Université Paris Diderot — **Programmation Orientée Objet**
L2, EIDD et M1 linguistiques — **Année 2015-2016**

TD n° 7

I/O et Exceptions

Dans cet TD nous allons travailler sur I/O et sur les Exceptions.

1 Input - Output

Exercice 1 On va créer une classe `Input` avec un attribut `Scanner`. Cette classe sera utilisée pour lire une chaîne de charactères qui est tapée par l'utilisateur à partir de son clavier. Le scanner aura `protected` comme modificateur. La raison pour laquelle on va choisir ce type de modificateur vous sera plus claire dans les exercices du groupe 2.

```
1  import java.io.*;
2  import java.util.*;
3  public class Input {
4
5          protected Scanner scanner;
6
7          public Input(){
8                  //...
9          }
10
11 }
```

1. Écrire le constructeur pour la classe `Input`. En particulier, il va créer un objet `Scanner` avec `System.in` comme paramètre.
2. Implémenter la méthode `public String readLine()`. La méthode renvoie un String, si la lecture est bien réussie, `null` sinon.
3. Implémenter la méthode `public String readLine(String message)`. Cette méthode prend en paramètre un message qui sera affiché avant de commencer la lecture de l'input.
4. En utilisant les méthodes de la classe `Scanner`, implémenter dans `Input` les méthodes suivantes :
   — `public int readInt(String message)`
   — `public double readDouble(String message)`
   — `public boolean readBoolean(String message)`
   La classe `Scanner` peut effectuer la lacture des chaînes de charactères, mais avant de renvoyer les résultats il faudra transformer le type de chaque élément (e.g. utiliser `Integer.parseInt(String s)` pour les entiers, ...).

5. Créer une classe `Main` utilisant un objet `Input` pour faire des tests. En regardant le code suivant :

```
1 Input i = new Input();
2 int data = i.readInt("Taper un entier");
3 System.out.println("Tu as tapé " + data);
```

Qu'est-ce qu'il se passe quand on va écrire :

(a) 123

(b) asdasdasd

(c) 1234567890987654321

Exercice 2 De manière similaire à ce qu'on a fait pour la classe `Input`, créer maintenant une classe `Output` qui va afficher à l'écran un objet de type String.

Exercice 3 Comment peut-on grouper `Input` et `Output` dans une autre classe "container" ? Faire un exemple en utilisant les interfaces et les deux classes ci-dessus.

Maintenant on va parler du mécanisme qui s'appelle "tokenization". Un objet répresenté par un String peut être imaginé comme une liste de plusieurs morceaux. Chaque morceau est normalement séparé des autres en utilisant un charactère particulier (par example, des tirets ou des espaces).

La méthode `public String[] split( String regex )` qui se trouve dans la classe `String` découpe la chaîne de caractères en plusieurs chaînes. Ces chaînes sont retournées dans un tableau de chaînes de caractères. L'argument `regex` est utilisé pour indiquer le charactère ou les charactères qui séparent chaque morceau. Pendant le découpage, la `regex` n'est pas stockée dans les morceaux.

Exercice 4 Regarder le morceau de code suivant :

```
1  String s;
2  s = "Hello - je m'appelle 'George' et je travaille - pour l'instant
       - chez Mario";
3  int count=0;
4  String [] tokens = s.split(" ");
5
6  for(String tok : tokens)
7  {
8          System.out.println(tok);
9          count++;
10 }
11 System.out.println("Count: "+count);
```

Qu'est-ce qu'il va afficher ? Quelle sera la valeur de la variable `count` ? Indiquer vos réponses dans le cas où on va remplacer le split par :

— `s.split("");` (String vide)
— `s.split(" ");` (espace)
— `s.split("-");` (tiret)
— `s.split("'");` (apostrophe)
— `s.split("' ");` (apostrophe et espace)

