Gestion des exceptions et I/O en Java

mariaiuliana.dascalu@gmail.com

FILS, 2013

- Les exceptions représentent le mécanisme de gestion des erreurs intégré au langage Java.

- Il se compose d'objets représentant les erreurs et d'un ensemble de trois mots clés qui permettent de détecter et de traiter ces erreurs (try, catch et finally) et de les lever ou les propager (throw et throws).

- Ces mécanismes permettent de renforcer la sécurité du code Java.

Nous pouvons éviter cette erreur:

Exemple : une exception levée à l'exécution non capturée

```
1. public class TestException {
2.    public static void main(java.lang.String[] args) {
3.       int i = 3;
4.       int j = 0;
5.       System.out.println("résultat = " + (i / j));
6.    }
7. }
```

Résultat :

```
1. C:>java TestException
2. Exception in thread "main" java.lang.ArithmeticException: /
3. by zero
4.         at tests.TestException.main(TestException.java:23)
```

Signature

Exemple (code Java 1.1) :

```
01. try {
02.     operation_risquée1;
03.     opération_risquée2;
04. } catch (ExceptionInteressante e) {
05.     traitements
06. } catch (ExceptionParticulière e) {
07.     traitements
08. } catch (Exception e) {
09.     traitements
10. } finally {
11.     traitement_pour_terminer_proprement;
12. }
```

Si un événement indésirable survient dans le bloc try, la partie éventuellement non exécutée de ce bloc est abandonnée et le premier bloc catch est traité. Si catch est défini pour capturer l'exception issue du bloc try alors elle est traitée en exécutant le code associé au bloc. Si le bloc catch est vide (aucune instruction entre les accolades) alors l'exception capturée est ignorée.

Exemple

Exemple (code Java 1.1) : erreur à la compil car Exception est traité en premier alors que ArithmeticException est une sous classe de Exception

```
01. public class TestException {
02.     public static void main(java.lang.String[] args) {
03.         // Insert code to start the application here.
04.         int i = 3;
05.         int j = 0;
06.         try {
07.             System.out.println("résultat = " + (i / j));
08.         }
09.         catch (Exception e) {
10.         }
11.         catch (ArithmeticException e) {
12.         }
13.     }
14. }
```

- Il faut faire attention à l'ordre des clauses catch pour traiter en premier les exceptions les plus précises (sous classes) avant les exceptions plus générales. Un message d'erreur est émis par le compilateur dans le cas contraire.

La Classe Throwable

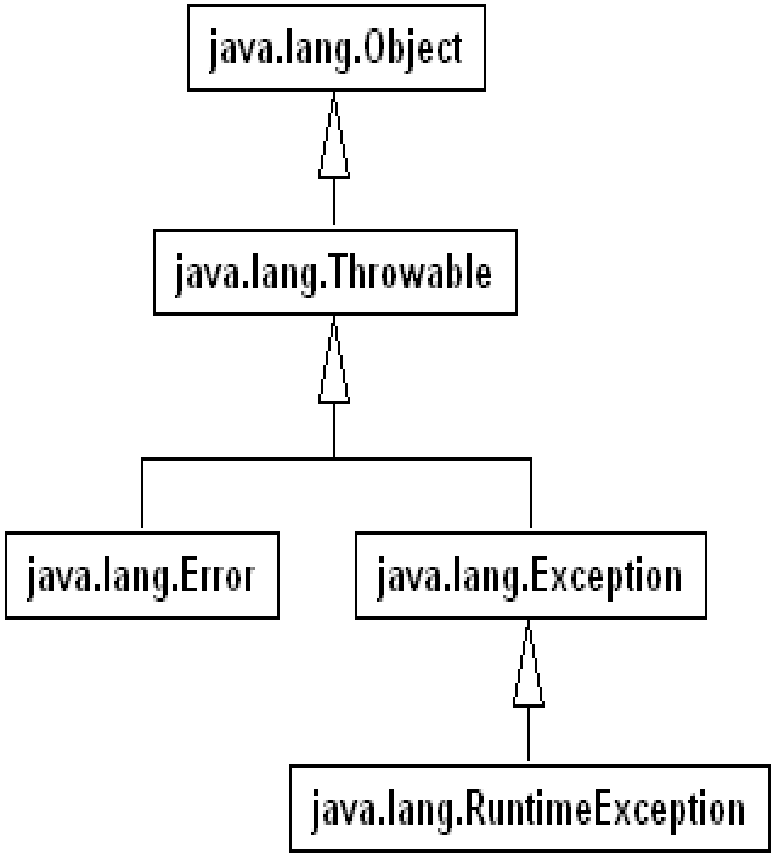
- Cette classe descend directement de la classe Object : c'est la classe de base pour le traitement des erreurs.
- Les principales méthodes de la classe Throwable sont :

