

UEF 1 : Informatique & Programmation

Faculté des Sciences de Nice

DEUG 2001-2002

Jérôme DURAND-LOSE

Jean-Paul ROY

COURS 9

T A B L E A U X

Définition, utilisation, parcours et recherches

9-2

Données de même type, même utilisation

Exemples : échantillonnage, relevés, coefficients,..
liste de prix, noms, machines, points, pays...

- 1 valeur => 1 variable
- 2, 3 ou 4 valeurs => possible avec des variables
- 10 valeurs => très lourd et très risqué
- 20 valeurs => irréaliste
- Nombre inconnu => impossible

Idéal : une seule variable, une seule déclaration et accès similaires pour chaque valeur

==> TABLEAU

Exemple :

0	1	2	3	4	5	6
1	6	15	20	15	6	1

9-3

Déclaration et création

/ déclaration */* <type> [] <identifiant>;

/ création */* <identifiant> = new <type> [<taille>];

taille est le nombre de *cases* désirées

ex. : **int**[] temperatures;
temperatures = new **int**[365];
double[] coefficients = new **double**[30];
int nombre = 3000;
String[] mots = new String[nombre];

valeurs
restant à
fournir

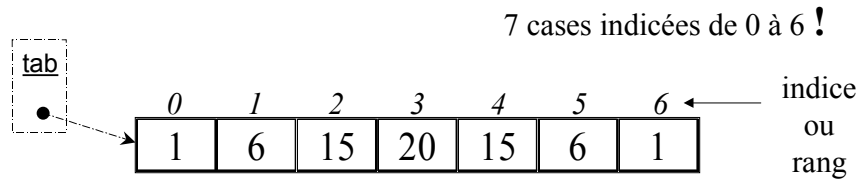
Avec les valeurs initiales :

int[] tab = new **int**[] { 1, 6, 15, 20, 15, 6, 1 };

taille
implicite
(7)

9-4

Accès à une case



Chaque case contient une valeur, on peut y accéder directement

/ accès à une case */* *<identifiant> [<indice>]*

```
int[] tab = new int[] { 1, 6, 15, 20, 15, 6, 1 };
```

```
....  
double s = 1.0;
```

```
int indice = 3;
```

```
....  
s = s + tab [ 0 ];
```

```
tab[ 4 ] = 5;
```

```
tab[indice] = (int) s;
```

Attention aux erreurs d'indices ...

```
tab[ -1 ]      tab[ 1.5 ]
```

```
tab[ 100 ]     tab[ "1" ]
```

```
tab[ 7 ]
```

... n'existent pas !

9-5

Taille et initialisation des données

On utilise l'attribut public `length` pour connaître la taille d'un tableau

```
int[] tabInt = new int[ 60 ];
```

```
System.out.println( "Le tableau a " + tabInt.length + " éléments" );
```

Le tableau a 60 éléments

Il faut donner une valeur à chacune des cases, *individuellement* :

```
double[] aleaTab = new double[ Numerik.randomInt ( 20, 25 ) ];
```

```
for ( int i = 0 ; i < aleaTab.length ; i ++ )  
    aleaTab[ i ] = 100 * Math.random ();
```

Dès que l'on manipule un tableau, on se retrouve avec des itérations !

9-6

Mouvements pour la tortue

On veut faire exécuter une série de mouvements à la tortue

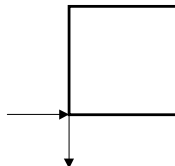
Ces mouvements sont dans un tableau et vont par deux :

un angle (rotation à gauche)

une distance (avancer la tortue)

Par exemple, pour faire un carré de côté 100

0, 100, 90, 100, 90, 100, 90, 100



La tortue se retrouve au même point, mais pas dans la même direction !

9-7

Mouvements pour une tortue

Pour cela on va faire une procédure (de classe) ayant comme arguments :

- une tortue
- un tableau contenant les mouvements

```
static void deplaceTortue ( Turtle t, int[] tm ) {  
    for ( int i = 0; i < tm.length; i += 2 ) {  
        t.left( tm[ i ] );  
        t.forward( tm[ i + 1 ] );  
    }  
}
```

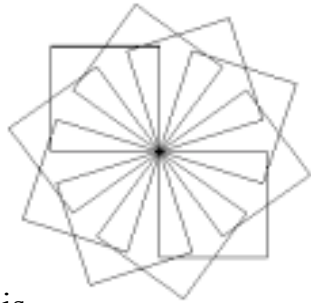
On avance par pas de 2,

1 pour la rotation et
1 pour le déplacement

9-8

Dessins possibles

En rappelant plusieurs fois de suite :



10 fois

{ 0, 100, 90, 100, 90, 100, 90, 100, 90, -18, 0 }

(un carré + rotation)



12 fois

{ 30, 40, 150, 40, -150, 20 }

(un « cran »)

9-9

Parcours séquentiel exhaustif

Il faut faire une boucle pour atteindre successivement tous les éléments !

