Algorithmique & Programmation Orientée Objet Semestre 2 ST

Les tableaux de variables

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 d'échec (vide ou occupé par telle ou telle pièce)

- 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'un indice entier (ou de plusieurs indices entiers)

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 de taille (s) définie(s)
- Type, dimension non modifiables par la suite
- Taille(s) modifiable(s) par la suite mais pas de façon simple (voir plus loin)

Syntaxe en langage algorithmique

- **Déclaration en dimension 1**

```
type [n] nomVariable
```

- Déclaration avec allocation pour un nombre fixé d'éléments
- n: Nombre d'éléments du tableau (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

- nomVariable: Nom de la variable

```
type [] nomVariable
```
- Déclaration sans allocation pour les éléments
- type: Type élémentaire du tableau
- nomVariable: Nom de la variable

- **Déclaration en dimension 2**

```
type [n1][n2] nomVariable
```
- Déclaration avec allocation pour un nombre fixé d'éléments
- n1 et n2: Tailles entières selon chacune des 2 dimensions (selon chacun des deux indices)
- type: Type élémentaire du tableau
- nomVariable: Nom de la variable

```
type [][] nomVariable
```
- Déclaration sans allocation pour les éléments
- type: Type élémentaire du tableau
- nomVariable: Nom de la variable

- **Généralisation en dimension D**

```
type [n1][n2]...[nD] nomVariable
```
- Déclaration avec allocation pour un nombre fixé d'éléments
- n1, n2, ..., nD: Tailles entières (selon chacune des D dimensions, selon chacun des D indices)
- type: Type élémentaire du tableau
- nomVariable: Nom de la variable

```
type []...[] nomVariable
```
- Déclaration sans allocation pour les éléments
- type: Type élémentaire du tableau
- nomVariable: Nom de la variable

ATTENTION: Utilisation directe impossible des tableaux déclarés sans allocation pour leurs éléments.

ATTENTION: Non initialisation des éléments des tableaux spécifiés avec une taille au moment de leur déclaration (i.e. les éléments ne sont pas initialisés et sont donc de valeur aléatoire)

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

- En consultation ou en affectation:

`nomVariable[indice]`
- Accès aux éléments d'un tableau de dimension 1
- nomVariable: Nom de la variable tableau
- indice: Constante entière (littérale ou non), variable entière ou expression à résultat entier

`nomVariable[indice1][indice2]`
- Accès aux éléments d'un tableau de dimension 2
- nomVariable: Nom de la variable tableau
- indice1 et indice2: Constantes entières (littérales ou non), variables entières ou expressions à résultat entier

`nomVariable[i1][i2]...[iD]`
- Accès aux éléments d'un tableau de dimension D
- nomVariable: Nom de la variable tableau
- 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 inclus (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 booleens      }
{ et initialisation a vrai                    }

action principale()
   booleen [8] 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 premieres valeurs de factoriel  }

constante entier N <- 50

action principale()
   entier [N] 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 noir                       }
{  - '.' pour case vide                       }
{ Affichage du tableau apres initialisation   }

action principale()
   caractere [10][10] 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 dynamique de la taille d'un tableau**
- Utilisation de fonctions ad hoc

`longueur(t)`

- Retour de la valeur entière définissant la taille selon le premier indice d'un tableau de dimension quelconque
- t: Tableau de dimension quelconque

`longueur(n,t)`

- Retour de la valeur entière définissant la taille selon le $n^{ième}$ indice d'un tableau de dimension au moins égale à n
- n: Numéro de la dimension (de 1 à D où D est la dimension du tableau)
- t: Tableau

- Pour les tableaux de dimension 2 ou supérieure: longueur(t) = longueur(1,t)
- **Exemple n°4: Accès dynamique à la taille et utilisation de cette valeur**
  Déclaration d'un tableau d'entiers, initialisation de ce tableau avec les premiers entiers pairs
  Déclaration d'un tableau de chaines de caractères à deux indices, initialisation de ce tableau avec la chaine de caractère vide

```
{ Initialisation d'un tableau d'entiers       }
{ de dimension 1                              }
{ Initialisation d'un tableau de chaines      }
{ de dimension 2                              }
{ Utilisation des fonctions longueur          }
{ pour determiner les tailles de ces tableaux }

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

action principale()
   entier [N] te
   chaine [N][M] 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**

- Trois syntaxes

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

- Déclaration avec définition explicite de la taille, allocation et initialisation implicite automatique à "zéro" des éléments
- type: Type élémentaire du tableau
- n: Taille du tableau
  - Constante entière littérale ou non
  - Variable entière
  - Expression numérique de résultat entier
- nomVariable: Nom de la variable tableau

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

- Déclaration avec définition implicite de la taille, allocation et initialisation explicite des éléments
- type: Type élémentaire du tableau
- V1, V2, ..., Vn: Valeurs d'initialisation affectées aux éléments du tableau
  (obligatoirement du type élémentaire du tableau)
  - Constantes (littérales ou non),
  - Variables
  - Expressions
- Nombre d'items entre accolades: Taille du tableau
- nomVariable: Nom de la variable tableau

```
type [] nomVariable;
```

- Déclaration sans définition de la taille ni allocation des éléments

- type: Type élémentaire du tableau
- nomVariable: Nom de la variable tableau

- **Syntaxe de déclaration en dimension 2**

- Trois syntaxes

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

- Déclaration avec définition explicite des deux tailles, allocation et initialisation implicite à "zéro" des éléments
- type: Type élémentaire du tableau
- n et m: Tailles du tableau
  - Constantes entière
  - Variables entière
  - Expressions numériques de résultat entier
- nomVariable: Nom de la variable tableau de dimension 2

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

- Déclaration avec définition implicite des deux tailles, allocation et initialisation explicite des éléments
- type: Type élémentaire du tableau
- V11, ..., Vnm: Valeurs d'initialisation (obligatoirement du type élémentaire du tableau)
  - Constantes littérales ou non
  - Variables
  - Expressions
- Nombre de lignes entre accolades: Taille du tableau selon le premier indice
- Nombre de colonnes entre accolades: Taille du tableau selon de second indice
- nomVariable: Nom de la variable tableau de dimension 2

