Exceptions

Programmation Orientée Objet

Jean-Christophe Routier
Licence mention Informatique
Université Lille 1

Université de Lille 1 SCIENCES ET TECHNOLOGIES
UFR IEEA Formations en Informatique de Lille 1

Principe :

- détecter le maximum d'erreurs à la compilation, mais... pas toujours possible

Gestion des erreurs à l'exécution :

- par une valeur de retour
  ↪ difficile à traiter
- mécanisme de gestion des "**exceptions**"
  ↪ signaler là où le problème se pose dans le code
  ↪ séparer le traitement des cas "normaux" de celui des cas exceptionnels
  ↪ les cas problématiques sont transmis au **gestionnaire d'exceptions** (*exception handler*)

Qu'est ce qu'un cas exceptionnel ?

- une situation qui ne correspond au "**fonctionnement normal**" du programme

diviser un nombre par 0	`ArithmeticException`
envoyer un message sur une référence `null`	`NullPointerException`
accéder à des cases d'un tableau en dehors des indices,	`ArrayIndexOutOfBoundsException`
réaliser un transtypage impossible	`ClassCastException`
tenter de référencer en lecture un fichier qui n'existe pas	`FileNotFoundException`
etc.	etc.

extrait de la javadoc classe `java.lang.String` :

substring
`public String substring(int beginIndex)`

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:
`"unhappy".substring(2)` returns `"happy"`
`"Harbison".substring(3)` returns `"bison"`
`"emptiness".substring(9)` returns `""` (an empty string)

Parameters:
`beginIndex` - the beginning index, inclusive.

Returns:
the specified substring.

Throws:
`IndexOutOfBoundsException` - if `beginIndex` is negative or larger than the length of this `String` object.

- en java les exceptions sont des objets
- elles sont toutes du type `Exception` et leur type "précis" est un "sous-type" de `Exception`
- les classes d'exceptions se nomment par convention *QuelqueChose*`Exception`

- des portions de code peuvent générer/**lancer** des exceptions, signes d'un "problème"
- le programmeur dispose d'un moyen pour **capturer** une exception et proposer une alternative/solution

Quelques méthodes du type `Exception`

- `public void printStackTrace()`
- `public String getMessage()`
- `public String getLocalizedMessage()`

Exemple (1)

```
1  public class TestException {
2     private int[] tab;
3     public TestException() {
4        this.tab = new int[2];
5        for(int i= 0; i < this.tab.length; i++) { this.tab[i]= i; }
6     }
7     public void timoleon(int limite) {
8        for(int i= 0; i < limite; i++) {
9           System.out.println(this.tab[i]);
10       }
11    }
12    public void go(int limite) {
13       this.timoleon(limite);
14       }
15    }
16    public static void main(String[] args) {
17       TestException ref = new TestException(3);
18       ref.go(Integer.parseInt(args[0]));
19       System.out.println("l'exécution continue ?");
20    }
21 }
   == EXECUTION ==
   $ java TestException 3
      0
      1
      Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:  2
         at TestException.timoleon(TestException.java:9)
         at TestException.go(TestException.java:13)
         at TestException.main(TestException.java:18)
```


"Capturer" une exception

Lorsqu'une portion de code est susceptible de **lancer**/**lever** une exception, il est possible d'**essayer** d'exécuter de ce code et de **capturer** l'exception et d'indiquer le traitement qui doit en être fait.

```
try {
   ...  code susceptible de lancer une exception
}
catch (ClasseDExceptionLancee e) {
   ..  traitement de l'exception
}
```

```
try {
   obj.method();
}
catch (NullPointerException e) {
   System.out.println("obj est null");
}
```

- Lorsqu'une exception levée est capturée, cela n'arrête pas le programme,
- On quitte un "bloc `try`" dès qu'une exception est levée dans ce bloc, le flux d'exécution reprend "après" le bloc.
- Si l'exception est capturée, le traitement associé à cette capture est exécuté
- Le bloc générateur de l'exception n'est pas réexécuté après le traitement dû à la capture (même si la cause de l'erreur a été corrigée par ce traitement)
- un même bloc peut être susceptible de lever plusieurs exceptions, il est possible de les traiter séparément ou globalement

```
try {
   x = 1/obj.getValue();
}
catch (NullPointerException e) {
   System.out.println("obj est null");
}
catch (ArithmeticException e) {
   System.out.println("val de obj == 0");
}
```

```
try {
   x = 1/obj.getValue();
}
catch (Exception e) {
   System.out.println("excep "+e);
}
```


