

# Algorithmique & Programmation Orientée Objet

## Semestre 2 ST

### Les tableaux de variables

[Tableaux de variables](#)

[Algorithmes de recherche et de tri](#)

[Précision, rapidité et complexité](#)

[Matrices de variables](#)

[Récurtivité](#)

[Informations et Archives](#)

[Travaux dirigés et travaux pratiques](#)

[Evaluation intermédiaire](#)

[Sujets de projet](#)

[Problématique](#)

[Les tableaux](#)

[Syntaxe en langage algorithmique](#)

[Syntaxe en langage Java](#)

[Tableaux en paramètres d'entête et en retour de sous-algorithme](#)

[Tableaux de variables de type agrégé](#)

[Tableaux dans un type agrégé](#)

[Affectation et test d'égalité entre tableaux](#)

[Exemples et exercices](#)

[Cours](#)

[TD](#)

[TP](#)

[Version PDF](#)

[Clavier.class](#) - [Ecran.class](#) - [Chaine.class](#) - [Documentation](#)

### Problématique

- Comment manipuler un très grand nombre d'informations de même nature?
- Exemples:
  - Un relevé de notes
  - Les informations de compte pour les clients d'une banque
  - Les positions et caractéristiques des unités pour un jeu de simulation de champ de bataille
  - Le contenu des cases pour un jeu de dames (blanc, noir ou vide)

- Définition d'autant de variables individuelles que de données devant être gérées
- Définition d'un type agrégé regroupant autant de champs que de données devant être gérées
- Solutions possibles en théorie mais impossibles en pratique car implantation très complexe des tâches nécessitant la manipulation de l'ensemble de ces variables (parcours, recherche, tri, ...).

## Solution

- Définir une variable agrégeant un grand nombre de sous-variables de même type élémentaire
- Autoriser l'utilisation des sous-variables individuelles (du type élémentaire) au sein de cette variable en les désignant de façon unique par la donnée d'indices entiers (ou de plusieurs indices)

## Les tableaux

- **Définition**
  - "Tableau" (array en anglais): Variable agrégeant un nombre arbitraire N de sous-variables ("composantes", "éléments") de même type
- N fonction de la "dimension" D du tableau (le nombre d'indices) et de la "taille" selon chaque dimension (D tailles)
  - N égal au produit des D tailles
- Type, dimension et taille(s) spécifiés lors de la déclaration d'un tableau
- Type, dimension et taille(s) non modifiables par la suite

## Syntaxe en langage algorithmique

- **Syntaxe de déclaration d'une variable tableau en dimension 1**

**Tableau**[n] **de type** nomVariable

- n: Taille du tableau selon son unique indice
  - Expression numérique de résultat entier (constante littérale ou non, variable, expression arithmétique, appel de fonction, ...)
- type: Type élémentaire du tableau

- **Syntaxe de déclaration d'une variable tableau en dimension 2**

**Tableau**[n1][n2] **de type** nomVariable

- n1 et n2: Tailles entières (selon chacune des 2 dimensions)
- type: Type élémentaire du tableau

- **Syntaxe de déclaration d'une variable tableau en dimension D**

**Tableau**[n1][n2]...[nD] **de type** nomVariable

- n1, n2, ..., nD: Tailles entières (selon chacune des D dimensions)
- type: Type élémentaire du tableau

**ATTENTION:** Non initialisation du contenu des tableaux au moment de leur déclaration (i.e. leurs composantes ne sont pas initialisées et sont donc de valeur aléatoire)

- **Syntaxe d'accès aux composantes d'une variable tableau de dimension 1**

En lecture ou en écriture:

nomVariable[indice]

- indice: Constante entière (littérale ou non), variable entière ou expression à résultat entier

- **Syntaxe d'accès aux composantes d'une variable tableau de dimension 2**

En lecture ou en écriture:

nomVariable[indice1][indice2]

- indice1 et indice2: Constantes entières (littérales ou non), variables entières ou expressions à résultat entier

- **Syntaxe d'accès aux composantes en dimension D**

En lecture ou en écriture:

nomVariable[i1][i2]...[iD]

- i1, i2, ..., iD: Constantes entières (littérales ou non), variables entières ou expressions à résultat entier

- **Exemples:** nomVariable[0], nomVariable[1], nomVariable[i], nomVariable[i+k], nomVariable[i][k]

**Attention:** Indices comptés à partir de 0 -> Indices définis de 0 à N-1 (pas de 1 à N) inclus si N est la taille selon la dimension considérée

- **Exemple n°1: Tableau de booléens**

Déclaration et initialisation d'un tableau de 8 booléens avec vrai

```
{ Declaration d'un tableau de 8 booléens      }  
{ et initialisation a vrai                    }
```

```

action principale()
  locales
    Tableau[8] de booléen tb
    entier i
  pour i de 0 à 7 faire
    tb[i] <- vrai
  fait
fin action

```

### • Exemple n°2: Tableau d'entiers

Déclaration et initialisation d'un tableau d'entiers avec les 50 premières valeurs de  $n!$  puis affichage du contenu du tableau

```

{ Initialisation d'un tableau d'entiers      }
{ avec les 50 premières valeurs de factoriel }

constante entier N <- 50

action principale()
  locales
    Tableau[N] de entier tf
    entier i
  tf[0] <- 1
  pour i de 1 à N-1 faire
    tf[i] <- tf[i-1]*i
  fait
  pour i de 0 à N-1 faire
    afficher(tf[i])
    Ecran.sautDeLigne()
  fait
fin action

```

### • Exemple n°3: Tableau en dimension 2 (2 indices)

Déclaration et initialisation d'un tableau de 10x10 caractères modélisant le damier d'un jeu de dames en début de jeu:

- Pions noirs représentés par le caractère 'N'
- Pions blancs représentés par le caractère 'B'
- Cases vides représentées par le caractère '.'

```

{ Initialisation d'un damier de jeu de dames }
{ - 'B' pour pion blanc                     }
{ - 'N' pour pion blanc                     }
{ - '.' pour case vide                      }
{ Affichage du tableau apres initialisation }

action principale()

```

```

locales
  Tableau[10][10] de caractere damier
  entier i,j
pour i de 0 à 9 faire
  pour j de 0 à 9 faire
    damier[i][j] <- '.'
  fait
fait
pour i de 0 à 8 pas 2 faire
  damier[i][0] <- 'N'
  damier[i+1][1] <- 'N'
  damier[i][2] <- 'N'
  damier[i+1][3] <- 'N'
  damier[i][6] <- 'B'
  damier[i+1][7] <- 'B'
  damier[i][8] <- 'B'
  damier[i+1][9] <- 'B'
fait
fin action

```

- **Détermination de la taille d'un tableau**

- Taille selon le premier indice d'un tableau de dimension quelconque:

longueur (t)

- t: Tableau de dimension quelconque

- Taille selon le  $n^{\text{ième}}$  indice d'un tableau de dimension au moins égale à n:

longueur (n, t)

- n: Numéro de la dimension
- t: Tableau

- Pour les tableaux de dimension 2 ou supérieure longueur(t) = longueur(1,t)

- **Exemple n°4: Accès à la taille et utilisation de cette valeur**

Déclaration d'un tableau d'entiers, initialisation de ce tableau avec les premiers entiers paires

Déclaration d'un tableau de chaînes de caractères à deux indices, initialisation de ce tableau avec la chaîne de caractère vide

```

{ Initialisation d'un tableau d'entiers          }
{ de dimension 1                                }
{ Initialisation d'un tableau de chaînes       }
{ de dimension 2                                }
{ Utilisation des fonctions longueur           }

```

```

{ pour determiner les tailles de ces tableaux }

constante entier N <- 50
constante entier M <- 30

action principale()
  locales
    Tableau[N] de entier te
    Tableau[N][M] de chaine tc
    entier i
    entier j
  pour i de 0 à longueur(te)-1 faire
    te[i] <- i*2
  fait
  pour i de 0 à longueur(1,tc)-1 faire
    pour j de 0 à longueur(2,tc)-1 faire
      tc[i][j] <- ""
    fait
  fait
fin action

```

## Syntaxe en langage Java

- Syntaxe de déclaration en dimension 1
- Deux syntaxes
  - Définition explicite de la taille et initialisation implicite automatique à "zéro" du contenu

```
type [] nomVariable = new type[n];
```

- n: Taille du tableau
  - Constante entière littérale ou non
  - Variable entière
  - Expression numérique de résultat entier
- type: Type élémentaire du tableau
- Définition implicite de la taille par initialisation explicite du contenu

```
type [] nomVariable = { V1,V2,...,Vn };
```

- V1, V2, ..., Vn: Constantes (littérales ou non), variables ou expressions à résultat du type déclaré affectées aux composantes du tableau
- type: Type élémentaire du tableau
- Nombre d'items entre accolades: Taille du tableau

- Syntaxe de déclaration en dimension 2

- Deux syntaxes

- Définition explicite de la taille et initialisation implicite à "zéro" du contenu

```
type [][] nomVariable = new type[n][m];
```

- n et m: Tailles du tableau
  - Constantes entière
  - Variables entière
  - Expressions numériques de résultat entier
- type: Type élémentaire du tableau

- Définition implicite de la taille par initialisation explicite du contenu

```
type [][] nomVariable = { { V11,V12,...,V1m },
                          { V21,V22,...,V2m },
                          ...
                          { Vn1,Vn2,...,Vnm } };
```

- V11, ..., Vnm: Constantes littérales ou non, variables ou expressions à résultat du type déclaré
- type: Type élémentaire du tableau
- Nombre de lignes entre accolades: Taille du tableau selon le premier indice
- Nombre de colonnes entre accolades: Taille du tableau selon de second indice

