I. Introduction

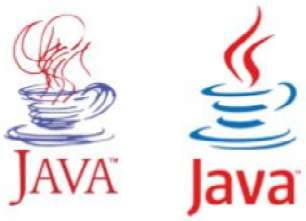
- Java est un langage de programmation orienté objet, évolué et à usage général. Il a été proposé à l'origine par Sun Microsystems et maintenant par Oracle depuis son rachat de Sun Microsystems en 2009.
- Java a été conçue avec deux objectifs principaux :
  - ➢ Permettre aux développeurs d'écrire des logiciels indépendants de l'environnement *hardware* d'exécution.
  - ➢ Offrir un langage orienté objet avec une bibliothèque standard riche

II. Bref historique de java

version	date	faits notables
1.0	janvier 1996	La naissance
1.1	février 1997	Ajout de JDBC et définition des JavaBeans
1.2	décembre 1998	Ajout de Swing, des collections (JCF), de l'API de réflexion. La machine virtuelle inclut la compilation à la volée (Just In Time)
1.3	mai 2000	JVM HotSpot
1.4	février 2002	support des regexp et premier parser de XML
5	septembre 2004	évolutions majeures du langage : autoboxing, énumérations, varargs, imports statiques, foreach, types génériques, annotations. Nombreux ajout dans l'API standard
6	décembre 2006	
7	juillet 2011	Quelques évolutions du langage et l'introduction de java.nio
8	mars 2014	évolutions majeures du langage : les lambdas et les streams et une nouvelle API pour les dates
9	septembre 2017	les modules (projet Jigsaw) et jshell
10	mars 2018	inférence des types pour les variables locales (mot-clé `var`)
11	septembre 2018	Nouvelle licence : la version propriétaire de Oracle JDK n'est plus utilisable en production. Oracle fournit cependant une version libre sous licence GPL (http://jdk.java.net/) Suppression de certains modules dépréciés en Java 9 (CORBA, JAXB, JAX-WS...). Nouvelle API de client HTTP.
12	mars 2019	Shenandoah : le nouvel algorithme de ramasse-miettes (*Garbage collector*). Extension de l'expression `switch`.

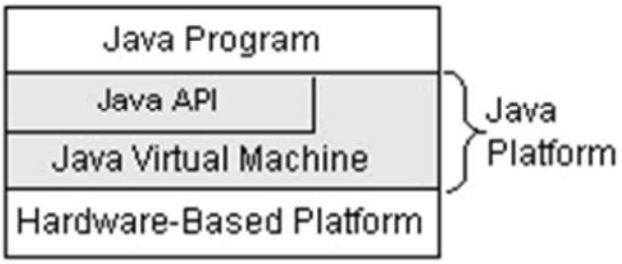
Remarque :

- Java a connu deux logos durant son histoire.



III. Caractéristiques

1. Java possède de multiples caractéristiques qui ont contribué à son énorme succès :
   - Java est orienté objet
   - Java est compilé/interprété
   - Java est portable
   - Java est fortement typé
   - Java est multitâche
   - Java est simple
   - Java assure la gestion de la mémoire
   - Java est sûre
   - Java est économe
   - Java est gratuit
2. Java est une plateforme
   - La plateforme Java, uniquement software, est exécutée sur la plateforme du système d'exploitation
   - La « Java Platform » est constituée de :
     - La « Java Virtual Machine » (JVM)
     - Des interfaces de programmation d'application (Java API)



3. Structure fondamentale du langage

- Le langage source Java est défini par la JLS (Java Language Specification) éditée par Sun-Oracle

- La syntaxe du langage Java est à l'origine très inspirée du C et du C++.
  - Java est sensible à la casse.
  - Les blocs de code sont encadrés par des accolades. Chaque instruction se termine par un caractère ';' (point−virgule).
  - Une instruction peut tenir sur plusieurs lignes

- Les programmes Java peuvent être écrits de deux manières :

a) Sous forme **d'une application Java** qui est composée d'une classe possédant une **méthode main ()** :

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

  //code à exécuter pour initialiser l'application

}
```

Exemple d'une application Java:

