

POO 2022

En Java

Jijel 2022



1

Introduction à Java

Introduction à Java

Définition

La technologie **Java** définit à la fois un langage de programmation **orienté objet** et une plateforme informatique. Créeée par l'entreprise Sun Microsystems (Sun) en **1995**, et reprise depuis par la société Oracle en **2009**.

Java s'appuie sur la syntaxe du langage **C** et du **C++**

Syntaxe de base de Java

```
1 package javaapp;
2 import java.util.Scanner;
3 //JavaApp.java: Somme de deux nombres entiers
4 public class JavaApp {
5 public static void main(String[] args) {
6     // TODO code application logic here
7     System.out.println("Premier programme Java !");
8     Scanner in = new Scanner(System.in);
9     int a,b,som;
10    System.out.print("a = ");
11    a = in.nextInt();
12    System.out.print("b = ");
13    b = in.nextInt();
14    som=a+b;
15    System.out.println("a+b = "+som);
16 }
17 }
```

Lecture/écriture

```
1 package javaapp;
2 import java.util.Scanner;
3 //JavaApp.java: Somme de deux nombres entiers
4 public class JavaApp {
5 public static void main(String[] args) {
6     // TODO code application logic here
7     System.out.println("Premier programme Java !");
8     Scanner in = new Scanner(System.in);
9     int a,b,som;
10    System.out.print("a = ");
11    a = in.nextInt();
12    System.out.print("b = ");
13    b = in.nextInt();
14    som=a+b;
15    System.out.println("a+b = "+som);
16 }
17 }
```

Lecture/écriture

```
1 Scanner input = new Scanner(System.in); //import java.util.Scanner  
2 ;  
3     int n=input.nextInt();  
4     double d=input.nextDouble();  
5     float f=input.nextFloat();  
6     String s=input.nextLine();  
7     char c=input.next().charAt(0);  
8     //and so on ...
```

Types de données primitifs (de base)

| Type | Description | Valeurs |
|---------|---|--|
| byte | un entier signé sur 8 bits | de -128 à +127 |
| short | un entier signé sur 16 bits | de -32 768 à +32 767 |
| int | un entier signé sur 32 bits | de -2147483648 à 2147483647 |
| long | un entier signé sur 64 bits | de l'ordre de (\pm) 9 milliards de milliards |
| float | un réel sur 32 bits | de l'ordre de 1.4E-45 à 3.4E38 |
| double | un réel sur 64 bits | de l'ordre de 4.9E-324 à 1.79E308 |
| boolean | vrai ou faux | true ou false |
| char | un caractère Unicode entier positif sur 16 bits | entre 0 et 65535 |

Types de données primitifs (de base)

| Type | Description | Valeurs |
|------|----------------------------|----------------|
| byte | un entier signé sur 8 bits | de -128 à +127 |

- R pour chaque type de données primitifs, des constantes indiquant les maximums et les minimums sont définies, par exemple : `Integer.MIN_VALUE`, `Integer.MAX_VALUE`, `Byte.MIN_VALUE`, `Byte.MAX_VALUE`, `Short.MIN_VALUE`, //etc.
- R En Java, une variable déclarée et non initialisée ne peut pas être utilisée et une erreur de compilation va survenir.

| | | |
|------|---|------------------|
| char | un caractère Unicode entier positif sur 16 bits | entre 0 et 65535 |
|------|---|------------------|

Opérateur arithmétiques et logiques

| Opérateur | Signification | Exemple |
|-----------------|------------------------|---------------------------------|
| + | addition | <code>n=a+b;</code> |
| - | soustraction | <code>n=a-b;</code> |
| * | multiplication | <code>n=a*b;</code> |
| / | division | <code>n=a/b</code> |
| % | modulo | <code>n=a%b;</code> |
| ++ | incrémente la variable | <code>i++;</code> |
| -- | décrémente la variable | <code>i--;</code> |
| <code>+=</code> | - | <code>n+=b; ⇒ n = n + b;</code> |
| <code>-=</code> | - | <code>n-=b; ⇒ n = n - b;</code> |
| = | affectation | <code>a = b;</code> |

