MOOC Intro POO 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 19 : Intégrité des données

Voici une solution possible :

```
import java.util.Scanner;
import java.util.ArrayList;

class SafeProject {

    private final static int NB_PROJECTS = 3;

    public static void main(String[] args) {
        ArrayList<Project> projects = new ArrayList<Project>();

        do {
            Project project = new Project();
            project.readProject();
            projects.add(project);
        } while (projects.size() < NB_PROJECTS);
    }
}

class Project {
    private String name = null;
    private String subject = null;
    private int duration = -1;

    public Project() {}

    // methode utilitaire utilis'ee par readProject pour la
    // lecture des entiers
    private int readInt(Scanner scanner) throws WrongDurationException {
        String strNumber = scanner.nextLine();
        int number;
        try {
            number = Integer.parseInt(strNumber);
        } catch (NumberFormatException e) {
            throw new WrongDurationException(strNumber + " is not a number !");
        }
        if (number <= 0) {
            throw new WrongDurationException("Duration should be stricly positive !");
        }
        return number;
    }

    // methode utilitaire utilis'ee par readProject pour la
    // lecture des String
    private String readString(Scanner scanner) throws NameTooLongException {
        String str = scanner.nextLine();
        if (str.length() > 50) {
            throw new NameTooLongException("Value should not exceed 50 characters");
        }
        return str;
    }


    // La methode readProject redemande les donn'ees
    // jsuqu'a ce qu'elles soient correctes
    public void readProject() {
        Scanner scanner = new Scanner(System.in);
        do {
            System.out.println("Donnez le nom du projet : ");
            try {
                name = readString(scanner);
            } catch (NameTooLongException e) {
                System.err.println("[ " + e.getMessage() + " ]");
            }
        } while (name == null);

        do {
            try {
                System.out.println("Donnez le sujet du projet : ");
                this.subject = readString(scanner);
            } catch (NameTooLongException e) {
                System.err.println("[ " + e.getMessage() + " ]");
            }
        } while (subject == null);
```

```
        do {
            try {
                System.out.println("Donnez la durée du projet : ");
                this.duration = readInt(scanner);
            } catch (WrongDurationException e) {
                System.err.println("[ " + e.getMessage() + " ]");
            }
        } while (duration < 0);
    }
}

class WrongDurationException extends Exception {
    public WrongDurationException(String msg) {
        super(msg);
    }

    public WrongDurationException() {
        super("Wrong duration !");
    }
}

class NameTooLongException extends Exception {

    public NameTooLongException(String s) {
        super(s);
    }

    public NameTooLongException() {
        super("Too long name !");
    }

}
```