```
public class HelloInf {

        /* Un style de commentaire
        sur plusieurs lignes. */

    public static void main (String [] args) {

          // Un commentaire sur une seule ligne
    System.out.println ("Bonjour à vous les L2 INF");
     }
    }
```

Remarques :

- ✓ toute classe publique doit être dans un fichier qui a le même nom que la classe
- ✓ tout code doit être à l'intérieur d'une classe, ça définit une unité de compilation
- ✓ Comme il y a une méthode main, cette classe est « exécutable »
- ✓ L'environnement d'exécution dépend de l'OS de la machine
- ✓ Pas de restriction au niveau des API
- ✓ Le code peut être écrit dans un simple éditeur de texte (Compilation et exécution du code en ligne de commande DOS) Ou en utilisant un environnement de développement (IDE) a titre d'exemple :
  - Eclipse (https://www.eclipse.org/)
  - Netbeans (https://netbeans.org)
  - ……
- ✓ Le code source d'une classe contenue dans un fichier est compilé avec la commande javac.Cela produit un code intermédiaire, appelé bytecode,
- ✓ Le bytecode d'une classe est destiné à être chargé par une machine qui doit l'exécuter avec la commande java. L'argument est le nom de la classe (sans l'extension .class)

b) Sous forme **d'une applet Java** qui comprend une classe publique dérivant de java.applet.Applet

Remarques

- ✓ L'environnement d'exécution dépend du browser Web
- ✓ Restrictions au niveau des API
- ✓ Généralement pas autorisée à lire ou écrire sur des fichiers locaux.
- ✓ Interdiction d'ouvrir des connections réseaux vers d'autres systèmes que la machine hôte qui a chargé l'applet

Dans ce cours nous allons nous intéressés particulièrement aux applications Java.

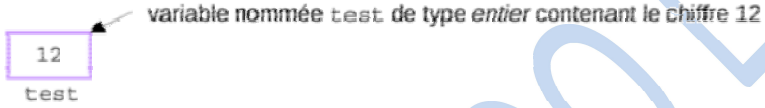
- Java adopte un style de nommage (très fortement) conseillé :
  - ✓ Style « chameau » (CamelCase) pour les indentificateurs
  - ✓ Première majuscule pour les classes (class HelloInf)
  - ✓ Première minuscule pour les variables/champs et les méthodes
  - ✓ Tout en majuscule pour les constantes (MAX_SIZE)

II. JAVA :

1. variables :

- Les variables constituent l'aspect le plus important de la programmation, puisqu'elles permettent de stocker dans un emplacement mémoire les données et de les transformer en utilisant des opérateurs.

- On peut représenter une variable comme une étiquette associée à une zone mémoire dans laquelle est rangée une valeur d'un certain type (entier ou réel) par exemple :



Remarque :

C'est en fonction du type de la variable que la taille de l'emplacement mémoire (en octet, soit 8 bits) et le codage binaire de la valeur seront déterminés.

1.1. Déclaration

- Avant de pouvoir utiliser une variable, il est nécessaire de la déclarer, cad d'associer la variable à un emplacement de la mémoire et de spécifier son type.
- La déclaration se fait par une instruction dont la syntaxe est :

typeVariable nomVariable;

1.2. Affectation :

nomVariable = uneValeur;

- L'affectation écrit dans l'emplacement mémoire associé à **nomVariable** la valeur **uneValeur**.
- On dit alors que **nomVariable** "prend la valeur" **uneValeur**.

Remarques :

- **uneValeur** doit être compatible avec le **type de nomVariable.**

- On peut également faire la **déclaration** et l'**initialisation** d'une variable en une seule instruction

typeVariable nomVariable = uneValeur;

- Le mot-clé final signifie en Java « affectation unique ».Il s'applique sur la variable qui est déclarée. Ce qui donne une constante.

1.3. Incrémentation et Décrémentation De Variables

- Les opérateurs d'incrémentation et de décrémentation sont : n++, ++n, n--, --n
- Si l'opérateur est placé avant la variable (préfixé), la modification de la valeur est immédiate sinon la modification n'a lieu qu'à l'issue de l'exécution de la ligne d'instruction (postfixé)

Exemple 1 :

```
System.out.println(x++);
// est équivalent à
System.out.println(x); x = x + 1;
```

L'opérateur ++ renvoie la valeur avant incrémentation

Exemple 2 :