Opérateurs arithmétiques et logiques

| Opérateur | Signification | Exemple |
|-----------|--------------------|--------------|
| > | supérieur | $a > b;$ |
| < | soustraction | $a < b$ |
| \geq | supérieur ou égale | $a \geq b$ |
| \leq | inférieur ou égale | $\leq b$ |
| \equiv | égale | $a \equiv b$ |
| \neq | différent à | $a \neq b$ |
| $\&\&$ | et logique | $a \&\& b$ |
| $\ $ | ou logique | $a \ b$ |
| ! | négation | $!trouve$ |

L'opérateur de cast (conversion de types)

```
1 //converting a smaller type to a larger type size
2 byte -> short -> char -> int -> long -> float -> double
3 //example:
```

```
4 int n = 1, p = 5;
5 float x = 1.5f;
6 float b= n * x + p; // Automatic casting: float to double
7 long a=n;           // Automatic casting: int to long
8 double d =n+x;     // Automatic casting: int to double
9 char c='a';
10 int e=c;           // Automatic casting: char to int
```

L'opérateur de cast (conversion de types)

```
1 //converting a smaller type to a larger type size  
2 byte -> short -> char -> int -> long -> float -> double  
3 //example:
```

```
1 // converting a larger type to a smaller size type  
2 double -> float -> long -> int -> char -> short -> byte  
3  
4 //Example:  
5 double d = 9.78d;  
6 int myInt = (int) d; // Manual casting: double to int  
7 float f=3;  
8 f=(float)(d+3); // Manual casting: double to float  
9 n= (int) f ; // Manual casting: float to int  
10 char c='a';  
11 c=(char) e; // Manual casting: int to char
```

Structures de contrôles

Les instructions de contrôle du déroulement lors de l'exécution des instructions ont la même syntaxe que pour le langage C.

Structures de contrôles

```
1 //if without else
2   if(test) //test is a logic expression
3     { block_of_instructions }
4
5 //if with one instruction
6   if(test) one_instruction;
7
8 //if with else
9   if(test)
10    { block_of_instructions }
11  else
12    { block_of_instructions }
13
14 //if with else if
15   if(test)
16    { block_of_instructions
17    }
18  else if
19    { block_of_instructions }
```

Structures de contrôles

```
1 //for with one instruction
2     for(int i=0;i<n;i++) one_instruction;
3
4 //for with several instructions
5     for(int i=0;i<n;i++)
6     {
7         block_of_instructions ;
8     }
9
10 //while with one instruction
11    while(condition) one_instruction;
12
13 //while with one instruction
14    while(condition)
15    {
16        block_of_instructions ;
17    }
18
19 //do ... while
20    do
21    {
22        block_of_instructions ;
23    } while(condition) ;
```

Tableaux

Plusieurs éléments de même type peuvent être regroupés en tableau. On peut accéder à chaque élément à l'aide d'un indice précisant le rang de l'élément dans le tableau. Le premier élément porte l'indice 0. Ils sont initialisés par défaut à 0 pour les nombres et à false pour les tableaux de booléens. En Java, on peut déclarer un tableau de différentes manières.

Tableaux

```
1 int tab1[] = {1, 2, 3}; // identique au langage C
2
3 int [] tab2 = {1, 2, 3};
4
5 int [] tab3 ;
6 tab3=new int[6]; /tab3 fait reference a un tableau de 6 entiers
7
8 int [] t1,t2; //t1 et t2 sont des references a des tableaux d'entiers
9
10 int t[10] ; // erreur: on ne peut pas indiquer de dimension ici
```

Tableaux

```
1 // ces trois déclarations sont équivalentes
2 int t [] [] ;
3 int [] t [] ;
4 int [] [] t ;
5
6 //un tableau de deux lignes et un nombre varie des colonnes
7 int t [] [] = {new int [3], new int [2]};
```

Tableaux

R

Les éléments d'un tableau peuvent être de type quelconque, et pas seulement d'un type primitif. Il est possible en Java de déclarer des tableaux des objets, par exemple :

```
1 Person [] t ;
2 t = new Person[3] ;//Person est une classe définie par le
                   programmeur
3
4 String[] prenoms = new String [2]; //tableau de 2 String
5 prenoms[0] = "Michel"; //prenoms[0] référence la chaîne "
                           Michel"
6 prenoms[1] = "Josette"; //prenoms[1] référence la chaîne "
                           Josette"
```

La même chose pour les tableaux à deux dimensions, on peut déclarer des tableaux d'objets à deux dimensions.

