R2.01 : Object-oriented development (OOD)

Coordinator : Isabelle Blasquez

My name: Cristina Onete

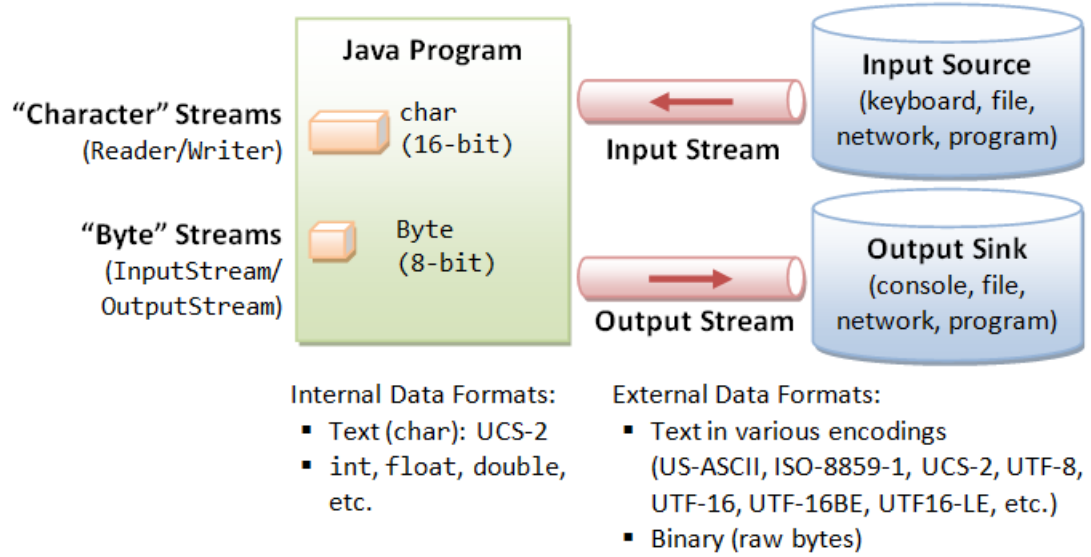
cristina.onete@gmail.com

Slides : https://www.onete.net/teaching.html

Input and output streams

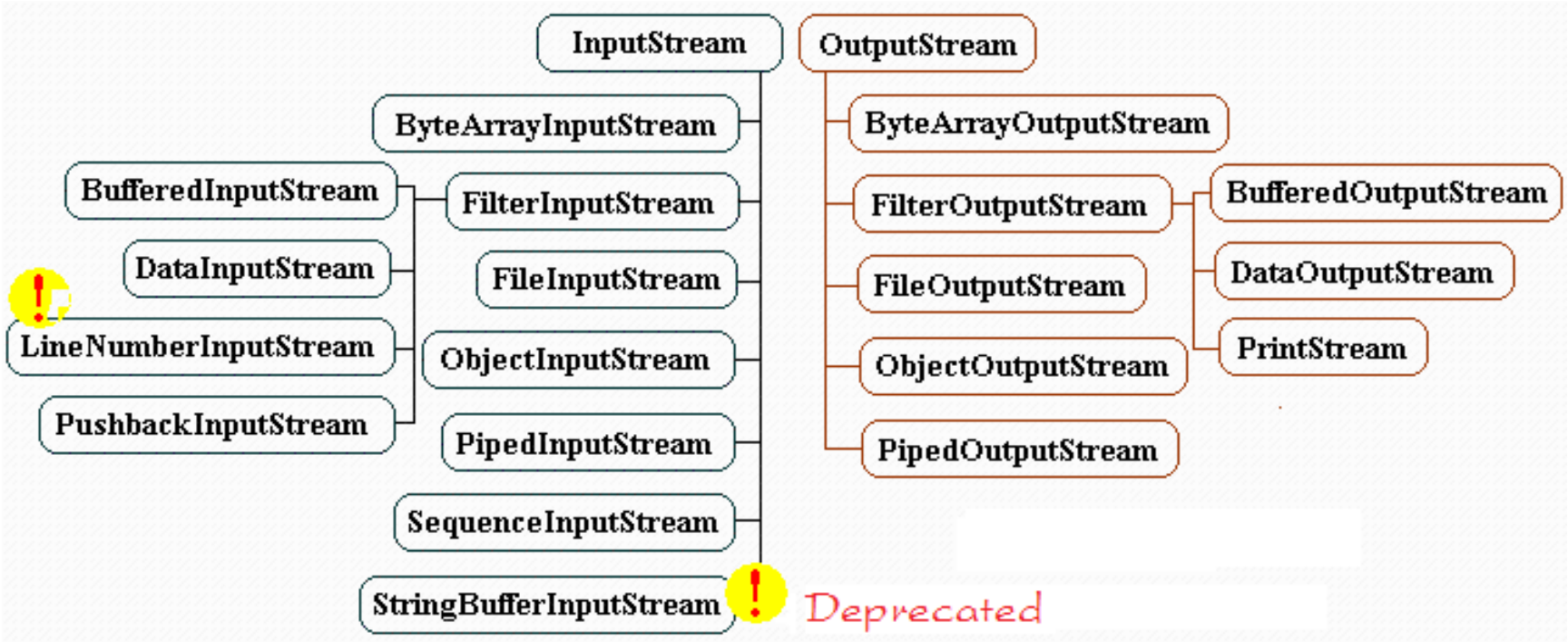
The java.io package

- Input/output **streams** are essential in Java
  - Such streams are unidirectional
  - There are multiple input sources and output destinations in Java
  - Input sources: keyboard input, file, network input, input from another program
  - Output destinations: Java console, file, another program, output on network
- Two types of streams : binary and character streams



