

Errata 1

Les exceptions

« Attrapez-les tous, sans exception ! »

1.1 Motivation

Un programme ne tourne pas dans un monde idéal.

Il doit pouvoir *résister aux défaillances* de l'environnement, telles que

- ▷ On tente d'ouvrir un fichier qui n'existe pas
- ▷ L'utilisateur entre des données incorrectes
- ▷ Une connexion à un site web ne se fait pas
- ▷ etc.

Tous ces événements *exceptionnels* viennent perturber le fonctionnement de votre programme, et doivent être gérés. Le langage Java offre un mécanisme, celui des exceptions.

À titre d'exemple, considérons le programme suivant

```
1 import java.util.Scanner;
  public class Affiche {
    /**
4     * Affiche un nombre entier lu au clavier.
    * @param args non utilisé
    */
7     public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        int nb;
10
        nb = clavier.nextInt();
        System.out.println(nb);
13    }
  }
```

java

Que se passe-t-il lorsque l'utilisateur entre une lettre, ou un nombre décimal ?

```
> javac Affiche.java
> java Affiche
a
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:857)
    at java.util.Scanner.next(Scanner.java:1478)
    at java.util.Scanner.nextInt(Scanner.java:2108)
    at java.util.Scanner.nextInt(Scanner.java:2067)
    at Affiche.main(Affiche.java:7)
```

Nom de l'exception

Pile d'appels (par où il est passé)

On constate qu'une Exception a été levée (ou « lancée »), ce qui résulte en l'arrêt prématuré du programme.

1.2 L'instruction try-catch

On l'a vu, une Exception peut arrêter l'exécution du programme. Cependant, le mécanisme autorise aussi d'intercepter (ou « attraper ») les Exceptions, afin de traiter l'erreur de façon transparente pour l'utilisateur. On utilise à cet effet une instruction try-catch, constituée de deux blocs de code :

- ▷ try : contient les instructions qui *peuvent mal se passer*
- ▷ catch : contient le code qui est *en charge* de gérer le problème

Quand une exception est levée dans un try

- ▷ Le code du try est interrompu
- ▷ Le code du catch est exécuté
- ▷ L'exécution reprend après le bloc try-catch

À titre d'exemple, on peut ainsi intercepter les exceptions levées par nextInt() comme ceci :

```
1 import java.util.Scanner;
   public class Affiche {
       /**
4      * Affiche l'entier lu au clavier, ou un message si ce n'est pas
        un entier.
        * @param args inutilisé.
        */
7      public static void main(String[] args) {
          Scanner clavier = new Scanner(System.in);
          int nb;
10         try {
            nb = clavier.nextInt();
            System.out.println(nb);
13         }
            catch(Exception e) {
16         System.out.println("Ce n'est pas un entier!");
            }
        }
    }
```

java

Afin d'être plus spécifique, on peut nommer dans le `catch` l'exception précise à intercepter, par exemple `InputMismatchException`.

1.3 L'instruction throw

Imaginons la situation où un programme doit demander un *entier positif* à l'utilisateur. Si le programme ne vérifie pas immédiatement que le nombre donné est entier et positif, il s'expose à des problèmes tels que plantages, résultats erronés, perte ou corruption de données, etc. avec la condition aggravante que le problème pourrait ne se signaler que plus loin dans le code, et plus tard dans le temps, rendant le débogage plus difficile. Cette constatation est à la base du mantra « Fail early, fail loudly. ».

Cas pratique : vérifier les *arguments* reçus. Si un paramètre est supposé être un entier positif, une saine gestion consiste à vérifier que c'est le cas et à lever une exception sinon.

```
/**
 * Calcule la racine carrée d'un nombre.
3  * @param nb le nombre dont on veut la racine carée.
 * @return la racine carrée de <code>nb</code>.
 * @throws IllegalArgumentException si <code>nb</code> est
   négatif.
6  */
public static double racineCarrée(double nb) {
    if (nb<0) {
9        throw new IllegalArgumentException("nb_doit_être_positif!");
    }
    // Traitement normal. On est sûr que le paramètre est OK.
12 }
}
```

java

```
try {
    System.out.println( racineCarrée( val ) );
3 } catch (Exception ex) {
    System.out.println( "Calcul_impossible!" );
}
```

java

Comme annoncé précédemment, on peut aussi préciser qu'on n'intercepte *que* les `IllegalArgumentException`

```
1 try {
    System.out.println( racineCarrée( val ) );
} catch (IllegalArgumentException ex) {
4    System.out.println( "Calcul_impossible!" );
}
```

java