Tableaux

Pour itérer ou parcourir un tableau, on utilise une boucle **for** (ou **while**). Aussi, il est nécessaire de connaître la taille d'un tableau. Pour cela, en java, on utilise l'instruction **tableau.length**.

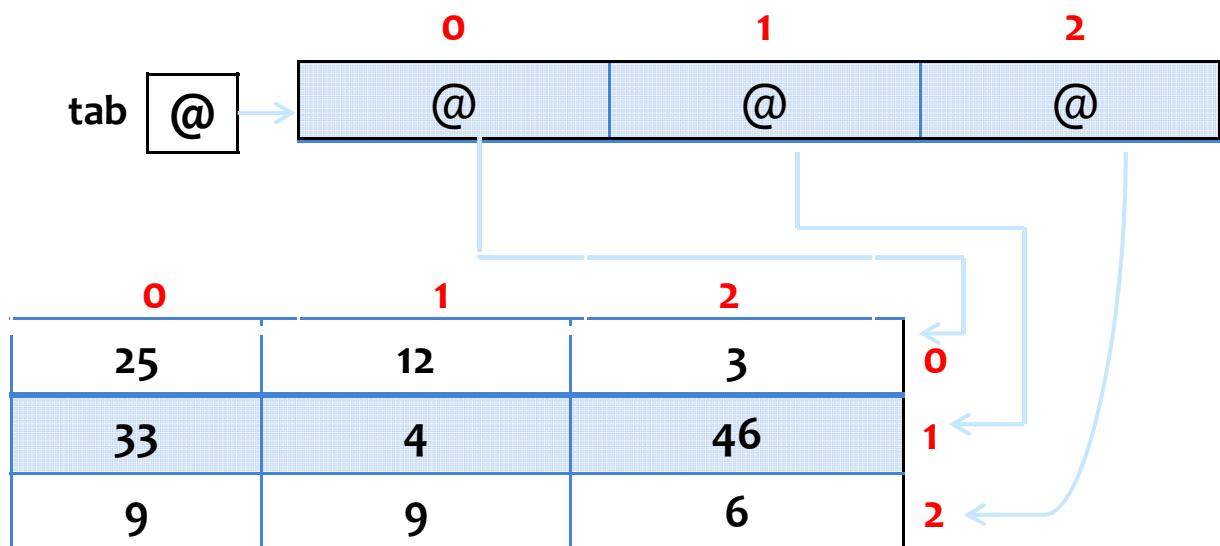
Tableaux

Pour itérer ou parcourir un tableau, on utilise une boucle **for** (ou **while**). Aussi, il est nécessaire de connaître la taille d'un tableau. Pour cela, en java, on utilise l'instruction **tableau.length**.

```
1 int t[] = new int[5] ;//tableau de 1 dimension
2 int somt=0, somtt;
3 for (int i =0; i<t.length; i++) somt+=t[i] ;
4 System.out.println("La somme des élément de t="+som);
5
6 int tt[] = new int[5][7] ;//tableau de 2 dimension
7 for (int i =0 ; i<tt.length ; i++)
{
8     for (int j=0; j<tt[i].length; j++) somtt+=t[i][j] ;
9 }
10
11 System.out.println("La somme des élément de tt="+somtt);
```

