IN201 : 8 – Exceptions

Solution

Author : Christophe Garion <garion@isae.fr>
Public : SUPAERO 2A
Date :

Résumé

Le but de ce TP est de manipuler des exceptions prédéfinies et de créer ses propres exceptions. On utilisera des fichiers au format XML.

1 Contenu

Ce document contient le corrigé de la partie « conception » du TP. Les sources Java des classes sont disponibles sur le site http://www.tofgarion.net/lectures/IN201. Attention, ce corrigé est *personnel*, j'ai fait des choix qui peuvent être contestés sans aucun problème.

2 Problématique

On souhaite récupérer des données provenant d'expériences d'aérodynamique dans lesquelles on relève des pressions. Ces données sont donc des réels qui doivent normalement être positifs. Les fichiers seront des fichiers au format XML [1]. XML est un langage de marquage qui permet d'ajouter du contenu sémantique à un fichier contenant du texte. Il est facilement lisible par un humain et par une machine, ce qui en fait son intérêt.
Un fichier XML représente un *arbre* contenant des *éléments* délimités par des *balises* auxquels on peut ajouter des *attributs*. Les fichiers de stockage des expériences auront le format suivant :

```
<experience resp-name="Allan Bonnet">
  <pression>25.754</pression>
  <pression>10.432</pression>
  <pression>30.6754</pression>
  <pression>42.543</pression>
  <pression>100.678</pression>
</experience>
```

Dans ces fichiers :
- l'élément représentant l'ensemble des relevés est délimité par la balise `experience`. On remarquera que le début d'un élément `balise` est représenté par une balise ouvrante `<balise>` et la fin de ce même élément par une balise fermante `</balise>`.
- la balise `experience` possède un attribut dont la clé est `resp-name` et dont la valeur représente le nom du responsable de l'expérience. Les valeurs des attributs ne peuvent être que des chaînes de caractères délimitées par `""` (ou `''`). Cet attribut peut ne pas être présent, on suppose dans ce cas que le responsable de l'expérience est inconnu.
- chaque relevé de pression est délimité par la balise `pression`.

Les données contenues dans les balises ne pourront être récupérées que comme des chaînes de caractères depuis Java.

On va donc créer une classe qui devra offrir les services suivants :
- lecture du fichier de données et vérification de la cohérence des données ;
- stockage des données ;
- accès au nom du responsable de l'expérience ;
- renvoi d'un itérateur sur l'ensemble de données.

Lorsqu'une erreur dans les champs représentant une valeur numérique du fichier est présente, on souhaite lever une exception et permettre à l'utilisateur de récupérer les chaînes de caractères des champs posant problème.

3 Conception de la classe `Acquisition`

La classe `Acquisition` est la classe qui va nous permettre de récupérer, puis de stocker les valeurs contenues dans le fichier.

1. quels est(sont) le(s) attribut(s) de la classe `Acquisition` ?

   Solution :

   Les attributs vont permettre de stocker les données lues dans le fichier. Il faut donc prévoir un attribut pour stocker ces valeurs. J'ai choisi ici d'utiliser une instance de `ArrayList` qui permet d'éviter d'avoir à gérer un tableau. On aurait très bien pu utiliser une instance de `ListeChainee` par exemple.

   J'ai choisi également de stocker le nom du fichier contenant les données comme attribut. Cela signifie que pour chaque fichier, on va utiliser un objet de type `Acquisition` différent. On aurait pu se libérer de cette contrainte et permettre de parcourir plusieurs fichiers avec le même objet.

   Enfin, le nom du responsable de l'expérience sera également stocké comme attribut.

2. quelles sont les méthodes de la classe `Acquisition` ?

   Solution :

   Il y a quatre méthodes dans la classe `Acquisition` :
   - **`private void`** `acquerirDonnees()` qui permet de lire les valeurs dans les fichiers et de les stocker dans l'`ArrayList`. Cette méthode est une méthode auxiliaire et sera appelée depuis le constructeur de `Acquisition` ;
   - **`public`** `String getResponsable()` qui renvoie le nom du responsable de l'expérience ;
   - **`public`** `Iterator<Double> iterator()` qui renvoie un itérateur sur la liste chaînée ;
   - **`public`** `String toString()` qui permet comme d'habitude d'avoir une représentation sous forme de chaîne de caractères de l'objet.

3. faudra-t-il utiliser des exceptions ? Si oui, faudra-t-il en définir vous-même ?

