Programmation Orientée Objet Java

Exceptions

v1.0 - 28.02.2023

Plan

- Introduction
- Types d’exception
- Mécanisme d’exception
- Syntaxe Java
- Bonnes pratiques

Introduction

Introduction

Une exception est un **évènement** qui se produit à l'exécution de votre programme et qui va provoquer l'arrêt de l'enchaînement "normal" de votre code.

Il va être possible d'intercepter (catcher) ces exceptions afin :

- D'empêcher que notre application ne plante
- De réaliser un traitement particulier

La **gestion d'exceptions** permet :

- de détecter une anomalie et de la traiter indépendamment de sa détection,
- de séparer la gestion des anomalies du reste du code.

Une anomalie peut être due, par exemple, à des données incorrectes, à une fin de fichier prématurée, à un événement non prévu par le programmeur.

Exception : Avantages

Les exceptions permettent d'intercepter des erreurs d'exécution de notre code et de pouvoir proposer une solution alternative.

Sans exception, notre application aurait planté !

<u>3 avantages :</u>

1. Séparation du code et code de traitement d'erreur (bloc **catch**)
2. Remontée de l'erreur dans la stack d'exécution du code
3. Groupe et différencie les types d'erreur

Exception : Inconvénients

Bien que les exceptions soient un moyen efficace de détecter des erreurs, c'est un mécanisme lent.

L'interception d'exception et son traitement ralentissent l'exécution d'un programme Java.
=> Il est préférable (si possible) d'anticiper une erreur d'exécution et de le prévoir plutôt que de laisser une exception se déclencher.

Types d’exception

Exceptions standards

Les causes de déclenchement d'exception peuvent être multiples.
Quelques cas classiques :

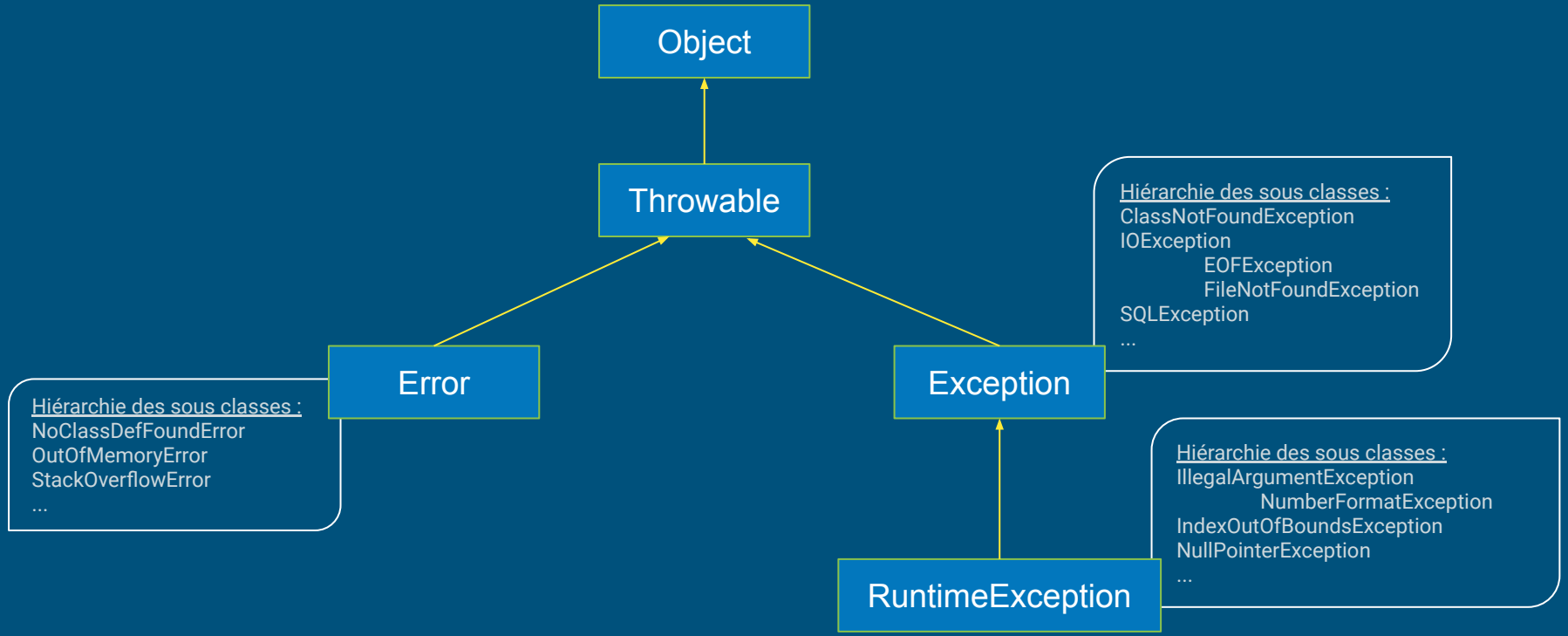
- L'invocation d'une méthode ou d'un attribut sur un objet *null* => java.lang.NullPointerException
- La conversion d'une chaîne de caractères de chiffres en nombre => java.lang.NumberFormatException
- L'accès à un fichier inexistant => java.io.FileNotFoundException
- L'accès à un élément d'un tableau dont l'indice est hors des bornes du tableaux => java.lang.IndexOutOfBoundsException
- …