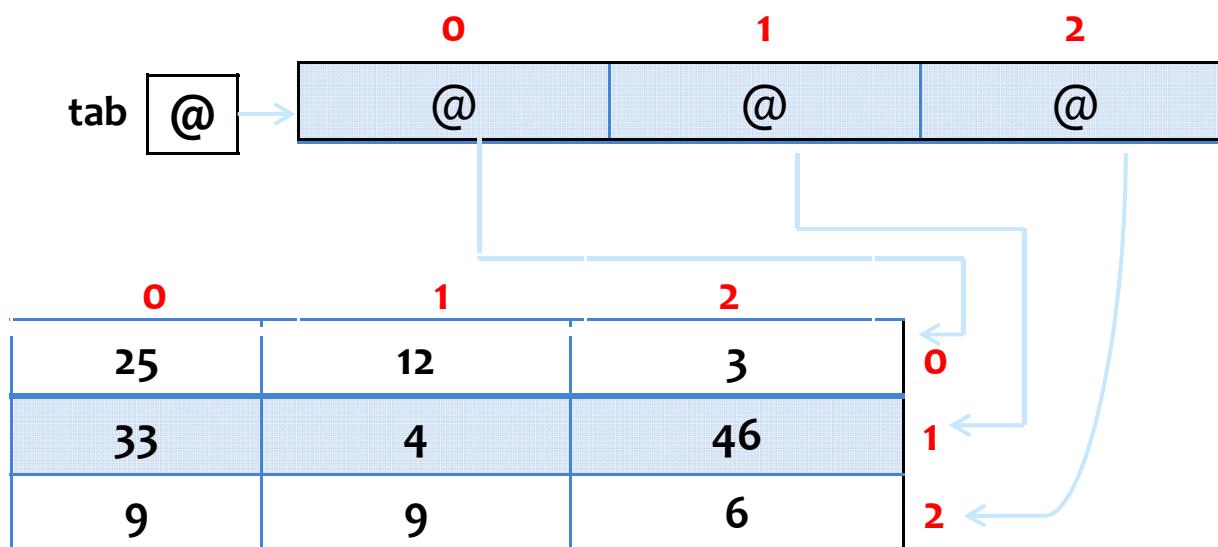
Tableaux

```
1 int [][] tab=new int [3][3];
2 // i=0
3 tab [0] [0]=25;tab [0] [1]=12;tab [0] [2]=3;
4 // i=1
5 tab [1] [0]=33;tab [1] [1]=4;tab [2] [2]=46;
6 // i=2
7 tab [2] [0]=9;tab [2] [2]=9;tab [2] [2]=6;
8
9 //une autre maniere d'initialisation du tab
10 int [][] tab={{25,12,3},{33,4,46},{9,9,6}};
```



Tableaux

```
1 int [][] tab=new int [3][3];
2 // i=0
3 tab [0] [0]=25; tab [0] [1]=12; tab [0] [2]=3;
4 // i=1
5 tab [1] [0]=33; tab [1] [1]=4; tab [1] [2]=46;
6 // i=2
7 tab [2] [0]=9; tab [2] [1]=9; tab [2] [2]=6;
8
9 //une autre maniere d'initialisation du tab
10 int [][] tab={{25,12,3},{33,4,46},{9,9,6}};
```



Chaines de caractères

pour créer et manipuler des chaînes de caractères, l'environnement JAVA définit trois classes: **String**, **StringBuilder** et **StringBuffer**.

Pour initialiser une variable de type **String**, on utilise les **guillemets ""** par exemple : **String m = "Bonjour";** Ou:

String m=new String("Bonjour");

Comme les tableaux, une variable de type **String** contient une **référence** vers une chaîne de caractères. Donc, la sémantique des opérateurs **d'affectation** et **d'égalité** est la même que les tableaux.

Chaines de caractères

| Méthode | Résultat renvoyé |
|-------------------------|---|
| <i>length</i> | Nombre de caractères contenus dans la chaîne courante |
| <i>charAt</i> | Le caractère dont la position dans la chaîne courante est passée en paramètre |
| <i>compareTo</i> | Compare la chaîne courante avec une chaîne passée en paramètre. renvoie zéro si les chaînes sont identiques, une valeur entière négative si l'objet courant est inférieur au paramètre, et une valeur entière positive si l'objet courant lui est supérieur |
| <i>endsWith</i> | Teste si la chaîne courante se termine par le suffixe passé en paramètre. Renvoie <i>true</i> ou <i>false</i> |
| <i>equals</i> | Compare cette chaîne à l'objet spécifié. Le résultat est vrai si et seulement si l'argument n'est pas <i>nul</i> et est un objet String qui représente la même séquence de caractères que cet objet |
| <i>equalsIgnoreCase</i> | Compare cette chaîne à une autre chaîne, en ignorant la casse (majuscule et minuscule). Renvoie <i>true</i> ou <i>false</i> |
| <i>contains</i> | Renvoie <i>true</i> si et seulement si cette chaîne contient une sous chaîne passée en paramètre |
| <i>indexOf</i> | renvoie la première position du caractère passé en paramètre dans la chaîne courante |
| <i>isEmpty</i> | Renvoie <i>true</i> si, et seulement si, <i>String.length</i> vaut 0 |
| <i>toLowerCase</i> | Convertit tous les caractères de cette chaîne en minuscule |
| <i>toUpperCase</i> | Convertit tous les caractères de cette chaîne en majuscule |