```
System.out.println(++x);
// est équivalent à
x = x + 1; System.out.println(x);
```

L'opérateur ++ renvoie la valeur après incrémentation.

2. Les Types Primitifs :

On distingue 4 catégories de types primitifs :

- entiers,
- réels,
- booléens,
- caractères.

Remarque :

- L'intervalle de valeurs représentables pour chacun des types peut varier en fonction de l'espace mémoire qu'ils occupent.

2.1 Les entiers

En Java, les différents types d'entiers sont les suivants :

Nom	Taille	Intervalle représentable
byte	1 octet	$[-128 \dots 127]$ ou $[-2^7 \dots 2^7-1]$
short	2 octets	$[-32768 \dots 32767]$ $[-2^{15} \dots 2^{15}-1]$
int	4 octets	$[-2^{31} \dots 2^{31}-1]$
long	8 octets	$[-2^{63} \dots 2^{63}-1]$

Remarques :

- On peut appliquer des opérateurs arithmétiques **(+,-,*, /)** à deux variables ou deux expressions de type entier. Le résultat de cette opération est également du type entier.

- Lorsque les deux opérandes sont de type entier, l'opérateur / calcule la division entière et l'opérateur **%** calcule le reste de cette division.

Exemple :

```
int  valA = 7;
int  valB = 2;
int  valC = valA / valB;     // valC contient la valeur 3
int valD = valA % valB;  // valD contient la valeur 1
```

2.2 **Les réels**

En Java, il existe deux types de représentation pour les nombres réels: simple et double précision (respectivement les types float et double).

Nom	Taille	Intervalle représentable
`float`	4 octets	$[-3.4.10^{38}, \cdots, -1.4.10^{-45}, 0, 1.4.10^{-45}, \cdots, 3.4.10^{38}]$
`double`	8 octets	$[-1.8.10^{308}, \cdots, -4.9.10^{-324}, 0, 4.9.10^{-324}, \cdots, 1.8.10^{308}]$

Remarque :

- Lorsqu'au moins une des opérandes est de type réel, l'opérateur / calcule la division réelle.

Exemple :

```
double reelA = 7;
double reelB = 2;
double division = reelA / reelB;  // La variable division contient la valeur 3.5
```

2.3 Les Booléens

Le **type booléen** est un type de variable très utile en informatique. Ce type prend deux valeurs **VRAI** ou **FAUX**.

Nom	Taille	Intervalle représentable
boolean	1 octet	[true,false]

Remarques :

- On peut appliquer des opérateurs logiques (et, ou, non) à des variables ou des expressions booléennes. Le résultat de cette opération est également du type booléen.
- && désigne l'opérateur et logique
- || désigne l'opérateur ou logique
- ! désigne l'opérateur non logique (qui transforme une valeur true en valeur false et inversement)

Exemple :

```
boolean boolA = TRUE;
boolean boolB = FALSE;
boolean nonA = !boolA;              // nonA vaut FALSE
boolean AetB = boolA && boolB;      // AetB vaut FALSE
```

2.4 Les caractères

Le type caractère peut correspondre à n'importe quel symbole du clavier (lettre en majuscule ou minuscule, chiffre, ponctuation et symboles).

Nom	Taille	Intervalle représentable
char	2 octets	[a...z,A...Z,0...9,:,;.?!...]

Remarque :

- Pour distinguer la valeur correspondant au caractère a de la variable dénommée a, on utilise l'apostrophe pour la première.

Exemple :

```
char  caractere = 'a';     // La variable caractere contient la valeur a
int a = 3;                 // La variable a contient la valeur 3
```

2.5 Transtypage

Le transtypage également appelée conversions d'un type primitif en un autre se fait de la manière suivante :

typeVariableA variableA = (typeVariableA) valeurB

Remarque :

- Si variableA et valeurB ne sont pas du même type, cette instruction affecte à variableA la conversion de la valeur de valeurB dans le type typeVariableA :
  - Entier vers Réel : la même valeur codée en réel
  - Réel vers Entier : la partie entière du réel
  - Entier vers caractère : le caractère dont le code est l'entier
  - Caractère vers Entier : le code numérique correspondant au caractère

Exemple :

