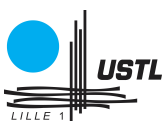
Exceptions

Programmation Orientée Objet



Jean-Christophe Routier
Licence mention Informatique
Université des Sciences et Technologies de Lille

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`**
comparer (par `compareTo` ou `equals`) des objets de classes différentes	**`ClassCastException`**
tenter de référencer en lecture un fichier qui n'existe pas	**`FileNotFoundException`**
etc.	etc.

- 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/*lever* des exceptions, signes d'un "problème"
- le programmeur dispose d'un moyen pour *capturer* une exception et proposer une alternative/solution

“Capturer” une exception

Lorsqu’une portion de code est susceptible de **lancer/lever** une exception, il est possible de **capturer** cette 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 est levée, 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 due 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

```
package calculette;
import po.calculette.*;
import calculette.operateur.*;
public class CalculetteInfixee implements Calculette {
   private final static Operateur PLUS = new Plus();
   ...
   private ValeurEditable ve = new ValeurEditable();
   private int accu = 0;
   private Operateur operateurCourant = null;
   ...
   public void enfoncerPlus() { exec(PLUS); }
   public void enfoncerMult() { exec(MULT); }
   public void enfoncerEgal() { exec(null); }
   ...
   private void exec(Operateur nouvelOperateur) {
      try {
         accu = operateurCourant.calcul(accu, ve.getValeur());
      catch (NullPointerException e) { accu = ve.getValeur(); }
      ve.nouveauNombre();
      operateurCourant = nouvelOperateur;
}
```

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 int index;
      public int compareTo(Object o) throws ClassCastException {{
         return this.index - ((AClass) o).index;
      }
}

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

- il est possible qu'un méthode "lance" 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`**

```
public class DaysOfWeek implements Comparable {
   ...
   public final int index;
   ...
   public int compareTo(Object o) throws ClassCastException {
      if (o instanceof DaysOfWeek) {
         return new Integer(this.index).compareTo(new Integer(((DaysOfWeek) o).index));
      }
      else {
         throw new ClassCastException("argument should be a DaysOfWeek");
      }
   }
} // DaysOfWeek
```

- 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
      ...  mais on laise done à false
   }
}
```

à suivre (COO - S5)

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