MOOC Init Prog Java
Corriges des exercices facultatifs semaine 6

Fonctions simples

```java
public class Simple {
    static public double min2(double a, double b) {
        if(a < b) {
            return a;
        }
        else {
            return b;
        }
    }

    // Sans utiliser min2 :
    static public double min3(double a, double b, double c) {
        if (a > b) {
            if (a > c) {
                return a;
            } else {
                return c;
            }
        } else {
            if (b > c) {
                return b;
            } else {
                return c;
            }
        }
    }

    /*
     * En utilisant min2 : our calculer le minimum de 3 nombres,
     * on peut aussi calculer d'abord le minimum des 2 premiers, et ensuite
     * calculer le minimum de ce résultat et du dernier nombre,
     * ce qui permet de réutiliser min2 :
     */
    static public double min3v2(double a, double b, double c) {
        return min2(min2(a, b), c);
    }
}
```

Sapin

```
public class Sapins {

    static public void etoiles(int nbEtoiles) {
        for (int i = 0; i < nbEtoiles; ++i) {
            System.out.print('*');
        }
    }

    static public void espaces(int nbEspaces) {
        for (int i = 0; i < nbEspaces; ++i) {
            System.out.print(' ');
        }
    }

    static public void triangle(int nbLignes) {
        for(int i = 0; i < nbLignes; ++i) {
            espaces(nbLignes - i);
            etoiles(2*i + 1);
            System.out.println();
        }
    }

    static public void triangleDecale(int nbLignes, int nbEspaces) {
        for(int i = 0; i < nbLignes; ++i) {
            espaces(nbEspaces + nbLignes - i);
            etoiles(2*i + 1);
            System.out.println();
        }
    }

    static public void sapin() {
        triangleDecale(2, 2);
        triangleDecale(3, 1);
        triangleDecale(4, 0);

        // le tronc:
        triangleDecale(1, 3);
    }

    /* Cette fonction permet d'afficher les 3 parties du beau sapin.
     * noLigneFin a le meme role que le parametre nbLignes
     * des fonctions triangle et triangleDecale.
     * noLigneDebut definit la ligne du haut du trapeze.
     */
static public void trapezeDecale(int noLigneDebut, int noLigneFin,
                                 int nbEspaces) {
        for(int i = noLigneDebut; i < noLigneFin; ++i) {
            espaces(nbEspaces + noLigneFin - i);
            etoiles(2 * i + 1);
            System.out.println();
        }
    }

    static public void beauSapin() {
        trapezeDecale(0, 3, 2);
        trapezeDecale(1, 4, 1);
```

```
        trapezeDecale(2, 5, 0);
        System.out.println("    |||");
    }

    /* Cette fonction n'etait pas demandee.
     * Notez qu'on peut avoir deux fonctions de meme nom, si leurs
     * parametres sont differents
     */
    static public void beauSapin(int nbEtages) {
        for(int i = 0; i < nbEtages; ++i) {
            trapezeDecale(i, 3 + i, nbEtages - i);
        }
        espaces(2 + nbEtages);
        System.out.println("|||");
    }

    public static void main(String[] args) {
        beauSapin(10);
    }
}
```

Calcul approché d'une intégrale

```java
import java.util.Scanner;

public class Integrale {

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

    public static void main(String[] args) {
        double a = demanderNombre();
        double b = demanderNombre();

        System.out.println("Intégrale de f(x) entre "+ a +" et "+ b + " : ");
        System.out.println(integre(a, b));
    }

    static public double f(double x) { return x*x; }
    // et d'autres fonctions possibles si on veut:
    // static public double f(double x) { return x*x*x ; }
    // static public double f(double x) { return 1.0/x ; }
    // static public double f(double x) { return sin(x); }

    static public double integre(double a, double b) {
        double res;
        res = 41.0 * (f(a) + f(b))
                + 216.0 * (f((5 * a + b) / 6.0) + f((5 * b + a) / 6.0))
                + 27.0 * (f((2 * a + b) / 3.0) + f((2 * b + a) / 3.0))
                + 272.0 * f((a + b) / 2.0);
        res *= (b - a) / 840.0;
        return res;
    }

    static public double demanderNombre() {
        double res;

        System.out.print("Entrez un nombre réel : ");
        res = clavier.nextDouble();

        return res;
    }
}
```

