MOOC Intro POO Java
Exercices semaine 6

Exercice 19 : Intégrité des données

Le programme fourni `SafeProject.java` ne fait aucune vérification sur l'intégrité des données servant à la création d'un objet de type `Project`.

Modifiez-le de sorte à :

- ce que ne soient créés que des projets dont le nom et le sujet n'excède pas les 50 caractères;
- garantir que la durée du projet soit bien un entier positif.

les nom, sujet et durée seront redemandés à l'utilisateur tant qu'ils sont introduits de façon incorrecte.

Vous introduirez pour cela deux classes d'exceptions personalisées `WrongDurationException` et `NameTooLongException`

Une `String`, `strNumber`, correspondant à un entier peut être transformée en `int` par l'appel à la méthode statique `parseInt` de la classe `Integer`(`Integer.parseInt(strNumber)`).

Si `strNumber` ne corresponds pas à un `int` une `RunTimeException` de type `NumberFormatException` sera lancée.

Le programme sera proprement modularisé.

Exercice 20 : Compile .. ou pas

Pas de programmation pour cet exercice! Répondez simplement aux questions posées dans le code fourni (`Exemple.java`) et repris ci-dessous.

Quelques incursions dans la documentation de l'API de Java vont être nécessaire pour répondre aux questions.

```
class Exemple {

    /*Expliquer pourquoi ce code ne compile pas
    */
    public void m1() {
        foo();
    }

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

    /*Expliquer pourquoi ce code n'est pas considéré comme bon
    */
    public void m2() {
        try {
            //do stuff...
        } catch (Exception e) {

        }
    }

    /*Expliquer pourquoi ce code ne compile pas
    */
    public void m3() {
        try {
            //do stuff...
        } catch (Exception e) {

        } catch (NullPointerException e) {

        }
    }

    /*Expliquer pourquoi ce code ne compile pas
    */
    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
    */
    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
```

```
    */
    public void m6() {
        bar();
    }

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

}
```

Exercice 21 : Attrapez-les toutes ...

Pour cet exercice, incorporez au projet Eclipse créé pour cet exercice, les fichiers fournis `UtilsMatrix.java` et `MatrixTest.java`, dans le dossier `src/`

La classe fournie `UtilsMatrix` vous fait revisiter le thème de la multiplication de matrices. Modifiez-la de sorte à ce qu'il lance une exception de type `CustomException` en toute situation d'erreur.

Certaines des méthodes utilisées lancent une `InputMismatchException`. Ces exceptions devront êtres interceptée et remplacées par des `CustomException`.

Test unitaires

Le fichier fourni `MatrixTest.java` contient des tests unitaires. Il permet de tester que votre solution fonctionne bien dans différents cas de figures. Il n'est pas nécessaire d'en comprendre le fonctionnement.

Commencez par configurer votre projet pour les tests unitaires : `clic-droit sur le projet -> Build Path -> Add librairies...` et sélectionner `junit` puis `JUnit4` dans la liste déroulante.

Pour lancer le test : `clic-droit sur MatrixTest > Run as > Junit test`

Si vous avez bien anticipé toutes les situations à gérer par des exceptions, il y aura 6 tests sur 6 réussis (ce sera indiqué dans Eclipse). Sinon, ceux qui auront échoué vous mettront sur la voie de ce qui manque.

Exercice 22 : Compression RLE

La compression RLE (*Run Length Encoding*) est une ancienne technique de compression, utilisée en particulier dans le codage des images. Le principe très simple de cette technique est le suivant: toute séquence de caractères *c* identiques, et de longueur *L* (assez grande) est codée par :

« *c flag L* » où *flag* est un caractère spécial, si possible peu fréquent dans les données à compresser

les autres séquences n'étant pas modifiées.

Lorsque le caractère spécial *flag* est rencontré dans la séquence d'origine, il est codé par la séquence spéciale «*flag* 0» (i.e. pas de caractère *c*, et *L* = 0).

Dans le fichier fourni `RLE.java`, commencez par compléter la classe `RLEAlgorithm` en y ajoutant les méthodes :

1. `static String` **`compresse`**`(String, data, char flag);`

   de sorte qu'elle retourne la chaîne correspondant à la compression par l'algorithme RLE de la séquence *data* donnée en argument, en utilisant *flag* comme caractère spécial.

   La longueur *'assez grande'* pour les séquences de caractères identiques à compresser est de 3.

   Comme nous codons la longueur sous forme de un caractère, la longueur **maximale** d'une séquence à compresser est donc 9.

   Dans le cas d'une séquence de plus de 9 caractères identiques, les 9 premiers seront codés de manières compressées puis la suite de la séquence sera considérée comme une nouvelle séquence à coder. [c.f. exemple de fonctionnement].

2. `static String` **`decompresse`**`(String rledata, char flag);`

   de sorte qu'elle retourne la chaîne `rledata` sous sa forme décompressée.
3. Complétez la classe fournie `RLEException` pour gérer les cas d'erreurs possibles dans la fonction de décompression.

   En cas d'erreur dans la chaîne à décoder, l'exception renvoyée devra contenir :

   1. un message d'erreur explicitant l'erreur qui s'est produite [cf l'exemple de fonctionnement]
   2. le message déjà décodé (i.e. avant l'erreur)
   3. le reste du message à décoder (i.e. après l'endroit où l'erreur s'est produite).

      Note : Pour extraire une sous-chaîne on pourra utiliser `s.substring(int beginIndex,int endIndex)`.

4. ***Optionnel :*** proposez (en commentaire à la fin du fichier) une amélioration de l'algorithme visant à mieux gérer
   i. les cas où les séquences de caractères identiques sont excessivement longues,
   ii. où le caractère spécial est présent fréquemment (et de manière groupée) dans les données à compresser.

Votre programme doit pouvoir être testé par la méthode `main` fournie, sans qu'elle ne soit retouchée,

Exemples de fonctionnement *(flag = '#')*

```
Entrez les données à comprimer : #aaaaa
Forme compressée:   #0a#5
[ratio = 83.3333%]
décompression ok!
Entrez les données à décompresser : #0a#3
décompressé : #aaa

Entrez les données : aa#4baaaabb#ddddddddddd###aaaaaaaaaaaaaaa
Forme compressée : aa#04ba#4bb#0d#9dd#0#0#0a#9a#6
[ratio = 73.17%]
décompression ok!
Entrez les données à décompresser : aa#04ba##4bb#0d
Erreur de décompression : caractère # incorrect après le flag #
décodé à ce stade : aa#4ba
non décodé        : ##4bb#0d
```