- Déclaration en dimension D

- Tableaux en dimension D strictement supérieure à 2 possibles en Java
- Syntaxes de déclaration à la dimension D généralisées à partir des syntaxes en dimension 2

- Accès aux composantes d'un tableau

- Syntaxe identique à la syntaxe algorithmique
- Nom de la variable tableau suivi de D indices entiers (constantes littérales ou non, variables ou expressions) spécifiés individuellement entre crochets (D dimension du tableau)
- **Exemples:** `nomVariable[0]`, `nomVariable[1]`, `nomVariable[i]`, `nomVariable[i+k]`, `nomVariable[i][k]`
- **Attention:** Indices comptés à partir de 0 -> Indices définis de 0 à N-1 inclus (pas de 1 à N) si N est la taille selon la dimension considérée

- Détermination de la taille d'un tableau

- Taille selon le premier indice d'un tableau de dimension quelconque:

`t.length`

t: Tableau de dimension quelconque

- Taille selon le deuxième indice d'un tableau de dimension égale au moins à 2:

`t[0].length`

t: Tableau de dimension supérieure ou égale à 2

- **Exemple n°1:**

Déclaration et initialisation d'un tableau de 8 booléens avec vrai

```
/* Declaration d'un tableau de 8 booleens      */
/* initialise a vrai                          */

public class InitialisationTableauBoolean {

/* Programme principal                        */

    public static void main(String [] args) {
        boolean [] tb = { true,true,true,true,
                          true,true,true,true };

        int n = tb.length;
        Ecran.afficherln(n);
        Ecran.afficherln(tb[0]);
        Ecran.afficherln(tb[n-1]);
        Ecran.afficherln(tb);
    }
}
```

### InitialisationTableauBoolean.java - Exemple d'exécution



```
InitialisationTableauBoolean
Fichier Propriétés
8
true
true
[Z@177ecd
```

- **Exemple n°2:**

Déclaration et initialisation d'un tableau d'entiers avec les 50 premières valeurs de n! puis affichage du contenu du tableau

```
/* Initialisation d'un tableau d'entiers      */
/* avec les 50 premieres valeurs de factoriel */

public class TableauFactoriels {

/* Programme principal                        */

    public static void main(String [] args) {
        final int N = 50;
        int [] tf = new int[N];
        int i;
        tf[0] = 1;
        for ( i = 1 ; i < tf.length ; i = i+1 ) {
            tf[i] = tf[i-1]*i; }
        for ( i = 0 ; i < tf.length ; i = i+1 ) {
            Ecran.formater("%2d! = %d\n",i,tf[i]); }
    }
}
```

### TableauFactoriels.java - Exemple d'exécution

```
TableauFactoriels
Fichier  Propriétés
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 1932053504
14! = 1278945280
15! = 2004310016
16! = 2004189184
17! = -288522240
```

Implantation avec des entiers long au lieu d'entiers int

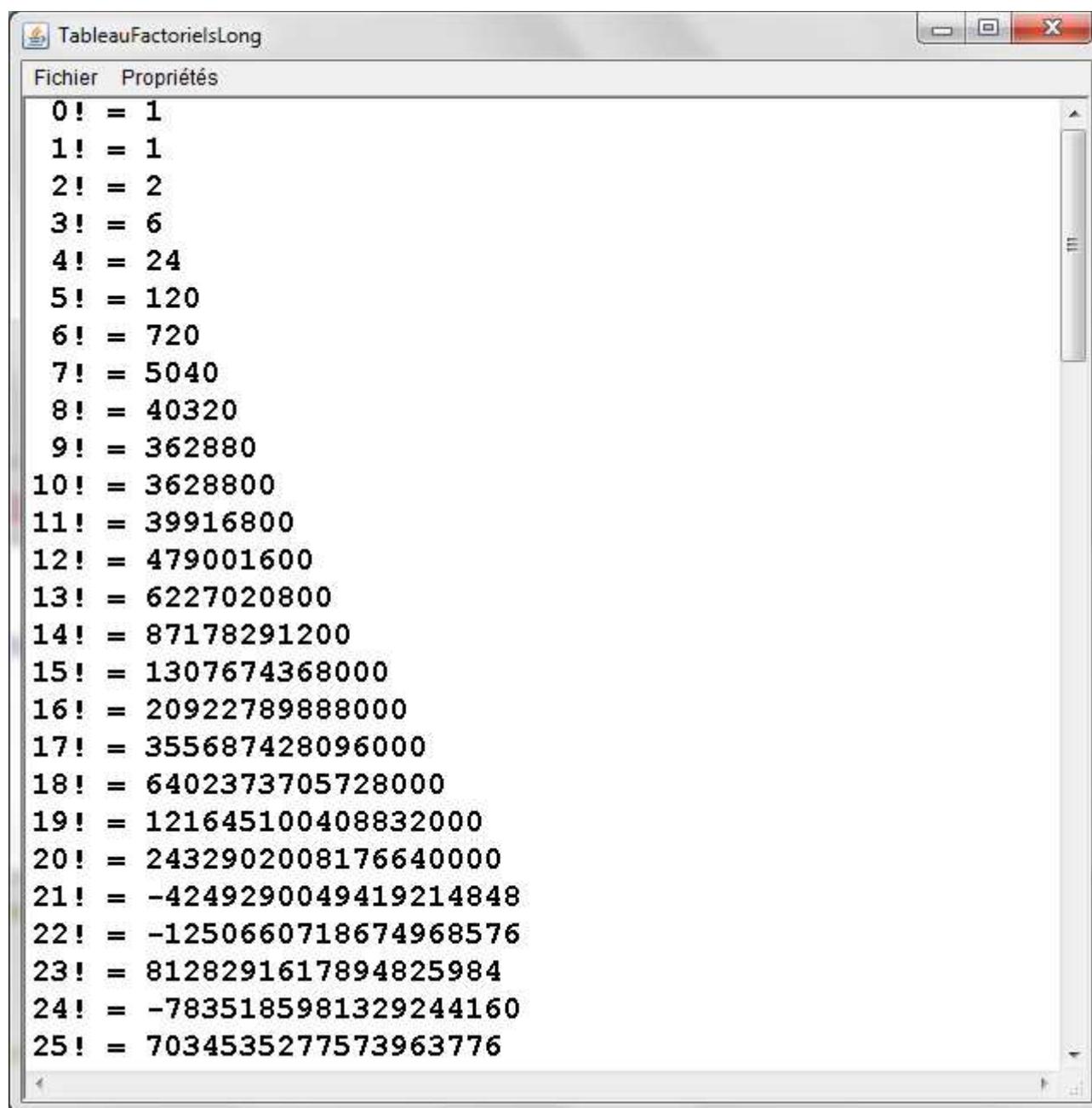
```
/* Initialisation d'un tableau d'entiers      */
/* avec les 50 premieres valeurs de factoriel */

public class TableauFactorielsLong {
```

```
/* Programme principal */

public static void main(String [] args) {
    final int N = 50;
    long [] tb = new long[N];
    int i;
    tb[0] = 1;
    for ( i = 1 ; i < N ; i = i+1 ) {
        tb[i] = tb[i-1]*i; }
    for ( i = 0 ; i < N ; i = i+1 ) {
        Ecran.formater("%2d! = %d\n",i,tb[i]); }
}
}
```

### TableauFactorielsLong.java - Exemple d'exécution



```
Fichier  Propriétés
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = -4249290049419214848
22! = -1250660718674968576
23! = 8128291617894825984
24! = -7835185981329244160
25! = 7034535277573963776
```

### • Exemple n°3:

Déclaration et initialisation d'un tableau de 10x10 caractères modélisant un damier de jeu de dames en début de jeu:

- Pions noirs représentés par le caractère 'N'
- Pions blancs représentés par le caractère 'B'
- Cases vides représentées par le caractère '.'

