M2 CCI – EP1 Algorithmique

TP3 (2h) – 06 oct. 2017

But de la séance : manipulation de tableaux 2D

Un tableau 2D n'est en rien différent d'un tableau 1D puis qu'il s'agit d'un tableau de tableaux. Il n'en reste pas moins au début une difficulté à se représenter l'organisation des données. En France on dit qu' »un petit dessin vaut mieux qu'on long discours »...

Les exercices suivants sont similaires à ceux du TD3. Ils apportent en plus un questionnement sur l'organisation mémoire des données. La partie algorithmique est supposée, c'est à dire qu'aucune question ne porte directement dessus. Toutefois, il est évident qu'elle est présente dans chaque question où un programme est attendu. C'est donc à vous de réfléchir à l'algorithme qu'il est nécessaire de concevoir afin de réaliser tel ou tel traitement, puis de transposer cet algorithme en langage Java.

Les notations utilisées sont celles habituelles :

- le langage Java est écrit en `police « courier »`
- le langage UML est écrit en couleur verte

Les exercices seront traités dans l'ordre, l'important n'est pas le nombre réalisé à la fin de la séance, mais votre compréhension et votre capacité à le refaire.

Table des matières

Exercice 1 – Matrice, organisation mémoire et opérations simples

On va réaliser une classe `Matrice` chargée de mémoriser une matrice de nombres réels composée de L lignes et de C colonnes. Cette matrice sera utilisée dans une autre classe, appelée `essaiMatrice`, qui contiendra le programme principal.

L'approche vue en cours consiste à toujours se placer du point de vue de l'utilisateur. On va donc essayer d'exprimer ce besoin afin de déterminer ensuite les méthodes de la classe `Matrice`.

Question 1

L'utilisateur doit pouvoir créer une matrice en précisant le nombre de lignes et le nombre de colonnes. La matrice créée sera nulle (tous ses coefficients vaudront 0).

ex. `Matrice m1 = new Matrice (5,3) ;`

pour créer une matrice de 5 lignes et 3 colonnes, soient 15 coefficients stockés dans l'objet m1. Programmer le constructeur Matrice(int,int)

La syntaxe ci-dessus correspond à celle d'un constructeur. Bien identifier le rôle de celui-ci, la signification des paramètres, s'il a besoin de stocker des informations, etc. En fonction de cela on pourra définir précisément ce que fait le constructeur, avec quoi et dans quel ordre. Il ne restera plus qu'à le coder. Rappel : les commentaires ne sont pas utiles qu'après avoir écrit le code. Il le sont surtout avant, car ils permettent de réfléchir et d'exprimer ce qu'on veut faire. Ils ne sont donc pas une charge, mais une aide !

Question 2

L'utilisateur doit pouvoir afficher une matrice dans la console. Elle apparaîtra ligne par ligne, les éléments alignés en colonnes, arrondis à 4 chiffres après la virgule. On supposera que tous les coefficients seront inférieurs à 1000[1].

ex. `m1.display() ;` pour afficher la matrice m1 précédemment créée.

Programmer la méthode display():void

Afin de faciliter la saisie d'une matrice, on propose de créer un nouveau constructeur, qui prendra un tableau crée au moyen d'un initialiseur. Ainsi il sera possible d'écrire :

```
double[][] tmp = { {1.5,2.2,3.0}, {4.6,5.2,6.9}, {7.1,8.3,9.7} };
Matrice m2 = new Matrice(tmp);
```

Rem. On pourrait aussi imaginer un constructeur qui recevrait un objet Matrice et qui en ferait un clone…

1 - Il y a différentes manières d'imposer un format d'affichage. Ici on peut utiliser la même syntaxe qu'en C par exemple. L'écriture de `String.format("%9.4f", Math.PI())` donne 3,1416. Le paramètre `"%9.4f"` signifie « sur 9 caractères de large dont 4 seront situés après la virgule pour des réels (aussi appelés flottants) ».

Question 3

Ecrire une première version du constructeur Matrice(double[][]) qui fait une copie des éléments passés en paramètres et tester en affichant la matrice.

Question 4

Vérifier au debugger ou en faisant afficher les adresses mémoire des tableaux que la variable passée en paramètre (notée `tmp` dans le programme principal) et le tableau (dans la méthode display()) sont bien physiquement séparées les unes des autres.

Question 5

Mettre en remarque le constructeur de la question 6 et écrire une nouvelle version du constructeur Matrice(double[][]) qui fait affecte juste la matrice interne avec celle reçue en paramètres (on ne duplique pas en mémoire les éléments reçus en paramètre). Contrôler en demandant à la matrice de s'afficher.

Question 6

Vérifier au debugger ou en faisant afficher les adresses mémoire des tableaux que la variable `tmp` (du programme principal) et le tableau (dans la méthode display()) désignent bien la même zone mémoire.

Question 7

Demander la suppression de la variable `tmp` dans le programme principal. Que va-t-il se passer si on demande l'affichage de la matrice `m2` ? Vérifier.