Types d’exception

En Java, il existe 3 types d’exception :

- Error
  Ce type modélise les erreurs d'exécution que l'on ne gère en général pas.
  Exemple : les divisions par zéro, les dépassements de capacité des tableaux, la saturation de la mémoire...
- Exception (=checked Exception)
  Ce type d’exception concerne les exceptions “prévisibles”
  Ce type d’exception doit obligatoirement être *catché* ou *throwé*
- RuntimeException (=unchecked Exception)
  Ce type d’exception concerne les exceptions “imprévisibles”
  Elles sont généralement liées à un bug !

Les exceptions et l'héritage



Exception : quelques méthodes

En Java, les exceptions possèdent plusieurs méthodes permettant de connaître plus précisément la cause de l'exception :

- getCause() : renvoie la cause ou null (si inconnue) de l'exception
- getMessage() et getLocalizedMessage() : renvoient une description de l'exception
- getStackTrace() : permet de récupérer la stacktrace* d'exécution
- printStackTrace() : permet d'afficher la stacktrace* d'exécution dans la console
- toString() : renvoie une courte description de l'exception

* Stacktrace : Pile d'appel de l'exécution d'un programme

Exception : Exemple

```
BufferedReader  br = new BufferedReader(new InputStreamReader(System.in));
try {
   input = br.readLine();
}
catch (IOException e) {  // Exception possible
   System.out.println("Lecteur ligne impossible !");
}
finally {
   try {
      if (br != null)  { br.close(); }
   }
   catch (IOException e) {
      e.printStackTrace(); // affiche la stacktrace dans la console
   }
}
```

Mécanisme d’exception

Exception : Où ? Propagation ?

Lorsqu'une erreur se produit dans une méthode, un objet **Exception** est créé.

Une exception peut se propager si elle n'est pas interceptée, voici comment :

1. Une exception est générée à l'intérieur d'une méthode
2. Si la méthode prévoit un traitement de cette exception, on va au point 4, sinon au point 3
3. L'exception est renvoyée à la méthode ayant appelé la méthode courante, on retourne au point 2
4. L'exception est traitée et le programme reprend son cours après le traitement de l'exception

Pourquoi pas nos propres exceptions ?

