MOOC Init Prog Java
Corriges semaine 4

Les corrigés proposés correspondent à l'ordre des apprentissages : chaque corrigé correspond à la solution à laquelle vous pourriez aboutir au moyen des connaissances acquises jusqu'à la semaine correspondante.

Exercice 12: Produit scalaire (Tableaux)

Les tableaux, ici à une dimension, sont typiquement parcourus au moyen de boucles `for`. Ces boucles sont utilisées pour remplir les vecteurs (par lecture de `doubles`) et pour effectuer le calcul du produit scalaire. La principale chose à remarquer ici est qu'avec les moyens à disposition au moment de la série, on ne sait pas réutiliser des portions de code lorsqu'elles sont requises à plusieurs endroits du programme (exemple: la lecture des vecteurs). Ceci implique que le code concerné est tout simplement dupliqué. Nous verrons à partir du cours prochain comment améliorer cet état des choses.

```
import java.util.Scanner;

class Scalaire {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int nMax = 10;
        int n = 0;

        // On exige l'introduction d'un entier entre 0 et 10
        while (n < 1 || n > nMax) {
            System.out.print("Quelle taille pour vos vecteurs [entre 1 et "
                        + nMax + "] ? ");
            n = scanner.nextInt();
        }

        // Déclaration-construction des deux vecteurs
        double [] v1 = new double [n];
        double [] v2 = new double [n];

        System.out.println("Saisie du premier vecteur :");
        for (int i = 0; i < n; i++) {
            System.out.print(" v1[" + i + "] = ");
            v1[i] = scanner.nextDouble();
        }

        System.out.println("Saisie du second vecteur :");
        for (int i = 0; i < n; i++) {
            System.out.print(" v2[" + i + "] = ");
            v2[i] = scanner.nextDouble();
        }

        // Calcul du produit scalaire
```

```
        double somme = 0.0;
        for (int i = 0; i < v1.length; i++) {
            somme += v1[i] * v2[i];
        }

        System.out.println("Le produit scalaire de v1 par v2 vaut " + somme);

        scanner.close();
    }
}
```

Exercice 13: Multiplication de matrices (Tableaux)

Encore un tableau à deux dimensions, mais cette fois il y a plusieurs tests d'intégrité à effectuer (taille non-nulle des matrices, matrices de tailles compatibles pour la mutliplication etc...)
La remarque faite lors des deux exercices précédents est valable ici aussi: des parties importantes du codes sont dupliquées. Un exercice ultérieur vous permettra de reprendre ce code en remédiant à cette situation.

```
import java.util.Scanner;

class MulMat {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int lignes = 0;
        int colonnes = 0;

        // Saisie de la 1ère matrice
        System.out.println("Saisie de la 1ere matrice :");

        // On vérifie que le nombre de lignes est plus grand que 0
        while (lignes < 1) {
            System.out.print("Nombre de lignes : ");
            lignes = scanner.nextInt();
        }

        // On vérifie que le nombre de colonnes est plus grand que 0
        while (colonnes < 1) {
            System.out.print("Nombre de colonnes : ");
            colonnes = scanner.nextInt();
        }

        // Déclaration-construction de la 1ère matrice
        double[][] mat1 = new double[lignes][colonnes];
        for (int row = 0; row < lignes; row++) {
            for (int col = 0; col < colonnes; col++) {
                System.out.print("  M[" + (row + 1) + "," + (col + 1) + "]=");
                mat1[row][col] = scanner.nextDouble();
            }
        }


        // ... et on refait la même chose pour la 2ème matrice
        lignes = 0;
        colonnes = 0;

        System.out.println("Saisie de la 2eme matrice :");
        while (lignes < 1) {
            System.out.print("Nombre de lignes : ");
            lignes = scanner.nextInt();
        }

        while (colonnes < 1) {
            System.out.print("Nombre de colonnes : ");
            colonnes = scanner.nextInt();
```

```
        }

        double[][] mat2 = new double[lignes][colonnes];
        for (int row = 0; row < lignes; row++) {
            for (int col = 0; col < colonnes; col++) {
                System.out.print("  M[" + (row + 1) + "," + (col + 1) + "]=");
                mat2[row][col] = scanner.nextDouble();
            }
        }

        // Ici on multiplie les matrices
        if (mat1[0].length != mat2.length) {
            System.out.println("Multiplication de matrices impossible !");
        } else {
            // Déclaration-construction de la matrice résultat
            double[][] prod = new double[mat1.length][mat2[0].length];
            for (int row = 0; row < mat1.length; row++) {
                for (int col = 0; col < mat2[0].length; col++) {
                    prod[row][col] = 0.0;
                    for (int i = 0; i < mat2.length; i++) {
                        prod[row][col] += mat1[row][i] * mat2[i][col];
                    }
                }
            }

            // Affichage du résultat
            System.out.println("Resultat :");
            for (int row = 0; row < prod.length; row++) {
                for (int col = 0; col < prod[row].length; col++) {
                    System.out.print(prod[row][col] + " ");
                }
                System.out.println();
            }
        }

        scanner.close();
    }
}
```

Exercice 14: Triangle de Pascal (Tableaux)

On a encore affaire à un tableau en deux dimensions, mais cette fois avec un nombre variable de colonnes sur chaque ligne. Ceci permet d'optimiser l'espace de stockage requis en tenant compte du fait que, pour un triangle de Pascal, certains éléments ne sont pas définis (exemple l'élément d'indices [0][2] n'existe pas). La matrice calculée a ici une forme triangulaire.

```
import java.util.Scanner;

class Pascal {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int size = 0;

        // On demande la taille
        System.out.println("Taille du triangle de Pascal : ");
        size = scanner.nextInt();

        // Ici on déclare le tableau, mais on ne construit que la première
        // dimension
        int[][] triangle = new int[size][];

        // On construit et initialise la 1ère ligne
        triangle[0] = new int[1];
        triangle[0][0] = 1;

        for (int row = 1; row < size; row++) {

            // Chaque ligne du triangle est un tableau à une dimension
            // dont la taille est celle de la ligne precedente + 1:
            // on construit ces lignes (new) au fur et à mesure que l'on
            // progresse dans le tableau
            triangle[row] = new int [triangle[row - 1].length + 1];

            // Remplissage du tableau:
            // les deux éléments aux deux extrémités des lignes valent 1.
            // Les autres sont liés par la relation:
            // triangle[row][j]= triangle[row-1][j-1] + triangle[row-1][j]

            for (int col = 0; col <= row; col++) {
                if ((col == 0) || (col == row)) {
                    triangle[row][col] = 1;
                } else {
                    triangle[row][col] = triangle[row - 1][col - 1]
                            + triangle[row - 1][col];
                }
            }
        }

        // Affichage du tableau
        for (int row = 0; row < size; row++) {
            for (int col = 0; col <= row; col++) {
                System.out.print(triangle[row][col] + " ");
            }
```

```
            System.out.println();
        }

        scanner.close();
    }
}
```
