MOOC Init Prog Java
Corriges semaine 6

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 18: Multiplication matricielle revisitée (Modularisation)

Pour faire cet exercice, il fallait créer les méthodes permettant de lire une matrice, mutliplier deux matrices et afficher une matrice. Le corps de ces méthodes contient les instructions nécessaires au traitement, copiées depuis la version non modularisée `MulMat.java`.

```
import java.util.Scanner;

class MulMatMod {


    private static Scanner scanner = new Scanner(System.in);

    /**
     * Point d'entrée du programme.
     * Demande à l'utilisateur
     * d'entrer deux matrices, les multiplie, et affiche
     * le résultat.
     */
    public static void main(String[] args) {
        double[][] matrice1 = lireMatrice();
        double[][] matrice2 = lireMatrice();
        if (matrice1[0].length != matrice2.length) {
            System.out.println("Multiplication de matrices impossible !");
        } else {
            double[][] produit = multiplierMatrice(matrice1, matrice2);
            System.out.println("Résultat :");
            afficherMatrice(produit);
        }
    }

    /**
     * Demande à l'utilisateur d'entrer un entier strictement positif.
     * @param message Le message affiché à l'utilisateur.
     * @return Un entier strictement positif.
     */
    static int lireEntier(String message) {
        int n = 0;
        do {
            System.out.print(message);
            n = scanner.nextInt();
        } while (n < 1);
        return n;
    }
```

```java
    /**
     * Demande à l'utilisateur de remplir une matrice.
     * @return Un tableau de double contenant la matrice.
     */
    static double[][] lireMatrice() {
        System.out.println("Saisie d'une matrice :");

        int lignes = lireEntier("  Nombre de lignes : ");
        int colonnes = lireEntier("  Nombre de colonnes : ");

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

    /**
     * Multiplie deux matrices.
     * @param mat1 Le premier opérande
     * @param mat2 Le deuxième opérande
     * @return Un tableau de double contenant le produit
     */
    static double[][] multiplierMatrice(double[][] mat1, double[][] mat2) {
        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];
                }
            }
        }
        return prod;
    }

    /**
     * Affiche une matrice à l'écran
     * @param matrice La matrice à afficher
     */
    static void afficherMatrice(double[][] matrice) {
        for (int row = 0; row < matrice.length; row++) {
            for (int col = 0; col < matrice[0].length; col++) {
                System.out.print(matrice[row][col] + " ");
            }
            // Retour à la ligne
            System.out.println();
        }
    }
}
```

Exercice 19: Passage par valeur (Passage de parametres)

Les explications et la correction sont données dans le code ci-dessous :

```
class ConcatIncorrecte {

    public static void main(String[] args) {
        String s = "China Blue";
        System.out.println(s);
        //version incorrecte
        //concatener(s, " Express");

        // version correcte:
        s = concatener(s, " Express");
        System.out.println(s);
    }

    /*
     * les raisons du comportement incorrect sont que:
     * 1. les opérations sur les chaines sont non destructives
     *(créent une autre chaine au lieu d'agir sur la chaine originale)
     * 2. l'objet s est une référence, mais les référence sont passées
     * par valeur (on peut altérer l'objet référencé, mais pas la référence
     * elle même):
     * + crée une nouvelle chaine
     * la référence de cette nouvelle chaine est affectée  s
     * (on essaie de changer la référence s et l'effet de cette modification n'
     * (revoir l'exemple du cours "méthode auxilliaires et réutilisabilité" ave
     * les tableaux)
     */

    /*
     * version incorrecte
    public static void concatener(String s,String s2 )
        {
            s += s2;
        }
    */

    // version corrigée
    public static String concatener(String s,String s2 ) {
        s += s2;
        return s;
    }
}
```

Exercice 20: Nombres amicaux (Modularisation, algorithme)

```java
class Amical {

    public static void main(String[] args) {
        int[] nombres = {1210, 45, 27, 220, 54, 284, 9890, 120, 1184};
        System.out.println("Les paires de nombres amicaux sont : ");
        afficherAmicaux(nombres);
    }

    /**
     * Vérifie si les deux nombres donnés (nb1 et nb2) sont amicaux.
     */
    public static boolean amical(int nb1, int nb2) {
        int somme = sommeDiviseur(nb1);
        return (nb1 + nb2 == somme
                    && sommeDiviseur(nb2) == somme);
    }

    /**
     * Calcule la somme des diviseurs du nombre passé en paramètre (nb1).
     */
    public static int sommeDiviseur(int nb1) {
        int somme = 0;
        for (int i = 1; i <= nb1; ++i){
            if ((nb1 % i) == 0){
                somme += i;
            }
        }
        return somme;
    }

    /**
     * Affiche tous les nombres amicaux contenus dans un tableau d'entiers.
     */
    public static void afficherAmicaux(int[] nombres){
        for (int i = 0; i < nombres.length; ++i){
            for (int j = i+1; j < nombres.length; ++j)
                if (amical(nombres[i],nombres[j])){
                    System.out.println(nombres[i] + " " + nombres[j]);
                }
        }
    }
}
```