Les Exceptions sont des classes dont on peut hériter afin de définir nos propres exceptions

```java
package fr.esgi.poo.exception;

/**
* MonException permet de catégoriser mon type d'Exception
*/
public class MonException extends Exception {

}
```

Syntaxe Java

Implémentation en Java

5 mots clés en Java sont utilisés pour gérer les exceptions :

- try
- catch
- finally
- throws
- throw

```
try {

   // Code exécuté

}
catch (Exception e) {
   // Code exécuté si Exception levée
}
finally {
   // Code exécuté dans tous les cas
}
```

Mot clé : try

try sert à déclarer un bloc de code où sera gérée l'interception d'exception.

```
try {
   myInt = Integer.parseInt("123");

   // Code exécuté si pas d'exception levée
}
catch (NumberFormatException nfe) {
   // Code exécuté si exception NumberFormatException levée
}

// Code exécuté dans tous les cas
```

Mot clé : catch

catch sert à déclarer le bloc de code d'interception de l'exception.

Dans le cas où plusieurs blocs catch sont définis et qu'une exception se produit, c'est le code du premier bloc catch correspondant à l'exception qui est exécuté.

L'ordre des blocs catch est important : il doit être du catch le plus spécialisé au plus général

```
try {
   myInt = Integer.parseInt("123");

   // Code exécuté si pas d'exception levée
}
catch (NumberFormatException nfe) {
   // Code exécuté si exception NumberFormatException levée
}

// Code exécuté dans tous les cas
```

Mot clé : catch

Le bloc catch peut être optionnel dans la gestion des exceptions si un bloc finally est présent.
Pas deux fois la même exception “catchée”.

Plusieurs blocs catch peuvent être définis pour un même try (de l’exception la plus spécifique à la plus générale).

```
try {
   // Code exécuté
}
catch (NumberFormatException nfe) {
   // Code exécuté si NumberFormatException levée
}
catch (ParseException pe) {
   // Code exécuté si ParseException levée
}
catch (Exception e) {
   // Code exécuté si Exception
}
```

Mot clé : catch

Un bloc catch (depuis **Java 7**) peut catcher plusieurs exceptions à la fois.
L'intérêt est d'avoir un code commun (un seul catch) pour plusieurs types d'exceptions différentes.
Pour cela, il faut définir les types d'exception séparées par un "|".

```
try {
   // Code exécuté
}
catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
   // Code exécuté si NumberFormatException ou ArrayIndexOutOfBoundsException levée
}
```

TD05.01-Exception : try..catch

Exercice :

1. Dans votre IDE, créez un nouveau projet Java nommé par exemple “ExceptionTryCatch”
2. Définissez un nom de package
3. Créez une classe et y ajouter la méthode public static void main(String[ ] args) mais laissez-la vide
4. Créez une autre classe nommée IntArray qui aura :
   a. 1 attribut “values” de type int[ ]
   b. 1 constante représentant la taille max du tableau à 5
   c. 1 constructeur permettant d’initialiser les valeurs du tableau int[0] = 0, int[1] = 1, ...
   d. 1 méthode display() qui affichera les valeurs du tableau dans la console à l’aide d’itération sur le tableau dont la taille est fourni par values.length
5. Créez une méthode displayWithException() qui va lister les valeurs du tableau à l’aide d’une boucle infinie et en utilisant le mécanisme d’exception (ArrayIndexOutOfBoundsException)
6. Testez ces méthodes dans le main

Mot clé : finally

finally sert à déclarer le bloc de code qui sera exécuté après que l'exception à laquelle il est rattaché se soit déclenchée ou pas.

Ce bloc finally est toujours exécuté si un bloc try lui est associé.

```
try {

   // Code exécuté

}
catch (Exception e) {
   // Code exécuté si Exception levée
}
finally {
   // Code exécuté dans tous les cas
}
```

Mot clé : finally

La fonction principale d'un bloc finally est de permettre quelque soit l'exécution du code du bloc try de "nettoyer" le code.

Le bloc finally est toujours placé à la fin du bloc try et après les blocs catch s'il y en a.

Le bloc finally est très souvent utilisé avec les IOException (exception d'Entrée/Sortie)

*nettoyer = fermer les flux (fichier, réseau), réinitialiser des valeurs, ...