Fonctions logiques

```
public class Logique {
    static public boolean nonEt(boolean a, boolean b) {
        return !(a && b);
    }

    /*
     * On peut remarquer que A ET A = A. NON(A ET A) vaut donc NON(A).
     * On peut alors écrire la fonction non ainsi :
     */

    static public boolean non(boolean a) {
        return nonEt(a, a);
    }

    /*
     * Comme NON(NON(C)) = C, on a NON(NON(A ET B)) = A ET B.
     * On peut donc écrire la fonction et ainsi:
     */
    static public boolean et(boolean a, boolean b) {
        return non(nonEt(a, b));
    }

    /*
     * NON(A OU B) = NON(A) ET NON(B).
     * Donc, (A OU B) = NON(NON(A OU B)) = NON(NON(A) ET NON(B)).
     * On peut donc écrire la fonction ou ainsi:
     */
    static public boolean ou(boolean a, boolean b) {
        return nonEt(non(a), non(b));
    }
}
```

Recherche par dichotmie

```java
import java.util.Scanner;
class Dichotomie {
    private static Scanner scanner = new Scanner(System.in);

    final static int MINIMUM = 0;
    final static int MAXIMUM = 20;

    public static void main(String[] args) {
        System.out.println("Pensez à un nombre entre " + MINIMUM + " et " +
                MAXIMUM);
        System.out.println("Votre nombre était: " +
                cherche(MINIMUM, MAXIMUM));
    }

    /**
     * Cherche un nombre dans l'intervalle [gauche, droite] en fonction des
     * indications données par l'utilisateur.
     * @param gauche La borne inférieure de l'intervalle
     * @param droite La borne supérieure de l'intervalle
     * @return Le pivot
     */
    static int cherche(int gauche, int droite) {
        if (droite < gauche) {
            System.out.println("Erreur: vous avez  répondu de façon incohérente
            return droite;
        } else if (droite == gauche) {
            return droite;
        } else {
            // Détermine le milieu de l'intervalle
            int pivot = (gauche + droite) / 2;

            char rep;
            do {
                System.out.println("Le nombre est-il <, > ou = à " + pivot + "
                rep = scanner.next().charAt(0);
            } while ((rep != '=') && (rep != '>') && (rep != '<'));

            switch (rep) {
            case '=':
                return pivot;
            case '<':
                return cherche(gauche, pivot - 1);
            case '>':
                return cherche(pivot + 1, droite);
            default:
                return droite;
            }
        }
    }
}
```

Recherche approchée de racine

```
import java.util.Scanner;

public class Reseq {
    private final static double EPSILON = 1e-6;

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

    static public double f(double x) { return (x-1.0)*(x-1.5)*(x-2.0); }

    static public double df(double x) { return (f(x+EPSILON)-f(x))/EPSILON; }

    static public double itere(double x) { return x - f(x) / df(x); }

    public static void main(String[] args) {
        double x1, x2;
        System.out.print("Point de départ ? : ");
        x2 = clavier.nextDouble();

        do {
            x1 = x2;
            System.out.println(" au point "+ x1 + " : ");
            System.out.println("    f(x)  = "+ f(x1));
            System.out.println("    f'(x)  = "+ df(x1));

            x2 = itere(x1);
            System.out.println("    nouveau pont = "+ x2);
        } while(Math.abs(x2-x1) > EPSILON);

        System.out.println("Solution : "+x2);
    }

}
```