Exercice 21: Césure (String, Modularisation, algorithme)

```java
import java.util.Scanner;

class Cesure {

    public static void main(String[] args) {
        String[] phrase = lirePhrase();
        System.out.println("Le résultat est : ");
        for (int i = 0; i < phrase.length; i++) {
            cesure(phrase[i]);
        }
    }

    /**
     * Lit une phrase depuis le terminal.
     * @return un tableau de Strings où chaque entrée
     * est un mot de la phrase lue.
     */
    static String[] lirePhrase() {
        Scanner scanner = new Scanner(System.in);

        int number = 0;
        while (number <= 0) {
            System.out.print("Donnez le nombre de mots de votre phrase : ");
            number = scanner.nextInt();
            if (number <= 0) {
                System.out.println("Entrez une valeur plus grande que 0");
            }
        }

        String[] phrase = new String [number];

        // se debarasser du \n
        scanner.nextLine();
        for (int i = 0; i < number; i++) {
            System.out.print("Donnez le mot " + (i + 1) + " : ");
            phrase[i] = scanner.nextLine();
        }

        scanner.close();

        return phrase;
    }

    /**
     * Réalise la césure pour un mot donné.
     * Pour ce faire, examine les différentes façons de couper
     * le mot: entre l'indice debut et l'indice i et entre i et
     * la longueur du mot (i variant de 1 à la longueur du mot et
     * debut étant initialisé à 0).
     * Si la régle de césure s'applique à ces deux parties
     * afficher le "-" et passer à la ligne suivante.
     * Puis, faire le même traitement en considérant la portion de mot
     * entre i et longueur du mot (debut prend la valeur de i)
     * @param mot une String représentant un mot de la phrase
     */
    static void cesure(String mot) {
```

```
        int debut = 0;
        for (int i = 1; i < mot.length(); i++) {
            char c1 = mot.charAt(i - 1);
            char c2 = mot.charAt(i);
            if ((voyelle(c1) && !voyelle(c2))) {
                String s1 = mot.substring(debut, i);
                String s2 = mot.substring(i, mot.length());
                if ((s1.length() > 1) && (s2.length() > 1)) {
                    if (!(queVoyelles(s1) || queVoyelles(s2))) {
                        System.out.println(s1 + "-");
                        debut = i;
                    }
                }
            }
        }

        String s1 = mot.substring(debut, mot.length());
        System.out.println(s1);
    }

    /**
     * Teste si un caractère est une voyelle.
     * @param c le caractère à tester
     * @return true si c est une voyelle et false sinon
     */
    static boolean voyelle(char c) {
        return ((c == 'a') ||
                    (c == 'e') ||
                    (c == 'i') ||
                    (c == 'o') ||
                    (c == 'u') ||
                    (c == 'y'));
    }

    /**
     * Teste si une chaîne de caractères ne contient que des voyelles.
     * @param s la chaîne à tester
     * @return true si s ne contient que des voyelles et false sinon
     */
    static boolean queVoyelles(String s) {
        for (int i = 0; i < s.length(); i++) {
            if (!voyelle(s.charAt(i))) {
                return false;
            }
        }
        return true;
    }
}
```

Exercice 22: Nombre de Fibonacci (récursivité)

```java
import java.util.Scanner;

class Fibonacci
{
   private static Scanner clavier = new Scanner(System.in);

   public static void main(String[] args) {
         char rep;
         do {
            int n = demanderNombre(0, 40);
            System.out.println("Méthode itérative :");
            System.out.println("    F(" + n + ") = " + FibonacciIteratif(n));
            System.out.println("Méthode récursive :");
            System.out.println("    F(" + n + ") = " + Fibonacci(n));


            do {
               System.out.print("Voulez-vous recommencer [o/n] ? ");
               rep = clavier.next().charAt(0);
            } while ((rep != 'o') && (rep != 'n'));

         } while (rep == 'o');
      }



/* ----------------------------------------------------------------------
 * Calcule de façon itérative le n-ieme nombre de Fibonacci.
 * Entrée : le nombre n
 * Sortie : F(n)
 * ---------------------------------------------------------------------- */
   static int FibonacciIteratif(int n)
      {
         int Fn = 0;    // stocke F(i)  , initialisé par F(0)
         int Fn_1 = Fn; // stocke F(i-1), initialisé par F(0)
         int Fn_2 = 1;  // stocke F(i-2), initialisé par F(-1)

         for (int i = 1; i <= n; ++i) {
            Fn   = Fn_1 + Fn_2;    // pour n>=1 on calcule F(n)=F(n-1)+F(n-2)
            Fn_2 = Fn_1;           // et on decale...
            Fn_1 = Fn;
         }
         return Fn;
      }

/* ----------------------------------------------------------------------
 * Calcule de façon récursive le n-ieme nombre de Fibonacci.
 * Entrée : le nombre n
 * Sortie : F(n)
 * ---------------------------------------------------------------------- */
   static int Fibonacci(int n)
      {
         if (n == 0)
            return 0;
         else if (n == 1)
            return 1;
```

```
         else
            return Fibonacci(n-1) + Fibonacci(n-2);
      }

/* ---------------------------------------------------------------
 * fonction demandant à l'utilisateur un nombre compris
 * dans un intervalle [a, b], ou supérieur ou égal à a
 * si b < a.
 * Entrée : les deux nombres a et b définissant l'intervalle
 * Sortie : le nombre saisi par l'utilisateur
 * --------------------------------------------------------------- */
   static int demanderNombre(int a, int b)
      {
          int res;
         do {
            System.out.print("Entrez un nombre entier ");
            if (a >= b){
               System.out.print("supérieur ou égal à " + a);
            }
            else {
               System.out.print("compris entre " + a + " et " + b);
            }

            System.out.print(" : ");
            /* La lecture de la valeur au clavier est ici délibérément simplifi
               on ne traite pas les situations où l'utilisateur se trompe
               au moment de la saisie*/
            res =  clavier.nextInt();
         } while ((res < a) || ((a < b) && (res > b)));

         return res;
      }
}
```