Affichage du damier

```

/* Initialisation d'un damier de jeu de dames */
/* 'B' pour pion blanc */
/* 'N' pour pion blanc */
/* '.' pour case vide */
/* Affichage du tableau apres initialisation */

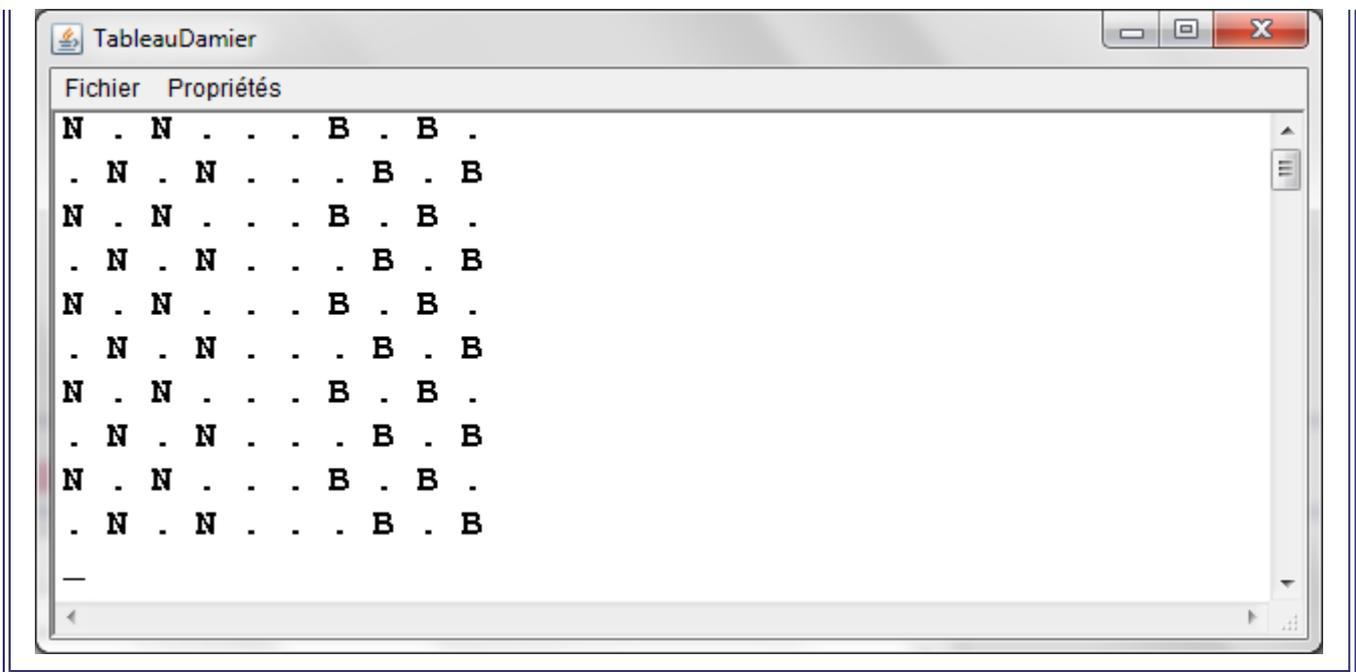
public class TableauDamier {

/* Programme principal */

public static void main(String [] args) {
    char [][] damier = new char[10][10];
    int nbl = damier.length;
    int nbc = damier[0].length;
    int i,j;
    for ( i = 0 ; i < nbl ; i = i+1 ) {
        for ( j = 0 ; j < nbc ; j = j+1 ) {
            damier[i][j] = '.'; } }
    for ( i = 0 ; i <= 8 ; i = i+2 ) {
        damier[i][0] = 'N';
        damier[i+1][1] = 'N';
        damier[i][2] = 'N';
        damier[i+1][3] = 'N';
        damier[i+1][9] = 'B';
        damier[i][8] = 'B';
        damier[i+1][7] = 'B';
        damier[i][6] = 'B'; }
    for ( i = 0 ; i < nbl ; i = i+1 ) {
        for ( j = 0 ; j < nbc ; j = j+1 ) {
            Ecran.afficher(damier[i][j]," "); }
        Ecran.sautDeLigne(); }
    }
}

```

[TableauDamier.java - Exemple d'exécution](#)



## Affectation et test d'égalité entre tableaux

### • Affectation

- En langage algorithmique et en langage Java, affectation impossible entre tableaux de types élémentaires ou de dimensions différentes
- En langage algorithmique et en langage Java, même comportement que pour les types agrégés si affectation (signe <- en PDL, signe = en Java) entre deux variables tableau ayant les mêmes type élémentaire et dimension:

Affectation des adresses mémoires et non affectation place pour place des composantes des tableaux

-> Confusion totale irréversible entre ces deux variables avec remplacement de celle à gauche du signe de l'opérateur affectation par celle située à droite

-> A tout jamais, même variable avec deux noms

-> Pertes des valeurs à gauche ainsi que de la taille (des tailles) remplacée(s) par celle(s) de la variable droite

**Solution:** Développer le code d'affectation place pour place des composantes des deux tableaux

**Recommandation:** Développer une action d'affectation (voir [plus loin](#))

```

/* Programme principal */
public static void main(String [] args) {
    int i;
    /* Declaration et initialisation de deux */
    /* tableaux: t1 et t2 */
    int [] t1 = { 0,1,2,3 };
    int [] t2 = { 0,0,0,0 };
    /* Affichage du contenu des deux tableaux */
}

```

```
Ecran.afficherln("Tableaux t1 et t2 initiaux");
Ecran.afficher("t1 : ");
for ( i = 0 ; i < t1.length ; i = i+1 ) {
    Ecran.afficher(t1[i]," "); }
Ecran.sautDeLigne();
Ecran.afficher("t2 : ");
for ( i = 0 ; i < t2.length ; i = i+1 ) {
    Ecran.afficher(t2[i]," "); }
Ecran.sautDeLigne();
/* Copie du contenu de t1 dans t2 puis */
/* réaffichage du contenu des deux tableaux */
Ecran.afficherln("t1 et t2 après copie de t1 dans t2");
for ( i = 0 ; i < t1.length ; i = i+1 ) {
    t2[i] = t1[i]; }
Ecran.afficher("t1 : ");
for ( i = 0 ; i < t1.length ; i = i+1 ) {
    Ecran.afficher(t1[i]," "); }
Ecran.sautDeLigne();
Ecran.afficher("t2 : ");
for ( i = 0 ; i < t2.length ; i = i+1 ) {
    Ecran.afficher(t2[i]," "); }
Ecran.sautDeLigne();
/* Remplissage de t1 avec des 0 puis */
/* réaffichage du contenu des deux tableaux */
Ecran.afficherln
("t1 et t2 après remplissage de t1 avec des 0.0");
t1[0] = 0;
t1[1] = 0;
t1[2] = 0;
t1[3] = 0;
Ecran.afficher("t1 : ");
for ( i = 0 ; i < t1.length ; i = i+1 ) {
    Ecran.afficher(t1[i]," "); }
Ecran.sautDeLigne();
Ecran.afficher("t2 : ");
for ( i = 0 ; i < t2.length ; i = i+1 ) {
    Ecran.afficher(t2[i]," "); }
Ecran.sautDeLigne();
/* Affichage des adresses mémoire */
/* de t1 et de t2 */
Ecran.afficherln("Adresses mémoire de t1 et t2");
Ecran.afficherln("t1 : ",t1);
Ecran.afficherln("t2 : ",t2);
/* Affectation de t1 avec t2 puis */
/* réaffichage du contenu des deux tableaux */
Ecran.afficherln
("t1 et t2 après affectation de t1 avec t2 (=)");
t1 = t2;
Ecran.afficher("t1 : ");
for ( i = 0 ; i < t1.length ; i = i+1 ) {
```

```
    Ecran.afficher(t1[i], " "); }
Ecran.sautDeLigne();
Ecran.afficher("t2 : ");
    for ( i = 0 ; i < t2.length ; i = i+1 ) {
        Ecran.afficher(t2[i], " "); }
Ecran.sautDeLigne();
    /* Réaffichage des adresses mémoire          */
    /* de t1 et de t2                            */
Ecran.afficherln("Adresses mémoire de t1 et t2");
Ecran.afficherln("t1 : ",t1);
Ecran.afficherln("t2 : ",t2);
    /* Remplissage de t1 avec des 0 puis          */
    /* réaffichage du contenu des deux tableaux */
Ecran.afficherln
("t1 et t2 après remplissage de t1 avec des 0");
t1[0] = 0;
t1[1] = 0;
t1[2] = 0;
t1[3] = 0;
Ecran.afficher("t1 : ");
    for ( i = 0 ; i < t1.length ; i = i+1 ) {
        Ecran.afficher(t1[i], " "); }
Ecran.sautDeLigne();
Ecran.afficher("t2 : ");
    for ( i = 0 ; i < t2.length ; i = i+1 ) {
        Ecran.afficher(t2[i], " "); }
Ecran.sautDeLigne();
    /* Remplissage de t2 avec des 1 puis          */
    /* réaffichage du contenu des deux tableaux */
Ecran.afficherln
("t1 et t2 après remplissage de t2 avec des 1");
t2[0] = 1;
t2[1] = 1;
t2[2] = 1;
t2[3] = 1;
Ecran.afficher("t1 : ");
    for ( i = 0 ; i < t1.length ; i = i+1 ) {
        Ecran.afficher(t1[i], " "); }
Ecran.sautDeLigne();
Ecran.afficher("t2 : ");
    for ( i = 0 ; i < t2.length ; i = i+1 ) {
        Ecran.afficher(t2[i], " "); }
Ecran.sautDeLigne();
}
```

[AffectationEntreTableaux.java](#) - [Exemple d'exécution](#)

```

AffectationEntreTableaux
Fichier Propriétés
Tableaux t1 et t2 initiaux
t1 : 0 1 2 3
t2 : 0 0 0 0
t1 et t2 après copie de t1 dans t2
t1 : 0 1 2 3
t2 : 0 1 2 3
t1 et t2 après remplissage de t1 avec des 0.0
t1 : 0 0 0 0
t2 : 0 1 2 3
Adresses mémoire de t1 et t2
t1 : [I@177ecd
t2 : [I@80bfe8
t1 et t2 après affectation de t1 avec t2 (=)
t1 : 0 1 2 3
t2 : 0 1 2 3
Adresses mémoire de t1 et t2
t1 : [I@80bfe8
t2 : [I@80bfe8
t1 et t2 après remplissage de t1 avec des 0
t1 : 0 0 0 0
t2 : 0 0 0 0
t1 et t2 après remplissage de t2 avec des 1
t1 : 1 1 1 1
t2 : 1 1 1 1

```

### • Test d'égalité et test de différence

- En langage algorithmique et en langage Java, tests d'égalité et de différence impossibles entre tableaux de types élémentaires ou de dimensions différentes
- En langage algorithmique et en langage Java, tests d'égalité et de différence possibles entre deux tableaux par utilisation des signes == et != mais attention le test porte sur leurs adresses mémoires respectives et ne porte donc pas sur leurs composantes place pour place  
-> Comportement différent de celui souhaité

**Solution:** Développer le code de test place pour place des composantes des deux tableaux

**Recommandation:** Développer une fonction de test (voir [plus loin](#))

```

/* Programme principal                                     */
public static void main(String [] args) {