```
int i;
double x = 2;
i = (int) (x * 42.3);      // i vaut 84
```

Remarques :

- La conversion peut entrainer une perte d'informations.
- Il n'existe pas en Java de fonction pour convertir : les conversions de type se font par des méthodes. La bibliothèque de classes API fournit une série de classes qui contiennent des méthodes de manipulation et de conversion de types élémentaires.

Classe	Rôle
String	pour les chaînes de caractères Unicode
Integer	pour les valeurs entières (integer)
Long	pour les entiers longs signés (long)
Float	pour les nombres à virgule flottante (float)
Double	pour les nombres à virgule flottante en double précision (double)

- Les classes portent le même nom que le type élémentaire sur lequel elles reposent avec la première lettre en majuscule.
- Ces classes contiennent généralement plusieurs constructeurs. Pour y accéder, il faut les instancier puisque ce sont des objets :

```
String montexte;
montexte = new String("test");
```

L'objet montexte permet d'accéder aux méthodes de la classe java.lang.String.

Exemple de Conversion d'un entier en chaine de caractère :

```
int    i  = 10;
String montexte = new String();
montexte = montexte.valueOf(i);
```

Exemple de conversion d'une chaine de caractère en un entier :

```
String  montexte  = new String("10");
Integer monnombre = new Integer(montexte);
int     i  = monnombre.intValue(); // conversion d'Integer en int
```

2.6 Les opérateurs de comparaison :

Les opérateurs de comparaison permettent de comparer deux variables d'un même type primitif (entier, flottant, booléen et caractère) et renvoient une valeur booléenne :

- **== comparaison d'égalité**
- **!= différence**
- **< inférieur strict**
- **<= inférieur ou égal**
- **> supérieur strict**
- **>= supérieur ou égal**

Remarques :

- Attention, l'opérateur = correspond à l'affectation alors que l'opérateur == correspond à la comparaison d'égalité,
- Pour les expressions composées l'utilisation de parenthèses est obligatoire :

Exemple :

```
double a = 8;
 boolean estDansIntervalle = ((a >= 0) && (a <= 10));  // vaut true ssi a appartient à [0,10]
```

3. Les Structures de Contrôle

Les structures de contrôle permettent de spécifier si l'exécution d'un traitement est conditionnée ou bien si elle se fait de manière répétée.

3.1 **Bloc d'instructions**

- Un bloc d'instructions est un ensemble d'instructions qui vont être exécutées les unes à la suite des autres.
- Les accolades {} permettent de délimiter un bloc d'instructions.
- Les variables déclarées dans un bloc sont accessibles à l'intérieur de ce bloc uniquement.
- Un bloc d'instructions peut, par exemple, être exécuté que lorsqu'une condition est vérifiée ou bien il peut être exécuté plusieurs fois de suite. Ce sont les structures de **contrôle conditionnelles et itératives** qui permettent d'exprimer cela.

3.2 **Structures conditionnelles**

- Les structures de contrôle conditionnelles permettent de spécifier à quelles conditions un bloc d'instructions va être exécuté.
- Cette **condition** est exprimée par une **expression logique**.
- Deux types de structures de contrôle conditionnelles existent : l'alternative et à choix multiples.

a. La Structure Alternative

```
if (condition) {

        // bloc d'instructions exécuté si condition est vraie

}
else {

        // bloc d'instructions exécuté si condition est fausse

}
```

Remarques :

- if (condition) équivalent à (condition == true)
- Cette structure de contrôle exprime une alternative.
- Il est possible de vouloir qu'un bloc soit exécuté sous une certaine condition et que sinon, aucune instruction ne soit exécutée. Dans ce cas, la clause else et son bloc sont supprimés.
- Les parenthèses autour de condition, qui est variable ou une expression à valeur booléenne, sont obligatoires.

Exemple :

```
double moyAnnee = 12;
if (moyAnnee >=  10) {
System.out.println (" Etudiant Admis");
}
else {
System.out.println (" Etudiant Ajourné");
}
```

b. La Structure Choix Multiples

Le second type de structure conditionnelle permet de faire plusieurs tests de valeurs sur le contenu d'une même variable. Sa syntaxe est la suivante :