```
type [][] nomVariable;
```

- Déclaration sans définition de la taille ni d'allocation des éléments
- type: Type élémentaire du tableau
- nomVariable: Nom de la variable tableau de dimension 2

- **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

ATTENTION: Utilisation directe impossible des tableaux déclarés sans allocation pour leurs éléments.

- **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 inclus) si N est la taille selon la dimension considérée

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

- Syntaxe particulière au langage Java

`t.length`
- Taille selon le premier indice d'un tableau de dimension quelconque
- t: Tableau

`t[0].length`
- Taille de la première ligne d'un tableau de dimension égale au moins à 2 = taille selon le 2ème indice si toutes les lignes ont la même taille
- 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       */
/* initialises a vrai                           */

public class InitialisationTableauBoolean {

/* Programme principal                          */

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

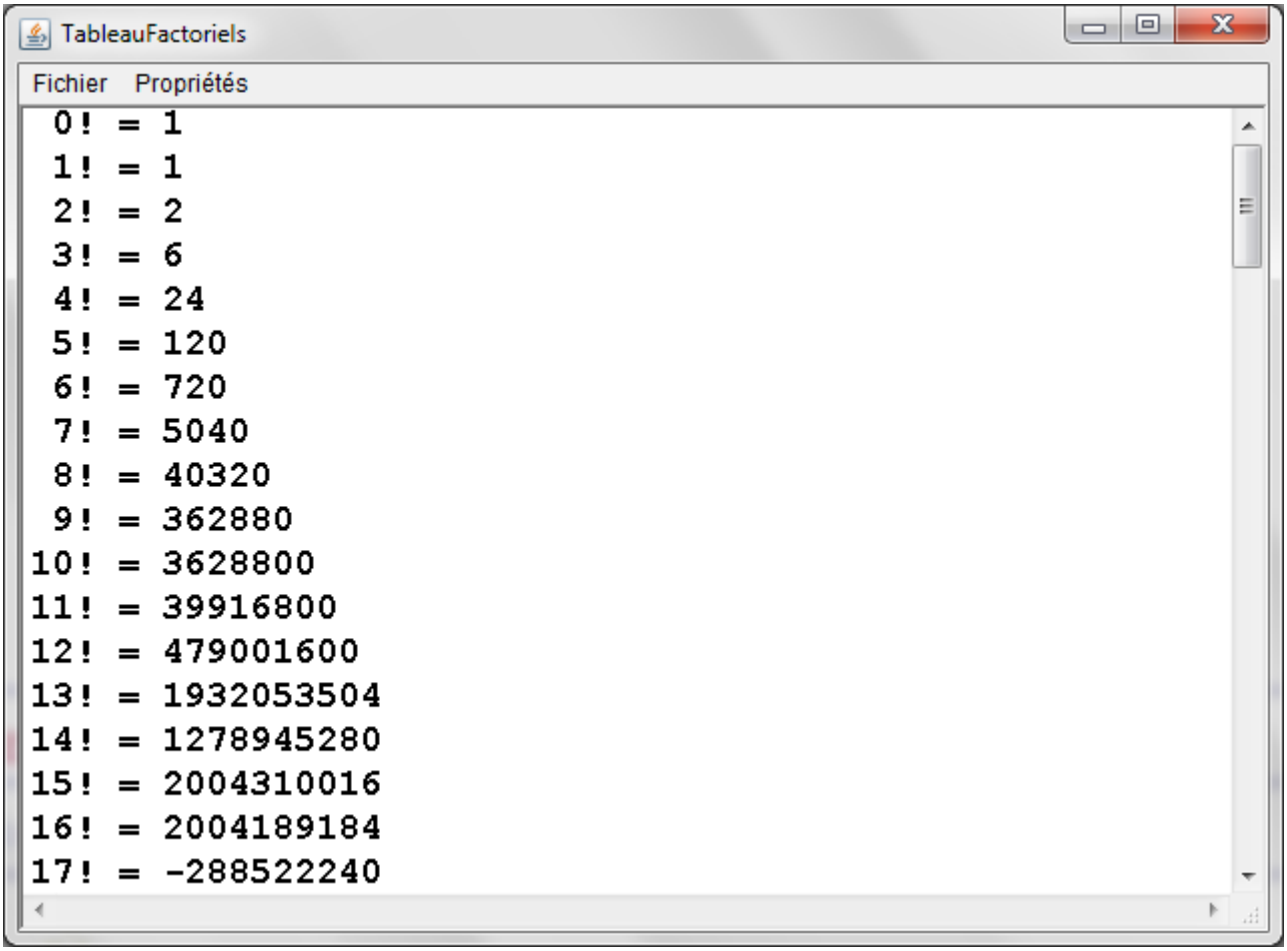
```
      int n = tb.length;
     Ecran.afficherln(n);
     Ecran.afficherln(tb[0]);
     Ecran.afficherln(tb[n-1]);
     Ecran.afficherln(tb);
   }
}
```

InitialisationTableauBoolean.java - Exemple d'exécution

- **Exemple n°2:**
  Déclaration d'un tableau de 50 entiers, remplissage de ce tableau avec les 50 premières valeurs de n! puis affichage du contenu de ce 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]); }
  }
}
```

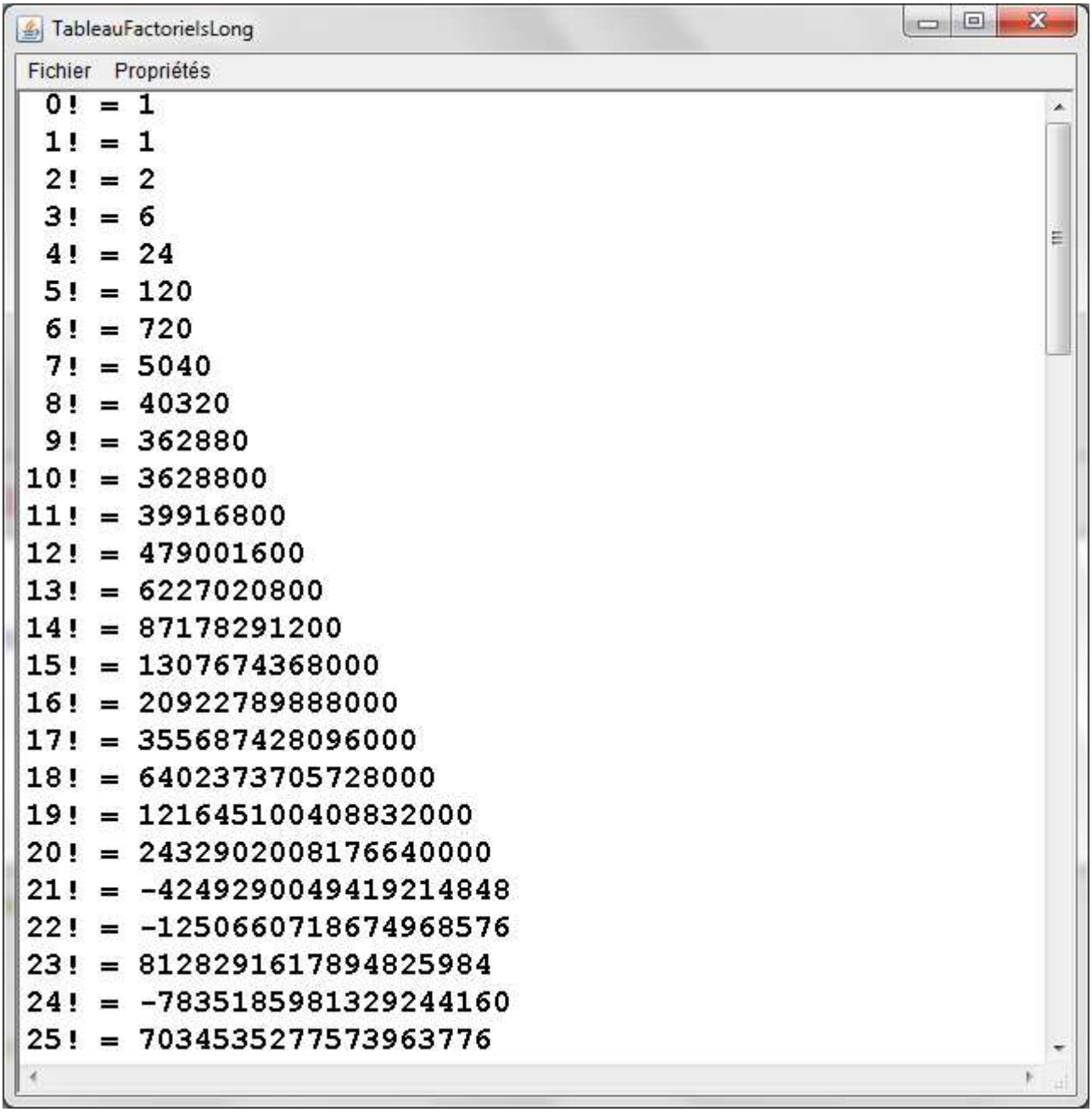
TableauFactoriels.java - Exemple d'exécution