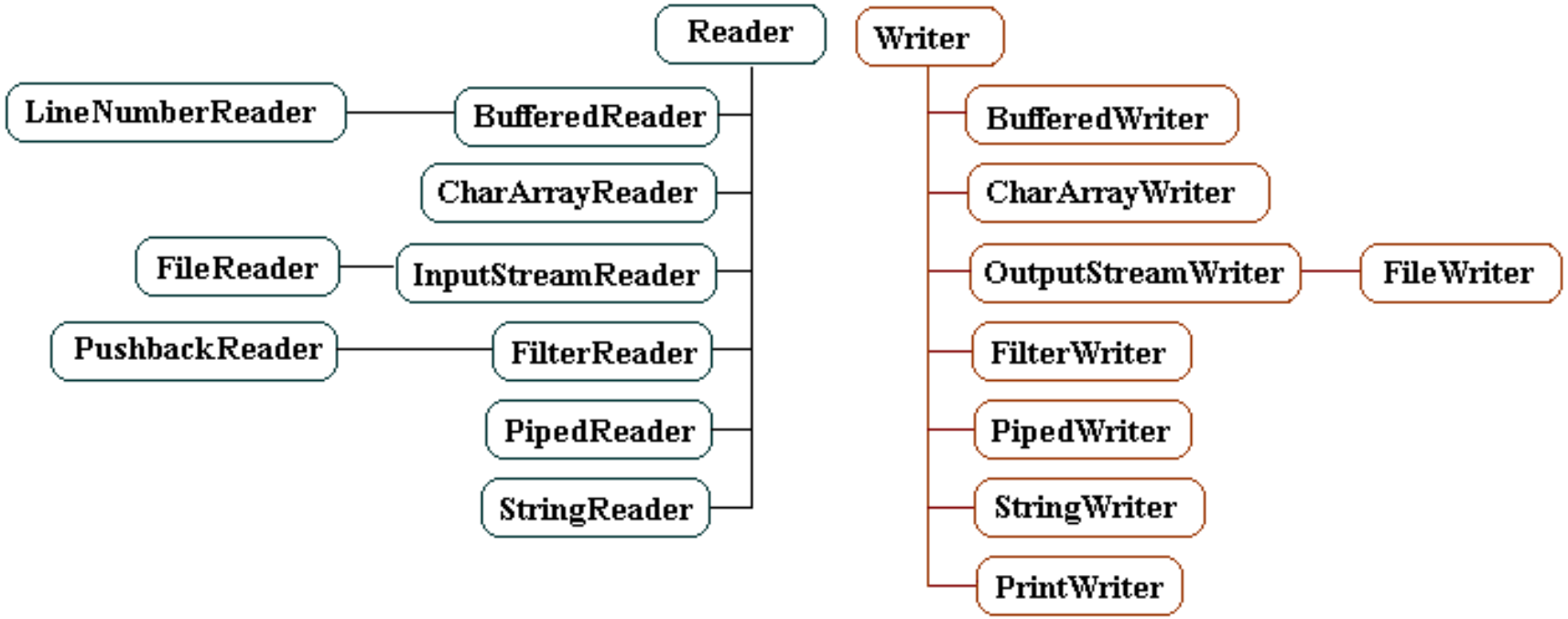
Source :
https://www.ntu.edu.sg

Binary Streams in Java



Source: https://o7planning.org/

Character streams



Source:
https://o7planning.org/

Reading a text file

- A correct manipulation of files in Java involves:
  - Correctly opening files
    - With a correct treatment of possible exceptions, including FileNotFound
  - Manipulation (for instance read/write operations)
  - Correctly closing the files
    - All files **must** be closed after use
    - In particular: even if an error occurs or an exception is triggered, the file must be correctly closed while the execution proceeds!
- Let's look at these steps one by one

Opening and reading a file

- A file can be read with a fileReader
  - We will use the constructor of class FileReader with the signature

    public FileReader(String fileName)

    throws FileNotFoundException
  - The methods in which we instantiate the FileReader must take into account the exceptions!

- Useful methods in class FileReader:
  - public int read(): reads a single character, throws an IOException
  - public int read(char[] buffer) : readers characters from a character array

    better efficiency, throws an IOException

Closing a file

- Always close files when you've finished using them
- 2 ways of doing this:
  - the finally block of a try-catch-finally block is always run
  - Since Java 7 try-catch suffices, due to the interface java.lang.Autocloseable
    - an interface implemented by most readers/writers and I/O streams
    - However, it is still a good idea to concretely close the files

```
import java.io.*;

public class Reading{
    public static void main(String[] args){
        try (FileReader reader = new
FileReader("C:/Documents/File.txt")) {
            // read character by character
            int character;
            while ( (character = reader.read()) != -1  )
                System.out.print((char)character);
        }
        catch(IOException e) {
