Programmation Orientée Objet – Java

Bertrand Estellon

Département Informatique et Interactions
Aix-Marseille Université

12 novembre 2014

Les exceptions

Un programme peut être confronté à une condition exceptionnelle (ou exception) durant son exécution.

Une exception est une situation qui empêche l'exécution normale du programme (elle ne doit pas être considérée comme un bug).

Quelques exemples de situations exceptionnelles :

- un fichier nécessaire à l'exécution du programme n'existe pas ;
- division par zéro ;
- débordement dans un tableau ;
- etc.

Mécanisme de gestion des exceptions

Java propose un mécanisme de gestion des exceptions afin de distinguer l'exécution normale du traitement de celles-ci afin d'en faciliter leur gestion.

En Java, une exception est concrétise par une instance d'une classe qui étend la classe `Exception`.

Pour lever (déclencher) une exception, on utilise le mot-clé `throw` :

```
if (problem) throw new MyException("error");
```

Pour capturer une exception, on utilise la syntaxe `try/catch` :

```
try { /* Problème possible */ }
catch (MyException e) { /* traiter l'exception. */ }
```

Définir son exception

Il suffit d'étendre la classe `Exception` (ou une classe qui l'étend) :

```
public class MyException extends Exception {

  private int number;

  public MyException(int number) {
    this.number = number;
  }

  public String getMessage() {
    return "Error "+number;
  }

}
```

La syntaxe try/catch

Pour capturer une exception, on utilise la syntaxe try/catch :

```
public static void test(int value) {
  System.out.print("A ");
  try {
    System.out.println("B ");
    if (value > 12) throw new MyException(value);
    System.out.print("C ");
  } catch (MyException e) { System.out.println(e); }
  System.out.println("D");
}
```

`test(11)` :

```
A B
C D
```

`test(13)` :

```
A B
MyException: Error 13
D
```

Exceptions et signatures des méthodes

Une méthode doit préciser dans sa signature toutes les exceptions qu'elle peut générer et qu'elle n'a pas traitées avec un bloc `try/catch` :

```
public class Test {

  public static void runTestValue(int value) throws MyException {
    testValue(value);
  }

  public static void testValue(int value) throws MyException {
    if (value>12) throw new MyException(i);
  }

  public static void main(String args[]) {
    try { runTestValue(13); }
    catch (MyException e) { e.printStackTrace(); }
  }
}
```

Pile d'appels

La méthode printStackTrace permet d'afficher la pile d'appels :

```
public class Test {
 public static void runTestValue(int value) throws MyException {
   testValue(value);
 }

 public static void testValue(int value) throws MyException {
  if (value>12) throw new MyException(value);
 }

 public static void main(String args[]) {
  try { runTestValue(13); } catch (MyException e) {e.printStackTrace();}
 }
}
```

```
MyException: Error 13
        at Test.testValue(Test.java:5)
        at Test.runTestValue(Test.java:2)
        at Test.main(Test.java:9)
```

La classe RuntimeException

Une méthode doit indiquer toutes les exceptions qu'elle peut générer sauf si l'exception étend la classe `RuntimeException` Bien évidemment, la classe `RuntimeException` étend `Exception`.

Quelques classes Java qui étendent `RuntimeException` :

- ArithmeticException
- ClassCastException
- IllegalArgumentException
- IndexOutOfBoundsException
- NegativeArraySizeException
- NullPointerException

Notez que ces exceptions s'apparentent le plus souvent à des bugs.

Capturer une exception en fonction de son type

Il est possible de capturer une exception en fonction de son type :

```
public static int divide(Integer a, Integer b) {
  try { return a/b; }
  catch (ArithmeticException exception) {
    exception.printStackTrace();
    return Integer.MAX_VALUE;
  } catch (NullPointerException exception) {
    exception.printStackTrace();
    return 0;
  }
}
```

`divide(12,0)` :

```
java.lang.ArithmeticException: / by zero
        at Test.diviser(Test.java:17)
        at Test.main(Test.java:28)
```

`divide(null,12)` :

```
java.lang.NullPointerException
        at Test.diviser(Test.java:17)
        at Test.main(Test.java:28)
```

Le mot-clé finally

Le bloc associé au mot-clé `finally` est toujours exécuté :

```
public static void readFile(String fileName) {
  try {
    FileReader fileReader = new FileReader(fileName);
          /* peut déclencher une FileNotFoundException. */
    try {
      int character = fileReader.read(); /* IOException ? */
      while (character !=-1) {
        System.out.println(character);
        character = fileReader.read(); /* IOException ? */
      }
    } finally { fileReader.close(); /* à faire dans tous les cas. */ }
  } catch (IOException exception) { exception.printStackTrace(); }
}
```

`FileNotFoundException` étend `IOException` donc elle est capturée.

Le mot-clé finally

il est toujours possible de capturer les exceptions en fonction de leur type :

```
public static void readFile(String fileName) {
  try {
    FileReader fileReader = new FileReader(fileName);
          /* peut déclencher une FileNotFoundException. */
    try {
      int character = fileReader.read(); /* une IOException ? */
      while (character !=-1) {
        System.out.println(character);
        character = fileReader.read(); /* une IOException ? */
      }
    } finally { /* à faire dans tous les cas. */
          fileReader.close();
    }
  } catch (FileNotFoundException exception) {
    System.out.println("File "+fileName+" not found.");
  } catch (IOException exception) { exception.printStackTrace(); }
}
```

Exemple

```
public class Stack<T> {
  private Object[] stack;
  private int size;

  public Stack(int capacity) {
    stack = new Object[capacity]; size = 0;
  }

  public void push(T object) throws FullStackException {
    if (taille == stack.length) throw new FullStackException();
    pile[taille] = object; taille++;
  }

  public T pop() throws EmptyStackException {
    if (size == 0) throw new EmptyStackException();
    size--; T object = (T)stack[size]; stack[size]=null;
    return object;
  }
}
```

Exemple

Définition des exceptions :

```
public class StackException extends Exception {
    public StackException(String msg) { super(msg); }
}
```

```
public class FullStackException extends StackException {
    public FullStackException() { super("Full stack."); }
}
```

```
public class EmptyStackException extends StackException {
    public EmptyStackException() { super("Empty stack."); }
}
```

Exemple

Exemples d'utilisation :

```
Stack<Integer> stack = new Stack<Integer>(2);
```

```
try {
  stack.push(1);
  stack.push(2);
  stack.push(3);
} catch (StackException e) {
  e.printStackTrace();
}
```

```
FullStackException: Full stack.
        at Stack.push(Stack.java:8)
        at Test.main(Test.java:4)
```

```
try {
  stack.push(1);
  stack.pop();
  stack.pop();
} catch (StackException e) {
  e.printStackTrace();
}
```

```
EmptyStackException: Empty stack.
        at Stack.pop(Stack.java:14)
        at Test.main(Test.java:4)
```