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



- **Exemple n°3:**

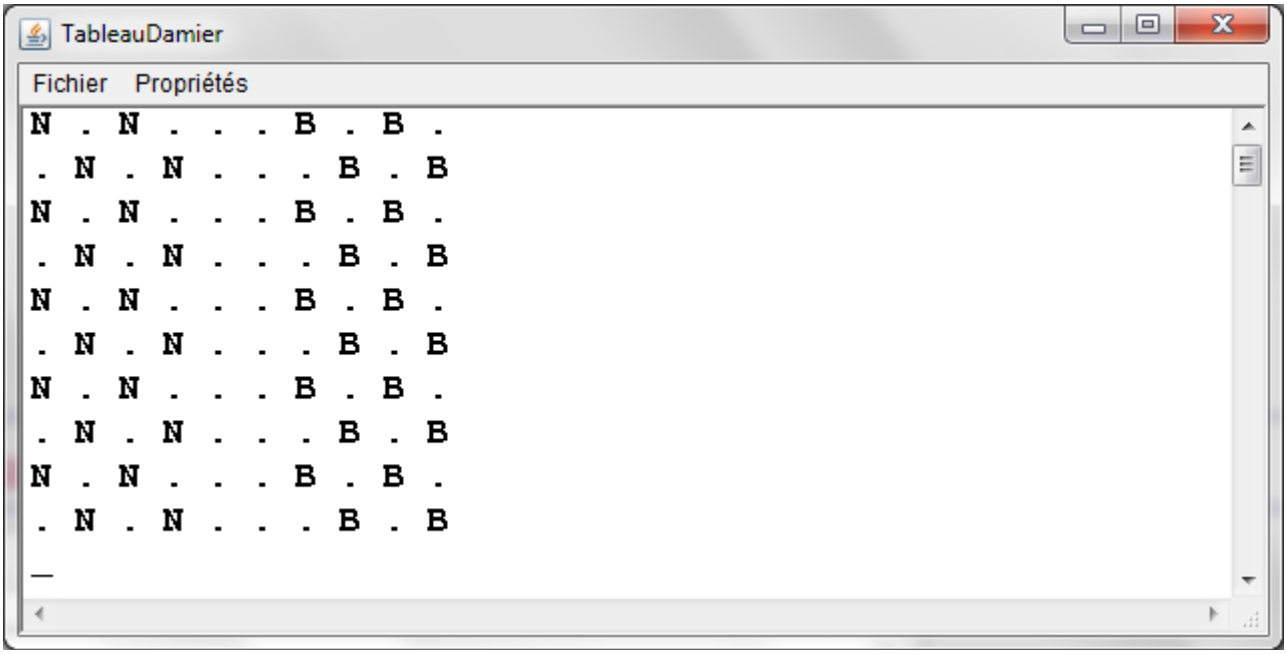
  Déclaration et initialisation d'un tableau de 10x10 caractères modélisant un damier de jeu de dames en début de jeu:
  - Cases avec pion noir représentées par le caractère 'N'
  - Cases avec pion blanc représentées par le caractère 'B'
  - Cases vides représentées par le caractère '.'