Chaines de caractères

```
1 String c1 = new String ( "bonjour " ) ; // ou String c1="bonjour "
2 String c2 ;
3 c2 = c1 + "á tous " ;
4 String c3 ="BonJour á Tous ";
5 System.out.println( c2 ) ;// bonjour á tous
6 System.out.println(c3) ;// BonJour á Tous
7 System.out.println( c2.length());
8 System.out.println ( c2.charAt( 3 ) );
9 System.out.println ( c2.indexOf('o') );
10 System.out.println( c1.contains("á tous ") );
11 System.out.println( c3.equalsIgnoreCase(c2));
```



Les chaînes de caractères d'un objet de type String ne peuvent pas être modifiés. Une même chaîne de caractères de la classe String peut être référencée plusieurs fois puisqu'on est sûr qu'elle ne peut pas être modifiée. Par contre, la classe *StringBuffer* permet la création et la modification d'une chaîne de caractères.

Tableaux dynamiques

Les **tableaux dynamiques** (**liste**) ont la particularité de pouvoir changer de taille pendant l'exécution du programme. En Java, on les définit à l'aide de type ***ArrayList*** de la bibliothèque:

java.util.ArrayList.

Pour ce faire, en écrit :

```
ArrayList <Type> myList;  
myList = new <Type> ArrayList();
```

Le type des éléments d'un **tableau dynamique** ne peut être un type simple (**int** ou **double** par exemple).

Tableaux dynamiques

Il doit être l'un des types **évolués** correspondant aux types de données simples. Par exemple : **Integer** pour **int**, **Float** pour **float**, **Double** pour **double**, etc.

Comme il peut être une classe spécifique (**Person** par exemple).

On peut aussi déclarer un tableau dynamique comme suit :

```
1 ArrayList myList;  
2 myList = new ArrayList();
```

Dans cette déclaration, il n'y a **aucune restriction** sur le **type** d'objets qu'on peut ajouter au tableau

Tableaux dynamiques

- ★ `tableau.add(valeur)` ajoute *valeur* à la fin de tableau
- ★ `tableau.size()` : renvoie la taille du tableau (de type int) ;
- ★ `tableau.set(i,valeur)` : affecte *valeur* à la case *i* du tableau
- ★ `tableau.get(i)` : renvoie l'élément à l'indice *i* dans le tableau (*i* doit être un entier compris entre 0 et `tableau.size()-1`)
- ★ `tableau.isEmpty()` : détermine si le tableau est vide ou non
- ★ `tableau.clear()` : supprime tous les éléments du tableau
- ★ `tableau.remove(i)` : supprime l'élément à l'indice *i* du tableau

```
1 ArrayList <String> sample = new ArrayList<String>();
2 sample.add("One Java");
3 sample.add("Two Java");
4 sample.add("Three Java");
5 System.out.println(sample.size());
6 System.out.println(sample.get(2));
7 System.out.println(sample.get(3));
```

Introduction à Java

```
1 //soit les tableaux suivants, tab1, tab2, tab3 et m
2 //Rechercher et afficher la valeur max dans les trois premiers
   tableaux
3
4     int tab1 []={2,5,16,9,3};
5     int tab2 []={1,5,6,9,13};
6     int tab3 []={12,5,6,9,3};
7     int m [] []={{12,5,6,9,3}, {1,5,6,9,13}, {2,5,16,9,3}};
8 //
```

Fonctions (méthodes)

Les **fonctions (Méthodes)** en Java en programmation sont des traitements, qui agissent sur des données.

Les **méthodes** sont des morceaux de code **réutilisables**, que l'on définit à un endroit du programme et que l'on peut appeler depuis un ou plusieurs autre(s) endroit(s)

Fonctions (méthodes)

Les méthodes permettent de:

- éviter la **duplication** de code
- facilement **maintenir** du code (si on veut changer le code, il suffit de changer le code de la fonction)
- avoir un programme bien **structuré** et facile à **comprendre**
- faciliter la **réutilisation** du code

Fonctions (méthodes)

Une fonction est caractérisée par :