TD05.02-Exception : try..catch..finally

Exercice :

1. Dans votre IDE, créez un nouveau projet Java nommé par exemple “ExceptionTryCatchFinally”
2. Définissez un nom de package
3. Créez une classe et y ajouter la méthode public static void main(String[ ] args) mais laissez-la vide
4. Créez une autre classe nommée Divider qui aura :
   a. 2 attributs “dividende” et “diviseur” de type int
   b. 1 constructeur permettant d’initialiser ces attributs
   c. 1 méthode divide() qui affichera le résultat de la division du dividende par le diviseur.
5. Instanciez plusieurs objets Divider et affichez la division
6. Faites le test en définissant la valeur 0 au diviseur. Que se passe t’il ?
7. Corrigez votre code sans Exception
8. Corrigez votre code avec la gestion de l’exception retournée qui affichera un message dans la console
9. Rajoutez un bloc finally qui affichera un autre message

Mot clé : throws

throws sert à déclarer qu'une ou plusieurs exceptions peuvent être renvoyées par une méthode.

throws doit donc être déclaré dans la signature de la méthode après les parenthèses.

```
/**
 * Le throws est optionnel mais permet d'indiquer que la méthode peut renvoyer l'exception de type NumberFormatException
 * @throws NumberFormatException
 */
public void methodWithThrowException() throws NumberFormatException {
   int value17 = Integer.parseInt("XVII"); // KO => NumberFormatException
   System.out.println("value17 : " + value17);
}
```

Mot clé : throw

throw sert à déclencher une exception.

```
public void methodWhichThrowsException() {
    int value123 = Integer.parseInt("123"); // OK
    System.out.println("value123 : " + value123);

    throw new NumberFormatException("Déclenche volontairement une NumberFormatException");
}
```

TD05.03-Exception : Exception custom et throw

Exercice :

1. Dans votre IDE, créez un nouveau projet Java nommé par exemple “MyOwnException”
2. Définissez un nom de package
3. Créez une classe et y ajouter la méthode public static void main(String[ ] args) mais laissez-la vide
4. Créez une autre classe nommée PositiveNumber qui aura :
   a. 1 attribut “value” de type int
   b. 1 méthode setValue(int v) qui affectera la “value” si et seulement si v est un nombre positif et renverra une NotPositiveNumberException sinon
5. Créez une autre classe nommée NotPositiveNumberException qui héritera de Exception
6. Dans la méthode main, instanciez plusieurs objets PositiveNumber et affectez leurs des valeurs positives et négatives
7. Vérifiez que votre code fonctionne bien, que la levée d’exception se fait correctement et qu’elle est bien interceptée dans le main

TD05.04-Exception et Constructeur

Exercice :

1. Dans votre IDE, créez un nouveau projet Java nommé par exemple “ExceptionConstructeur”
2. Définissez un nom de package
3. Créez une classe et y ajouter la méthode public static void main(String[ ] args) mais laissez-la vide
4. Créez une autre classe nommée EvenNumber qui aura :
   a. 1 attribut “value” de type int
   b. 1 constructeur prenant un int et affectera l’attribut “value” avec ce int s’il est pair et renverra une Exception sinon
   c. 1 constructeur prenant un int (et un boolean par exemple) et affectera l’attribut “value” avec ce int s’il est pair et renverra une RuntimeException sinon
   d. 1 accessor get pour l’attribut “value”
5. Dans la méthode main, instanciez un objet EvenNumber avec différentes valeurs et constatez le fonctionnement du programme
   Que constatez-vous ?

Bonnes pratiques

A retenir

- Ne jamais ignorer une exception
  → Ne jamais laisser un catch vide. Au moins afficher un message !
- Utiliser la clause throws de manière exhaustive
  → Renvoyer toutes les exceptions nécessaires
- Les exceptions ne sont pas faites pour le contrôle de flux
  → Cela n'est pas fait pour dérouter l'exécution de votre programme
- Les exceptions et les entrées/sorties
  → Indispensable pour blinder le code et libérer les ressources
- Attention au return dans un bloc finally !
  → Car le bloc finally est toujours exécuté
- Utiliser les exceptions standards
  → Ne réinventez pas des exceptions déjà existantes dans Java

Questions/Réponses