  Affichage du damier

```
/* Initialisation d'un damier de jeu de dames  */
/* 'B' pour pion blanc                         */
/* 'N' pour pion noir                          */
/* '.' 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**

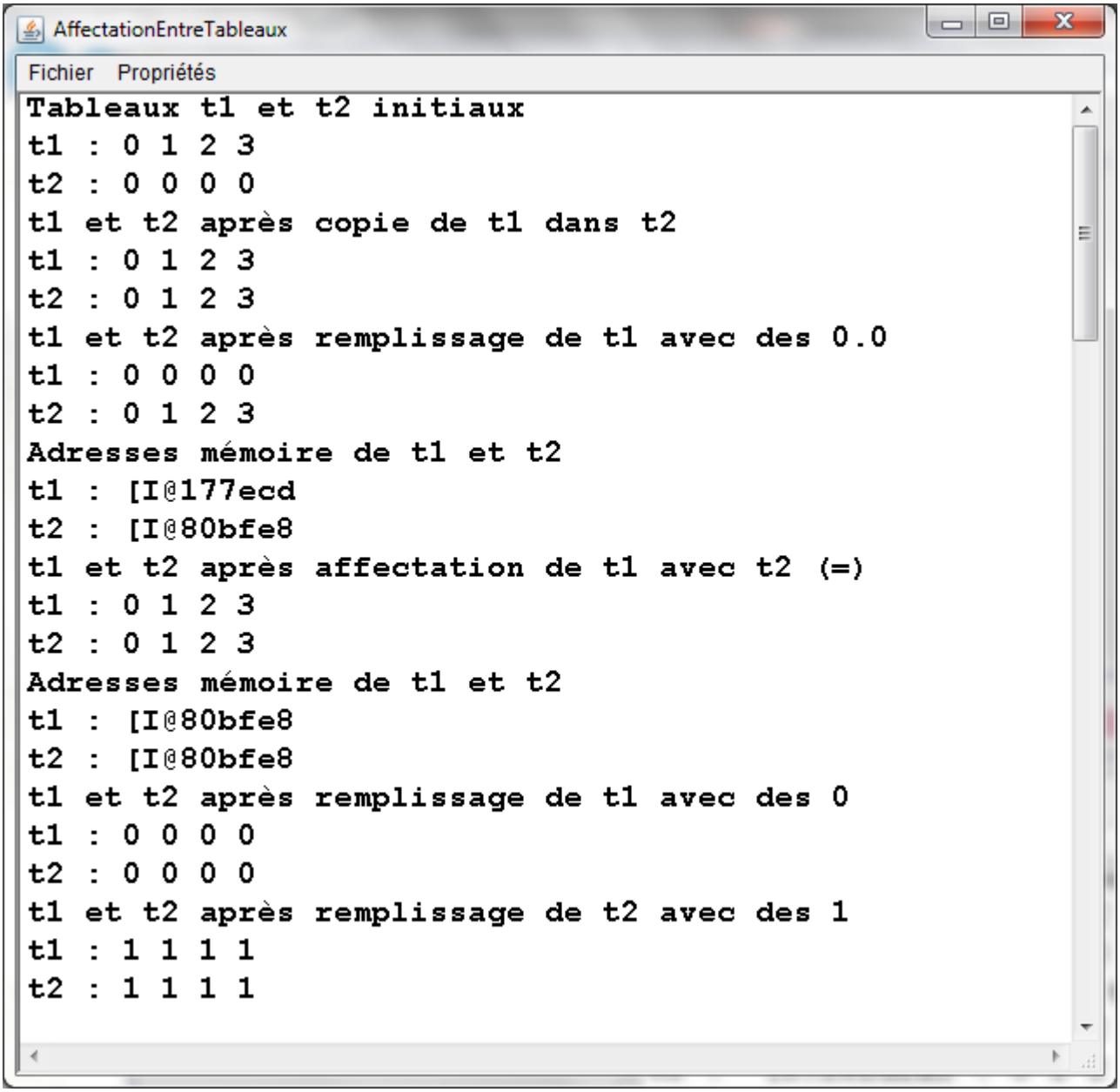
- En langage algorithmique et en langage Java, affectation impossible entre tableaux de types élémentaires ou de dimensions différentes
- En langage algorithmique et en langage Java, même comportement que pour les types agrégés si affectation (signe <- en PDL, signe = en Java) entre deux variables tableau ayant les mêmes type élémentaire et dimension:
  Affectation des adresses mémoire 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
  -> Perte des valeurs à gauche ainsi que de la taille (des tailles) remplacée(s) par celle(s) de la variable droite
  Solution pour réaliser une "vraie" affectation: Développer le code d'affectation place pour place des composantes des deux tableaux
  Recommandation: Développer un sous-algorithme 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



- **Test d'égalité et test de différence**

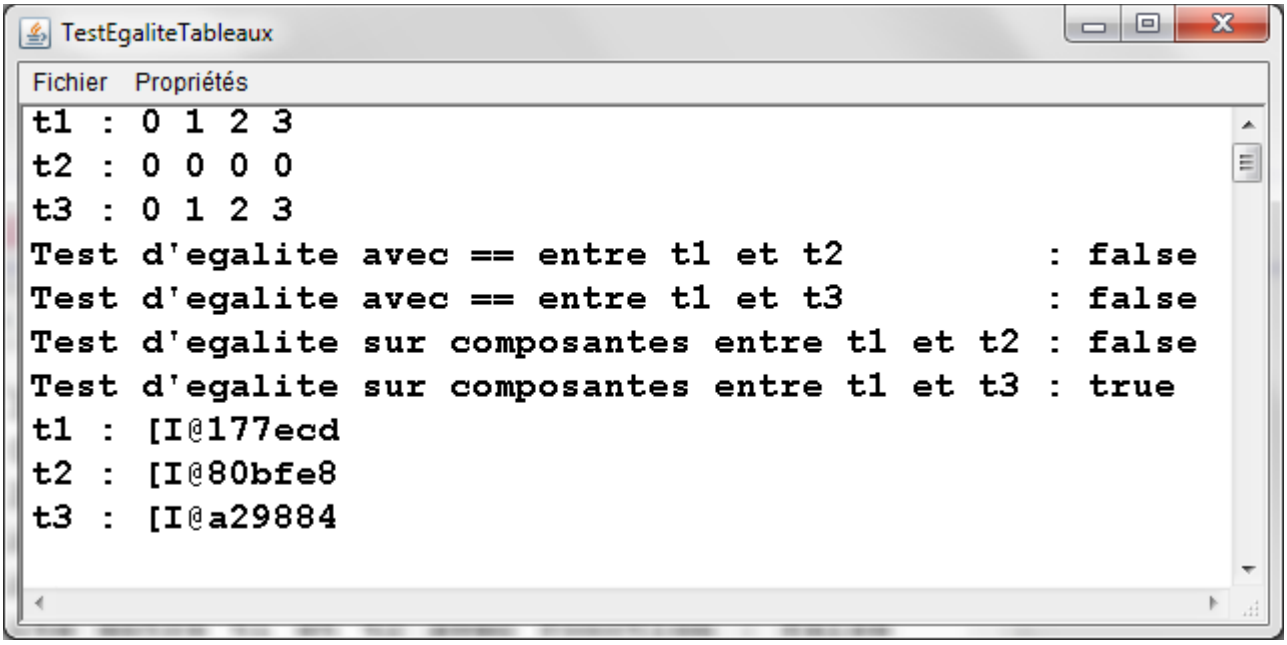
- En langage algorithmique et en langage Java, tests d'égalité et de différence impossibles entre tableaux de types élémentaires ou de dimensions différentes
- En langage algorithmique et en langage Java, tests d'égalité et de différence possibles entre deux tableaux ayant les mêmes dimension et type élémentaire. Utilisation des signes == et !=. ATTENTION : le test porte sur les adresses mémoires respectives et ne porte donc pas sur les composantes place pour place -> Comportement différent de celui souhaité
  Solution pour un "vrai" test entre tableaux: Développer le code de test place pour place des composantes des deux tableaux
  Recommandation: Développer un sous-algorithme 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 mémoire                */
   /* des trois tableaux                            */
  Ecran.afficherln("t1 : ",t1);
  Ecran.afficherln("t2 : ",t2);
  Ecran.afficherln("t3 : ",t3);
}
```

TestEgaliteTableaux.java - Exemple d'exécution



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 fonctionnaliés du langage utilisé permettant de récupérer dynamiquement les tailles

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

- **Exemples**

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

action affichage(-> entier [] t)
   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(-> entier [] tab ->)
   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                             }