Exemple (2)

```
public class TestException {
   ...  idem ...
   public void go(int limite) {
      try {
         this.timoleon(limite);
      }
      catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("avant");
         System.out.println("*** FAIRE QUELQUE CHOSE ***");
         System.out.println(" ...  probleme corrige et detecte ");
         System.out.println("apres");
      }
   }

   public static void main(String[] args) {
      ...  idem ...
   }
}
== EXECUTION ==
$ java TestException 3
   0
   1
   avant
   *** FAIRE QUELQUE CHOSE ***
   ...  probleme corrige et detecte
   apres
   l'exécution continue ?
```


Exemple (autre)

```
...
public class Book {
   ...
   public boolean equals(Object o) {
      try{
         Book theOther = (Book) o;    // peut générer une ClassCastException
         return this.title.equals(theOther.title)
            && this.author.equals(lAutre.author)
            && this.year == theOther.year;
      }
      catch (ClassCastException e) {
         return false;
      }
   }
}
```


Cas de non capture

Si une méthode contient une portion de code susceptible de lancer une exception et que l'on ne souhaite pas (ou ne peut pas) traiter l'exception dans le corps de la méthode, il est nécessaire d'informer l'utilisateur de la méthode que celle-ci peut générer une exception

"`throws` *ClasseDException*" dans la signature de la méthode
JavaDoc : tag `@exception`

```
public class AClass {
   private String name;
   ...
   public int suffixe(int longueurSuffixe) throws IndexOutOfBoundsException {
      return this.name.substring(this.name.length()-longueurSuffixe);
   }
}

public FileReader openFile(String file) throws FileNotFoundException {
   return new FileReader(file);
}
```

Exemple (3)

```
public class TestException {
   ...  idem ...
   public void go(int limite) throws ArrayIndexOutOfBoundsException {
      this.timoleon(limite);
   }

   public static void main(String[] args) {
      TestException ref = new TestException(3);
      try {
         ref.go(Integer.parseInt(args[0]));
      }
      catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("*** capture de l'exception AIOOBE dans le main");
      }
   }
}
== EXECUTION ==
$ java TestException 3
   0
   1
   *** capture de l'exception AIOOBE dans le main
```


- il est possible qu'un méthode soit susceptible de déclencher plusieurs exceptions :

```
public void someMethod(args)
      throws Exception1, ..., ExceptionN {
   ...
}
```


Lever une exception

Pour lever une exception explicitement dans une portion de code :

1. créer un objet exception (de la classe d'exception voulue)
2. "lancer" l'exception à l'aide de `throw`

Exemple (4)

```
public class TestException {
   ...  idem ...
   public void go(int limite) throws IllegalStateException {
      if (limite > this.tab.length) {
          throw new IllegalStateException("limite dépassée");
      }
      this.timoleon(limite);
   }

   public static void main(String[] args) {
      TestException ref = new TestException(3);
      try {
         ref.go(Integer.parseInt(args[0]));
      }
      catch(IllegalStateException e) {
         System.out.println("*** capture de l'exception ISE dans le main");
      }
   }
}
== EXECUTION ==
$ java TestException 3
   0
   1
   *** capture de l'exception ISE dans le main
```

- Réexécuter le bloc `try` après correction d'une erreur :

```
boolean done = false;
while (!  done) {
   try {
      ...  traitement avec levée d'exceptions possible
      done = true;
   } catch(ClasseDException e) {
      ...  correction ''problème''
      done = false; // ou done déjà à false
   }
}
```


- exemple : saisie d'un nombre

```
public class Input {
   public static int readInt() throws java.io.IOException {
      try {
         return Input.scanner.nextInt();
      } catch (Exception e) {
         Input.scanner.skip(".*");
         throw new java.io.IOException();
      }
   }
} // Input
```

```
==========================================================================================
```

```
   int value = -1;
   while (value < 0) {
      try {
         System.out.print("Enter a number between 1 and 6 :  ");
         value = Input.readInt();
      } catch(IOException e) {
         // input string is not a
         System.out.println("You must provide a number between 1 and 6.");
      }
   }
```


plus tard

- pourquoi toutes les exceptions n'ont pas nécessairement besoin d'être capturées ou signalées (ce sont les "`RuntimeExceptions`") ?
- créer ses propres exceptions