Algorithmique & Programmation
Semestre 2 ST
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 de dames (blanc, noir ou vide)

- Définition d'autant de variables individuelles que de données devant être gérées
- Solution possible en théorie mais impossible 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 d'une sous-variable individuelle (du type élémentaire) au sein de cette variable par la donnée d'un indice (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 indice (D tailles)
  - N égal au produit des D tailles
- Type et taille(s) spécifiés lors de la déclaration d'un tableau
- Type et taille(s) non modifiables par la suite

Syntaxe en langage algorithmique

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

```
nomTableau : Tableau[N] de type
```

  - N: Constante entière littérale ou non littérale définissant la taille selon l'unique indice

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

```
nomTableau : Tableau[N][M] de type
```

  - N et M: Constantes entières littérales ou non définissant les tailles (selon chacune des 2 dimensions)

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

`nomTableau : Tableau[N1][N2]...[ND] de type`

- N1, N2, ..., ND: D constantes entières littérales ou non définissant les tailles (selon chacune des n dimensions)

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'un tableau de dimension 1**

  En lecture ou en écriture:
  `nomTableau[indice]`
  - indice: Constante entière, variable entière ou expression à résultat entier

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

  En lecture ou en écriture:
  `nomTableau[indice1][indice2]`
  - indice1 et indice2: Constantes entières, variables entières ou expressions à résultat entier

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

  En lecture ou en écriture:
  `nomTableau[i1][i2]...[iD]`
  - i1, i2, ..., iD: Constantes entières, variables entières ou expressions à résultat entier

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

ATTENTION: Pour un tableau de taille N, indices définis de 0 à N-1 et pas de 1 à N

- **Exemple n°1**:
  Déclaration et initialisation d'un tableau de 8 booléens avec vrai

```
{ Initialisation a vrai d'un tableau          }
{ de 8 booleens                               }

action principale()
   locales
    tb : Tableau[8] de booleen
```

```
    i : entier
   pour i de 0 à 7 faire
    tb[i] <- vrai
   fait
fin action
```

- **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   }

constante entier N <- 50

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

- **Exemple n°3:**
  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
    damier : Tableau[10][10] de caractere
    i,j : entier
```

```
    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
```

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
    - Variable entière
    - Expression numérique de résultat entier
  - Taille du tableau définitivement fixée après déclaration

  - Définition implicite de la taille par initialisation explicite du contenu

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

  - V1, V2, ..., Vn: Constantes, variables ou expressions à résultat du type déclaré affectées aux composantes 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
- Tailles du tableau définitivement fixées après déclaration

- Définition implicite de la taille par initialisation explicite du contenu

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

- V11, ..., Vnm: Constantes, variables ou expressions à résultat du type déclaré
- 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 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 du tableau suivi de D indices entiers (constantes, variables ou expressions) spécifiés individuellement entre crochets (D dimension du tableau)
- Pour chaque dimension, indice compté à partir de 0 jusqu'à la taille du tableau selon cette dimension - 1
- **Exemples**: `nomTableau[0]`, `nomTableau[1]`, `nomTableau[i]` , `nomTableau[i+k]` , `nomTableau[i][k]`

- Syntaxe Java de détermination de la taille selon le premier indice d'un tableau de dimension quelconque:

```
t.length
```

t: Tableau de dimension quelconque

- Syntaxe Java de détermination la 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

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

public class InitialisationTableauBoolean {

/* Programme principal                         */

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

InitialisationTableauBoolean.java - Exemple d'exécution

- **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 [] tb = new int[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.afficher(tb[i]);
      Ecran.sautDeLigne(); }
```

```
  }
}
```

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.afficher(tb[i]);
      Ecran.sautDeLigne(); }
  }
}
```

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:
  - Pions noirs représentés par le caractère 'N'
  - Pions blancs représentés par le caractère 'B'
  - Pases 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 i,j;
     for ( i = 0 ; i < 10 ; i = i+1 ) {
       for ( j = 0 ; j < 10 ; 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 < 10 ; i = i+1 ) {
       for ( j = 0 ; j < 10 ; j = j+1 ) {
        Ecran.afficher(damier[i][j]); }
      Ecran.sautDeLigne(); }
  }
}
```

TableauDamier.java - Exemple d'exécution

Tableaux en paramètre et en résultat d'action

- Tableaux utilisables en paramètre d'action
- Tableaux utilisables en retour d'action (de fonction)