- **un corps** : la portion de code à exécuter
- **un nom** : celui par lequel on désignera cette fonction
- **des paramètres** (des **arguments**) : ensemble de variables (extérieures à la fonction) que la fonction prend en entrée, et dont le corps a besoin pour fonctionner
- **un type et une valeur de retour** : la sortie de la fonction, ce qu'elle renvoie au reste du programme
- **l'appel** est l'endroit où l'on utilise la méthode. Il contient le nom de la méthode et les **arguments** que l'on veut lui passer

Fonctions (méthodes)

Syntaxe:

```
1 return_type name_method (type p1, type p2, ...)  
2 {  
3     // Body of the method goes here  
4 }
```

```
1 public static void main(String[] args) {  
2     int x = 10, y = 15;  
3     System.out.print(somme(x,y)); //appel de la fonction somme  
4 }  
5  
6 //définition de la fonction somme  
7 static int somme(int a, int b) {  
8     return a+b;  
9 }
```



Parfois, une méthode ne renvoie pas de valeur. Le mot clé *void* est utilisé comme type de retour si une méthode ne renvoie aucune valeur.

Introduction à Java

```
1 //soit les tableaux suivants, tab1, tab2, tab3 et m
2 //1. Rechercher et afficher la valeur max dans les trois premiers
   tableaux
3
4     int tab1 []={2,5,16,9,3};
5     int tab2 []={1,5,6,9,13};
6     int tab3 []={12,5,6,9,3};
7     int m [][]={{12,5,6,9,3}, {1,5,6,9,13}, {2,5,16,9,3}};
8
9 //2. Créer une fonction MaxTable qui renvoie comme valeur la valeur
   max dans un tableau
10 //3. Utiliser cette fonction pour afficher la valeur max des
    tableaux précédents
11 //4. Créer une fonction MaxMatrice qui renvoie comme valeur la
    valeur max dans une matrice en faisant appelle à la fonction
    MaxTab
12 //5. Utiliser cette fonction pour afficher la valeur max de la
    matrice m
```

Fonctions (méthodes)

Signature d'une méthode:

Une méthode a une **signature**, qui identifie de **manière unique** la méthode dans un contexte particulier.

La signature d'une méthode est la combinaison des quatre parties suivantes :

- ☆ **Nom** de la méthode
- ☆ **Nombre** de paramètres
- ☆ **Ordre** de paramètres
- ☆ **Types** des paramètres

Fonctions (méthodes)

Signature d'une méthode:

```
1  int somme(int a){  
2      return a+a;  
3  }  
4  
5  int somme(int a, int b){  
6      return a+b;  
7  }  
8  
9  int somme(int a, int b, int c){  
10     return a+b+c;  
11  }  
12  
13  double somme(double a, double b){  
14      return a+b;  
15  }
```

Fonctions (méthodes)

Évaluation d'un appel de méthode :

L'appel de méthode se passe par cinq étapes :

- 1) Les expressions passées en **argument** sont évaluées (en peut passer, lors de l'appel d'une fonction, comme arguments soit des **valeurs concrètes**, soit des **variables** soit des **expressions**)
- 2) Les valeurs correspondantes sont **affectées** aux **paramètres de la méthode** (les valeurs sont **copiées** dans les paramètres de la méthode)
- 3) Le corps de la méthode **s'exécute** avec ces valeurs
- 4) L'expression suivant la première commande **return** rencontrée est évaluée
- 5) La valeur obtenue est **retournée** comme résultat de l'appel : cette valeur remplace l'expression de l'appel

Fonctions (méthodes)

Évaluation d'un appel de méthode :

R

Si une méthode est sans arguments, les étapes 1 et 2 n'ont pas lieu. Ainsi, si la méthode est sans valeur de retour (type de retour *void*), les étapes 4 et 5 n'ont pas lieu.

