Exceptions

Alexis Nasr (d'après les slides de Arnaud Labourel)

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 toujours être considérée comme un bug). Quelques exemples de situations exceptionnelles :

- un fichier nécessaire à l'exécution du programme n'existe pas,
- une division par zéro,
- un débordement dans un tableau,
- un besoin de se connecter à un serveur et celui-ci est injoignable,
- un dépilement d'une pile vide,
- ...

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 concrétisée 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 les mots-clés `try` et `catch` :

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

Exemple d'exceptions existantes en java

- `ArithmeticException` : opération arithmétique impossible comme la division par 0.
- `IndexOutOfBoundsException` : dépassement d'indice dans un tableau, un vecteur, ...
- `NullPointerException` : accès à un attribut/méthodes/case pour les tableaux d'une référence valant `null`, argument `null` alors que ce n'est pas autorisé.
- `FileNotFoundException` : échec de l'ouverture d'un fichier à partir d'un chemin.
- `IllegalArgumentException` : argument incorrect (en dehors des valeurs autorisées) lors de l'appel d'une méthode.
- `NoSuchElementException` : `next` alors que l'itération est finie, dépilement d'une pile vide, ...
- ...

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)`	`test(13)`
A B C D	A B MyException: Error 13 D

Exceptions et signatures des méthodes

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

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

La méthode `testValue` peut lever une exception de type `MyException`.

`runTestValue` doit indiquer qu'elle peut lever une exception car elle ne gère pas l'exception provoquée par l'appel `testValue(value)`.

Gestions des exceptions : règle

La méthode `runTestValue` peut lever une exception (de type `MyException`).

Lorsqu'on fait un appel à la méthode `runTestValue`, il est vérifié à la compilation que l'une des deux propriétés suivante est vraie :

- la méthode appelant `runTestValue` est indiquée comme pouvant lever l'exception `MyException` (en écrivant `throws MyException` à la signature de la méthode).
- l'exception potentielle est capturée par un loc `try/catch`.

Si aucune des deux propriétés est vérifiée alors il y a une erreur à la compilation.

```
Error:(YY, XX) java: unreported exception MyException;
      must be caught or declared to be thrown
```

Exceptions et signatures des méthodes

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

On doit donc écrire :

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

Techniquement la méthode `main` pourrait indiquer qu'elle génère l'exception `MyException` mais cela n'aurait pas beaucoup de sens car cela voudrait dire qu'on ne gère pas vraiment l'exception.

Méthode printStackTrace

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

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

```
MyException: Error 13
    at Test.testValue(Test.java:23)
    at Test.runTestValue(Test.java:20)
    at Main.main(Main.java:6)
```

Exceptions pour des piles (1/2)

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

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

Exceptions pour des piles (2/2)

```
public class Stack<T> {
  public void push(T object) throws FullStackException {
    if (size == stack.length)
      throw new FullStackException();
    stack[size] = object;
    size++;
  }
  public T pop() throws EmptyStackException {
    if (size == 0) throw new EmptyStackException();
    size--;
    T object = (T)stack[size];
    stack[size]=null;
    return object;
  }
}
```

Définition des exceptions pour les piles

```
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 d'utilisation (1/2)

```
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:13)
    at Main.main(Main.java:10)
```

Exemple d'utilisation (2/2)

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

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

```
EmptyStackException: Empty stack.
    at Stack.pop(Stack.java:18)
    at Main.main(Main.java:10)
```

La classe `RuntimeException`

Une méthode doit indiquer toutes les exceptions qu'elle peut lever 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`
- `NoSuchElementException`

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

Description de la classe `RuntimeException` dans la documentation Java

`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

`RuntimeException` and its subclasses are unchecked exceptions. Unchecked exceptions do not need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.

Règle générale : ajout cas `RuntimeException`

Lorsqu'on fait un appel à une méthode `canThrow` pouvant lever une exception `MyException` qui n'étend pas `RuntimeException`, il est vérifié à la compilation que l'une des deux propriétés suivantes est vraie :

- la méthode appelant `canThrow` dans son code est indiquée comme pouvant lever une exception de type `MyException` ou une de ses super-classes (en écrivant `throws MyException` à la signature de la méthode).
- l'exception potentielle est capturée par un bloc `try/catch`.

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`

On rajouter un bloc `finally` après des blocs `try`. Le bloc associé au mot-clé `finally` est toujours exécuté :

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