Placement sans recouvrement

```java
import java.util.Scanner;

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

    public static void main(String[] args) {
        boolean [][] grille = new boolean [10][10];
        initGrille(grille);
        ajouterElements(grille);
        System.out.println("\nMerci d'avoir joué!");
    }

    /**
     * Essaie de remplir la grille avec un objet de longueur donnée
     * dans une direction donnée en partant des coordonnées
     * (targetRow, targetCol).
     * @param grille La grille à remplir
     * @param targetRow coordonnée Y du début du remplissage
     * @param targetCol coordonnée X du début du remplissage
     * @param direction direction du remplissage (N, S, E, O)
     * @param longueur nombre de cases à remplir
     * @return true si le placement est réussi et false sinon
     */
    static boolean remplitGrille(boolean [][] grille, int targetRow,
                int targetCol, char direction, int longueur) {

        int deltaRow = 0;
        int deltaCol = 0;
        int dim = grille.length;

        // On calcule ici la direction dans laquelle aller depuis
        // les coordonnées indiquées par l'utilisateur.
        // Ex: pour aller au nord, il faut monter d'une
        // ligne en restant sur la même colonne.
        switch (direction) {
        case 'N':
            deltaRow = -1;
            deltaCol = 0;
            break;
        case 'S':
            deltaRow = 1;
            deltaCol = 0;
            break;
        case 'E':
            deltaRow = 0;
            deltaCol = 1;
            break;
        case 'O':
            deltaRow = 0;
            deltaCol = -1;
            break;
        default:
            // Arrête la méthode et retourne false si direction est invalide
            return false;
        }
```

```
    // On se place sur la case de départ
    int tempRow = targetRow;
    int tempCol = targetCol;

    // Est-il possible de placer l'élément ?
    boolean possible = true;

    // Avant de modifier la grille il faut vérifier s'il est
    // possible de mettre tout l'objet
    for (int i = 0; (i < longueur && possible); i++) {
        if ((tempRow < 0) || (tempRow >= dim)) {
            possible = false;
        } else if ((tempCol < 0) || (tempCol >= dim)) {
            possible = false;
        } else if (grille [tempRow][tempCol]) {
            possible = false;
        }

        // On se déplace sur la case suivante
        tempRow += deltaRow;
        tempCol += deltaCol;
    }

    // Le placement est possible, et on peut donc mettre
    // à jour la grille
    if (possible) {
        tempRow = targetRow;
        tempCol = targetCol;

        for (int i = 0; i < longueur; i++) {
            grille [tempRow][tempCol] = true;
            tempRow += deltaRow;
            tempCol += deltaCol;
        }
    }

    return possible;
}

/**
 * Initialise toutes les cases de la grille à false.
 * @param grille La grille à initialiser
 */
static void initGrille(boolean[][] grille) {
    int dim = grille.length;
    for (int row = 0; row < dim; row++) {
        for (int col = 0; col < dim; col++) {
            grille[row][col] = false;
        }
    }
}

/**
 * Permet à l'utilisateur de demander le placement d'objets
 * sur la grille.
 * @param grille
 */
static void ajouterElements(boolean[][] grille) {
```

```
        int coordX = 0;
        int coordY = 0;
        char dir = 'N';
        int length = 0;
        System.out.print("Entrez la taille " +
                "(négative pour arrêter le programme): ");
        length = scanner.nextInt();
        // L'utilisateur signifie qu'il veut arrêter
        // d'introduire des objets en spécifiant une
        //longueur négative
        while (length >= 0) {
            do {
                System.out.print("Entrez coord. x : ");
                coordX = scanner.nextInt();
            } while ((coordX < 0) || (coordX >= grille.length));

            do {
                System.out.print("Entrez coord. y : ");
                coordY = scanner.nextInt();
            } while ((coordY < 0) ||(coordY >= grille.length));

            do {
                System.out.print("Entrez direction (N,S,E,O) : ");
                dir = scanner.next().charAt(0);
            } while ((dir != 'N') && (dir != 'S') && (dir != 'E')
                     && (dir != 'O'));
            System.out.print("Placement en (" + coordX + "," + coordY +
                        ") direction " + dir + " longueur " + length + " -> ");

            if (remplitGrille(grille, coordY, coordX, dir, length)) {
                System.out.println("Succès");
                afficheGrille(grille);
            } else {
                System.out.println("Echec");
            }
            System.out.println();
            System.out.print("Entrez la taille " +
                        "(négative pour arrêter le programme): ");
            length = scanner.nextInt();
        }
    }

    /**
     * Affichage de la grille.
     * @param grille La grille a afficher
     */
    static void afficheGrille(boolean[][] grille) {
        int dim = grille.length;
        for (int row = 0; row < dim; row++) {
            for (int col = 0; col < dim; col++) {
                if (grille[row][col]) {
                    System.out.print("#");
                } else {
                    System.out.print(".");
                }
            }
            System.out.println();
        }
```

```
        }
}
```