Fonctions (méthodes)

Les références et passage de paramètres :

En programmation, **de façon générale**, on dira que :

- ☆ Un **argument** est passé par **valeur** si la méthode ne peut pas le **modifier**, la méthode **crée une copie locale** de cet argument (travaille avec une **copie**)
- ☆ Un **argument** est passé par **référence** (adresse) si la méthode peut le **modifier** (on connaît son adresse)

En **Java**, **il n'existe que le passage par valeur**, mais cela a des conséquences différentes selon que le type du paramètre est simple (int, double, etc.) ou évolué (Objet)

Fonctions (méthodes)

Les références et passage de paramètres :

Par exemple:

- pour les entiers le passage est par valeur.
- Par contre, pour les tableaux, le passage des paramètres s'effectue par adresse (par référence). On peut **accéder** et **modifier** les valeurs du tableau passé en paramètre

Fonctions (méthodes)

Les références et passage de paramètres :

```
1 // Qu'affiche ce code:  
2  
3 public static void main(String[] args) {  
4     int x = 10, y = 15;  
5     System.out.print(x + " " + y + " ");  
6  
7     f1(x, y);  
8     System.out.println(x + " " + y + " ");  
9  
10    String s1 = "abcd", s2 = "efgh";  
11    System.out.print(s1 + s2);  
12  
13    f2(s1, s2);  
14    System.out.print(s1 + s2);  
15 }  
16  
17 static void f1(int a, int b) {  
18     int tmp = a; a = b; b = tmp;  
19 }  
20  
21 static void f2(String s, String t) {  
22     String tmp = s; s = t; t = tmp;  
23 }
```

Fonctions (méthodes)

Les références et passage de paramètres :

```
1 // Qu'affiche le code suivant:  
2 public static void main(String[] args)  
3 {  
4     int[] tab = {1, 2, 3, 4};  
5     System.out.println(tab[0] + " " + tab[1]);  
6     f(tab, 0, 1);  
7     System.out.println(tab[0] + " " + tab[1]);  
8 }  
9  
10 static void f(int[] t, int i, int j)  
11 {  
12     int tmp = t[i];  
13     t[i] = t[j];  
14     t[j] = tmp;  
15     t = new int[4];  
16 }
```

Fonctions (méthodes)

Les références et passage de paramètres :

R

Les modifications faites dans la méthode sur la référence elle-même ne sont pas visibles à l'extérieur de la méthode.

```
1 public static void main(String[] args)
2 {
3     StringBuilder st=new StringBuilder ("Bonjour");
4
5     add(st);
6     replace(st);
7     System.out.println(st);
8
9 }
10
11
12 static void add(StringBuilder s){
13     s.append(" tout le monde");
14 }
15
16 static void replace(StringBuilder s){
17     StringBuilder ch=new StringBuilder("Hello every one");
18     s=ch;
19 }
```

Fonctions (méthodes)

```
1 // Crée un programme qui affiche les nombres premier de 1 à n en cr  
    éant les méthodes suivantes:  
2 //div(int i, int j): renvoie true si i est divisible sur j  
3 //prime(int n): renvoie true si n est premier, false sinon  
4 // liste_prime(int n) renvoie une liste des nombres premier de 1 à  
    n  
5 // afficherliste(n) affiche les éléments d'une liste  
6 //modifier le programme pour qu'il affiche les n premiers nombres  
    premiers (ajouter une méthode)
```

Introduction à Java

Exercice 13 En mathématiques, deux nombres entiers n et m sont dits amicaux ou aimables ou amiabiles si la somme des diviseurs de n (n exclus) vaut m et la somme des diviseurs de m (m exclus) vaut n . Autrement dit, si la somme des diviseurs de l'un q , coïncide avec la somme des diviseurs de l'autre ; et la somme des deux nombres vaut q .

Par exemple 220 et 284 sont amicaux car, la somme :

- ☆ des diviseurs de 220 = $1+2+4+5+10+11+20+22+44+55+110+\mathbf{220} = 504$
- ☆ des diviseurs de 284 = $1+2+4+71+142+\mathbf{284} = 504$
- ☆ $220 + 284 = 504$.

D'une autre manières :

- ☆ La somme des diviseurs **propres** de **220** est 284 : $\text{divp}(220) = 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110$ et la somme de ces nombres est **284**.
- ☆ La somme des diviseurs **propres** de **284** est 220 : $\text{divp}(284) = 1, 2, 4, 71, 142$ et la somme de ces nombres est **220**.

Voici quelques paires de nombres amicaux : (1184, 1210), (2620, 2924), (5020, 5564) et (6232, 6368).

Travail demandé : Modularisez le code au moyen de méthodes auxiliaires pour écrire une méthode **Amicaux** qui prend en entrée un tableau d'entiers et affiche toutes les paires de nombres amicaux qu'il contient. Chaque paire ne sera affichée qu'une seule fois.