Exercice 20 : Compile ... ou pas

Voici une solution possible :

```
class Exemple {

    /*Expliquer pourquoi ce code ne compile pas
    SOLUTION : Il faut entourer l'appel de la méthode foo() par un bloc try/catch
    */
    public void m1() {
        foo();
    }

    public int foo() throws Exception {
        throw new Exception();
    }

    /*Expliquer pourquoi ce code n'est pas considéré comme bon
    SOLUTION : Le premier bloc interceptes les exception de type "Exception",
    donc il intercepte toutes les exceptions possibles. Mais si le code situé
    à l'intérieur du bloc try lève des exceptions précises (NullPointerException,
    IllegalArgumentException, etc...) on perd l'information sur le type d'exception
    qui a été levée.
    */
    public void m2() {
        try {
            //do stuff...
        } catch (Exception e) {

        }
    }

    /*Expliquer pourquoi ce code ne compile pas
    SOLUTION : Le premier bloc interceptes les exception de type "Exception",
    donc il intercepte toutes les exceptions possibles, y-compris une exception
    de type "NullPointerException". Le deuxième bloc n'est donc jamais atteint.
    */
    public void m3() {
        try {
            //do stuff...
        } catch (Exception e) {

        } catch (NullPointerException e) {

        }
    }

    /*Expliquer pourquoi ce code ne compile pas
    SOLUTION : Il manque "throws CustomCheckedException" à la signature
    de la méthode.
    */
    public void m4() {
        throw new CustomCheckedException();
    }

    private class CustomCheckedException extends Exception {

        private static final long serialVersionUID = -7944813576443065516L;

        public CustomCheckedException() {
            //nothing
        }

    }

    /*Expliquer pourquoi ce code ne compile pas
    SOLUTION : La variable "age" n'est pas initialisée. Si le déroulement
    de la méthode "getAccessCode()" ne lance pas d'exception, ce n'est pas un
    problème. Mais dans le cas contraire, la variable "age" ne se verra affecter
    aucune valeur. Il faut donc lui donner une valeur "par défaut".
    */
    public int m5() {
        int age;
        String s = "24";
        try {
            age = getAccessCode();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
        return age;
    }

    public int getAccessCode() throws IllegalAccessException {
        throw new IllegalAccessException();
    }

    /*Expliquer pourquoi ce code COMPILE
```

```
    SOLUTION : Toutes les exceptions héritant du type RuntimeException ne doivent pas obligatoirement
    être traitées (unchecked exceptions), le compilateur ne signalera ainsi aucune erreur,
    même si l'on n'utilise pas de bloc try-catch. Il n'est pas non plus demandé d'ajouter l'instruction
    "throws RuntimeException" dans la signature de la méthode;
    Exemple de "unchecked exceptions" :
    NullPointerException, NumberFormatException, IllegalArgumentException, etc...
    */
    public void m6() {
        bar();
    }

    public int bar() {
        throw new RuntimeException();
    }


}
```

Exercice 21 : Attrapez-les toutes

Voici une solution possible :

```
import java.util.InputMismatchException;
import java.util.Scanner;

class CustomException extends Exception {

    public CustomException(String string) {
        super(string);
    }

}

final class UtilsMatrix {

    public static int[][] multiply(int[][] mat1, int[][] mat2) throws CustomException {

        checkMatrix(mat1);
        checkMatrix(mat2);

        int rows1 = mat1.length;
        int cols1 = mat1[0].length;
        int rows2 = mat2.length;
        int cols2 = mat2[0].length;

        if (cols1 != rows2) {
            throw new CustomException("Matrix dimensions must fits");
        }

        int[][] result = new int[rows1][cols2];
        for (int i = 0; i < rows1; i++) {
            for (int j = 0; j < cols2; j++) {
                for (int k = 0; k < rows2; k++) {
                    result[i][j] += mat1[i][k] * mat2[k][j];
                }
            }
        }
        return result;
    }

    private static int tryReadInt(Scanner scanner) throws CustomException {
        int val = -1;
        try {
            val = scanner.nextInt();
        } catch (InputMismatchException e) {
            throw new CustomException("Vous devez entrez un nombre");
        }
        if (val <= 0) {
            throw new CustomException("Vous devez entrez des valeurs strictement positives !");
        }
        return val;

    }

    public static int[][] readMatrix() throws CustomException {
        Scanner scanner = new Scanner(System.in);

        int row = -1;
        int col = -1;

        System.out.println("Nouvelle matrice");

        System.out.print("\t Entrez nombre de lignes : ");
        row = tryReadInt(scanner);
        System.out.print("\t Entrez nombre de colonnes : ");

        col = tryReadInt(scanner);

        int[][] result = new int[row][col];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                System.out.print("\t Contenu cellule [" + i + "][" + j + "] : ");
                try {
                    result[i][j] = scanner.nextInt();
                } catch (InputMismatchException e) {
                    throw new CustomException("Vous devez entrez un nombre");
                }
            }
        }
        return result;
    }

    /**
     * Ensure that the matrix is not null nor empty and in the right format
     * @param mat
```

```
     * @throws CustomException
     */
    public static void checkMatrix(int[][] mat) throws CustomException {
        if (mat == null) {
            throw new CustomException("Matrix should be initialized");
        }
        if (mat.length == 0) {
            throw new CustomException("Matrix should not be empty");
        }
        int lineLength = mat[0].length;
        for (int[] lines : mat) {
            if (lineLength != lines.length){
                throw new CustomException("This is clearly not a matrix !");
            }
        }
    }

    public static void display(int[][] mat) {
        for (int[] lines : mat) {
            for (int item : lines) {
                System.out.print(item + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] mat1 = null;
        int[][] mat2 = null;
        boolean dataOk = true;
        do {
            try {
                dataOk = true;
                mat1 = readMatrix();
                mat2 = readMatrix();
            } catch (CustomException e) {
                System.err.println(e.getMessage());
                dataOk = false;
            }
        } while (!dataOk);

        int[][] prod = null;
        try {
            prod = multiply(mat1, mat2);
            display(prod);
        } catch (CustomException e) {
            System.err.println(e.getMessage());
        }
    }
}
```