```
switch (variable) {
case valeur1 :
      Liste d'instructions // exécutées si (variable == valeur1)
 break;
case valeur2 :
     Liste d'instructions // exécutées si (variable == valeur2)
break;
...
case valeurN :
      Liste d'instructions // exécutées si (variable == valeurN)
break;
default:
      Liste d'instructions // exécutées sinon
 }
```

Remarques :

- Le mot clé **default** précède la liste d'instructions qui sont exécutées lorsque variable a une valeur différentes de valeur1,..,valeurN.
- Le mot clé **break** indique que la liste d'instructions est terminée.

3.3 Structures Itératives

Java propose 3 formes de structures itératives, chacune posséde un cadre d'utilisation bien spécifique:

1. while(condition)
2. do ... while(condition)
3. for

3.3.1 La boucle while()

- Une boucle while() permet d'exécuter l'instruction ou le bloc de code qui la suit tant que la condition booléenne est évaluée à true.
- La condition est évaluée en début de chaque itération.
- La syntaxe générale est de la forme :

```
while (condition) {

// code à exécuter dans la boucle
// bloc d'instructions répétées tant que condition est vraie.
// condition doit être modifiée dans ce bloc
}
```

Remarques :

- L'itération répétée tant qu'une condition est vraie
- Si avant l'instruction while, la condition est fausse, alors le code de la boucle ne sera jamais exécuté.
- Ne pas mettre de ; après la condition sinon le corps de la boucle ne sera jamais exécuté.

3.3.2 La boucle do while()

- C'est une variante de la boucle while()
- Permet d'exécuter l'instruction ou le bloc de code qui la suit tant que la condition booléenne est évaluée à vrai. La condition est évaluée en fin de chaque itération, cad après que les instructions ont été exécutées.
- La syntaxe générale est de la forme :

```
do {

// bloc d'instructions exécutées

// condition doit être modifiée dans ce bloc

} while (condition);   // si condition est vraie le bloc est exécuté à nouveau
```

Remarques :

- Ne pas oublier le ; après la condition d'arrêt.
- Cette boucle est au moins exécutée une fois quelle que soit la valeur du booléen.

3.3.3 La boucle for

- La boucle for permet de répéter un bloc d'instructions un nombre de fois fixé.

```
for (initialisation; condition; mise à jour de valeurs) {

//  Instructions

}
```

- Dans sa syntaxe, il faut :
  - déclarer et initialiser la variable qui sert de compteur de tours de boucle,
  - indiquer la condition sur le compteur pour laquelle la boucle s'arrête

Exemples :

```
for (int compteur = 0 ; compteur < n ; compteur = compteur + 1) {
// bloc instructions répétées n fois
}
Ou
for (int compteur = n ; compteur > 0 ; compteur = compteur - 1) {
// bloc instructions répétées n fois
}
```

Remarques :

- L'initialisation, la condition et la modification du compteur sont optionnelles.
- Dans l'initialisation, on peut déclarer une variable qui servira de compteur et qui sera dans ce cas locale à la boucle.
- Il est possible d'inclure plusieurs traitements dans l'initialisation et la modification de la boucle : chacun des traitements doit être séparé par une virgule :

```
for (i = 0, j = 0 ; i * j < 1000; i++, j+= 2) {....}
```

- La condition peut ne pas porter sur le compteur de la boucle :

```
boolean trouve = false;
for (int i = 0 ; !trouve ; i++ ) {
if (tableau[i] == 1) {
trouve = true;
... //gestion de la fin du parcours du tableau
}
}
```

3.4 Les débranchements BREAK / CONTINUE

- **BREAK**: achève immédiatement la boucle
- **CONTINUE**: ignore le reste des instructions et recommence au début de la boucle

Exemples :

```
for (int i=0; i<10 ;i++){

    if (i==5) continue; // Si i=5, on recommence au début

    if (i==7) break;   /* Si i=7, on sort de la boucle et
                          les instructions suivantes sont
                          exécutées */

