# Chapitre 7: Exceptions

## I. Introduction

Dans un programme, il faut soigner la gestion des erreurs. Ce n'est pas toujours facile avec les langages classiques. Java propose une approche très différente des approches traditionnelles, à travers le mécanisme des exceptions. Une exception est une sorte de signal indiquant qu'une erreur ou une situation anormale a eu lieu.

On dit qu'une méthode ayant détecté une situation anormale déclenche (throws) une exception. Cette exception pourra être capturée (catch) par le code.

On peut distinguer deux types de situations anormales : les exceptions et les erreurs.

Les erreurs sont en principe des erreurs fatales et le programme s'arrête à la suite de ce type de situation (classe java.lang.Error).

Les exceptions ne sont pas uniquement des erreurs systèmes. Le programmeur peut définir des erreurs (non fatales) pour assurer que son programme est robuste (classe java.lang.Exception).

Par exemple, le débordement d'un tableau est une exception.

Lorsqu'une méthode déclenche une exception la JVM remonte la suite des invocations des méthodes jusqu'à atteindre une méthode qui capture cette exception. Si une telle méthode n'est pas rencontrée, l'exécution est arrêtée.

L'utilisation des exceptions permet de :

- séparer le code correspondant au fonctionnement normal d'un programme du code concernant la gestion des erreurs,
- propager de proche en proche les exceptions d'une méthode à la méthode appelante jusqu'à atteindre une méthode capable de gérer l'exception. Il n'est donc pas nécessaire que la gestion d'une exception figure dans la méthode qui est susceptible de déclencher cette exception. Une méthode peut ignorer la gestion d'une exception à condition qu'elle transmette l'exception à la méthode appelante,
- regrouper par types la gestion des exceptions.

## II. Qu'est-ce qu'une exception

C'est un objet de la classe java.lang.Throwable qui est la classe mère de toutes les erreurs et exceptions du langage Java. Seuls les objets qui sont des instances de cette classe (ou d'une classe dérivée) sont déclenchés par la JVM et apparaissent comme arguments d'une clause catch. Nous allons voir ci-après les sous-classes principales de la classe java.lang.Throwable.

- java.lang.Error est la classe des erreurs, qui indiquent un problème grave qui doit conduire à l'arrêt de l'application en cours. On ne demande pas aux méthodes de déclarer une telle erreur dans la clause throws, puisqu'elle n'est pas susceptible d'être capturée. Un certain nombre d'erreurs dérivent de cette classe, par exemple OutOfMemoryError, et d'autres...
- java.lang.Exception est la classe des exceptions qui indiquent qu'une application devrait raisonnablement les capturer, c'est-à-dire traiter ces cas de situations anormales, sans arrêter le programme. Voici des exceptions classiques qui dérivent de cette classe : java.io.IOException, FileNotFoundException,et bien d'autres... A chaque objet de la classe java.lang.Exception(ou d'une classe dérivée) est associé un message que l'on peut récupérer avec la méthode getMessage() de la classe java.lang.Throwable
- RuntimeException est une classe dérivée de la précédente, et c'est la classe mère des exceptions qui peuvent être déclenchées au cours de l'exécution d'un programme. Supposons qu'une méthode soit susceptible de lever une exception de type RuntimeException, il n'est pas obligatoire de le signaler dans sa clause throws. En effet, les exceptions de type RuntimeException peuvent être levées mais ne pas être capturées, générant ainsi un arrêt du programme. Voiciq uelques exemples de sous-classes de la classe RuntimeException:
  - ArrayStoreException,
  - ArithmeticException,
  - NullPointerException,
  - NumberFormatException...

## 1. Capturer une exception

On l'a dit précédemment, lorsqu'une exception est lancée, elle se propage dans la pile des méthodes jusqu'à être capturée. Si elle ne l'est pas, elle provoque la fin du programme, et la pile des méthodes traversées est indiquée à l'utilisateur.

Supposons qu'une instruction instr d'une méthode uneMethode lance une exception alors :

- si instr se trouve dans un bloc **try**, suivi d'un bloc **catch** alors,
  1. les instructions du bloc try suivant instr ne sont pas exécutées,
  2. les instructions du bloc catch sont exécutées,
  3. le programme reprend son exécution normalement avec l'instruction suivant le bloc catch.
- si instr ne se trouve pas dans un bloc try comme décrit précédemment, alors la méthode uneMethode est terminée. Si uneMethode est la méthode main, le programme se termine, et l'exception n'a pas été capturée. Sinon, on se retrouve dans une méthode qui a appelé la méthode uneMethode via une instruction instr2 qui lance à son tour l'exception.