            e.printStackTrace();
        }
}
```

Optimising reading

- Reading character by character is inefficient
- To improve efficiency we can read windows of several characters at a time
  - We can choose the window size
- For large file, the second method can make all the difference

```
import java.io.*;

public class Reading{
     public static void main(String[] args){
          try{
               FileReader reader = new
FileReader("C:/Documents/File.txt");
               char[] window = new char[128];
               while ( (reader.read(window)) != -1  ){
                    for (int i=0; i<128; i++){
                         System.out.print(window[i]);
                    }
               }
          }
          catch(IOException e) {
               e.printStackTrace();
          }
}
```

Writing to a file

- Use a file writer: FileWriter
- Two interesting FileWriter constructors:
  - public FileWriter(String filename) : throws IOException
  - public FileWriter(String filename, boolean append) : throws IOException
    - if append == true, then the input data is written at the end of the file; oterwise they are written at the start of it
    - very useful in a log or in a file in which order is important
- We can end the line and start on a fresh line by using "\n".

Writing into the file

```
import java.io.*;

public class Writing{
    public static void main(String[] args){
        try{
            FileWriter writer = new FileWriter("C:/Documents/File.txt", true);
            writer.write("I'm continuing to write in this file \n and here is the rest
of my sentence");
        }
        catch(IOException e) {
            e.printStackTrace();
        }
}
```

Of further use...

- The BufferedReader class optimizes reading for files
- Class File

- More information :
  - https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html
  - https://docs.oracle.com/javase/7/docs/api/java/io/File.html

Exceptions

What is an exception?