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

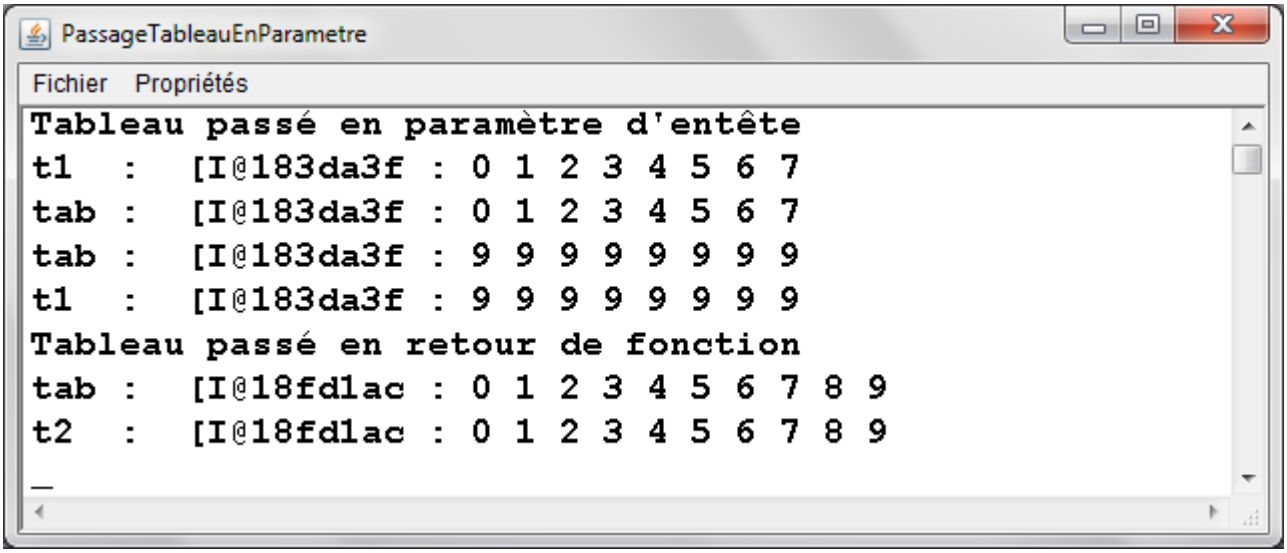
```
{ Action principale                                  }

action principale()
   entier [8] t1
   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.lda

```
/* 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



- **Gestion du retour des fonctions**

- Pas de problème:
  - Affectation du tableau retourné à une variable tableau de mêmes caractérisques (dimension et type élémentaire) en utilisant l'opérateur d'affectation (= en Java, <- en PDL)
  -> Disparition du contenu de l'ancienne variable tableau remplacé par le tableau retourné
  -> En Java, pas nécessaire de faire d'allocation mémoire (2 syntaxes, voir plus haut) dans le programme appelant car elle est fait dans la fonction sur la variable retournée en fin de fonction
  -> En langage algorithmique, pas nécessaire de spécifier la taille du tableau destiné à être affecté, car l'allocation est faite dans la fonction appelée et l'affectation écrase l'ancienne taille

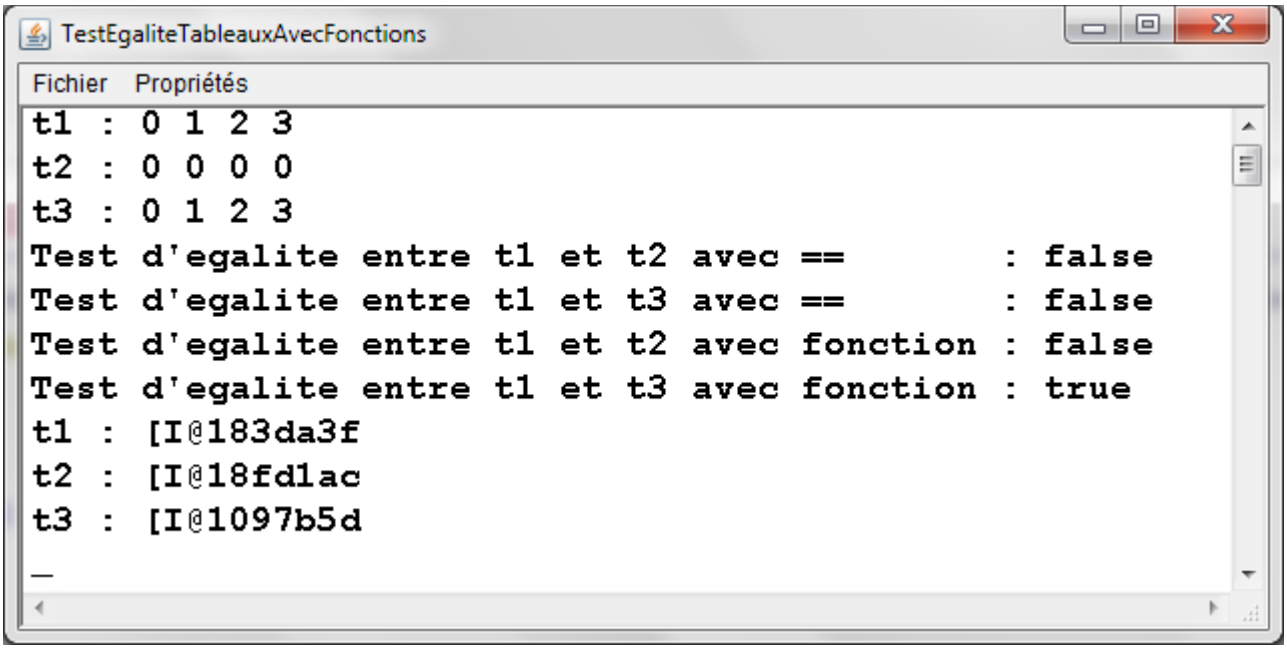
- **Exemples supplémentaires**: Réécriture des deux programmes exemples du paragraphe sur l'affectation et le test d'égalité entre tableaux
-> utilisation de fonctions

```
/* 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

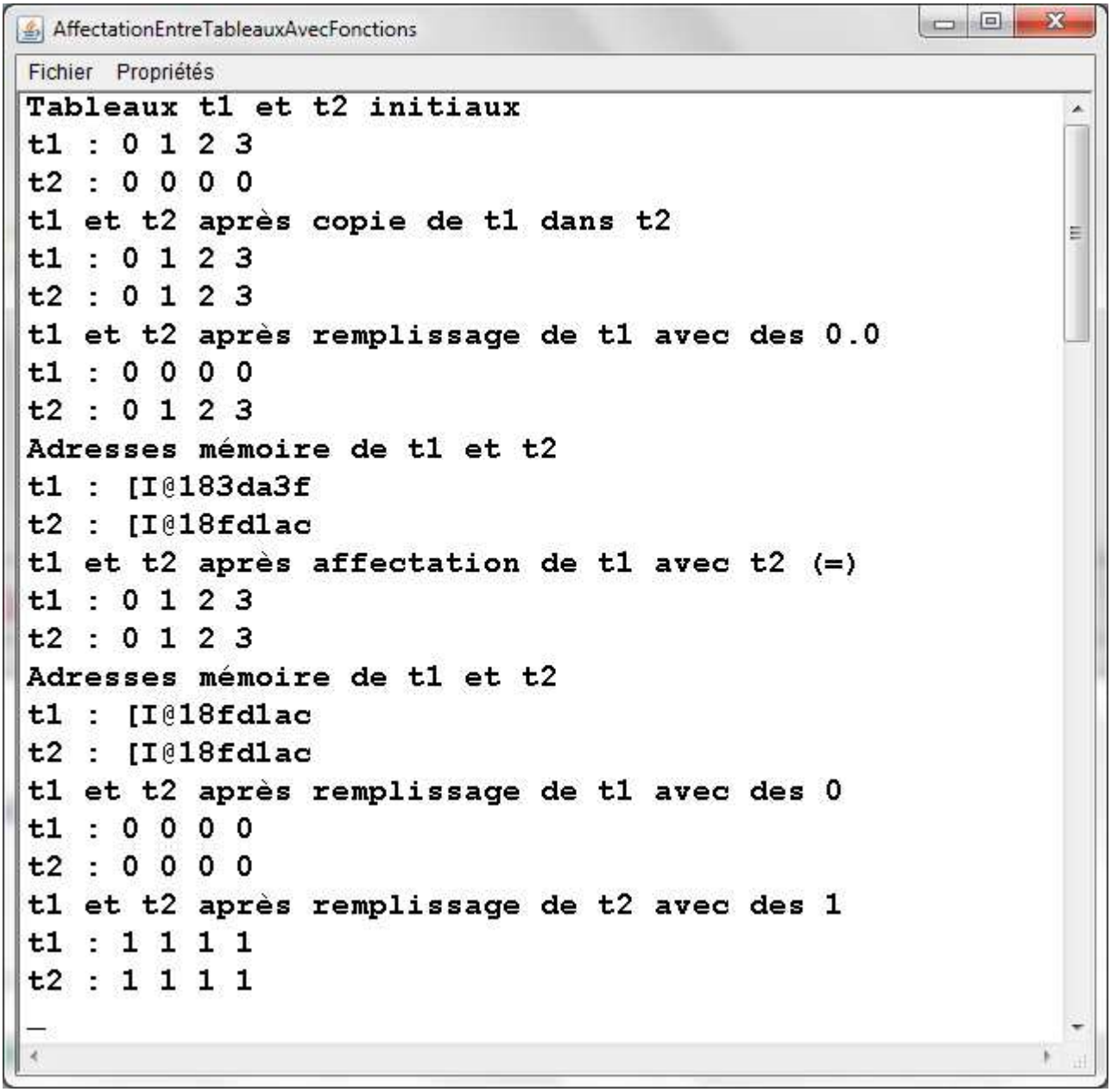