Une méthode susceptible de lancer une exception sans la capturer doit l'indiquer dans son entête avec la clause throws. Cependant, comme précisé précédemment, on est dispensé de déclarer le lancement des erreurs les plus courantes, comme parexemple :

- ArrayOutOfBoundsException,
- ArrayStoreException,
- ArithmeticException,
- NullPointerException,
- NumberFormatException...

**Exemple :**

```
class AttrapExcep{
    static int moyenne(String[] liste) {
      int somme=0, entier, nbNotes=0;
      for (int i=0;i<liste.length;i++) {
          try{
              entier=Integer.parseInt(liste[i]);
              somme+=entier;
              nbNotes++;
          }
          catch(NumberFormatException e) {
              System.out.println("La "+(i+1)+"ième note pas entière");
          }
    }
    return somme/nbNotes;
}
public static void main(String [] arg) {
    System.out.println("La moyenne est :"+moyenne(arg));
  }
}
```

Voici quelques exemples d'exécution du programme précédent :

```
chaouiya/GBM2/coursJava/Notes_cours$ java AttrapExcep 5 b 10
La 2ième note n'est pas un entier
La moyenne est :7

chaouiya@pccc:~/GBM2/coursJava/Notes_cours$ java AttrapExcep 5 10 15
La moyenne est :10

chaouiya@pccc:~/GBM2/coursJava/Notes_cours$ java AttrapExcep 5 10 15 n
La 4ième note n'est pas un entier
La moyenne est :10
```

```
chaouiya@pccc:~/GBM2/coursJava/Notes_cours$ java AttrapExcep 10.5 xx
La 1ième note n'est pas un entier
La 2ième note n'est pas un entier
java.lang.ArithmeticException: / by zero
        atAttrapExcep.moyenne(AttrapExcep.java:14)
        atAttrapExcep.main(AttrapExcep.java:17)
```

### 2. Définir de nouveaux types d'exceptions

Les exceptions sont des objets d'une classe dérivée de java.lang.Exception.

Si l'on veut signaler un évènement inattendu, non prévu par l'API de Java, il faut dériver la classe Exception et définir une nouvelle classe qui ne contient en général pas d'autre champ qu'un ou plusieurs constructeur(s) et éventuellement une redéfinition de la méthode toString. Lors du lancement d'une telle exception, on crée une instance de cette nouvelle classe.

### Exemple :

```
Class ExceptionRien extends Exception {
    public String toString() {
        return("Aucune note n'est valide'\n");
    }
}
```

### 3. Lancer et capturer une exception

Rien ne vaut un exemple, reprenons celui de I.Charon:

```
Class ExceptionThrow {
  Static int moyenne(String[] liste) throws ExceptionRien {
      int somme=0,entier, nbNotes=0;
      int i;
      for (i=0;i <liste.length;i++) {
          try{
              entier=Integer.parseInt(liste[i]);
              somme+=entier;
              nbNotes++;
          }
          catch (NumberFormatException e){
              System.out.println("La "+(i+1)+" eme note n'est "+
                                  "pas entiere");
      }
}
      if (nbNotes==0) throw new ExceptionRien();
      return somme/nbNotes;
}
public static void main(String[] argv) {
      try {
          System.out.println("La moyenne est "+moyenne(argv));
      }
      catch (ExceptionRien e) {
              System.out.println(e);
      }
}
```

### 4. Blocs finally

La clause finally est en général utilisée pour "faire le ménage" (par exemple fermer les fichiers, libérer les ressources, ...). Un bloc finally est utilisé en association avec un bloc try. On sort d'un bloc try par une instruction break ou returnou continue ou par une propagation d'exception. Un bloc finally suit un bloc try suivi, en général, d'un bloc catch. Dans tous les cas, quelque soit la façon dont onest sorti du bloc try, les instructions du bloc finally sont exécutées.

Voici un exemple, toujours tir´e du support de cours d'Irène Charon, qui n'ad'autre objectif que d'illustrer l'effet du bloc finally:

```
Class MonException extends Exception {
    MonException() {
        System.out.println("me voila");
    }
}
class Propagation {
    static booleanprobleme=true;

static void methodeBasse() throws MonException {
    try {
        if (probleme) throw new MonException();
        System.out.println("et moi ?");
    }
    finally {
        System.out.println("hauteur basse : il faudrait etre ici");
    }
    System.out.println("pas mieux");
}
Static void methodeMoyenne() throws MonException {

    try {
        methodeBasse();
        System.out.println("et ici ?");
    }
    finally {
        System.out.println("moyenne hauteur : ou bien etre la");
    }
}
Static void methodeHaute() {
    try {
        methodeMoyenne();
    }
    Catch (MonException e) {
        System.out.println("attrape...");
    }
}
static public void main(String[] argv) {
    methodeHaute();
}
}
```