- En langage algorithmique, passages de paramètre:
  - En donnée
  - En résultat
  - En donnée/résultat

- En langage Java, un seul type de passage de paramètre:
  - En donnée/résultat

```
constante entier N <- ???

{ Passage d'un tableau en parametre           }
{ Modification du tableau a l'interieur       }
{ de la methode                               }
{ -> Modification du tableau sur lequel       }
{    la methode a ete appelee                 }

action testPassage(tab)
   Données / Résultats
```

```
    tab : Tableau [N] de entier
   Locales
    i : entier
   pour i de 0 à N-1 faire
    Ecran.afficher(tab[i]," ")
   fait
  Ecran.sautDeLigne()
   pour i de 0 à N-1 faire
    tab[i] <- 9
   fait
   pour i de 0 à N-1 faire
    Ecran.afficher(tab[i]," ")
   fait
  Ecran.sautDeLigne()
fin action

{ Retour d'un tableau par une action          }
{ apres declaration et initialisation         }

Tableau [N] de entier fonction testRetour()
   Locales
    tab : Tableau [N] de entier
  tab[0] <- 0
  tab[1] <- 0
  tab[2] <- 0
  tab[3] <- 0
  tab[4] <- 1
  tab[5] <- 1
  tab[6] <- 1
  tab[7] <- 1
   retourner tab
fin fonction
```

```
/* Test de l'utilisation des tableaux          */
/* en parametre de methode                     */

public class PassageTableauEnParametre {

/* Passage d'un tableau en parametre           */
/* Modification du tableau a l'interieur       */
/* de la methode                               */
/* -> Modification du tableau sur lequel       */
/*    la methode a ete appelee                 */

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

```
       tab[i] = 9; }
      for ( i = 0 ; i < tab.length ; i = i+1 ) {
       Ecran.afficher(tab[i]," "); }
     Ecran.sautDeLigne();
   }

/* Retour d'un tableau par une action          */
/* apres declaration et initialisation         */

   static int [] testRetour() {
     int i;
     int [] tab = { 7,6,5,4,3,2,1,0 };
     for ( i = 0 ; i < tab.length ; i = i+1 ) {
      Ecran.afficher(tab[i]," "); }
    Ecran.sautDeLigne();
     return tab;
   }

/* Programme principal                         */

   public static void main(String [] args) {
     int i;
/* Test de la methode testPassage              */
     int [] t1 = { 0,1,2,3,4,5,6,7 };
    testPassage(t1);
     for ( i = 0 ; i < t1.length ; i = i+1 ) {
      Ecran.afficher(t1[i]," "); }
    Ecran.sautDeLigne();
    Ecran.sautDeLigne();
/* Test de la methode testRetour               */
     int [] t2;
    t2 = testRetour();
     for ( i = 0 ; i < t2.length ; i = i+1 ) {
      Ecran.afficher(t2[i]," "); }
    Ecran.sautDeLigne();
   }
}
```

PassageTableauEnParametre.java - Exemple d'exécution

Tableau 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

```
nomTableau : Tableau[N1][N2]...[ND] de nomType
```

N1, N2, ..., ND: Constantes entières littérales ou non définissant les tailles (selon chacune des D dimensions)

- nomType: Nom du type agrégé nomType: Nom du type agrégé

Si initialisation à la déclaration pour le type agrégé **nomTypes**
-> Initialisation implicite de chaque composante du tableau
Dans le cas contraire
-> Initialisation impérative par programme dans l'algorithme
-> En langage algorithmique, 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:

```
nomType [] nomTableau = new nomType[n];
for ( int i = 0 ; i < n ; i = i+1 )
  nomTableau[i] = new nomType();
```

- n: Constante entière, variable entière ou expression numérique à résultat entier définissant la taille selon l'unique dimension

En dimension D (D indices):

```
nomType [][]...[] nomTableau = new nomType[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 ) {
      nomTableau[i1][i2]...[iD] = new nomType(); } } }
```

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

Si initialisation à la déclaration pour le type agrégé **nomType**
-> Initialisation implicite de chaque composante du tableau
Dans le cas contrairele cas contraire
-> Initialisation impérative par programme dans l'algorithme
-> En langage Java, 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:
  nomVariableTableau[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
  x : reel <- 0.0
  y : reel <- 0.0
fin structure

{ Programme principal                                  }

constante entier TAILLE <- 10

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