Exercice 22 : Compression RLE

Voici une solution possible :

```
import java.util.Scanner;

class RLE {

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

    private final static char FLAG = '#';

    public static void main(String[] args) {

        System.out.println("Entrez les données à comprimer : ");
        String dta = scanner.nextLine();
        String rle = RLEAlgorithm.compress(dta, FLAG);
        System.out.println("Forme compressée:   " + rle + "\n[ratio = " + rle.length()*100.0/dta.length() + "%]");
        String dcp = "";
        try {
            dcp = RLEAlgorithm.decompresse(rle, FLAG);
        } catch (RLEException e) {
            e.printStackTrace();
        }
        if (!dcp.equals(dta)) {
            System.out.println("Erreur - données corrompues:" + dcp);
        } else {
            System.out.println("décompression ok!");
        }

        // teste la decompression
        System.out.println("Entrez les données à décompresser : ");
        dta = scanner.nextLine();
        try {
            dcp = RLEAlgorithm.decompresse(dta, FLAG);
            System.out.println("décompressé : " + dcp);
        }
        catch (RLEException exception) {
            System.out.println("Erreur de décompression : " + exception.getMessage() + "\n");
            System.out.println("décodé à ce stade : " + exception.getDecoded() + "\n");
            System.out.println("non décodé        : " + exception.getRemaining());
        }
    }
}

class RLEException extends Exception {
    private String decoded;
    private String remaining;

    public RLEException(String message, String decoded, String remaining) {
        super(message);
        this.decoded = decoded;
        this.remaining = remaining;
    }

    public String getRemaining() {
        return remaining;
    }

    public String getDecoded() {
        return decoded;
    }

}

class RLEAlgorithm {

    public static String compress(String data, char flag) {
        char currentChar;         // caractère extrait
        int  repetitionCount;         // nombre de répétitions à produire
        int currentIndex = 0; // position dans la chaine à décompresser
        String result = "";     // chaine compressée

        while (currentIndex < data.length()) {
            repetitionCount = 1;
            if ((currentChar = data.charAt(currentIndex)) == flag) {
                ++currentIndex;
                result += currentChar;
                result += 0;
            } else {
                while ((++currentIndex < data.length()) && (repetitionCount < 9) && (data.charAt(currentIndex) == currentChar)) {
                    repetitionCount++;
                }
                if (repetitionCount >= 3) {
                    result += currentChar;
                    result += flag;
                    result += repetitionCount;
```

```
                } else {
                    while (repetitionCount-- > 0) {
                        result += currentChar;
                    }
                }
            }
        }
        return result;
    }

    public static String decompresse(String rledata, char flag) throws RLEException {
        char currentChar;         // caractère extrait
        int  repetitionCount;         // nombre de répétitions à produire
        int currentIndex; // position dans la chaine à décompresser
        String out = "";     // chaine décompressée

        for (currentIndex = 0; currentIndex < rledata.length(); currentIndex++) {
            currentChar = rledata.charAt(currentIndex);
            if ((currentIndex+1 < rledata.length())
                    && (rledata.charAt(currentIndex+1) == flag)) {
                // flag en p+1 détecté ?
                currentIndex += 2;
                if (currentIndex >= rledata.length()) {
                    // erreur : on devrait avoir qqchose derrière le FLAG