Question 8

On exécute le code source suivant dans le programme principal :

```
double[][] tmp = { {1.5,2.2,3.0},{4.6,5.2,6.9},{7.1,8.3,9.7} };
Matrice m2 = new Matrice(tmp);
tmp[1][1] = -1;
m2.display();
```

Quel est le résultat de l'affichage ? Est-ce prévisible ? Concluez en faisant un schéma de l'organisation en mémoire des données et vérifiez qu'il est cohérent avec ce que vous avez observé aux questions 4, 6 et 8.

On continue de s'intéresser à l'allocation mémoire. On affiche les adresses des blocs mémoire liés au tableau tmp dans le programme principal :

```
double[][] tmp = { {1.5,2.2,3.0},{4.6,5.2,6.9},{7.1,8.3,9.7} };
System.out.println("@tmp = " + tmp);
System.out.println("@[0] = " + tmp[0]);
System.out.println("@[1] = " + tmp[1]);
System.out.println("@[2] = " + tmp[2]);
```

Un résultat pour une adresse est par exemple : `@tmp = [[D@4bd66d2f`

Structure : [[= tableau 2D — Type : Double — Adresse mémoire en base 16

Un *double* est codé sur 8 octets en Java. Le but est de savoir si l'allocation faite pour tmp est contiguë en mémoire ou non, c'est à dire si l'ensemble des lignes forme un bloc compact en mémoire. Pour cela on va chercher à connaître les adresses en mémoire des différents tableaux produits.

Question 9

Calculer l'écart qui existe entre `tmp[0]`, `tmp[1]` et `tmp[2]`. Qu'en concluez-vous quand à la contiguïté des lignes du tableau `tmp` ?

Question 10

On recommence le même calcul, sur le tableau situé dans la version de la matrice obtenue à la question 3 (on alloue et on recopie les éléments). Qu'en concluez-vous quand à la contiguïté des lignes de ce tableau alloué par new ?

Additionner deux matrices peut se faire lorsque celles-ci ont les mêmes dimension. Dans ce cas, on additionne terme à terme, c'est à dire qu'on somme les termes situés aux mêmes coordonnées. Si on note A et B les matrices à additionner, et C le résultat, on pourra écrire pour tout couple {x,y} de coordonnées valide la relation :

$$C[x][y] = A[x][y] + B[x][y]$$

Question 11

Ecrire la méthode add(Matrice):Matrice dans la classe Matrice qui réalise l'addition de telle façon que l'écriture suivante soit correcte :

```
double[][] tmp1 = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
Matrice m4 = new Matrice(tmp1);
Matrice m5 = new Matrice(tmp1);
Matrice m6 = m4.add(m5) ;
```

La transposée d'une matrice A est une matrice B où les lignes de A sont devenues les colonnes de B (et donc les colonnes de A sont devenues les lignes de B). Notée A', on a par exemple :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}' = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Question 12

Ecrire la méthode trans():Matrice dans la classe Matrice qui calcule la transposée de telle façon que l'écriture suivante soit correcte :

ex. `Matrice m7 = m4.trans() ;`

Question 13

Que signifie alors l'expression `m4.add(m4.trans()).display();` ? Tester pour vérifier qu'elle est valide et que le résultat est correct.

Exemple d'affichage en console du programme complet

```
         0,0000      0,0000      0,0000
         0,0000      0,0000      0,0000
         0,0000      0,0000      0,0000
         0,0000      0,0000      0,0000
         0,0000      0,0000      0,0000

Q3 : @ tableau dans objet = [[D@10875750
         1,5000      2,2000      3,0000
         4,6000      5,2000      6,9000
         7,1000      8,3000      9,7000
Q3 : @tmp dans main() = [[D@580283d3

Q5 : @ tableau dans objet = [[D@2ac510e3
         1,5000      2,2000      3,0000
         4,6000      5,2000      6,9000
         7,1000      8,3000      9,7000
Q7 :
         1,5000      2,2000      3,0000
         4,6000      5,2000      6,9000
         7,1000      8,3000      9,7000
Q8 :
         1,5000      2,2000      3,0000
         4,6000     -1,0000      6,9000
         7,1000      8,3000      9,7000
Q9 :
      @tmp = [[D@333c339f
      @[0] = [D@d1cbec9
      @[1] = [D@5a3184d8
      @[2] = [D@6908af2a
Q10 :
      @data = [[D@4633c1aa
      @[0]  = [D@6fefa3e7
      @[1]  = [D@5df1cc1a
      @[2]  = [D@2d8eef25
Q11 :
      //    double[][] tmp1 = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
      //    Matrice m4 = new Matrice(tmp1);
      //    Matrice m5 = new Matrice(tmp1);
      //    Matrice m6 = m4.add(m5);

         2,0000      4,0000      6,0000
         8,0000     10,0000     12,0000
        14,0000     16,0000     18,0000
Q12 :
      //    Matrice m7 = m4.trans();

         1,0000      4,0000      7,0000
         2,0000      5,0000      8,0000
         3,0000      6,0000      9,0000
Q13 :
      //    m4.add(m4.trans()).display();

         2,0000      6,0000     10,0000
         6,0000     10,0000     14,0000
        10,0000     14,0000     18,0000
```