```
/* Declaration de tableaux de variables                */
/* de type agrege position en deux dimensions          */

public class TableauPosition2D {

/* 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 = 10;
   /* Declaration du tableau                            */
     Position2D [] tpos = new Position2D[TAILLE];
   /* Initialisation des composantes par programme      */
```

```
      for ( i = 0 ; i < TAILLE ; i++ )
       tpos[i] = new Position2D();
    /* ************************************  */
      for ( i = 0 ; i < tpos.length ; i++ )
       Ecran.afficherln(tpos[i].x," ",tpos[i].y);
  }
}
```

TableauPosition2D.lda - TableauPosition2D.java - Exemple d'exécution

Version alternative avec utilisation d'une action pour l'affichage de chaque Position2D

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

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

{ Action d'affichage d'une position 2D              }

action afficher(p)
   donnees
    p : position2D
  Ecran.afficherln("[",p.x,",",p.y,"]")
fin action

{ Programme principal                               }

constante entier TAILLE <- 10

action principale()
   locales
    i : entier
    tpos : Tableau[TAILLE] de position2D
   pour i de 0 à TAILLE-1 faire
    afficher(tpos[i])
   fait
  Ecran.afficherln()
fin action
```

```
/* Declaration de tableaux de variables             */
/* de type agrege position en deux dimensions       */

public class TableauPosition2Dv2 {

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

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

/* Affichage d'une position 2D                          */

   static void afficher(Position2D p) {
    Ecran.afficherln("[",p.x,",",p.y,"]");
  }

/* Programme principal                                  */

   public static void main(String [] args) {
     int i;
     final int TAILLE = 10;
   /* 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                      */
    Position2D [] tpos2 = new Position2D[TAILLE];
   /* Initialisation des composantes par programme      */
     for ( i = 0 ; i < TAILLE ; i++ )
      tpos2[i] = new Position2D();
   /* ******************************************        */
     for ( i = 0 ; i < tpos1.length ; i++ )
      afficher(tpos1[i]);
    Ecran.sautDeLigne();
     for ( i = 0 ; i < tpos2.length ; i++ )
      afficher(tpos2[i]);
    Ecran.afficherln();
  }
}
```

TableauPosition2Dv2.lda - TableauPosition2Dv2.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 nomType
  nomTableau : Tableau[N1][N2]...[ND] de type
  ...
fin structure
```

- N1, N2, ..., ND: Constantes entières littérales ou non définissant les tailles selon chacune des dimensions
- nomType: Le nom du type agrégé

- En langage Java, syntaxe en dimension D pour un tableau de variables de type prédéfini:

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

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

- 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:

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   }
{ parmis ses champs: Un ensemble de reels        }

constante entier TAILLE <- 15

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

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

{ Action d'initialisation à zéro du tableau      }
{ d'un ensemble de reels                         }

action initialisation(e)
   Résultats
    e : ensembleDeReel
```

```
    locales
     i : entier
    pour i de 0 à TAILLE-1 faire
     e.t[i] <- 0.0
    fait
fin action

{ Programme principal                                    }

action principale()
    Locales
     i : entier
     e : ensembleDeReel
  initialisation(e)
   pour i de 0 à 4 faire
    e.t[i] <- 1.0
  e.n <- 5
   pour i de 0 à e.n faire
    Ecran.afficher(e.t[i])
   fait
fin action
```

```
/* Declaration d'un type agrege avec un tableau     */
/* parmis ses sous-variables: Un ensemble de double */

public class TypeAgregeAvecTableau {

   final static int TAILLE = 15;

/* Type agrege de stockage d'un ensemble de double  */

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

/* Programme principal                              */

   public static void main(String [] args) {
     int i;
     EnsembleDeDouble e = new EnsembleDeDouble();
     for ( i = 0 ; i < 5 ; i = i+1 ) {
      e.t[i] = 1.0; }
    e.n = 5;
     for ( i = 0 ; i < e.n ; i = i+1 )
      Ecran.afficherln(e.t[i]);
  }
}
```