Variable

table : tableau

Algorithme pour affichage

parcourir tableau

afficher la case courante

Valeur renvoyée

somme : nombre approché

Algorithme pour moyenne

somme ← 0

parcourir tout le tableau

ajouter la valeur de la case courante à somme

moyenne ← somme / nombre de cases

```
double[] table;
```

```
....
```

```
/* affiche tous les éléments */
```

```
for ( int i = 0 ; i < table.length ; i ++ )
```

```
    System.out.println ( table[ i ] );
```

```
...
```

```
/* calcul de la somme */
```

```
double somme = 0;
```

```
for ( int i = 0 ; i < table.length ; i ++ )
```

```
    somme += table[ i ];
```

```
/* affichage de la moyenne */
```

```
Console.println( somme / table.length );
```

9-10

Recherche de l'indice du minimum

```
/* cherche l'indice du minimum */
```

```
int iMin = 0;
```

```
/* invariant : iMin est l'indice de la plus petite  
valeur des cases d'indices de 0 à i-1 */
```

```
for ( int i = 0 ; i < table.length ; i ++ )
```

```
    if ( table[ i ] < table[ iMin ] )
```

```
        iMin = i;
```

```
/* iMin est l'indice de la plus petite valeur de table */
```

Variable

table : tableau

iMin : entier = premier indice

Valeur renvoyée

résultat : indice du minimum

Algorithme de recherche de l'indice du minimum

parcourir tableau

si la valeur de la case courante < celle de la case iMin alors

iMin ← indice courant

résultat ← iMin

9-11

Recherche d'un élément

On évite de
parcourir
inutilement le
tableau !

```
int i = 0;
```

```
int iTrouve;
```

```
/* cherche premier indice pour lequel la valeur est négative */
```

```
while ( ( i < table.length ) && ! ( table[ i ] < 0 ) )
```

```
    i ++;
```

```
if ( i < table.length )
```

```
    iTrouve = i;
```

```
else
```

```
    iTrouve = NON_TROUVE;
```

```
/* iTrouve vaut
```

```
- soit le plus petit indice i pour lequel table[i] est négative,
```

```
- soit NON_TROUVE et il n'y a pas de valeur négative dans le tableau */
```

Coupe-circuit

Variables

table : tableau

i : entier

Valeur renvoyée

iTrouvé : indice de la première valeur négative, Non trouvé s'il n'y en a pas

Algorithme de recherche du plus petit indice pour une valeur négative

parcourir tableau tant que l'on ne l'a pas trouvé

avec i comme indice

si i est un indice valide alors

iTrouvé ← i

sinon

iTrouvé ← Non Trouvé

9-12

Recherche d'un élément

On peut aussi
arrêter
le parcours !

Variables

table : tableau
i : entier

Valeur renvoyée

iTrouvé : indice de la première valeur négative, Non trouvé s'il n'y en a pas

Algorithme de recherche du plus petit indice pour une valeur négative

iTrouvé ← Non Trouvé
parcourir tableau
si la case courante est celle que l'on cherche alors
iTrouvé ← indice de la case courante
mettre fin au parcours

```
int iTrouvé = NON_TROUVE;
for ( int i = 0; ( i < table.length ); i++ ) {
    if ( table[ i ] < 0 ) {
        iTrouvé = i;
        break;
    }
}
```

Fin prématurée
de la boucle

9-13

Vecteur sous forme de tableau

Vecteur en dimension d ➤ d coordonnées ➤ tableau de longueur d
On peut écrire une classe pour les manipuler

Données

u, v : vecteurs de même dimension

Variables

prodS : nombre approché = 0

Valeur renvoyée

résultat : nombre approché, valeur du produit scalaire

Algorithme de calcul du produit scalaire

parcourir u et v en parallèle
ajouter à prodS le produit des cases courantes de u et v
résultat ← prodS

```
static double produitScalaire( double[] u, double[] v ){
    double prodS = 0;
    for ( int i = 0; i < u.length; i++ )
        prodS += u[ i ] * v[ i ];
    return prodS;
}
```

on peut mettre
des tableaux
en paramètre !

9-14

Renvoyer un tableau

```
double [] v1 = { 1, 0, 0.5, 0 };
double [] v2 = { 0, 2, 0.5, 0.7 };
double [] v = ManipulationVecteur.somme ( v1, v2 );
ManipulationVecteur.afficheVecteur ( v );
```

(1.0, 2.0, 1.0, 0.7)

il est récupéré là

1.0 0.0 0.5 0.0

0.0 2.0 0.5 0.7

1.0 2.0 1.0 0.7

On peut renvoyer
un tableau !

il est créé ici

```
static double [] somme ( double [] u, double [] v ){
    double [] w = new double [ u.length ];
    for ( int i = 0; i < u.length; i++ )
        w[ i ] = u[ i ] + v[ i ];
    return w;
}
```

il est retourné ici

9-15

Un tableau est un objet !

Dans les appels et les renvois de méthodes,
c'est sa référence que l'on manipule,
non une copie de sa valeur !

En particulier, ce n'est pas un type primitif

Si une méthode le modifie, il reste modifié

9-16

Modifier un tableau

```
double [] v = { 1, 1, 1, 1 };  
ManipulationVecteur.normalise ( v );  
ManipulationVecteur.afficheVecteur ( v );
```

(0.25, 0.25, 0.25, 0.25)

1.0	1.0	1.0	1.0
0.25	0.25	0.25	0.25

deux références
mais
un seul tableau !

```
static void normalise ( double [] u ) {  
    double nrm = Math.sqrt ( produitScalaire ( u, u ) );  
    if ( nrm != 0 )  
        for ( int i = 0 ; i < u.length ; i ++ )  
            u [ i ] = u [ i ] / nrm;  
}
```

calcul de
la norme