```
/* Fonction d'affichage d'un tableau d'entiers */
/* t : Le tableau de int a afficher            */

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



Tableaux de variables de type agrégé

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

```
nomStructure [n1][n2]...[nD] nomVariable
```

- Déclaration d'un tableau dont le type élémentaire est un type agrégé
- nomStructure: Nom du type agrégé élémentaire
- n1, n2, ..., nD: Constante entière (littérale ou non), variable entière ou expression numérique à résultat entier définissant les tailles (selon chacune des D dimensions)
- nomVariable: Nom de la variable tableau

- En langage Java, pas de déclaration d'un tableau de variables de type agrégé en une seule ligne de code car toute variable de type agrégé ou de type tableau doit être déclarée par un new or on a ici un tableau qui contient des composantes de type agrégé
  - Déclaration du tableau par opérateur new
  - Déclaration obligatoire par opérateur new de chacune des composantes de type agrégé du tableau

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

- Code de déclaration d'un tableau de type élémentaire agrégé en Java en dimension 1
- TypeAgrégé: Nom du type agrégé élémentaire
- n: Constante entière (littérale ou non), variable entière ou expression numérique à résultat entier définissant la taille selon l'unique dimension
- nomVariable: Le nom de la variable tableau

```
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(); } } }
```

- Code de déclaration d'un tableau de type élémentaire agrégé en Java en dimension D (D indices)
- TypeAgrégé: Nom du type agrégé élémentaire
- n1, n2, ..., nD: Constantes entières, variables entières ou expressions numériques à résultat entier définissant les tailles selon chacune des D dimensions du tableau
- nomVariable: Le nom de la variable tableau

- **Recommandation**: En Java, utiliser une fonction pour réaliser l'implantation du code de déclaration (isolation du problème)

- 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`
  - Pour une variable tableau de dimension 2:
    `nomVariable[indice1][indice2].nomChamp`

- Pour une variable tableau de dimension D:

  `nomVariable[indice1][indice2]...[indiceD].nomChamp`

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

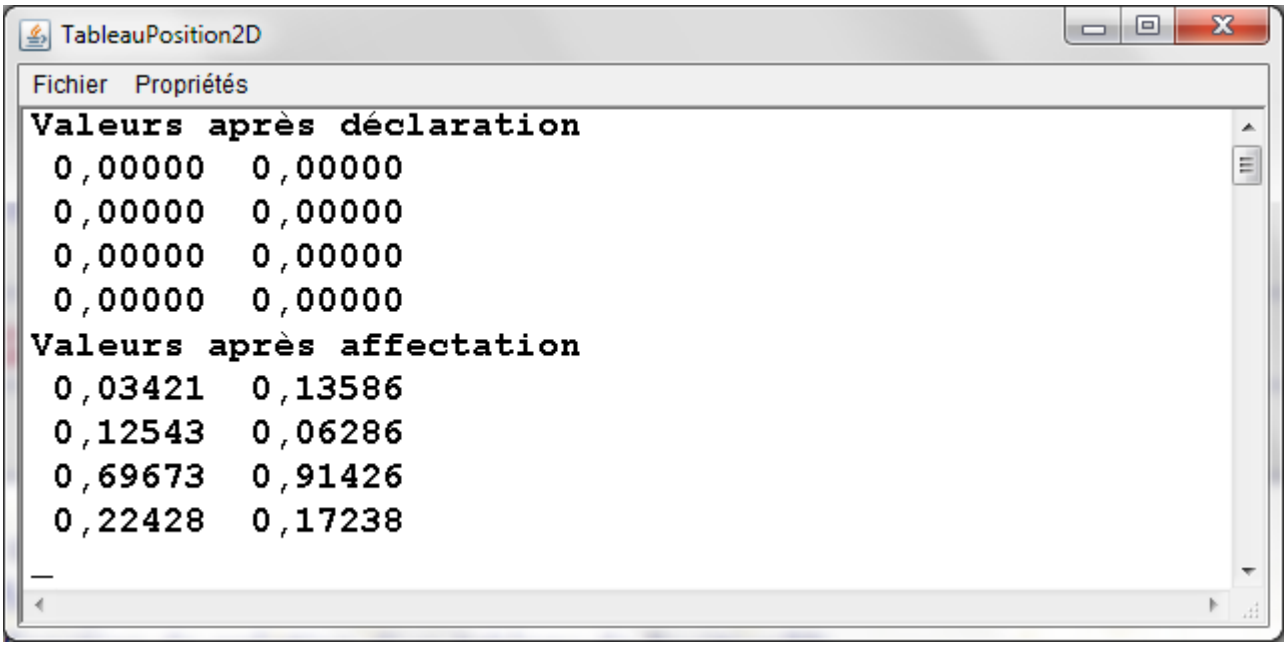
```
{ Declaration d'un tableau de variables                }
{ de type agrege position en deux dimensions           }

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

{ Programme principal                                  }

constante entier TAILLE <- 3

action principale()
   entier i
   position2D [TAILLE] 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.lda