```

```
boolean egal;
int i;
/* Declaration et initialisation de trois */
/* tableaux: t1, t2 et t3 */
int [] t1 = { 0,1,2,3 };
int [] t2 = { 0,0,0,0 };
int [] t3 = { 0,1,2,3 };
/* Affichage du contenu des trois tableaux */
Ecran.afficher("t1 : ");
for ( i = 0 ; i < t1.length ; i = i+1 ) {
    Ecran.afficher(t1[i]," "); }
Ecran.sautDeLigne();
Ecran.afficher("t2 : ");
for ( i = 0 ; i < t2.length ; i = i+1 ) {
    Ecran.afficher(t2[i]," "); }
Ecran.sautDeLigne();
Ecran.afficher("t3 : ");
for ( i = 0 ; i < t3.length ; i = i+1 ) {
    Ecran.afficher(t3[i]," "); }
Ecran.sautDeLigne();
/* Tests d'egalite par utilisation du == */
egal = (t1 == t2);
Ecran.afficherln
("Test d'egalite avec == entre t1 et t2 : ",egal);
egal = (t1 == t3);
Ecran.afficherln
("Test d'egalite avec == entre t1 et t3 : ",egal);
/* Tests d'egalite par implantation */
/* du test composante par composante */
egal = true;
if ( t1.length != t2.length ) {
    egal = false; }
else {
    i = 0;
    while ( ( egal == true ) && ( i < t1.length ) ) {
        if ( t1[i] != t2[i] ) {
            egal = false; }
        i = i+1; } }
Ecran.afficherln
("Test d'egalite sur composantes entre t1 et t2 : ",egal);
egal = true;
if ( t1.length != t3.length ) {
    egal = false; }
else {
    i = 0;
    while ( ( egal == true ) && ( i < t1.length ) ) {
        if ( t1[i] != t3[i] ) {
            egal = false; }
        i = i+1; } }
Ecran.afficherln
```

```
("Test d'egalite sur composantes entre t1 et t3 : ",egal);
  /* Affichage des adresses memoire          */
  /* des trois tableaux                      */
Ecran.afficherln("t1 : ",t1);
Ecran.afficherln("t2 : ",t2);
Ecran.afficherln("t3 : ",t3);
}
```

### TestEgaliteTableaux.java - Exemple d'exécution

```
TestEgaliteTableaux
Fichier  Propriétés
t1 : 0 1 2 3
t2 : 0 0 0 0
t3 : 0 1 2 3
Test d'egalite avec == entre t1 et t2          : false
Test d'egalite avec == entre t1 et t3          : false
Test d'egalite sur composantes entre t1 et t2  : false
Test d'egalite sur composantes entre t1 et t3  : true
t1 : [I@177ecd
t2 : [I@80bfe8
t3 : [I@a29884
```

## Tableaux en paramètre d'entête et en retour de sous-algorithme

- Tableaux utilisables en paramètre d'action et de fonction
- Tableaux utilisables en retour de fonction
- **Passages de paramètre**
- En langage algorithmique, passages de paramètre d'entête:
  - En entrée
  - En sortie
  - En entrée/sortie
- En langage Java, un seul type de passage de paramètre d'entête:
  - En entrée/sortie
- **Syntaxes PDL et Java**
- Disparition de la spécification des tailles dans la définition des paramètres d'entête de type tableau
  - > Dans le corp des sous-algorithmes, utilisation des fonctionnalités permettant de récupérer les tailles

-> Développement de sous-algorithmes admettant en entrée, en sortie et en retour des tableaux de taille quelconque

## • Exemples

```
{ Passage d'un tableau en parametre           }
{ Affichage de ses composantes               }
{ t : Le tableau d'entiers à afficher        }
```

```
action affichage(t)
```

```
  Entrées
```

```
    Tableau [] de entier t
```

```
  Locales
```

```
    entier i
```

```
  pour i de 0 à longueur(t)-1 faire
```

```
    afficher(t[i], " ")
```

```
  fait
```

```
  Ecran.sautDeLigne()
```

```
fin action
```

```
{ Passage d'un tableau en parametre           }
{ Modification du tableau a l'interieur       }
{ de l'action                                 }
{ -> Modification du tableau sur lequel      }
{   l'action a ete appelee                   }
{ tab : Le tableau d'entiers à modifier      }
```

```
action testPassage(tab)
```

```
  Entrées / Sorties
```

```
    Tableau [] de entier tab
```

```
  Locales
```

```
    entier i
```

```
    entier n <- longueur(tab)
```

```
  affichage(tab)
```

```
  pour i de 0 à n-1 faire
```

```
    tab[i] <- 9
```

```
  fait
```

```
  affichage(tab)
```

```
fin action
```

```
{ Retour d'un tableau par une fonction        }
{ apres declaration et initialisation         }
{ dans le corps                              }
{ n : La taille du tableau d'entiers à créer }
{   et retourner                             }
```

```
Tableau [] de entier fonction testRetour(n)
```

```
  Entrées
```

```
    entier n
```

```

    Locales
    Tableau [n] de entier t
    entier i
    pour i de 0 à n-1 faire
        t[i] <- i
    fait
    affichage(t)
    retourner t
fin fonction

{ Action principale }

action principale()
    Locales
    Tableau[8] de entier t1
    Tableau[10] de entier t2
{ Test de l'action testPassage }
    t1[0] <- 0
    t1[1] <- 1
    t1[2] <- 2
    t1[3] <- 3
    t1[4] <- 4
    t1[5] <- 5
    t1[6] <- 6
    t1[7] <- 7
    affichage(t1)
    testPassage(t1)
    affichage(t1)
    Ecran.sautDeLigne();
{ Test de la fonction testRetour }
    t2 <- testRetour(10);
    affichage(t2)
    Ecran.sautDeLigne()
fin action

```

### PassageTableauEnParametre.Ida

```

/* Passage d'un tableau en parametre */
/* Fonction d'affichage d'un tableau d'entiers */
/* t : Le tableau d'entiers à afficher */

static void affichage(int [] t) {
    int i;
    for ( i = 0 ; i < t.length ; i = i+1 ) {
        Ecran.afficher(t[i], " "); }
    Ecran.sautDeLigne();
}

/* Fonction d'affichage d'un tableau d'entiers */
/* avec affichage préalable d'un message */

```

```
/* alphabétique et de l'adresse mémoire      */
/* du tableau                                */
/* t : Le tableau d'entiers à afficher       */
/* message : Le message préalable           */

static void affichage(int [] t,String message) {
    Ecran.formater("%s : %11s : ",message,t);
    affichage(t);
}

/* Passage d'un tableau en parametre        */
/* Modification du tableau a l'interieur    */
/* de la fonction                           */
/* -> Modification du tableau sur lequel    */
/*     la fonction a ete appelee            */
/* tab : Le tableau d'entiers à modifier    */

static void testPassage(int [] tab) {
    int i;
    affichage(tab,"tab");
    for ( i = 0 ; i < tab.length ; i = i+1 ) {
        tab[i] = 9; }
    affichage(tab,"tab");
}

/* Retour d'un tableau par une fonction     */
/* apres declaration et initialisation      */
/* dans le corps                            */
/* n : La taille du tableau d'entiers à créer */
/*     et retourner                          */

static int [] testRetour(int n) {
    int i;
    int [] tab = new int[n];
    for ( i = 0 ; i < tab.length ; i = i+1 ) {
        tab[i] = i; }
    affichage(tab,"tab");
    return tab;
}

/* Programme principal                       */

public static void main(String [] args) {
    int [] t1 = { 0,1,2,3,4,5,6,7 };
    int [] t2;
    /* Test de la fonction testPassage      */
    Ecran.afficherln
    ("Tableau passé en paramètre d'entête de fonction");
    affichage(t1,"t1 ");
    testPassage(t1);
}
```

```

    affichage(t1,"t1 ");
/* Test de la fonction testRetour */
    Ecran.afficherln("Tableau passé en retour de fonction");
    t2 = testRetour(10);
    affichage(t2,"t2 ");
}

```

### PassageTableauEnParametre.java - Exemple d'exécution

```

PassageTableauEnParametre
Fichier  Propriétés
Tableau passé en paramètre d'entête
t1 : [I@183da3f : 0 1 2 3 4 5 6 7
tab : [I@183da3f : 0 1 2 3 4 5 6 7
tab : [I@183da3f : 9 9 9 9 9 9 9 9
t1 : [I@183da3f : 9 9 9 9 9 9 9 9
Tableau passé en retour de fonction
tab : [I@18fd1ac : 0 1 2 3 4 5 6 7 8 9
t2 : [I@18fd1ac : 0 1 2 3 4 5 6 7 8 9

```

- **Gestion du retour des fonctions**
- Pas de problème:
  - Affectation du tableau retourné à une variable tableau de mêmes caractéristiques en utilisant l'opérateur d'affectation (= en Java, < - en PDL)
    - > Disparition de l'ancienne variable tableau remplacée par le tableau retourné
    - > En Java, pas nécessaire de faire un new dans le programme appelant car il est fait dans la fonction sur la variable retournée en fin de fonction
- **Exemples supplémentaires:** Réécriture des deux programmes exemples du paragraphe sur l'affectation et le test d'égalité entre tableaux -> utilisation de fonctions

```

/* Fonction d'affichage d'un tableau d'entiers */
/* t : Le tableau de int a afficher */

static void affichage(int [] t) {
    int i;
    for ( i = 0 ; i < t.length ; i = i+1 ) {
        Ecran.afficher(t[i]," "); }
    Ecran.sautDeLigne();
}

/* Fonction d'affichage d'un tableau d'entiers */