   On utilisera évidemment des méthodes qui pourront lever des exceptions dans la méthode `acquerirDonnees`, par exemple lorsque l'on va construire l'arbre XML à partir du fichier de données. Il faudra également créer une exception particulière pour signaler qu'un champ représentant une valeur numérique n'est pas correct (valeur négative ou chaîne de caractères ne représentant pas une valeur numérique). Elle sera représentée par la classe `FormatDonneesException`. Cette classe sera une exception particulière, puisqu'elle aura un attribut de type `ArrayList<String>` contentant les valeurs incorrectes sous forme de chaînes de caractères.

   Lorque le fichier XML sera lu, il faudra stocker les erreurs dans une liste, puis lever une exception à la fin de la lecture du fichier si une erreur était présente. Lorque l'on « transforme » les chaînes de caractères récupérées depuis le fichier en **`double`** grâce à la classe `Double`, une exception de type `NumberFormatException` peut être levée (cf. javadoc). On utilisera donc un bloc **`try`**/**`catch`** avec cette exception pour trouver les erreurs de format numérique des champs.

   Enfin, il faut lever une exception si le nom du responsable de l'expérience n'a pas été précisé.

   Reste maintenant à savoir s'il faut propager ces exceptions ou les traiter localement. Il me semble qu'il est assez naturel de propager toutes les exceptions, car on ne sait pas a priori les traiter localement. C'est l'utilisateur qui devra donc les gérer...

4. construire le diagramme UML de la classe.

   Solution :

   Le diagramme est représenté sur la figure 1. Notez que la méthode `acquerirDonnees` est privée, qu'elle propage les exceptions levées à l'intérieur et qu'elle est normalement appelée depuis le constructeur. Afin de ne pas surcharger le diagramme, j'ai choisi de mettre juste une note sur le constructeur pour rappeler qu'il propagera les exceptions levées par `acquerirDonnees`. J'ai également regroupé sous le nom `IOXMLException` les exceptions liées aux problèmes d'entrées/sorties ou au *parser* du fichier XML, toujours par soucis de lisibilité.

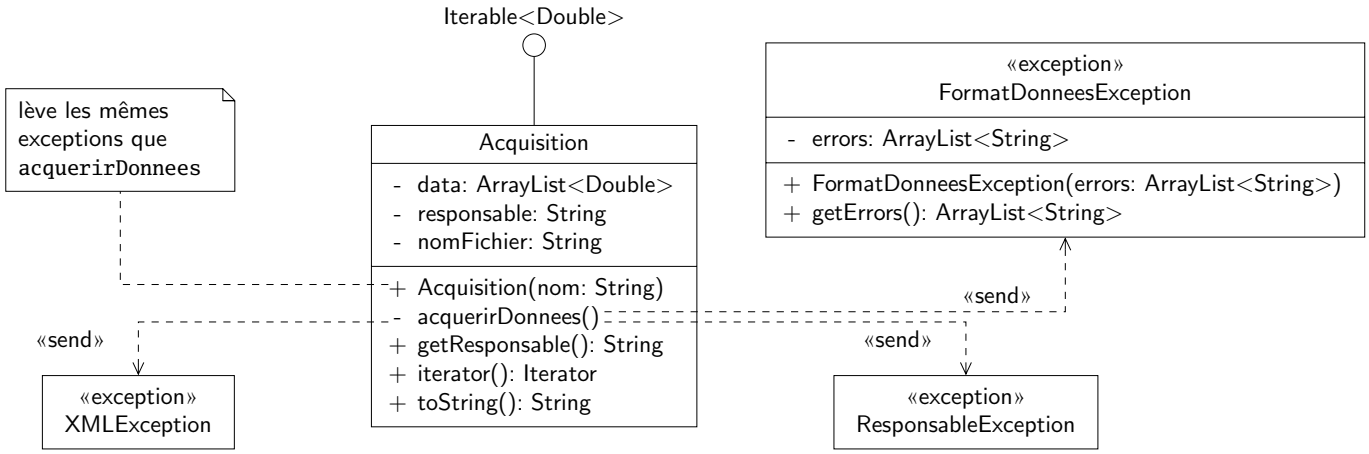

FIGURE 1 – La classe `Acquisition` et les exceptions associées

4 Implantation de la classe `Acquisition`

Il faut maintenant implanter la classe `Acquisition`.