    System.out.println ("la valeur de i est : " + i);
}
```

Remarques :

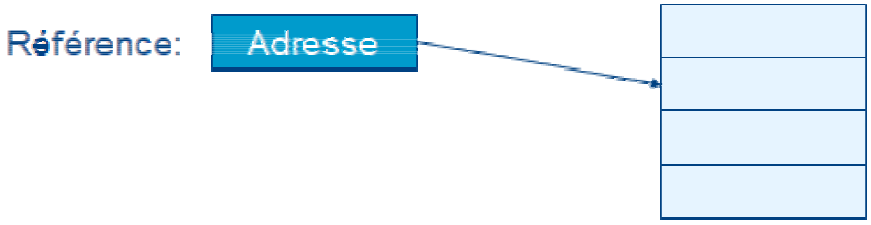
- L'instruction break permet de quitter immédiatement une boucle ou un branchement. Elle est utilisable dans tous les contrôles de flot.
- L'instruction continue s'utilise dans une boucle pour passer directement à l'itération suivante
- break et continue peuvent s'exécuter avec des blocs nommés. Il est possible de préciser une étiquette pour indiquer le point de retour lors de la fin du traitement déclenché par le break.
- Une étiquette est un nom suivi d'un caractère deux-points qui définit le début d'une instruction.
- l'utilisation dans le code des débranchements n'est pas recommandée.
- Il est possible de nommer une boucle à l'aide d'une étiquette pour permettre de l'interrompre même si cela est peu recommandé :

```
int compteur = 0;
boucle:
while (compteur < 100) {

  for (int compte = 0 ; compte < 10 ; compte ++) {
    compteur += compte;
    System.out.println ("compteur = "+compteur);
    if (compteur > 40) break boucle;
  }
}
```

4. Les types de Références

- **Une référence**: contient l'adresse mémoire où l'information relative à l'objet est réellement stockée



- **Les types de références** sont tous les types hormis les types primitifs. Ils sont définis soit dans les APIs (java.lang.Object, java.lang.String, java.util), soit ces des types cachés (tableau), soit par l'utilisateur.

Remarques :

- Une référence est un « Pointeur implicite » sur un objet.
- Les objets sont manipulés via des références.

4.1 La classe STRING

- Une chaîne de caractères est simplement une suite de caractères où chaque caractère fait partie du jeu de caractères du langage. Dans le langage de programmation Java, et dans presque tous les langages, une chaîne de caractères s'écrit entre des double-quote (").

  Exemple : "Java" est une chaîne de caractères.

- Les chaînes de caractères sont des objets de la classe String.
- String n'est pas un type primitif, c'est une classe.
- La classe String est prédéfinie en Java.

- La déclaration de deux chaines de caractères s1 et s2 se fait par:
  String s1, s2;

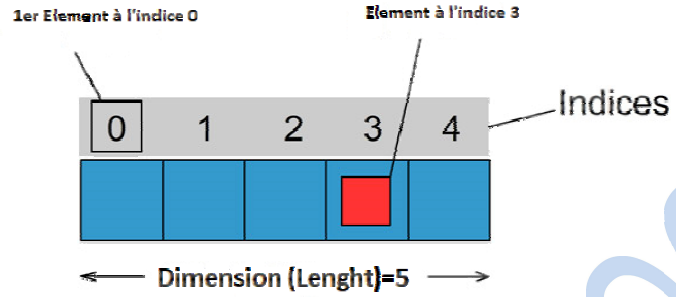
- Initialisation :
  S0= null ;
  s1 = "Bonjour";
  s2 = "les L2 informatique";

- Ou Déclaration et initialisation :
  String s3 = "Bonjour";

- Concaténation :
  String s4 = s1 + " " + s2;

4.2 Les tableaux

- Les **tableaux** (arrays en anglais), ne sont pas des objets mais une structure de données ordinaire que l'on trouve dans tous les langages de programmation.
- Un tableau est l'équivalent d'une table de taille fixe contenant des valeurs d'un type donné. La taille de la table est appelée sa dimension.
- Le type tableau est un type générique. Ils ne sont pas extensibles.



- Un tableau peut contenir des valeurs scalaires de type int, double, boolean, etc. Il peut également contenir des références sur des objets d'un type donné.
- Une variable de type tableau se déclare en faisant suivre le type des éléments du tableau par des crochets [].

Exemple :

