Chapitre 8: Exceptions et I/O Streams

Présentation pour

Java Software Solutions

Foundations of Program Design
Second Edition

by John Lewis and William Loftus

Java Software Solutions is published by Addison-Wesley

Exceptions et I/O Streams

- Nous pouvons explorer deux sujets reliés: les exceptions et les streams d'entrée / sortie
- Le chapitre 8 se concentre sur:
  - L'énoncé try-catch
  - La propagation d'exceptions
  - Créer et lancé des exceptions
  - Types de I/O streams
  - Le traitement de la classe Keyboard
  - Lecture et écriture de fichiers de texte
  - Sérialisation d'objets

Exceptions

- Une *exception* est un objet qui décrit une situation inhabituelle ou erronée
- Les exceptions sont *lancées* par un programme, et peuvent être *attrapées* et *traitées* par une autre partie du programme
- Un programme peut donc être séparé entre un flot d'exécution normal et un *flot d'exécution d'exception*
- Une *erreur* est aussi représentée comme un objet en Java, mais représente habituellement une situation irrécouvrable et ne devrait pas être attrapée

Traitement des Exceptions

- Un programme peut traiter une exception de trois façons:
  - L'ignorer
  - La traiter là où elle arrive
  - La traiter un autre endroit dans le programme
- La façon avec laquelle une exception est traitée est une considération importante lors de la conception

Traitement des Exceptions

- Si une exception est ignorée par le programme, l'exécution va s'arrêter et produira un message approprié
- Le message inclut une *call stack trace* indiquant à quelle ligne l'exception est arrivée
- La *call stack trace* montre aussi l'appel de la méthode qui a conduit à l'exécution de la ligne offensante
- Voir `Zero.java` (page 379)

L'énoncé `try`

- Pour traiter une exception lors de son occurrence, la ligne qui lance l'exception est exécutée à l'intérieur d'un bloc *try*
- Un bloc *try* est suivit par un ou plusieurs clauses *catch*, qui contiennent le code pour traiter l'exception
- Chaque clause *catch* est associée à un type d'exception
- Lors de l'occurrence d'une exception, le traitement continu à la première clause *catch* qui correspond au type de l'exception
- Voir `ProductCodes.java` (page 381)

La Clause `finally`

- **Un énoncé *try* peut avoir une clause optionnelle désignée par le mot réservé `finally`**
- **Si aucune exception est générée, l'énoncé dans la clause *finally* est exécutée après que les instructions dans le bloc *try* soit exécutées**
- **Aussi, si une exception est générée, les instructions dans la clause *finally* sont exécutée après que les instructions dans la clause *catch* appropriée soit complétées**

7

La Propagation des Exceptions

- **S'il n'est pas approprié de gérer l'exception où elle se produit, elle peut être traitée à un plus haut niveau**
- **Les exceptions se *propagent* vers le haut à travers la hiérarchie d'appels de méthodes jusqu'à ce qu'elles soient attrapées et traitées ou qu'elles atteignent le niveau le plus haut**
- **Un Bloc *try* contenant une appel a une méthode dans laquelle une exception est lancée peut être utilisé pour attraper cette exception**
- **Voir `Propagation.java` (page 384)**
- **Voir ExceptionScope.java (page 385)**

8

L'énoncé `throw`

- **Un programmeur peut définir une exception en dérivant la classe appropriée**
- **Les exceptions sont lancées en utilisant l'énoncé *throw***
- **Voir `CreatingExceptions.java` (page 388)**
- **Voir `OutOfRangeException.java` (page 389)**
- **Habituellement, l'énoncé *throw* est imbriqué à l'intérieur d'un if qui évalue la condition pour voir si une exception devrait être lancée**

9

Exceptions Vérifiées

- **Une exception est soit *vérifiée* ou *non vérifiée***
- **Une exception vérifiée ne peut être lancée qu'à l'intérieur d'un bloc *try* ou encore à l'intérieur d'une méthode qui est désignée pour lancer une exception**
- **Le compilateur se plaindra si une exception vérifiée n'est pas traitée de façon appropriée**
- **Une exception non vérifiée n'a pas besoin de traitement explicite quoique qu'elle pourrait en avoir**

10

I/O Streams

- **Un *stream* est une séquence d'octets qui coulent d'une source vers une destination**
- **Dans un programme, nous lisons de l'information d'un *input stream* et écrit de l'information à un *output stream***
- **Un programme peut gérer plusieurs *streams* à la fois**
- **Le package java.io contient plusieurs classes qui nous permettent de définir des *streams* variées avec des caractéristiques spécifiques**

Catégories d'I/O Stream

- **Les classes dans le package I/O divisent les *streams* d'entrée et sortie dans d'autres catégories**
- **Un I/O *stream* est soit**
  - ***character stream*, traitent des données de texte**
  - ***byte stream*, traitent des octets de données**
- **Un I/O stream est aussi, soit**
  - ***data stream*, qui peut être une source ou une destination**
  - ***processing stream*, qui change ou gère les informations dans le stream**

I/O Standard

- **Il y a trois I/O streams standards :**
  - ***standard input* – définie par `System.in`**
  - ***standard output* – définie par `System.out`**
  - ***standard error* – définie par `System.err`**
- **Nous utilisons `System.out` lors de l'exécution de `println`**
- **`System.in` est déclaré être une référence générique à un `InputStream`, et donc, doit habituellement être projeté vers un stream utile avec des caractéristiques spécifiques**

La Classe Keyboard

- **La classe `Keyboard` a été écrite par les auteurs du livre pour faciliter la lecture de données de *standard input***
- **Nous pouvons maintenant examiner le traitement de la classe `Keyboard` plus en détails**
- **La classe `Keyboard` :**
  - **Déclare un *input stream* standard qui est utile**
  - **Gère les exceptions qui peuvent être lancées**
  - **Fait l'analyse syntaxique des lignes en entrées vers des valeurs séparées**
  - **Convertit les strings en entrées vers les types attendus**
  - **Gère les problèmes de conversion**

L'*Input Stream* Standard

- **La classe `Keyboard` déclare les *input streams* suivantes:**

```
InputStreamReader isr =
    new InputStreamReader (System.in)
BufferedReader stdin = new BufferedReader (isr);
```

- **L'objet `InputStreamReader` convertit le *byte stream* original vers un *character stream***
- **L'objet `BufferedReader` nous permet d'utiliser la méthode `readLine` pour obtenir une ligne entière en entrée**

Fichiers Texte

- **Les informations peuvent être lues de et écrites dans des fichiers texte en déclarent et utilisant les *streams I/O* appropriées**
- **La classe `FileReader` représente un fichier d'entrée contenant des données de caractères**
- **Voir `Inventory.java` (page 397)**
- **Voir `InventoryItem.java` (page 400)**
- **La classe `FileWriter` représente un fichier de sortie de texte**
- **Voir `TestData.java` (page 402)**

Sérialisation d'Objet

- ***La sérialisation d'objets* est l'action de sauvegarder un objet, et son état courant, pour qu'il puisse être utilisé dans un autre programme**
- **L'idée qu'un objet puisse exister après le programme qui l'a créé est appelé *persistance***
- ***La sérialisation d'objets* est accomplie en utilisant les classes `ObjectOutputStream` et `ObjectInputStream`**
- **La sérialisation tient compte des autre objets qui sont référencés par un objet qui est sérialisé, les sauvegardant aussi**