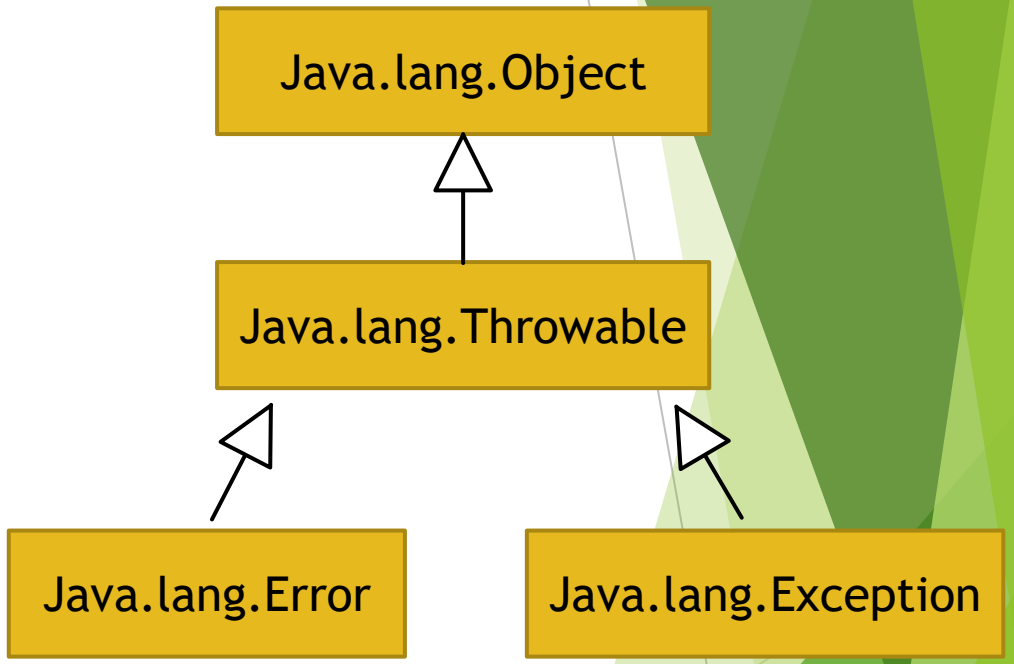
- An exception is an object (instance of a class modelling it)
- An exception is instantiated whenever some event interrupts the normal flow of an algorithm
- An exception can appear when a program halts, or upon some aberrant flow

Some errors can be anticipated and the program, modified to account for them

Others are harder to spot and can lead to premature halts

Different kinds of exceptions

- **Error**:
  - Terminal exception: halts the program entirely
  - Errors signal the existence of a serious flaw and we must let the program halt if they occur
  - VirtualMachineError, OutOfMemory, ...
- **Exception:**
  - Less serious than an error, but can still create some problems
  - Two kinds: exceptions at compilation, exceptions at runtime
  - IOException, SQLException, NullPointerException



Exceptions are thrown...

- If a technical failure interrupts the flow of the program
  - For instance if an index falls out of bounds (in an array)

    IndexOutOfBoundsException
  - Or if the Java virtual machine encounters some error:

    OutOfMemoryError
- If an application failure interrupts the flow of the program
  - For instance if the program tries to access an inexistent file:

    FileNotFoundException

Throwing exceptions

- Keyword: throw
- Raise exception in if statement:

```
if (age > 99){
    throw new RuntimeException("My students
can't be that old!");
}
```

- We can also do it in a "try-catch" block:

```
try{
    // code we ideally want to run
}
catch (FileNotFoundException e) {
    // in case the file doesn't exist, Java will raise an exception
}
```

What's the difference ?

Checked and unchecked exceptions

- Exceptions can be checked or unchecked
- Checked exceptions:
  - Exceptions which Java knows can occur in particular circumstances
    - For instance: when trying to open a file that doesn't exist, Java is able to throw a FileNotFoundException
  - The possibility of such an exception is anticipated by the compiler, which instructs the user to provide for it (typically try-catch)
- Unchecked exceptions:
  - Exceptions raised at runtime, can be caused by some errors or an abnormal program flow
  - Example: `NullPointerException`

Intermezzo: null

Null references

- NullPointerException:
  - Raised when Java stumbles on an object that does not exist
  - For instance the undefined (but existing) n-th element of an array

- How can we prevent this?
  - Check if the object exists: `if(object == null){...}`
  - a `try-catch` block around the code involving the exception

- Careful: null is not an object!
  - We do not use `object.equals(null)` or `null.equals(object)`
  - The latter even yields a `NullPointerException`!