TypeAgregeAvecTableau.lda - TypeAgregeAvecTableau.java - Exemple d'exécution

Affectation et test d'égalité entre tableaux

Affectation

- En langage algorithmique, affectation directe considérée comme possible entre deux variables de types tableau identiques (dimension, taille(s) et type élémentaire identiques)
  Utilisation du signe <-
- En langage Java, même comportement que pour les types agrégés si affectation entre deux variables de type tableau (signe =):
  Confusion totale irréversible entre ces deux variables
  -> A tout jamais, même variable avec deux noms
  Solution: Développer le code d'affectation place pour place des composantes des deux tableaux
  Recommandation: Développer une action d'affectation

```
/* Tests d'affectation entre tableaux            */

public class AffectationEntreTableaux {

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

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

/* Action d'affectation des composantes         */
/* d'un tableau d'entiers cible                 */
/* avec les composantes                         */
/* d'un tableau d'entier source                 */

   static void affectation(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) {
     int [] t1 = { 0,1,2,3 };
     int [] t2 = { 0,0,0,0 };
     int [] t3 = { 3,2,1,0 };
```

```
    affichage(t1);
    affichage(t2);
    affectation(t1,t2);
    affichage(t1);
    affichage(t2);
    t1[0] = 0;
    t1[1] = 0;
    t1[2] = 0;
    t1[3] = 0;
    affichage(t1);
    affichage(t2);
    Ecran.sautDeLigne();
    affichage(t1);
    affichage(t3);
    t1 = t3;
    affichage(t1);
    affichage(t3);
    t1[0] = 0;
    t1[1] = 0;
    t1[2] = 0;
    t1[3] = 0;
    affichage(t1);
    affichage(t3);
  }
}
```

AffectationEntreTableaux.java - Exemple d'exécution

Test d'égalité

- En langage algorithmique, test d'égalité considéré comme possible entre deux variables de types tableau identiques (dimension, taille(s) et type élémentaire identiques)
  Utilisation du signe =
- En langage Java, test d'égalité impossible de manière simple entre deux tableaux par utilisation du signe ==
  Solution: Développer le code de test place pour place des composantes des deux tableaux
  Recommandation: Développer une fonction de test

```
/* Tests d'egalite entre 2 tableaux                */

public class TestEgaliteTableaux {

/* Fonction de test de l'egalite de 2 tableaux  */
/* d'entiers                                       */
```

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

/* Programme principal                                  */

   public static void main(String [] args) {
     int [] t1 = { 0,1,2,3 };
     int [] t2 = { 0,0,0,0 };
     int [] t3 = { 0,1,2,3 };
     boolean egal;
    egal = (t1 == t2);
    Ecran.afficherln(egal);
    egal = (t1 == t3);
    Ecran.afficherln(egal);
    egal = testEgalite(t1,t2);
    Ecran.afficherln(egal);
    egal = testEgalite(t1,t3);
    Ecran.afficherln(egal);
  }
}
```

TestEgaliteTableaux.java - Exemple d'exécution

Exemples d'utilisation des tableaux

- Actions de déclaration et 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

```
constante entier N <- ...

{ Initialisation d'un tableau                        }
{ de reel avec des 0.0                               }

action initialisation(tab)
```

```
   Résultats
    tab : Tableau[N] de reel
   Locales
    i : entier
   pour i de 0 à N-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   }

action initRand(tab)
   Résultats
    tab : Tableau[N] de reel
   Locales
    i : entier
   pour i de 0 à N-1 faire
    tab[i] <- random()
   fait
fin action

{ Creation d'un tableau de reel initialise avec }
{ une serie croissante de nombres aleatoires    }

Tableau [N] de reel fonction initRandCroissant()
   Locales
    i : entier
    tab : Tableau[N] de reel
  tab[0] <- random()
   pour i de 1 à N-1 faire
    tab[i] <- tab[i-1]+random()
   fait
   retourner tab
fin fonction
```

```
/* Methode d'initialisation d'un tableau       */
/* de double avec des 0.0                      */

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

/* Methode d'initialisation d'un tableau       */
/* de double avec des reels tires au sort      */
/* entre 0.0 et 1.0                            */

static void initRand(double [] tab) {
```

```
    int i;
    for ( i = 0 ; i < tab.length ; i = i+1 ) {
     tab[i] = Math.random(); }
}

/* Methode de creation d'un tableau de double  */
/* initialise avec une serie croissante        */
/* de nombres aleatoires                       */

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

InitialisationTableauReel.lda - InitialisationTableauReel.java - Exemple d'exécution

- Action d'affichage d'un tableau de réels

```
{ Affichage d'un tableau de reels              }

constante entier N <- ...

action affichageTableau(t)
    Données
     t : Tableau [N] de reel
    Locales
     i : entier
    pour i de 0 à N-1 faire
     Ecran.afficherln(t[i]," ")
    fait
fin action
```

```
/* Methode d'affichage des valeurs contenues   */
/* dans un tableau de double                   */

static void affichageTableau(double [] t) {
    int i;
    for ( i = 0 ; i < t.length ; i = i+1 ) {
     Ecran.afficherln(t[i]); }
}
```