```

```
/* avec affichage préalable d'un message */
/* alphabétique */
/* message : La chaine a afficher préalablement */
/* t : Le tableau de int a afficher */

static void affichage(String message,int [] t) {
    Ecran.formater("%s : ",message);
    affichage(t);
}

/* Fonction de test de l'egalite */
/* de deux tableaux d'entiers et de retour */
/* du booleen true si c'est le cas, false sinon */
/* t1 : Le premier tableau de int du test */
/* t2 : Le second tableau de int du test */

static boolean testEgalite(int [] t1,int [] t2) {
    boolean egal = true;
    int i;
    if ( t1.length != t2.length ) {
        egal = false; }
    else {
        i = 0;
        while ( ( egal == true ) && ( i < t1.length ) ) {
            if ( t1[i] != t2[i] ) {
                egal = false; }
            i = i+1; } }
    return egal;
}

/* Programme principal */

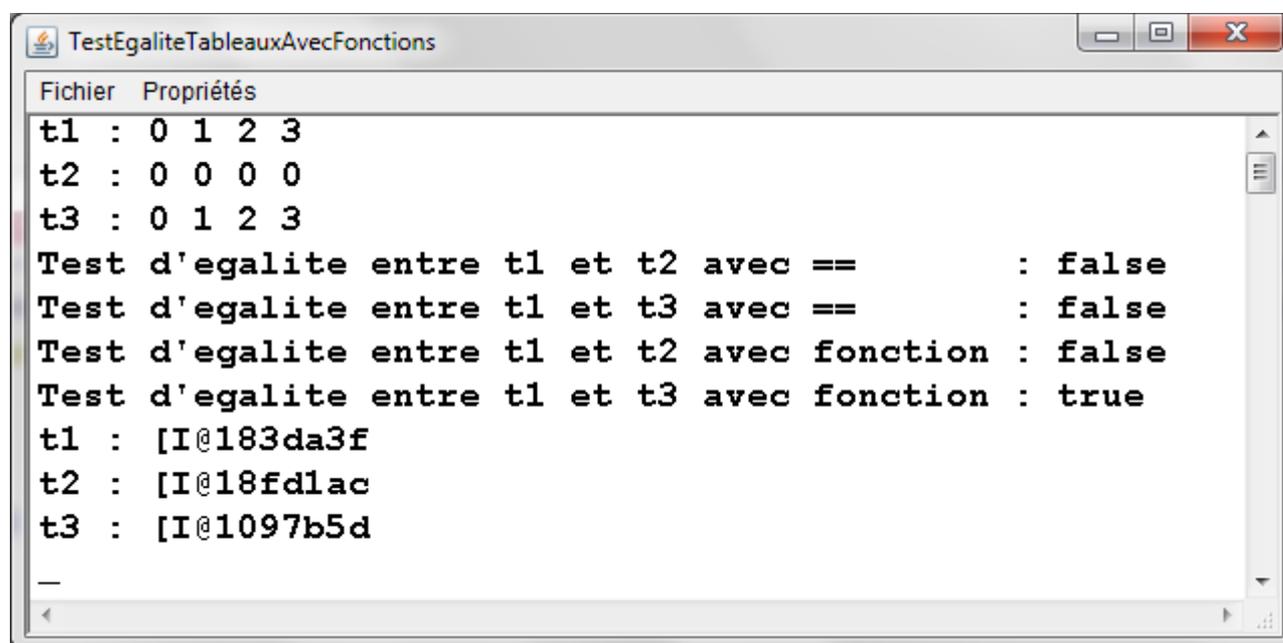
public static void main(String [] args) {
    boolean egal;
    /* Declaration et initialisation de trois */
    /* tableaux: t1, t2 et t3 */
    int [] t1 = { 0,1,2,3 };
    int [] t2 = { 0,0,0,0 };
    int [] t3 = { 0,1,2,3 };
    /* Affichage du contenu des trois tableaux */
    affichage("t1",t1);
    affichage("t2",t2);
    affichage("t3",t3);
    /* Tests d'egalite par utilisation du == */
    egal = (t1 == t2);
    Ecran.afficherln
("Test d'egalite entre t1 et t2 avec == : ",egal);
    egal = (t1 == t3);
    Ecran.afficherln
("Test d'egalite entre t1 et t3 avec == : ",egal);
}
```

```

    /* Tests d'egalite par implantation          */
    /* du test composante par composante        */
    egal = testEgalite(t1,t2);
    Ecran.afficherln
("Test d'egalite entre t1 et t2 avec fonction : ",egal);
    egal = testEgalite(t1,t3);
    Ecran.afficherln
("Test d'egalite entre t1 et t3 avec fonction : ",egal);
    /* Affichage des adresses mémoire          */
    /* des trois tableaux                      */
    Ecran.afficherln("t1 : ",t1);
    Ecran.afficherln("t2 : ",t2);
    Ecran.afficherln("t3 : ",t3);
}

```

### TestEgaliteTableauxAvecFonctions.java - Exemple d'exécution



```

TestEgaliteTableauxAvecFonctions
Fichier Propriétés
t1 : 0 1 2 3
t2 : 0 0 0 0
t3 : 0 1 2 3
Test d'egalite entre t1 et t2 avec ==          : false
Test d'egalite entre t1 et t3 avec ==          : false
Test d'egalite entre t1 et t2 avec fonction : false
Test d'egalite entre t1 et t3 avec fonction : true
t1 : [I@183da3f
t2 : [I@18fd1ac
t3 : [I@1097b5d

```

```

/* Fonction d'affichage d'un tableau d'entiers */
/* t : Le tableau de int a afficher           */

static void affichage(int [] t) {
    int i;
    for ( i = 0 ; i < t.length ; i = i+1 ) {
        Ecran.afficher(t[i]," "); }
    Ecran.sautDeLigne();
}

/* Fonction d'affichage d'un tableau d'entiers */
/* avec affichage préalable d'un message      */
/* alphabétique                               */
/* message : La chaine a afficher préalablement */
/* t : Le tableau de int a afficher           */

static void affichage(String message,int [] t) {

```

```
Ecran.formater("%s : ",message);
affichage(t);
}

/* Fonction de copie des valeurs contenues      */
/* dans un tableau d'entiers source             */
/* vers un tableau d'entiers cible             */
/* source : Le tableau de int source           */
/* cible : Le tableau de int cible             */

static void copie(int [] source,int [] cible) {
    int i;
    for ( i = 0 ; i < source.length ; i = i+1 ) {
        cible[i] = source[i]; }
}

/* Programme principal                          */

public static void main(String [] args) {
    /* Declaration et initialisation de deux    */
    /* tableaux: t1 et t2                       */
    int [] t1 = { 0,1,2,3 };
    int [] t2 = { 0,0,0,0 };
    /* Affichage du contenu des deux tableaux  */
    Ecran.afficherln("Tableaux t1 et t2 initiaux");
    affichage("t1",t1);
    affichage("t2",t2);
    /* Copie du contenu de t1 dans t2 puis     */
    /* réaffichage du contenu des deux tableaux */
    Ecran.afficherln("t1 et t2 après copie de t1 dans t2");
    copie(t1,t2);
    affichage("t1",t1);
    affichage("t2",t2);
    /* Remplissage de t1 avec des 0 puis      */
    /* réaffichage du contenu des deux tableaux */
    Ecran.afficherln
("t1 et t2 après remplissage de t1 avec des 0.0");
    t1[0] = 0;
    t1[1] = 0;
    t1[2] = 0;
    t1[3] = 0;
    affichage("t1",t1);
    affichage("t2",t2);
    /* Affichage des adresses mémoire         */
    /* de t1 et de t2                         */
    Ecran.afficherln("Adresses mémoire de t1 et t2");
    Ecran.afficherln("t1 : ",t1);
    Ecran.afficherln("t2 : ",t2);
    /* Affectation de t1 avec t2 puis         */
    /* réaffichage du contenu des deux tableaux */
}
```

```
Ecran.afficherln
("t1 et t2 après affectation de t1 avec t2 (=)");
t1 = t2;
affichage("t1",t1);
affichage("t2",t2);
/* Réaffichage des adresses mémoire      */
/* de t1 et de t2                        */
Ecran.afficherln("Adresses mémoire de t1 et t2");
Ecran.afficherln("t1 : ",t1);
Ecran.afficherln("t2 : ",t2);
/* Remplissage de t1 avec des 0 puis      */
/* réaffichage du contenu des deux tableaux */
Ecran.afficherln
("t1 et t2 après remplissage de t1 avec des 0");
t1[0] = 0;
t1[1] = 0;
t1[2] = 0;
t1[3] = 0;
affichage("t1",t1);
affichage("t2",t2);
/* Remplissage de t2 avec des 1 puis      */
/* réaffichage du contenu des deux tableaux */
Ecran.afficherln
("t1 et t2 après remplissage de t2 avec des 1");
t2[0] = 1;
t2[1] = 1;
t2[2] = 1;
t2[3] = 1;
affichage("t1",t1);
affichage("t2",t2);
}
```

### [AffectationEntreTableauxAvecFonctions.java - Exemple d'exécution](#)

```

AffectationEntreTableauxAvecFonctions
Fichier Propriétés
Tableaux t1 et t2 initiaux
t1 : 0 1 2 3
t2 : 0 0 0 0
t1 et t2 après copie de t1 dans t2
t1 : 0 1 2 3
t2 : 0 1 2 3
t1 et t2 après remplissage de t1 avec des 0.0
t1 : 0 0 0 0
t2 : 0 1 2 3
Adresses mémoire de t1 et t2
t1 : [I@183da3f
t2 : [I@18fd1ac
t1 et t2 après affectation de t1 avec t2 (=)
t1 : 0 1 2 3
t2 : 0 1 2 3
Adresses mémoire de t1 et t2
t1 : [I@18fd1ac
t2 : [I@18fd1ac
t1 et t2 après remplissage de t1 avec des 0
t1 : 0 0 0 0
t2 : 0 0 0 0
t1 et t2 après remplissage de t2 avec des 1
t1 : 1 1 1 1
t2 : 1 1 1 1

```

## Tableaux de variables de type agrégé

- Déclaration de tableaux de variables de type agrégé possible dans la très grande majorité des langages
  - Ok en langage algorithmique
  - Ok en langage Java
- En langage algorithmique, déclaration de la même manière que tout tableau