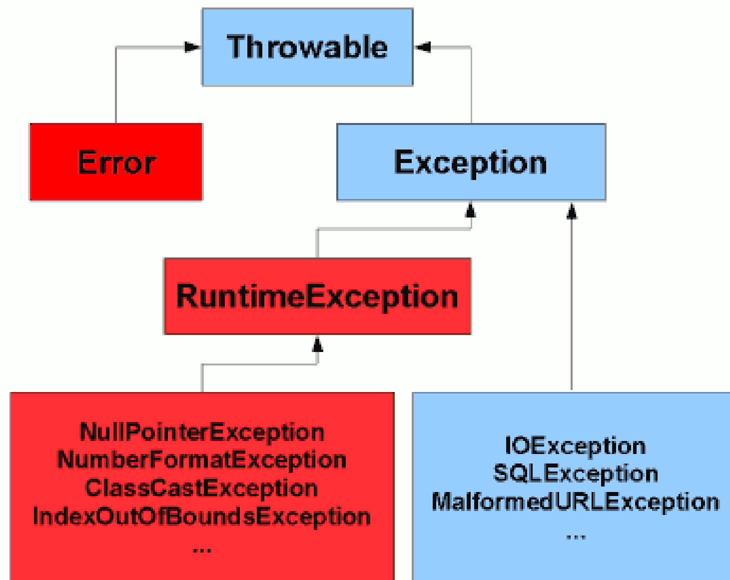

FIGURE 1 – Arbre des exceptions

2 Exceptions

Les exceptions représentent un mécanisme de gestion des erreurs pendant l'exécution du code. Une exception est un objet qui représente le type d'erreur qui peut être soit traité, soit propagé.

Par exemple, le code suivant :

```
1 int [] array = new int[10];
2 int data = array[-1];
```

va créer une exception qui n'est pas explicitement traitée et qui aura comme effet l'arrêt du programme avec ce message :

```
1 Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
    : -1
2         at Main.main(Main.java:10)
```

Regarder la Figure 1 pour mieux comprendre le mécanisme des Exceptions.

2.1 Le block try - catch - finally

Le méchanisme de base pour traiter les exceptions est répresenté par le bloc try - catch - finally.

- `try` : bloc qui contient du code pouvant créer des problèmes ;
- `catch` : en cas de problème, l'exception est capturé par ce bloc.
- `finally` : quel que soit l'execution des niveaux précedents, la partie finally sera excecutée.

L'instruction `finally` n'est pas obligatoire.

```
try{
        //Du code qui peut générer des exceptions
} catch (MyException e1){
        //Du code qui sera executé quand l'exception
        //du type MyException est capturée;
} catch (MyException2 e2){
        //Du code qui sera executé quand l'exception
        //du type MyException2 est capturée;
} finally {
        //Du code qui sera EN TOUT CAS excecuté.
}
```

Exercice 5 Comment peut-on modifier l'exemple avec `array[-1]` en utilisant le bloc try - catch - finally ?

2.2 throw et throws

Il y a la possibilté de programmer la création d'une exception à l'interieur de votre code. Il s'agit de créer un objet de la classe `Exception` (ou bien d'une sous-classe, ou d'une classe personalisée qui hérite de Exception, ...) et de le propager.

```
Scanner sc = new Scanner(System.in);
String result = sc.readLine();

if(result.equals("exception")){
        throw new RuntimeException("Je n'aime pas les exceptions.")
            ;
}
```

Il y a une raison pour laquelle on a choisi de créer un objet `RuntimeException` : il n'est pas obligatoire de traiter toutes les exceptions ! On regarde encore la Figure 1 : les classes qui sont marquées en rouge n'ont pas besoin d'être traitées, alors que pour les autres il faut les gérer.

En tout cas, le block try - catch n'est pas la seule manière de traiter les exceptions : si nécessaire, on peut propager l'exception vers le niveau supérieur. Du coup, on va indiquer que la méthode courante va propager une exception, ce que l'on fait en indiquant l'Exception qui est propagée dans la signature de la méthode :

```
public void boolean monException() throws Exception{
        throw new Exception("Je n'aime pas les exceptions.");
}
```

Exercice 6 Dans la classe Input, modifier la méthode `readInt(String message)` en utilisant le mécanisme des exceptions et le block try - catch ou throws.

2.3 File I/O

Un objet `Scanner` peut prendre en argument un objet de la classe `File`. Un objet `File` est la représentation d'un fichier à l'interieur du système de fichiers.

```
File f = new File("/home/user/Documents/myfile.dat");
Scanner s = new Scanner(f);
```

Dans ce cas là, le constructeur de `s` va propager une exception, qui doit être soit capturée, soit déclarée.

Exercice 7 Créer une classe `InputFromFile` qui hérite de `Input`. Ajouter un constructeur qui prenne en argument un String qui représente le fichier. Si aucune String n'est specifié, `InputFromFile` doit marcher exactement comme `Input`.

Attention : il faudra penser à la gestion des exceptions.

L'écriture dans un fichier textuel peut être implementée en utilisant un objet de type `FileWriter`.

```
1 FileWriter f = new FileWriter(new File("myfile.dat"));
2 f.write("Hello");
3 f.write("Another");
4 f.write("Last\n");
5
6 f.close();
```

Exercice 8 Créer une classe `OutputToFile` qui hérite de `Output` et qui va modéliser l'écriture dans un fichier textuel.