```
/* 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



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(-> 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(-> position2D [] tpos)
   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()
  position2D [TAILLE] tpos
  affichage(tpos)
fin action
```

TableauPosition2DAvecFonctions.lda

```
/* 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 = 5;
   /* Declaration du tableau tpos1                 */
   /* par utilisation de la fonction ad hoc        */
  Position2D [] tpos1 = tableauDePosition2D(TAILLE);
   /* Affichage des Position2D du tableau tpos1    */
  Ecran.afficherln("Premier tableau");
  affichage(tpos1);
   /* Declaration du tableau tpos1                 */
   /* avec intialisation de ses champs             */
   /* lors de la declaration                       */
  Position2D [] tpos2 = { new Position2D(),
                          new Position2D(),
                          new Position2D() };
   /* Affichage des Position2D du tableau tpos2    */
  Ecran.afficherln("Second tableau");
  affichage(tpos2);
}
```

TableauPosition2DAvecFonctions.java - Exemple d'exécution

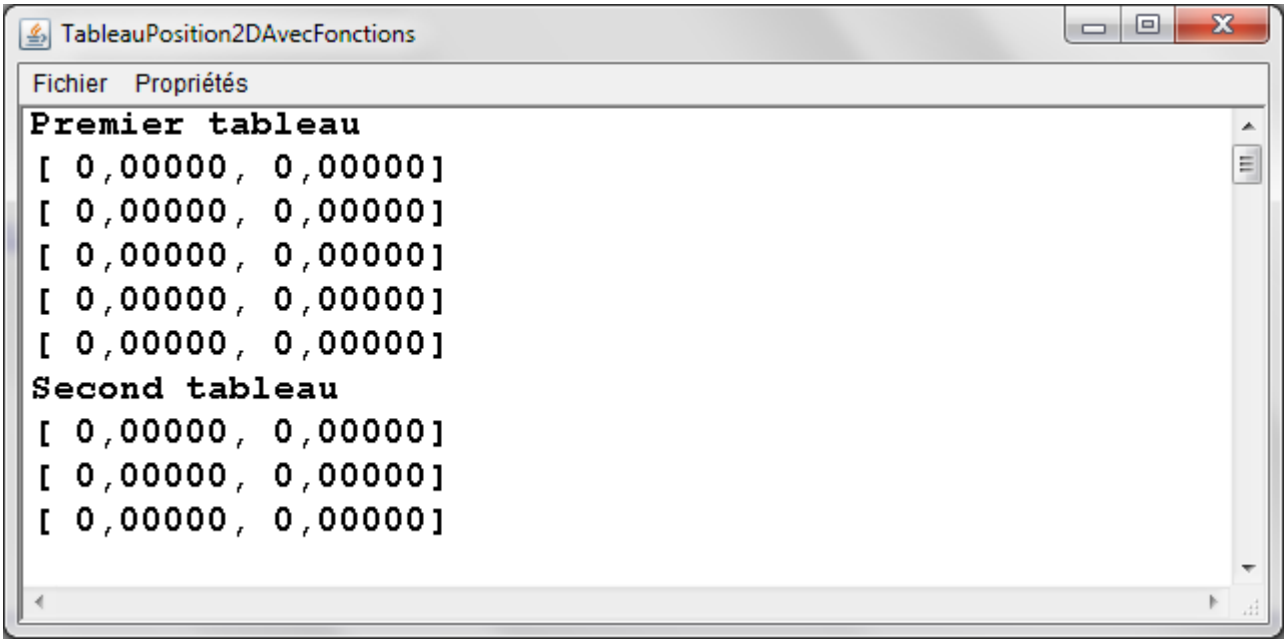


Tableau en tant que champ dans un type agrégé

- Champs de type tableau autorisés dans les types agrégés
- En langage algorithmique

```
structure nomType
  type [n1][n2]...[nD] nomChamp
  ...
fin structure
```

  - Syntaxe en dimension D
  - nomType: Le nom du type agrégé créé
  - type: Le type élémentaire du champ défini
  - n1, n2, ..., nD: Constantes entières (littérales ou non), variables entières ou expressions numériques à résultat entier définissant les tailles selon chacune des D dimensions
  - nomChamp: Le nom du champ de type tableau créé au sein du type agrégé nomType

- En langage Java

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

  - Syntaxe en dimension D pour un tableau de variables de type prédéfini
  - NomType: Le nom du type agrégé créé
  - type: Le type élémentaire du champ de type tableau

- n1, n2, ..., nD: Constantes entières (littérales ou non), variables entières ou expressions numériques à résultat entier définissant les tailles selon chacune des D dimensions
- nomChamp: Le nom du champ de type tableau créé au sein du type agrégé NomType

- Syntaxes d'utilisation identiques en langage algorithmique et en langage Java pour les variables déclarées de type agrégé incluant un ou des champs de type tableau:

  - Pour une variable agrégée ayant un champ de type tableau de dimension 1:
    `nomVariable.nomChampTableau[indice]`
  - Pour une variable agrégée ayant un champ de type tableau de dimension 2:
    `nomVariable.nomChampTableau[indice1][indice2]`
  - Pour une variable agrégée ayant un champ de type tableau de dimension D:
    `nomVariable.nomChampTableau[indice1][indice2]...[indiceD]`

- **Exemple**: Une structure de stockage d'un ensemble d'au maximum TAILLE 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
   reel [TAILLE] 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
(-> entier nb,-> réel v,ensembleDeReel e ->)
   entier i
  e.n <- nb
   pour i de 0 à nb-1 faire
    e.t[i] <- v
```

```
   fait
fin action

{ Programme principal                                    }

action principale()
   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.lda

```
/* 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(reel [] tab ->)
   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(reel [] tab ->)
   entier i
   pour i de 0 à longueur(tab)-1 faire
    tab[i] <- random()
   fait
fin action