**Tableau**[n1][n2]...[nD] **de** nomStructure nomVariable

- n1, n2, ..., nD: Constante entière (littérale ou non), variable entière ou expression numérique à résultat entier définissant les tailles (selon chacune des D dimensions)
- nomStructure: Nom du type agrégé élémentaire

Si initialisation à la déclaration pour les champs du type agrégé  
**nomStructure**

-> Initialisation implicite des champs de chaque composante du tableau

Dans le cas contraire

-> Initialisation impérative par programme dans l'algorithme

-> Initialisation des types agrégés à la déclaration très recommandée

- En langage Java, pas de déclaration d'un tableau de variables de type agrégé en une seule ligne de code
  - Déclaration du tableau par opérateur new
  - Déclaration obligatoire par opérateur new de chacune des composantes du tableau

En dimension 1: En dimension 1:

```
TypeAgrege [] nomVariable = new TypeAgrege[n];
for ( int i = 0 ; i < n ; i = i+1 ) {
    nomVariable[i] = new TypeAgrege(); }
```

- n: Constante entière (littérale ou non), variable entière ou expression numérique à résultat entier définissant la taille selon l'unique dimension
- typeAgrégé: Nom du type agrégé

En dimension D (D indices):

```
TypeAgrege [][]...[] nomVariable = new TypeAgrege[n1]
[n2]...[nD];
for ( int i1 = 0 ; i1 < n1 ; i1 = i1+1 ) {
    for ( int i2 = 0 ; i2 < n2 ; i2 = i2+1 ) {
        ...
        for ( int iD = 0 ; iD < nD ; iD = iD+1 ) {
            nomVariable[i1][i2]...[iD] = new TypeAgrege(); } } }
```

- n1, n2, ..., nD: Constantes entières, variables entières ou expressions numériques à résultat entier définissant les tailles selon chacune des D dimensions du tableau
- typeAgrégé: Nom du type agrégé

**Recommandation:** Utiliser une fonction pour réaliser l'implantation du code de déclaration (isolation du problème)

Si initialisation à la déclaration des champs du type agrégé `nomType`

-> Initialisation implicite des champs de chaque composante du tableau

Dans le cas contraire

-> Initialisation impérative par programme dans l'algorithme

-> Initialisation des types agrégés à la déclaration très recommandée

- Syntaxes d'utilisation identiques en langage algorithmique et en langage Java pour les variables déclarées de type tableau de types agrégés:

Pour une variable tableau de dimension 1:

```
nomVariable[indice].nomChamp
```

- **Exemples:** Déclaration et utilisation d'un tableau de positions dans le plan

```
{ Declaration d'un tableau de variables          }
{ de type agrege position en deux dimensions    }

structure position2D
    reel x <- 0.0
    reel y <- 0.0
fin structure

{ Programme principal                            }

constante entier TAILLE <- 3

action principale()
    locales
        entier i
        Tableau[TAILLE] de position2D tpos
    pour i de 0 à TAILLE-1 faire
        afficherln(tpos[i].x, " ", tpos[i].y)
    fait
Ecran.sautDeLigne()
    pour i de 0 à TAILLE-1 faire
        tpos[i].x = Math.random()
        tpos[i].y = Math.random()
    fait
    pour i de 0 à TAILLE-1 faire
        afficher(tpos[i].x, " ", tpos[i].y)
    fait
fin action
```

### TableauPosition2D.Ida

```
/* Type agrege de stockage d'une position du plan */

static class Position2D {
    double x = 0.0;
    double y = 0.0; };

/* Programme principal */

public static void main(String [] args) {
    int i;
    final int TAILLE = 4;
    /* Declaration du tableau */
    Position2D [] tpos = new Position2D[TAILLE];
    */
```

```

/* Initialisation des composantes par programme */
for ( i = 0 ; i < TAILLE ; i++ ) {
    tpos[i] = new Position2D(); }
/* Affichage des Position2D du tableau tpos */
Ecran.afficherln("Valeurs après déclaration");
for ( i = 0 ; i < tpos.length ; i++ ) {
    Ecran.formater("%8.5f %8.5f\n",tpos[i].x,tpos[i].y); }
/* Affectation de valeurs tirees au sort */
/* aux Position2D du tableau tpos */
for ( i = 0 ; i < tpos.length ; i++ ) {
    tpos[i].x = Math.random();
    tpos[i].y = Math.random(); }
/* Reaffichage des Position2D du tableau tpos */
Ecran.afficherln("Valeurs après affectation");
for ( i = 0 ; i < tpos.length ; i++ ) {
    Ecran.formater("%8.5f %8.5f\n",tpos[i].x,tpos[i].y); }
}

```

### TableauPosition2D.java - Exemple d'exécution

```

TableauPosition2D
Fichier Propriétés
Valeurs après déclaration
0,00000 0,00000
0,00000 0,00000
0,00000 0,00000
0,00000 0,00000
Valeurs après affectation
0,03421 0,13586
0,12543 0,06286
0,69673 0,91426
0,22428 0,17238

```

Version alternative avec utilisation d'une action pour l'affichage d'une Position2D,  
d'une action pour l'affichage d'un tableau de Position2D,  
et, pour le programme Java, d'une fonction de création d'un tableau de Position2D

```

{ Declaration d'un tableau de variables }
{ de type agrege position en deux dimensions }

structure position2D
    reel x <- 0.0
    reel y <- 0.0
fin structure

```

```

{ Action d'affichage d'une position 2D      }
{ p : La position2D à afficher              }

action affichage(p)
  Entrées
    position2D p
  afficher("[",p.x,",",",p.y,"]")
fin action

{ Action d'affichage d'un tableau de position 2D }
{ tpos : Le tableau de position 2D à afficher   }

action affichage(tpos)
  Entrées
    tableau [] de position2D tpos
  Locales
    entier i
  pour i de 0 à longueur(tpos)-1 faire
    affichage(tpos[i])
    Ecran.sautDeLigne()
  fait
fin action

{ Programme principal                        }

constante entier TAILLE <- 3

action principale()
  locales
    Tableau[TAILLE] de position2D tpos
  affichage(tpos)
fin action

```

### TableauPosition2DAvecFonctions.la

```

/* Type agrege de stockage d'une position du plan */

static class Position2D {
  double x = 0.0;
  double y = 0.0; };

/* Fonction de creation et retour          */
/* d'un tableau de Position2D              */
/* n : La taille du tableau de position 2D a creer */

static Position2D [] tableauDePosition2D(int n) {
  int i;
  /* Declaration du tableau                */
  Position2D [] tpos = new Position2D[n];
  /* Initialisation des composantes par programme */

```

```
    for ( i = 0 ; i < n ; i++ ) {
        tpos[i] = new Position2D(); }
    return tpos;
}

/* Fonction d'affichage d'une position 2D          */
/* p : La position2D à afficher                    */

static void affichage(Position2D p) {
    Ecran.formater("[%8.5f,%8.5f]",p.x,p.y);
}

/* Fonction d'affichage d'un tableau de Position2D */
/* tpos : Le tableau de position 2D à afficher    */

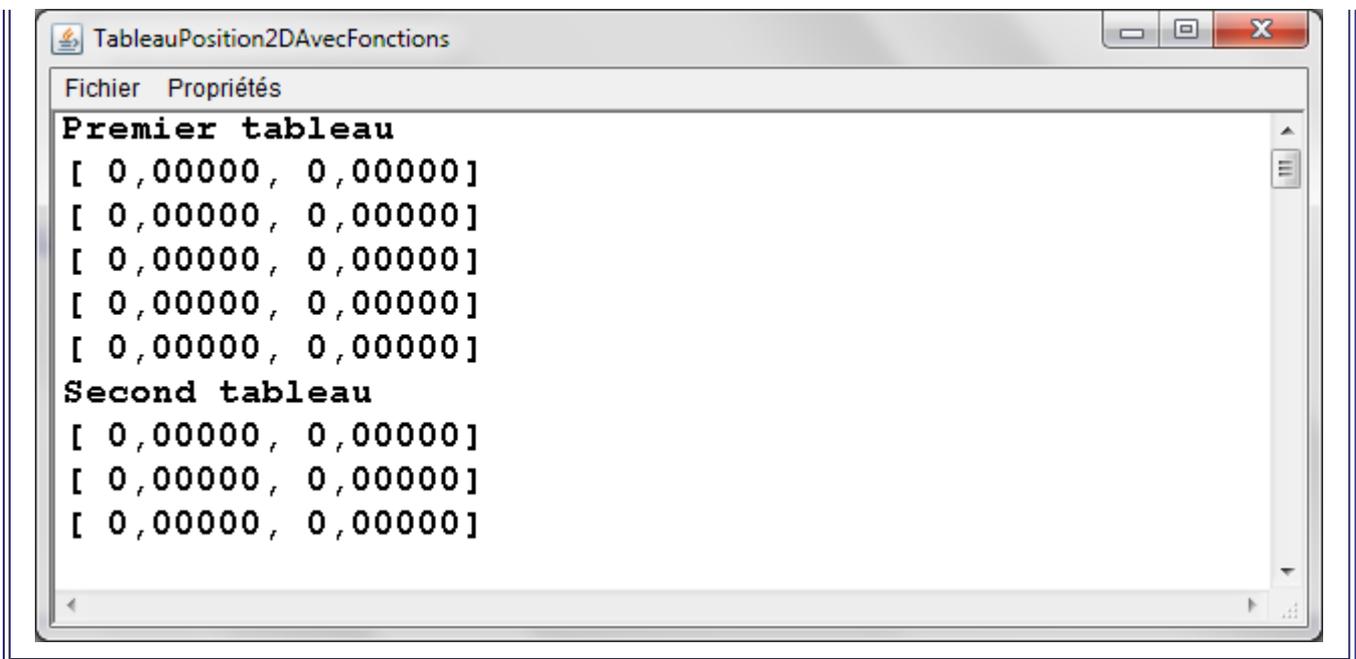
static void affichage(Position2D [] tpos) {
    int i;
    for ( i = 0 ; i < tpos.length ; i++ ) {
        affichage(tpos[i]);
        Ecran.sautDeLigne(); }
}

/* Programme principal                             */

public static void main(String [] args) {
    final int TAILLE = 3;
    /* Declaration du tableau tpos1                 */
    /* avec intialisation de ses champs             */
    /* lors de la declaration                       */
    Position2D [] tpos1 = { new Position2D(),
                            new Position2D(),
                            new Position2D(),
                            new Position2D(),
                            new Position2D() };

    /* Declaration du tableau tpos2                 */
    /* par utilisation de la fonction ad hoc        */
    Position2D [] tpos2 = tableauDePosition2D(TAILLE);
    /* Affichage des Position2D du tableau tpos1   */
    /* puis du tableau tpos2                       */
    /* par utilisations de la fonction ad hoc      */
    Ecran.afficherln("Premier tableau");
    affichage(tpos1);
    Ecran.afficherln("Second tableau");
    affichage(tpos2);
}
```

