Licence Informatique — **UE J1IN4W01**
Université de Bordeaux — **Programmation 2**
2015-2016 — **Travaux Dirigés - feuille 8**

Exercice 1.
On considère le code suivant :

```
public class TestException{
    static void usage() {
        System.out.println("java␣TestException␣<entier>\n");
        System.exit(1);
    }

    public static void main( String[] args){
        if (args.length != 1)
            usage();
        int u = Integer.parseInt(args[0]);
    }
}
```

et un extrait de la documentation de la méthode `parseInt` de la classe `Integer` :

```
public static int parseInt(String s) throws NumberFormatException
Throws:
    NumberFormatException - if the string does not contain a parsable integer.
```

1. Expliquer le résultat de l'exécution suivante :
   ```
   $ java TestException 8.8
   Exception in thread "main" java.lang.NumberFormatException: For input string: "8.8"
   at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
   at java.lang.Integer.parseInt(Integer.java:481)
   at java.lang.Integer.parseInt(Integer.java:514)
   at TestException.main(TestException.java:10)
   ```
2. Utiliser un bloc `try ... catch ...` de façon à capturer l'exception `NumberFormatException` et à obtenir le résultat suivant :
   ```
   $ java TestException 8.8
   erreur : For input string: "8.8"
   java TestException <entier>
   ```

Exercice 2.
On considère l'interface `ShapeStack` suivante :

```
public interface ShapeStack {

    public boolean isEmpty();
    public void pop();
    public void push(Shape s);
    public Shape top() throws EmptyStackException;

    public boolean equals(ShapeStack s);
    public ShapeStack clone();
    public String toString();
}
```

1. L'exception `EmptyStackException` est une exception d'utilisation : cette exception est levée si la méthode `top` est invoquée sur une pile vide. Cette exception peut donc être déclarée dans la signature de la méthode `top`.
   — Écrire la classe `EmptyStackException`. L'exception `EmptyStackException` est une exception contrôlée, et hérite donc de la classe `Exception`.
   — Modifier en conséquence le code écrit pour la feuille5
   (voir http ://dept-info.labri.fr/ENSEIGNEMENT/programmation2/corriges/ pour un corrigé de cette feuille).
2. L'exception `FullStackException` est une exception d'implémentation car elle est liée à une implémentation particulière d'une pile : lorsqu'une pile est implémentée par un tableau de taille fixe, cette exception est levée si la méthode `push` est invoquée sur une pile pleine. Cette exception ne doit donc pas être déclarée dans la signature de la méthode `push`.
   — Écrire la classe `FullStackException`. L'exception `FullStackException` est une exception non contrôlée, et hérite donc de la classe `RuntimeException`.
   — Modifier en conséquence le code écrit pour la feuille5.

Exercice 3.
a) Écrire un programme récursif qui calcule la suite entière suivante (où la division est une division entière) :

$$\forall n > 1, \begin{cases} si\ n\ pair, & u_n = u_{n-1}/u_{n-2} - u_{n-2}/u_{n-1} \\ si\ n\ impair, & u_n = u_{n-2}/u_{n-1} - u_{n-1}/u_{n-2} \end{cases}$$

Les valeurs n, u_0 et u_1 sont données en paramètres du programme.

b) Que se passe-t-il pour les valeurs suivantes : $n = 10$, $u_0 = 3$, $u_1 = 1$?
c) Modifier ce programme pour intercepter correctement l'erreur et afficher le message suivant en français.

```
[ERREUR]java.lang.RuntimeException: Division par 0
```

Exercice 4. Pour savoir à quel moment l'erreur s'est produite dans l'évaluation de la suite, il faut remonter la pile des appels de fonctions lorsqu'il y a une erreur.
Pour cela nous allons créer une classe d'exception dans laquelle nous ajouterons les informations nécessaires pendant la remontée dans la pile d'appel des fonctions.

a) Écrire une classe `ImpossibleEvaluationException` qui hérite de la classe `Exception` et qui comporte :
   — Une variable membre privée `traces` de type `ArrayList<String>`,
   — Un constructeur qui prend en paramètre une chaîne de caractères et qui appelle le constructeur de la classe parent avec cette chaîne.
   — Une fonction membre `addTrace(String trace)` qui ajoute la chaîne `trace` dans le vecteur `traces`.
   — Une fonction membre `toString` qui transforme le vecteur `trace` en chaîne de caractères.
b) Ajouter cette exception dans le programme de l'exercice 1.
Cette opération consiste à modifier la méthode de calcul de la suite et à y ajouter les *try*, *catch*, *throw*, *throws* nécessaires.

```
$ java Suite 10 3 1
ImpossibleEvaluationException: Division par 0
[dans suite (5,3,1),
appelé par suite (6,3,1),
appelé par suite (7,3,1),
appelé par suite (8,3,1),
appelé par suite (9,3,1),
appelé par suite (10,3,1)]
```