IN201 : 13 – Exceptions

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

Solution

Résumé

Le but de ce TP est de manipuler des exceptions prédéfinies et de créer ses propres exceptions.

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'un fichier. Ces données sont des réels qui doivent normalement être positifs (par exemple, ils représentent une pression). 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 ;
– renvoi d'un itérateur sur l'ensemble de données.

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.

2. quelles sont les méthodes de la classe `Acquisition` ? Faudra-t-il utiliser des exceptions ? Si oui, faudra-t-il en définir vous-même ?

   Solution :
   Il y a deux méthodes dans la classe `Acquisition` :
   – **`public void`** `acquerirDonnees()` qui permet de lire les valeurs dans les fichiers et de les stocker dans l'`ArrayList` ;
   – **`public`** `Iterateur getIterateur()` qui renvoie un itérateur sur la liste chaînée.
   On utilisera évidemment des méthodes qui pourront lever des exceptions dans la méthode `acquerirDonnees`, par exemple lorsque l'on va ouvrir un flux vers le fichier. Il faudra également créer une exception particulière pour signaler qu'une valeur dans le fichier n'est pas strictement positive. Elle sera représentée par la classe `ReelNegatifException`. De plus, il faudra « transformer » les chaînes de caractères récupérées depuis le fichier en **`double`** grâce à la classe `Double`. Cette «transformation » peut lever une `NumberFormatException`.
   Reste maintenant à savoir s'il faut propager ces exceptions ou les traiter localement. J'ai fait un choix :
   – les exceptions `NumberFormatException` et `ReelNegatifException` qui seront levées seront propagées, car il s'agit d'exceptions concernant des erreurs d'écriture du fichier que l'on ne peut pas corriger localement ;
   – les exceptions concernant des erreurs d'entrée/sortie seront traitées localement (la plupart du temps, affichage d'un message d'erreur).

3. construire le diagramme UML de la classe.

Solution :
Le diagramme est représenté sur la figure 1.

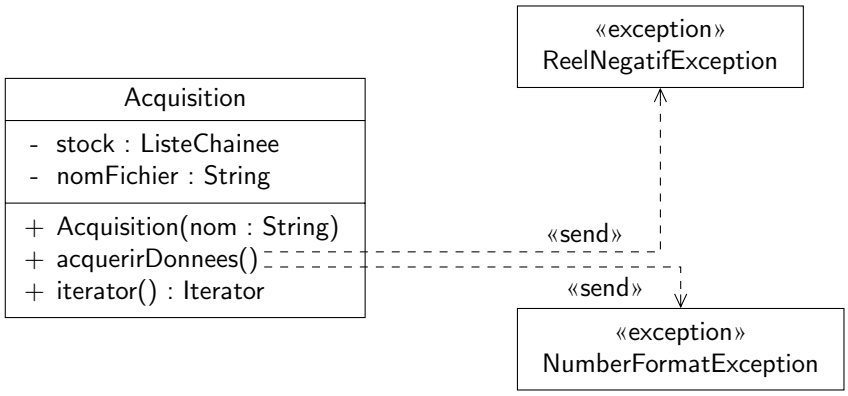

FIGURE 1 – La classe `Acquisition` et les exceptions associées

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. utiliser la documentation de l'API d'entrée/sortie de Java ;

   Solution :
   Pas de problème particulier. Les exemples étaient assez clairs. Petite subtilité, j'ai utilisé la clause **`finally`** pour fermer le fichier, même en cas d'exception. Il fallait alors déclarer la référence vers `FileReader` en dehors de la boucle **`try`** pour qu'elle soit visible dans la clause **`finally`**. On pourra vérifier quand la clause **`finally`** est exécutée grâce à l'affichage.

3. construire les exceptions dont vous avez besoin ;

   Solution :
   Il fallait définir la classe `ReelNegatifException` qui avait été prévue 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.

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

   Solution :
   Cette question a été traitée dans la partie conception.

5. utiliser les fichiers proposés sur le site pour vérifier que la classe est opérationnelle. En particulier, réfléchir 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 sur le site 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 ligne par ligne dans le fichier de test.