### TableauPosition2DAvecFonctions.java - Exemple d'exécution



### Tableau en tant que champ dans un type agrégé

- Champs de type tableau autorisés dans les types agrégés
- En langage algorithmique, syntaxe en dimension D:

```

structure nomStructure
  Tableau[n1][n2]...[nD] de type nomChamp
  ...
fin structure

```

- n1, n2, ..., nD: Constantes entières (littérales ou non), variables entières ou expressions numériques à résultat entier définissant les tailles selon chacune des D dimensions
- En langage Java, syntaxe en dimension D pour un tableau de variables de type prédéfini:

```

static class NomType {
  type [][]...[] nomChamp = new type[n1][n2]...[nD];
  ...
};

```

- n1, n2, ..., nD: Constantes entières (littérales ou non), variables entières ou expressions numériques à résultat entier définissant les tailles selon chacune des D dimensions
- Syntaxes d'utilisation identiques en langage algorithmique et en langage Java pour les variables déclarées de types agrégés incluant un ou des champs de type tableau: champs de type tableau:

Pour une variable agrégée ayant un champ de type tableau de dimension 1:

`nomVariable.nomChampTableau[indice]`

- **Exemple:** Une structure de stockage d'un ensemble d'au maximum 15 réels

```

{ Declaration d'un type agrege avec un tableau      }
{ parmi ses champs: Un ensemble de reels          }

{ Constante de définition du nombre maximum        }
{ de valeurs stockables dans un ensemble          }
{ de reels                                          }

constante entier TAILLE <- ...

{ Type agrege de stockage d'un ensemble de reels }

structure ensembleDeReel
  entier n <- 0
  Tableau[TAILLE] de reel t
fin structure

{ Action d'initialisation d'un ensemble de reels }
{ nb : Le nombre de valeurs a initialiser        }
{ v : La valeur d'initialisation                 }
{ e : L'EnsembleDeReel a initialiser            }

action initialisation(nb,v,e)
  Entrées
    entier nb
    réel v
  Sorties
    ensembleDeReel e
  Locales
    i entier
  e.n <- nb
  pour i de 0 à nb-1 faire
    e.t[i] <- v
  fait
fin action

{ Programme principal                               }

action principale()
  Locales
    entier i
    ensembleDeReel e
  initialisation(5,1.0,e)
  afficher("Nombre valeurs: ",e.n)
  pour i de 0 à e.n faire

```

```
    afficher(e.t[i])
  fait
fin action
```

### TypeAgregeAvecTableau.Ida

```
/* Constante de définition du nombre maximum          */
/* de valeurs stockables dans un ensemble de double  */

final static int TAILLE = 15;

/* Type agrege de stockage d'un ensemble de double  */
/* n: nombre de valeurs stockees                    */
/* t: tableau ou sont stockees les valeurs           */
/*   aux indices 0 à n-1                            */

static class EnsembleDeDouble {
    int n = 0;
    double [] t = new double[TAILLE]; };

/* Fonction d'initialisation d'un EnsembleDeDouble  */
/* nb : Le nombre de valeurs a initialiser          */
/* v : La valeur d'initialisation                   */
/* e : L'EnsembleDeDouble a initialiser             */

static void initialisation(int nb,double v,
                           EnsembleDeDouble e) {

    int i;
    e.n = nb;
    for ( i = 0 ; i < e.n ; i = i+1 ) {
        e.t[i] = v; }
}

/* Fonction d'affichage des valeurs contenues        */
/* dans un EnsembleDeDouble                          */
/* e : L'EnsembleDeDouble a afficher                */

static void affichage(EnsembleDeDouble e) {
    int i;
    for ( i = 0 ; i < e.n ; i = i+1 )
        Ecran.afficherln(e.t[i]);
}

/* Programme principal                                */

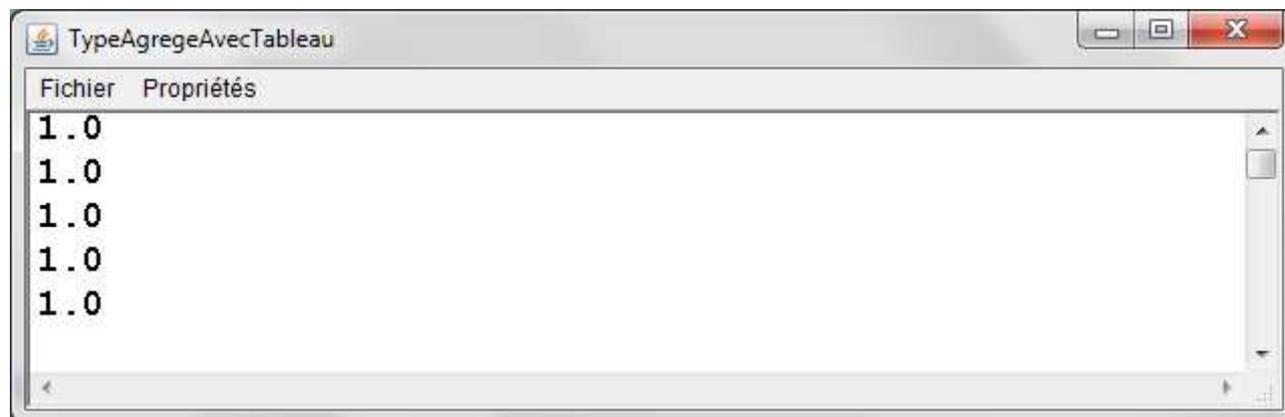
public static void main(String [] args) {
    /* Declaration d'un EnsembleDeDouble            */
    EnsembleDeDouble e = new EnsembleDeDouble();
    /* Initialisation de cet EnsembleDeDouble       */
    /* avec 5 valeurs 1.0                            */
}
```

```

    initialisation(5,1.0,e);
    /* Affichage de cet EnsembleDeDouble */
    affichage(e);
}

```

### TypeAgregeAvecTableau.java - Exemple d'exécution



### Exemples d'utilisation des tableaux

- Actions de d'initialisation d'un tableau de réels:
  - Avec des zéro
  - Avec des nombres tirés au sort entre 0.0 et 1.0
  - Avec une suite croissante de nombres réels

```

{ Initialisation d'un tableau }
{ de reel avec des 0.0 }
{ tab : Le tableau de réels à initialiser }

action initialisation(tab)
  Sorties
  Tableau[] de reel tab
  Locales
  entier i
  pour i de 0 à longueur(tab)-1 faire
    tab[i] <- 0.0
  fait
fin action

{ Initialisation d'un tableau de reel avec }
{ des valeurs tirees au sort entre 0.0 et 1.0 }
{ tab : Le tableau de réels à initialiser }

action initRand(tab)
  Sorties
  Tableau[] de reel tab
  Locales
  entier i

```

```

    pour i de 0 à longueur(tab)-1 faire
        tab[i] <- random()
    fait
fin action

{ Creation d'un tableau de reel initialise avec }
{ une serie croissante de nombres aleatoires   }
{ tab : Le tableau de réels à initialiser      }

action initRandCroissant(tab)
    Sorties
        Tableau[] de reel tab
    Locales
        entier i
    tab[0] <- random()
    pour i de 1 à longueur(tab)-1 faire
        tab[i] <- tab[i-1]+random()
    fait
fin action

```

### InitialisationTableauReel.lida

```

/* Fonction d'affichage d'un tableau de double */
/* tab : Le tableau de double a afficher      */

static void affichage(double [] tab) {
    int i;
    for ( i = 0 ; i < tab.length ; i = i+1 ) {
        Ecran.formater("%6.3f",tab[i]); }
    Ecran.sautDeLigne();
}

/* Fonction d'initialisation d'un tableau */
/* de double avec des 0.0                */
/* tab : Le tableau de double a initialiser */

static void initialisation(double [] tab) {
    int i;
    for ( i = 0 ; i < tab.length ; i = i+1 ) {
        tab[i] = 0.0; }
}

/* Fonction d'initialisation d'un tableau */
/* de double avec des reels tires au sort */
/* entre 0.0 et 1.0                       */
/* tab : Le tableau de double a initialiser */

static void initRand(double [] tab) {
    int i;
    for ( i = 0 ; i < tab.length ; i = i+1 ) {