1. choisir un attribut permettant de stocker les valeurs ;

   Solution :
   Le choix a été fait précédemment : il s'agit d'une instance de `ArrayList`. Ce choix nous permet de récupérer facilement une instance de `Iterator` sur l'ensemble des données et donc de rendre `Acquisition Iterable`.

2. pour lire les données contenues dans le fichier, utiliser la documentation de l'API JAXP ;

   Solution :
   Pas de problème particulier, les exemples étaient assez clairs mais il fallait faire attention et bien respecter l'algorithme proposé. Eclipse vous aidait pour gérer les exceptions.

3. construire les exceptions nécessaires ;

   Solution :
   Il fallait définir les classes `ResponsableException` et `FormatDonneesException` qui avaient été prévues lors de la conception. J'ai également rajouté une classe `ArgumentException` qui permet de gérer les cas où le nombre d'arguments passés en ligne de commande est différent de 1.
   En ce qui concerne `FormatDonneesException`, l'implantation n'était pas très compliquée.

4. les exceptions levées dans les méthodes doivent-elle être rattrapées et traiter dans la classe `Acquisition` ou propagées ?

   Solution :
   Cette question a été traitée dans la partie conception. Si on s'intéresse à la gestion des erreurs dans les champs qui doivent contenir les résultats des expériences, voici le code correspondant :

```
72          NodeList childNodes = root.getChildNodes();
73
74          ArrayList<String> errors = new ArrayList<String>();
75
76          for(int i = 0; i < childNodes.getLength(); i++) {
77              Node n = childNodes.item(i);
78              if (n.getNodeType() == Node.ELEMENT_NODE) {
79                  verifyElement((Element) n, "pression");
80
81                  String content = n.getTextContent();
82
83                  try {
84                      double d = Double.parseDouble(content);
85
```

```
86                      if (d < 0) {
87                          errors.add(content + " n'est pas positif");
88                      } else {
89                          data.add(d);
90                      }
91                  } catch (NumberFormatException e) {
92                      errors.add(content + " ne represente pas un nombre");
93                  }
94              }
95          }
96
97          if (! errors.isEmpty()) {
98              throw new FormatDonneesException("Erreur dans les valeurs numeriques",
99                                               errors);
100         }
```

Voici un bon exemple de traitement local des exception : je traite localement les exceptions de type `NumberFormatException` pour pouvoir remplir la liste d'erreurs.

5. utiliser les fichiers proposés dans le dépôt pour vérifier que la classe est opérationnelle (ajouter le répertoire `data` contenant les fichiers au CLASSPATH). Réfléchir également à l'écriture d'une classe de test JUnit.

   Solution :

   Pas de problèmes particulier. Il suffisait de lancer par exemple `java TestAcquisition donnees1` pour avoir un premier test.

   Pour écrire une classe de test JUnit pour `Acquisition`, j'ai utilisé le *tag* `@Test(expected=Exception.class)` pour écrire des méthodes de test vérifiant que des exceptions étaient bien levées. Les fichiers fournis dans le dépôt sont utilisés pour cela. On remarquera que la méthode de test d'acquisition des données est assez lourde : on fait l'acquisition des données, puis on vérifie que les données sont les bonnes en les comparant aux données lues dans le fichier de test. Pour les cas levant une instance de `FormatDonneesException`, je traite localement l'exception pour vérifier que la « bonne erreur » est trouvée et je repropage l'exception.

6. (facultatif) comment vérifier que le fichier XML a le bon format ?

   Solution :

   Pour vérifier que le fichier a le bon format, j'ai utilisé la méthode `getTagName` de `Element` pour vérifier que le nom des balises était correct. J'ai également vérifié que le nœud racine du document était bien un élément XML. J'ai donc créé une classe d'exceptions `ElementXMLException` qui est utilisée lorsque le fichier XML n'a pas la syntaxe attendue. Cette exception est propagée tout comme `ResponsableException` et `FormatDonneesException` car il s'agit d'erreurs d'écriture du fichier.

Références

[1] Wikipedia contributors. *XML*. 2103. URL : http://en.wikipedia.org/wiki/XML.

License CC BY-NC-SA 3.0

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported license (CC BY-NC-SA 3.0)

You are free to Share (copy, distribute and transmite) and to Remix (adapt) this work under the following conditions:

Attribution – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Noncommercial – You may not use this work for commercial purposes.

Share Alike – If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

See http://creativecommons.org/licenses/by-nc-sa/3.0/ for more details.