AffichageTableauReel.lda - AffichageTableauReel.java - Exemple d'exécution

- Action de test de l'égalité de deux tableaux de réels

```
constante entier N <- ...
constante entier M <- ...

{ Test de l'egalite du contenu                  }
{ de deux tableaux de reel                      }

booleen fonction testEgalite(t1,t2)
   Données
    t1 : Tableau [N] de reel
    t2 : Tableau [M] de reel
   Locales
    egalite : booleen
    i : entier
  egalite <- vrai
   si N <> M alors
    egalite <- faux
     sinon
    i <- 0
     tant que ( egalite et ( i < N ) ) faire
      egalite <- (t1[i] = t2[i])
      i <- i+1
     fait
   fsi
   retourner egalite
fin fonction
```

```
/* Methode de test de l'egalite du contenu     */
/* de deux tableaux de double                  */

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

TestEgaliteTableauxReel.lda - TestEgaliteTableauxReel.java - Exemple d'exécution

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

```
constante entier N <- ...

{ Calcul de la valeur minimale                    }
{ d'un tableau de reel                            }

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

```
/* Methode de calcul de la valeur minimale      */
/* d'un tableau de double                       */

static double minimum(double [] t) {
   double min;
   int i;
   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.lda - ValeurMinimaleTableauReel.java - Exemple d'exécution

- Manipulations de type "Chaîne de caractères <-> Tableau de caractères"

```
{ Creation d'une chaine de caracteres          }
{ a partir d'un tableau de caracteres          }

constante entier N <- 5

action principale()
   Locales
    s : chaine
    t : Tableau [N] de caractere
    i : entier
  s <- ""
   t[0] <- 'A'
  t[1] <- 'a'
  t[2] <- '.'
   t[3] <- 'z'
  t[4] <- 'Z'
   pour i de 0 à N-1 faire
    s <- s+t[i]
   fait
  Ecran.afficherln(N," : ",s)
fin action
```

```
{ Extraction sous forme de tableaux            }
{ de caracteres successifs des caracteres      }
{ d'un tableau de chaines de caracteres        }
{ Affichage de ces tableaux de caracteres      }

constante entier N <- 4
constante entier M <- ...

action principale()
   Locales
    ts : Tableau [N] de chaine
    i,j : entier
    tcl : entier
    tc : Tableau [M] de caractere
  ts[0] <- "ABC"
  ts[1] <- "!!&&%%"
   ts[2] <- ""
   ts[3] <- "âçéè"
   pour i de 0 à N-1 faire
    tc <- ts[i].toCharArray()          { ??? }
    tcl <- ts[i].length()              { ??? }
    Ecran.afficher(tcl," : ")
```

```
      pour j de 0 à tcl-1 faire
       Ecran.afficher(tc[j])
      fait
     Ecran.sautDeLigne()
    fait
fin action
```

```
/* Creation d'une chaine de caracteres          */
/* a partir d'un tableau de caracteres          */
/*                                              */
/* Extraction du contenu d'une chaine           */
/* de caracteres dans un tableau de caracteres */

public class UtilisationChaineEtTableauCaracteres {

/* Programme principal                          */

   public static void main(String [] args) {
/* Creation d'une chaine de caracteres          */
/* a partir d'un tableau de caracteres          */
     char [] t = { 'A','a','.','z','Z' };
     String s = String.valueOf(t);
    Ecran.afficherln(s.length()," caracteres : ",s);
/* Extraction sous forme de tableaux            */
/* de caracteres successifs des caracteres      */
/* d'un tableau de chaines de caracteres        */
/* Affichage de ces tableaux de caracteres      */
     String [] ts = { "ABC","!!&&%%","","àçéè" };
     char [] tc;
     int i,j;
     for ( i = 0 ; i < ts.length ; i++ ) {
      tc = ts[i].toCharArray();
      Ecran.afficher(tc.length," caracteres : ");
       for ( j = 0 ; j < tc.length ; j++ ) {
        Ecran.afficher(tc[j]); }
      Ecran.sautDeLigne(); }
    Ecran.sautDeLigne();
  }
}
```

UtilisationChaineEtTableauCaracteres1.lda - UtilisationChaineEtTableauCaracteres2.lda
UtilisationChaineEtTableauCaracteres.java - Exemple d'exécution

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