```

```
        tab[i] = Math.random(); }
    }

/* Fonction d'initialisation d'un tableau          */
/* de double avec une serie croissante           */
/* de nombres aleatoires                         */
/* tab : Le tableau de double a initialiser      */

static void initRandCroissant(double [] tab) {
    int i;
    tab[0] = Math.random();
    for ( i = 1 ; i < tab.length ; i = i+1 ) {
        tab[i] = tab[i-1]+Math.random(); }
    }

/* Programme principal                            */

public static void main(String [] args) {
    int i;
    double [] t1 = { 0.0,1.0,0.0,1.0,0.0 };
    double [] t2 = new double[6];
    double [] t3 = new double[6];
    /* Test de la fonction d'initialisation a 0.0 */
    Ecran.afficherln("Un tableau de 5 réels avant");
    Ecran.afficherln("et après initialisation à 0.0");
    affichage(t1);
    initialisation(t1);
    affichage(t1);
    Ecran.sautDeLigne();
    /* Test de la fonction d'initialisation avec   */
    /* des nombres aleatoires entre 0.0 et 1.0    */
    Ecran.afficherln
("Un tableau de 6 réels avant et après");
    Ecran.afficherln
("initialisation avec des nombres aléatoires");
    affichage(t2);
    initRand(t2);
    affichage(t2);
    Ecran.sautDeLigne();
    /* Test de la fonction de creation d'une serie */
    /* croissante de nombres aleatoires           */
    Ecran.afficherln
("Un tableau de 6 réels avant et après initialisation");
    Ecran.afficherln
("avec une série croissante de nombres aléatoires");
    affichage(t3);
    initRandCroissant(t3);
    affichage(t3);
}
```

InitialisationTableauReel.java - Exemple d'exécution

```

InitialisationTableauReel
Fichier Propriétés
Un tableau de 5 réels avant
et après initialisation à 0.0
 0,000 1,000 0,000 1,000 0,000
 0,000 0,000 0,000 0,000 0,000

Un tableau de 6 réels avant et après
initialisation avec des nombres aléatoires
 0,000 0,000 0,000 0,000 0,000 0,000
 0,603 0,486 0,239 0,524 0,486 0,951

Un tableau de 6 réels avant et après initialisation
avec une série croissante de nombres aléatoires
 0,000 0,000 0,000 0,000 0,000 0,000
 0,512 0,672 1,296 2,012 2,494 2,882

```

- Action de recherche de la valeur minimale présente dans un tableau de réels

```

{ Fonction de calcul et retour          }
{ de la valeur minimale d'un tableau de reel }
{ t : le tableau de réels où la recherche   }
{     est effectuée                       }

```

```

reel fonction minimum(t)
  Entrées
  Tableau [] de reel t
  Locales
  reel min
  entier i
  dans le cas de longueur(t)
  0 :
  min <- NaN
  1 :
  min <- t[0]
  autres cas :
  min <- t[0]
  pour i de 1 à longueur(t)-1 faire
  si t[i] < min alors
  min <- t[i]
  fsi
  fait
  fcas

```

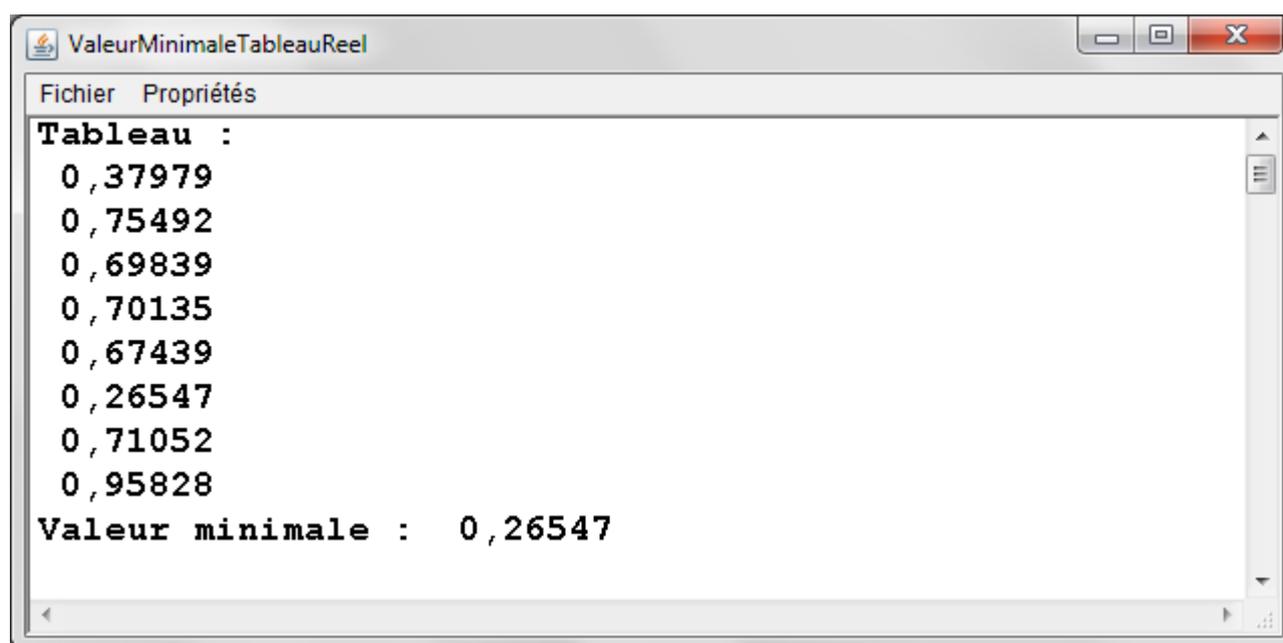
```
    retourner min
fin fonction
```

### ValeurMinimaleTableauReel.lida

```
/* Fonction de calcul et retour de la valeur      */
/* minimale d'un tableau de double                */
/* t : le tableau de réels où la recherche        */
/* est effectuée                                  */

static double minimum(double [] t) {
    int i;
    double min;
    switch (t.length) {
        case 0 : {
            min = Double.NaN; }
            break;
        case 1 : {
            min = t[0]; }
            break;
        default : {
            min = t[0];
            for ( i = 1 ; i < t.length ; i++ ) {
                if ( t[i] < min ) {
                    min = t[i]; } } } }
    return min;
}
```

### ValeurMinimaleTableauReel.java - Exemple d'exécution



```
ValeurMinimaleTableauReel
Fichier Propriétés
Tableau :
0,37979
0,75492
0,69839
0,70135
0,67439
0,26547
0,71052
0,95828
Valeur minimale : 0,26547
```

## Exercices

1) Développer un algorithme principal réalisant les traitements suivants:

- Déclaration d'un tableau unidimensionnel de 100 booléens
- Remplissage complet de ce tableau avec des booléens tirés au sort avec 70% de chance que la valeur tirée soit "vrai".

2) Développer un algorithme principal réalisant les traitements suivants:

- Déclaration d'un tableau bidimensionnel de 20x30 caractères
- Remplissage complet de ce tableau avec des caractères alphabétiques minuscules tirés au sort avec équiprobabilité
- Comptage puis affichage du nombre de 'e' présents dans ce tableau

3) Développer et valider un sous-algorithme réalisant les traitements suivants:

- Déclaration d'un tableau unidimensionnel de n booléens (n entier donné -> passé en paramètre de sous-algorithme)
- Remplissage complet de ce tableau avec des booléens tirés au sort avec équiprobabilité
- Retour du tableau ainsi créé

4) Développer et valider un sous-algorithme réalisant les traitements suivants:

- Affichage complet des valeurs contenues dans un tableau unidimensionnel de booléens

5) On définit le type agrégé suivant:

```
static class Duree {  
    int mn = 0;  
    int s = 0; };
```

Ce type agrégé est utilisé pour coder des durées formées d'un nombre de minutes et d'un nombre de secondes ([0,59]).

a) Développer et valider un sous-algorithme réalisant les traitements suivants:

- Déclaration d'un tableau unidimensionnel de n Duree (n entier donné -> passé en paramètre de sous-algorithme)
- Retour du tableau ainsi créé

b) Développer et valider un sous-algorithme réalisant les traitements suivants:

- Déclaration d'un tableau unidimensionnel de n Duree (n entier donné -> passé en paramètre de sous-algorithme)
- Remplissage complet de ce tableau avec des durées tirées au sort: nombre de minutes tiré dans l'intervalle [0,10], nombre de secondes tiré dans l'intervalle [0,59].
- Retour du tableau ainsi créé

c) Développer et valider un sous-algorithme réalisant les traitements suivants:

- Calcul de la somme des durées contenues dans un tableau de Duree td donné (td donné -> passé en paramètre de sous-algorithme)
- Retour de la durée ainsi calculée

6) On souhaite coder des polygones du plan en mémoire d'une application informatique. Un polygone est caractérisé par la liste ordonnée des positions de ses n sommets.

a) Développer un type agrégé permettant de stocker un polygone

b) Développer et valider un sous-algorithme réalisant les traitements suivants:

- Affichage du nombre de sommets d'un polygone
- Affichage de la position de tous les sommets d'un polygone

c) Développer et valider un sous-algorithme réalisant les traitements suivants:

- Création d'un Polygone pour qu'il représente un polygone régulier à n sommets, qu'il soit centré sur l'origine et qu'il soit de rayon r. L'entier n et le réel r sont donnés et donc passés en paramètre de sous-algorithme.
- Retour de ce polygone

d) Développer et valider un sous-algorithme réalisant les traitements suivants:

- Calcul du périmètre d'un Polygone p donné (p donné -> passé en paramètre de sous-algorithme)
- Retour de la valeur ainsi calculée

---

Auteur: Nicolas JANEY  
UFR Sciences et Techniques  
Université de Besançon  
16 Route de Gray, 25030 Besançon  
[nicolas.janey@univ-fcomte.fr](mailto:nicolas.janey@univ-fcomte.fr)

---