                    String msg = "Flag " + flag + " sans rien derrière";
                    String decode = out + currentChar;
                    String remaining = flag + rledata.substring(currentIndex);
                    throw new RLEException(msg, decode, remaining);

                } else if ((rledata.charAt(currentIndex) >= '0')
                        && (rledata.charAt(currentIndex) <= '9')) {
                    repetitionCount = Integer.parseInt("" + rledata.charAt(currentIndex)); // on récupère l
                    if (repetitionCount >= 1) {
                        while (repetitionCount-- > 0) {
                            out += currentChar; // décompression des l x c
                        }
                    } else {
                        // l=0 -> le flag était dans les données
                        out += currentChar;
                        out += flag;
                    }
                } else {
                    // erreur : ce qui suit le FLAG n'est pas correct
                    String msg = "Caractère " + rledata.charAt(currentIndex) + " incorrect après le flag " + flag;
                    String decoded = out + currentChar;
                    String remaining = flag + rledata.substring(currentIndex);
                    throw new RLEException(msg,decoded,remaining);
                }
            } else {
                // caractère seul
                out += currentChar;
                if (currentChar == flag) {
                    ++currentIndex; // saute le 0 dans le cas d'un flags au début ("#0")
                }
            }
        }
        return out;
    }

}
/*
 * Question subsidiaire :
 *
 * - pour coder de grandes séquences consécutives, on peut par exemple utiliser
 *   les valeurs 'spéciales' de l (dans notre cas 0, 1 et 2, puisque les
 *   chaînes de longueurs inférieur à 3 ne sont pas compressées. Nous utilisons
 *   déjà le 0 pour le flag seul) pour indiquer une extension du codage du
 *   nombre de répétitions. Par exemple :
 *     1) l = 1 -> codage sur par exemple 2 digits => on passe à une longueur
 *                 max de 99 caractères
 *     2) l = 2 -> codage sur par exemple 3 digits => l max = 999
 *   et on peut continuer le raisonnement :
 *     avec l = 1, la longueur 'minimale' codée de manière étendue devrait être
 *     10 (sinon on code comme avant).
 *       1b) on peut donc encoder l avec un décalage de 10 -> l code donc de 10
 *           (#100) à 109 (#199), lmax devient donc 109 au lieu de 99.
 *       1c) on peut utiliser à nouveau tout ou une partie de ces valeurs pour
 *           prolonger l'encodage étendu...
 *
 *       Exemple: on doit coder 105 caractères 'c' consécutifs:
 *          cas 1:   c#199c#6    (1->99 puis 105-99=6 codés en normal)
 *          cas 2:   c#2105      (2->105 sur 3 digits)
 *          cas 1b:  c#195       (1->95+biais = 95+10 = 105)
 *
 * - On peut utiliser la même technique pour coder des répétitions du flag ;
 *   dans ce cas, on peut utiliser par exemple la valeur spéciale '2' pour
 *   indiquer que le flag est effectivement compressé, et le nombre de
 *   répétitions est encodé sur le prochain caractère.
 *
```

```
*   Exemple:
*             à compresser: aa#####bb
*                         -> aa#25bb
*
*   On constate que dans ce cas, il devient intéressant de coder des
*   répétitions de 2 flags déjà (3 caractères (#22) contre 4 avec la version
*   '#0' (#0#0))
*
* En résumé, il est possible d'utiliser les valeurs 'impossibles' de l pour
* indiquer des cas spéciaux, ou encore encoder les valeurs avec un certain
* biais. On étend ainsi légèrement la plage de valeurs représentable, pour un
* très faible surcoût de complexité.
* À ce propos, l'un des avantage de la compression RLE est justement sa
* faible complexité algorithmique (i.e. temps consommé pour réaliser la
* compression).
*/
```