```
String [] names ; // un tableau de références sur des String

int [] values ;   // un tableau de valeurs de type int
```

- Une variable de type tableau doit être initialisée en créant un tableau. Sinon, un accès à un élément du tableau provoquera une erreur.

- L'initialisation se fait avec l'instruction new suivi du type des éléments du tableau suivi de la dimension entre crochet.

Exemple :

```
String [] names = new String [10] ; // dimension 10

int [] values = new int[5] ;       // dimension 5
```

Remarques :

- La dimension du tableau est choisie à la création du tableau. Elle ne peut plus être modifiée.
- Les éléments d'un tableau de dimension n sont numérotés de 0 à n-1.
- Le numéro d'un élément dans un tableau est appelé son indice.
- Pour accéder à un élément du tableau à partir de son indice, on fait suivre le nom de la variable de type tableau du crochet ouvrant, de l'indice et du crocher fermant.

Exemple 1:

Si l'on a déclaré et initialisé la variable names :

```
String[] names = new String[10] ;
```

On peut accéder à l'élément d'indice 7 avec la notation names[7] :

```
names[7] = "java" ;
System.out.println("names[7] = “ + names[7]) ;
```

- En plus des tableaux unidimensionnels nous pouvons parfois travailler avec des tableaux bidimensionnels ou des tableaux multiples (tableaux de tableaux).
- length est une propriété (property) du tableau. En cas de tableaux bidimensionnels,
- cette propriété est le nombre de lignes du tableau.
- Java fournit quelques méthodes statiques utilitaires pour les tableaux, telles que le tableau d'arrangement, l'assignement des valeurs à tous les éléments du tableau, la recherche, la comparaison des tableaux etc. Ces méthodes sont définies dans la classe Arrays.
- à partir de Java 5, java fournit une boucle for-each qui vous permet de traverser (traverse) tous les éléments d'un tableau sans utiliser de variable d'index.

Exemple2 :

```
public class ArrayExample2 {

    public static void main(String[] args) {

        // Déclarez un tableau de 5 éléments
        int[] myArray = new int[5];

        // Imprimez à l'écran l'éléments count
        System.out.println("Array Length=" + myArray.length);

        // Utilisez une boucle for pour assigner la valeur aux éléments du tableau.
        for (int index = 0; index < myArray.length; index++) {
            myArray[index] = 100 * index * index + 3;
        }

        // Imprimez à l'écran l'élément à l'index  3
        System.out.println("myArray[3] = " + myArray[3]);
    }

}
```

Exemple3

```
import java.util.Arrays;
import java.util.List;

public class ArraysExample {

public static void main(String[] args) {

int[] years = new int[] { 2001, 1994, 1995, 2000, 2017 };

// Arrangez
Arrays.sort(years);

for (int year : years) {
System.out.println("Year: " + year);
}

// Convertissez un tableau en chaîne (string)
String yearsString = Arrays.toString(years);

System.out.println("Years: " + yearsString);

// Créez une liste (List) de plusieurs valeurs.
List<String> names = Arrays.asList("Tom", "Jerry", "Donald");

for (String name : names) {
System.out.println("Name: " + name);
}
```

Exemple 4 : Trois façons pour déclarer un tableau bidimensionnel

```
// Déclarez un tableau de 5 lignes, 10 colonnes
MyType[][] myArray1 = new MyType[5][10];

// Déclarez un tableau bidimensionnel de cinq lignes.
// (Tableau de tableaux)
MyType[][] myArray2 = new MyType[5][];

// Déclarez un tableau bidimensionnel, en précisant la valeur des éléments.
MyType[][] myArray3 = new MyType[][] {

   { value00, value01, value02 , value03 },
   { value10, value11, value12 }
};

// ** Remarque :
// MyType est soit du type primitif (byte, char, double, float, long, int, short, boolean)
//ou soit du type de référence.
```

Remarques :

- Si les valeurs des éléments d'un tableau de types primitifs (byte, char, double, float, long, int, short, boolean) ne sont pas précisées à la déclaration, ces éléments adoptent des valeurs par défaut.
  - La valeur par défaut 0 correspond à **byte**, **double**, **float**, **long**, **int**, **short**.
  - La valeur par défaut **false** correspond au type **boolean**.
  - La valeur par défaut ' **u0000**' (caractère null) correspond au type **char**.