End of Intermezzo

Checked exceptions

- Can be checked within a method directly (in a try-catch block)

```
public FileReader read(String filePath) {
    try{
        return (new FileReader(filePath));
    } catch (FileNotFoundException e){
        //treatment of exception
    }
}
```

- Or we can check it when we use the method (at call-time)

```
public FileReader read(String filePath) throws
FileNotFoundException{
    // ... some code
    return (new FileReader(filePath));
}
```

Try-catch blocks

- Try-catch blocks consist of :
  - An original `try` block, in which we write the code we would like to run
  - A first `catch` block indicating what to do when throwing a first exception
  - We can have multiple `catch` blocks
    - In that case, we are treating exceptions in reverse hierarchy
    - Subclasses before superclasses: `NullPointerException` before `RuntimeException`
- Optionally, we can have a `finally` block:
  - Always executed, even if an exception was thrown previously
  - Allows us to close open processes (for instance if a file is open)

Handy instructions

- System.err.println(String): a method that allows us to print (in red) a text that is meant to be printed when an exception is thrown
- printStackTrace(): method that can be run for any instance of a class that implements the interface Throwable; when called, this method tracks the cause of the error or exception that was thrown

Example: index out of bounds

Check out this code

MonException.java

```
1
2  public class MonException {
3      public static void main(String[] args) {
4          MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
5          int taille = 3;
6          int[] monTableau = new int[taille];
7          int monEntree = 10;
8          monTableau[0] = monEntree;
9          System.out.println("La premiere composante du tableau est " + monTableau[0]);
10         objet.doubler(monEntree);
11         monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
12         System.out.println("La deuxieme composante du tableau est " + monTableau[1]);
13
14     }
15
16     public int augmenter (int index) {
17         System.out.println("La valeur initiale de l'index est " + index);
18         index = index + 1;
19         System.out.println("L'index a ete augmente a " + index);
20         return index;
21     }
22
23     public int doubler (int valeur) {
24         System.out.println("Nous allons doubler la valeur " +valeur);
25         valeur = valeur * valeur;
26         System.out.println("Valeur doublee " + valeur);
27         return valeur;
28     }
29 }
30
```

Lors d'une execution normale

Problems Javadoc Declaration Console

<terminated> MonException [Java Application] C:\Program Files\Java\

```
La premiere composante du tableau est 10
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
La deuxieme composante du tableau est 100
```

Let's add an arror

ErrorProgram/src/MonException.java

```
1
2  public class MonException {
3      public static void main(String[] args) {
4          MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
5          int taille = 1;
6          int[] monTableau = new int[taille];
7          int monEntree = 10;
8          monTableau[0] = monEntree;
9          System.out.println("La premiere composante du tableau est " + monTableau[0]);
10         objet.doubler(monEntree);
11         monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
12         System.out.println("La deuxieme composante du tableau est " + monTableau[1]);
13
14     }
15
```

changed size of array

Problems Javadoc Declaration Console

<terminated> MonException [Java Application] C:\Program Files\Java\jre1.8.0_

```
La premiere composante du tableau est 10
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
Exception in thread "main" L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
java.lang.ArrayIndexOutOfBoundsException: 1
        at MonException.main(MonException.java:11)
```

Throw exception

```
1
2  public class MonException {
3      public static void main(String[] args) {
4          MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
5          int taille = 1;
6          int[] monTableau = new int[taille];
7          int monEntree = 10;
8          monTableau[0] = monEntree;
9          System.out.println("La premiere composante du tableau est " + monTableau[0]);
10         objet.doubler(monEntree);
11         try {
12             monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
13             System.out.println("La deuxieme composante du tableau est " + monTableau[1]);
14         }
15         catch(IndexOutOfBoundsException e) {
16             System.out.println("Taille du tableau depassee !");
17         }
18     }
19
20     public int augmenter (int index) {
21         System.out.println("La valeur initiale de l'index est " + index);
22         index = index + 1:
```