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

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

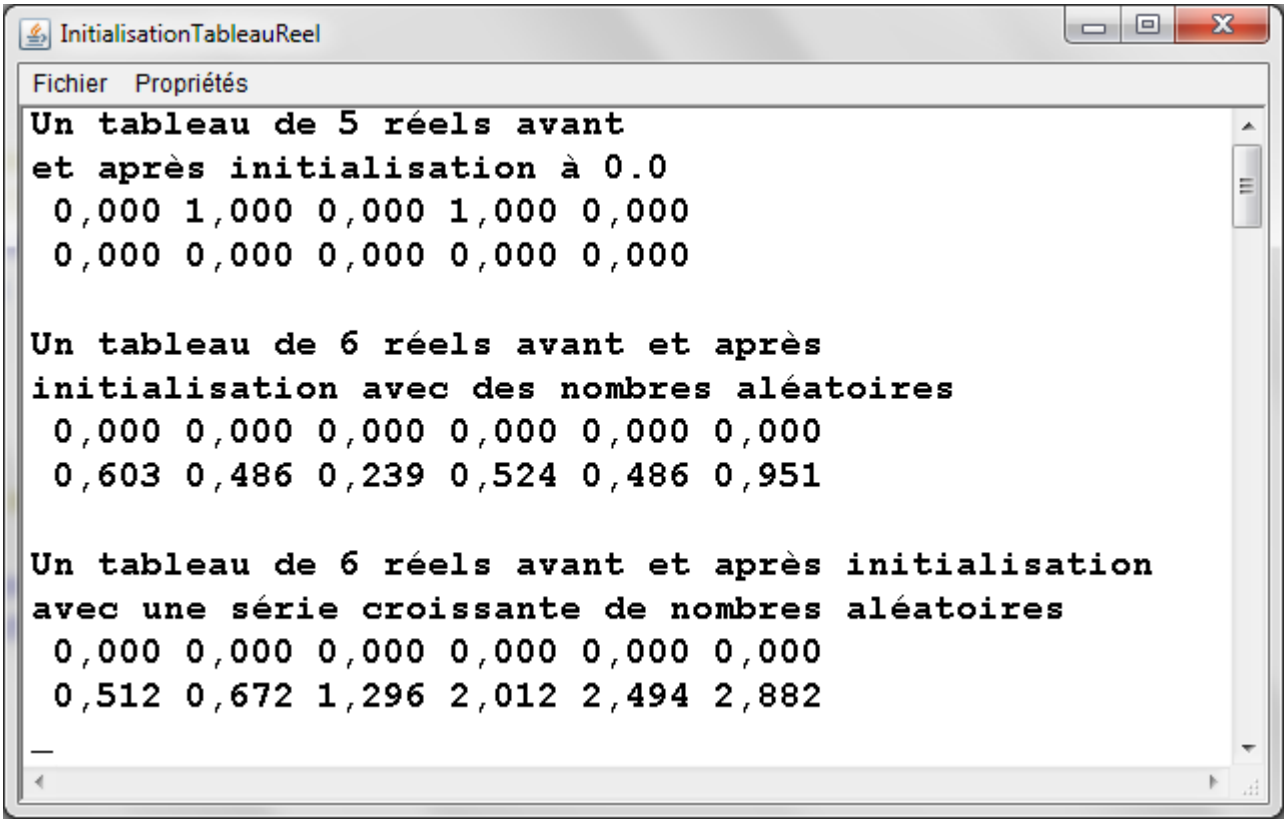
InitialisationTableauReel.lda

```
/* 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



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

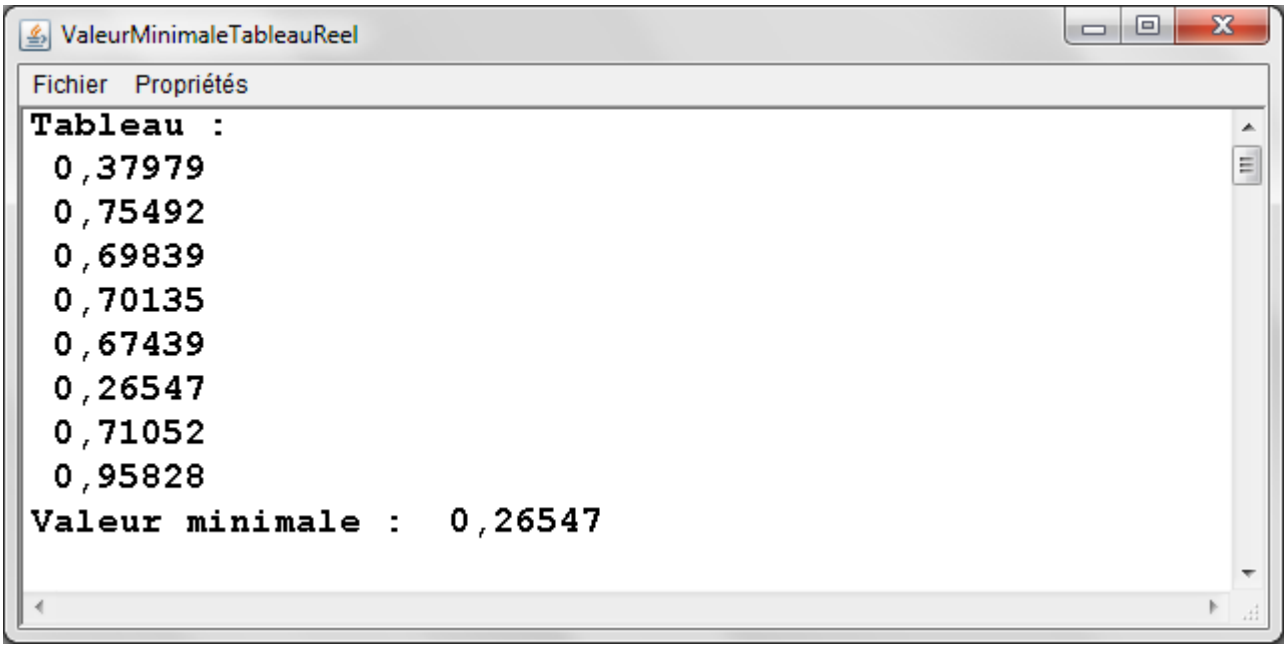
```
{ Fonction de calcul et retour                  }
{ de la valeur minimale d'un tableau de reel    }
{ t : le tableau de réels où la recherche       }
{     est effectuée                             }

reel fonction minimum(-> reel [] t)
   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.lda

```
/* 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



Exercices

1) Développer un algorithme principal réalisant les traitements suivants:

- Déclaration d'un tableau unidimentionnel 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 bidimentionnel 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 unidimentionnel 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 unidimentionnel 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 (valeur entière positive ou nulle) et d'un nombre de secondes (valeur entière comprise dans l'intervalle [0,59]).

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

- Déclaration d'un tableau unidimentionnel 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 unidimentionnel 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 est 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