Méthodes	Rôle
String getMessage( )	lecture du message
void printStackTrace( )	affiche l'exception et l'état de la pile d'exécution au moment de son appel
void printStackTrace(PrintStream s)	Idem mais envoie le résultat dans un flux

Exemple (code Java 1.1) :

```
01. public class TestException {
02.     public static void main(java.lang.String[] args) {
03.         // Insert code to start the application here.
04.         int i = 3;
05.         int j = 0;
06.         try {
07.             System.out.println("résultat = " + (i / j));
08.         }
09.         catch (ArithmeticException e) {
10.             System.out.println("getmessage");
11.             System.out.println(e.getMessage());
12.             System.out.println(" ");
13.             System.out.println("toString");
14.             System.out.println(e.toString());
15.             System.out.println(" ");
16.             System.out.println("printStackTrace");
17.             e.printStackTrace();
18.         }
19.     }
20. }
```

Les classes Exception, RunTimeException et Error



- La classe Error représente une erreur grave intervenue dans la machine virtuelle Java ou dans un sous système Java. L'application Java s'arrête instantanément dès l'apparition d'une exception de la classe Error.

- La classe Exception représente des erreurs moins graves. Les exceptions héritant de la classe RuntimeException n'ont pas besoin d'être détectées impérativement par des blocs try/catch.

The *java.io package*

- Streams IO sont les moyens par lesquels les programmes Java communiquent avec l'entrée et / ou des dispositifs de sortie tels que le clavier, l'écran, le disque dur, l'imprimante, etc
- Le java.io.package fournit un support pour la lecture des données et l'écriture de / vers différents dispositifs.
- Quatre catégories de classes:

IOStreams are the means by which Java programs communicate with the input and/or output devices such as the keyboard, the display, the hard disk, the printer etc.

The java.io.package provides support for data reading and writing from/to various devices.

Four categories of classes:

1. Input Byte Streams: **BufferedInputStream**, **DataInputStream**, **FileInputStream** e **StringBufferInputStream.** Extensions of the abstract class **InputStream**.

   Input Character Streams: **Reader**, **BufferedReader**, **FileReader** e **StringReader**. Extensions of the abstract class **Reader**

2. Output Byte Streams: **OutputStream**, **FileOutputStream**, **BufferedOutputStream**, **PrintStream** e **StringBufferOutputStream**. Extensions of the abstract class **OutputStream**.

   Output Character Streams: **Writer**, **BufferedWriter**, **FileWriter** e **PrintWriter**. Extensions of the abstract class **Writer**

3. **FileInputStream, FileOutputStream, File** and **RandomAccessFile**.
4. **StreamTokenizer**

Lit ligne par ligne à partir d'un fichier et écrire les lignes d'un autre fichier

```
        try
        {
            // reads line by line from a file and write the lines to another file
            LineNumberReader lnr = new LineNumberReader(new
FileReader("example2in.txt"));
            PrintWriter outFile = new PrintWriter(new BufferedWriter(new
FileWriter("example2out.txt")));
            String s="";
            while ((s=lnr.readLine()) != null)
            {
                outFile.println(lnr.getLineNumber()+" "+s);
            }
            lnr.close();
            outFile.close();
        }
        catch(FileNotFoundException e)
        {
            System.out.println("File Not Found");
        }
        catch (EOFException e)
        {
            System.out.println("End of File");
        }
        catch(IOException e)
        {
            System.out.println("IOException");
        }
```