Problems Javadoc Declaration Console

```
<terminated> MonException [Java Application] C:\Program Files\Jav
La premiere composante du tableau est 10
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
Taille du tableau depassee !
```

try-catch block

Java looks out for an IndexOutOfBounds exception

exception is thrown

Tracking down the source of the error

```
2 public class MonException {
3     public static void main(String[] args) {
4         MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
5         int taille = 1;
6         int[] monTableau = new int[taille];
7         int monEntree = 10;
8         monTableau[0] = monEntree;
9         System.out.println("La premiere composante du tableau est " + monTableau[0]);
10        objet.doubler(monEntree);
11        try {
12            monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
13            System.out.println("La deuxieme composante du tableau est " + monTableau[1]);
14        }
15        catch(IndexOutOfBoundsException e) {
16            System.err.println("Taille du tableau depassee !");
17            e.printStackTrace();
18        }
19        finally {
20            System.out.println("Dans le bloc finally");
21        }
22    }
```

Problems Javadoc Declaration Console

```
<terminated> MonException [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Jan 17, 2018, 1:43:26 PM)
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
Taille du tableau depassee !
java.lang.ArrayIndexOutOfBoundsException: 1
        at MonException.main(MonException.java:12)
Dans le bloc finally
```

System.out.println replaced by System.err.println

tracks down error

Tracking down the source of the error

```
2  public class MonException {
3      public static void main(String[] args) {
4          MonException objet = new MonException(); // nous avons besoin d'un objet pour pouvoir utiliser les methodes en bas
5          int taille = 1;
6          int[] monTableau = new int[taille];
7          int monEntree = 10;
8          monTableau[0] = monEntree;
9          System.out.println("La premiere composante du tableau est " + monTableau[0]);
10         objet.doubler(monEntree);
11         try {
12             monTableau[objet.augmenter(0)] = objet.doubler(monEntree);
13             System.out.println("La deuxieme composante du tableau est " + monTableau[1]);
14         }
15         catch(IndexOutOfBoundsException e) {
16             System.err.println("Taille du tableau depassee !");
17             e.printStackTrace();
18         }
19         finally {
20             System.out.println("Dans le bloc finally");
21         }
22     }
```

Problems Javadoc Declaration Console

```
<terminated> MonException [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Jan 17, 2018, 1:43:26 PM)
Nous allons doubler la valeur 10
Valeur doublee 100
La valeur initiale de l'index est 0
L'index a ete augmente a 1
Nous allons doubler la valeur 10
Valeur doublee 100
Taille du tableau depassee !
java.lang.ArrayIndexOutOfBoundsException: 1
        at MonException.main(MonException.java:12)
Dans le bloc finally
```

A finally block

Making our own exceptions

Catching new errors

- Remember this example?

```
if (age > 99){
    throw new RuntimeException("My students
can't be that old!");
}
```

- Here, we are throwing an exception which is not normally speaking an exception: an age larger than 99

- In this example we throw a generic RuntimeException...
  - Could we be more specific and create our own exception ?

Create a new exception

- We can, in fact, create new exceptions in Java, which will inherit from superclasses of exceptions:
  - Checked exceptions inherit from class Exception
  - Unchecked exceptions inherit from class RuntimeException

- The new exception will be a new class, with attributes and methods

GP1: Do not create a new Java exception if you can use existing ones!

Example: StudentTooOld

```
public class StudentTooOldException extends RuntimeException{
    // customized constructor using the superclass constructor
    public StudentTooOld(String m){
        super(m); // we will use the constructor of RuntimeException
    }
}
```

```
public class MainClass{
    public static void main(String[] args) {
        Student anneLeclerc = new Student();
        if (anneLeclerc.getAge() > 99){
            throw new StudentTooOldException("This student is too old!");
        }
    }
 }
```

Other good practices

- Crucial to catch the exceptions potentially touching our code

GP2: Use the best approximation you can have of your exception (subclass rather than Exception/RuntimeException directly)

- Try-catch-finally blocks have special structures

GP3: Do not use catch blocks as regular else blocks in the code!

GP4 : Never raise an exception using a `return` statement

Questions ?