Schülerakademie Schelklingen 2008

Kurs 4.2 - eXtreme Programming
Exceptions und Logging in Java

Manuel Eberl

7.8.2008

Inhaltsverzeichnis

1 Wozu Exceptions? [JLS 11]

Exceptions, zu Deutsch “Ausnahmen”, sind ein Konzept in vielen Programmiersprachen, das es erlaubt, auf Fehler, die in einem Programm auftreten, zu reagieren. Bemerkt ein Programmteil einen Fehler, zum Beispiel beim Versuch, eine nicht existente Datei zu laden, so löst er eine Exception aus. Diese Exception wird durch die Aufrufhierarchie des Programms nach oben weitergereicht, bis sie einen Programmteil erreicht, der sie verarbeiten kann, und von ihm abgefangen wird. Ohne Exceptions müsste jede Funktion eine Fehlerbehandlung durchführen, wofür sie auch sämtliche relevanten Informationen zur Reaktion auf den Fehler erhalten müsste. Dies macht den Programmcode unübersichtlich und unflexibel.

2 Exceptions und Errors in Java

2.1 Throwables [Insel S. 476 f.]

Throwable ist ein Interface, von dem sich alle Objekte ableiten, die ausgelöst und abgefangen werden können. Die einzigen Klassen der Java-Bibliothek, die Throwable implementieren, sind Exception und Error. Ausgehend von Throwable und diesen beiden Klassen sowie RuntimeException, einer von Exception abgeleiteten Klasse, erstreckt sich die Vererbungshierarchie von Exceptions und Errors:

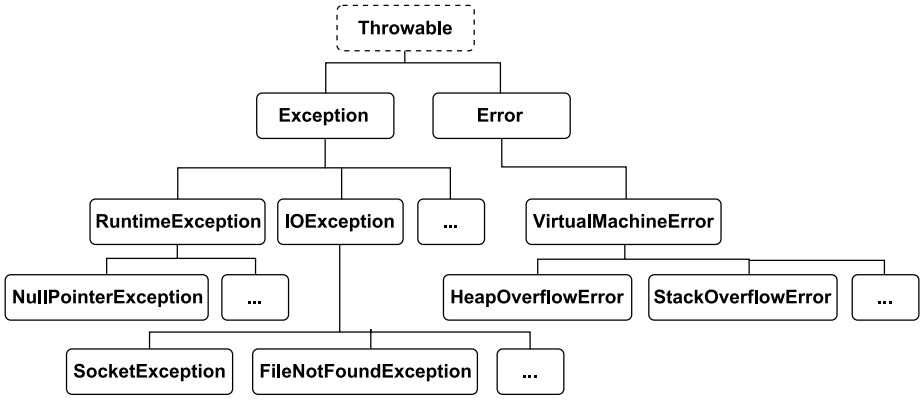

Abbildung: Klassenhierarchie der Throwables in Java
[API, Throwable, http://java.sun.com/javase/6/docs/api/java/lang/Throwable.html]

2.2 Checked und unchecked Exceptions [Insel S. 480]

In Java wird ein besonderes Prinzip von Exceptions verwendet: Es gibt checked und unchecked Exceptions, wobei letzere vom Typ RuntimeException sind oder von ihm ableiten und erstere nicht. Java prüft bereits bei der Kompilierung des Programmcodes, ob alle checked Exceptions, die in einer Methode auftreten können, auch gefangen oder (per ’throws’) an die aufrufende Funktion weitergeleitet werden. Ist dies nicht der Fall, wird die Kompilierung mit einen Fehler abgebrochen. Unchecked Exceptions werden normalerweise, weil sie nicht gefangen werden, durch die gesamte Aufrufhierarchie nach oben gereicht, bis sie von der Java-Umgebung abgefangen werden und dadurch das Programm abgebrochen wird.

2.3 Syntax in Bezug auf Throwables in Java

Um eine Exception auszulösen, wird in Java das Schlüsselwort “throw” verwendet:

```
throw new IllegalArgumentException("Ungültiger Parameter: "+param);
```

'IllegalArgumentException' ist hierbei der Typ des Fehlers. Dieser Typ wird ausgelöst, wenn eine Funktion einen ungültigen Wert übergeben bekommen hat. '“Ungültiger Parameter. "+param' ist die Fehlermeldung, die möglichst aussagekräftig sein sollte, um sofort auf den Grund des Fehlers schließen zu können.

2.3.1 Behandeln von Throwables

try-catch/finally-Block [Insel S. 465 ff., S. 472 ff.]:
Ein try-catch-Block fängt alle Exceptions ab, die in den catch-Blöcken angegeben werden. catch (Exception1 e) fängt dabei alle Exceptions ab, die vom Typ Exception1 sind oder von ihm ableiten. Von mehreren passenden catch-Blöcken wird immer der erste ausgeführt. Ein finally-Block wird immer ausgeführt, egal ob der Code im try-Block eine Exception auslöst oder nicht. Dies ist zum Beispiel nützlich, wenn im try-Block eine Ressource, etwa eine Datei, angefordert wird, die nach der Bearbeitung wieder für andere Prozesse freigegeben werden sollte.

```
try {
    Programmcode der Exceptions vom Typ Exception1, Exception2 auslösen kann
} catch (Exception1 e) {
    Fehlerbehandlung für Exception1
} finally {
    Programmcode, der immer ausgeführt wird.
}
```

2.3.2 Weiterreichen von Exceptions [Insel S. 469 ff.]

Falls eine Funktion nicht die erforderlichen Informationen besitzt, um einen Fehler zu verarbeiten, so kann sie einen auftretenden Fehler an die aufrufende Funktion weiterreichen. Damit dies vom Compiler zugelassen wird, muss die Exception in die throws-Deklaration der Funktion aufgenommen werden. Die Syntax dafür lautet:

```
public void foo() throws Exception1
{
    Code, der Exception1 verursachen kann, diese aber nicht abfängt
}
```

Errors und RuntimeExceptions sind unchecked und sollten in den meisten Fällen nicht gefangen werden, deshalb müssen sie auch nicht in die throws-Deklaration mit aufgenommen werden.

2.4 Exceptions [Insel S. 465]

“Normale” Exceptions, das heißt, Typen die von Exception (nicht aber von RuntimeException oder einem ihrer Untertypen) abgeleitet sind, werden in Java für Fehler benutzt, mit denen in

einer normalen Ausführung des Programms gerechnet werden muss, ein gutes Beispiel ist das oben genannte Problem einer nicht gefundenen Datei (FileNotFoundException).

2.5 RuntimeExceptions und Errors [Insel S. 479 ff.]

In Java gibt es außer Exceptions noch Errors, die sich ebenso wie Exceptions auslösen und fangen lassen, aber unchecked sind. Errors jedoch sind Fehler, die abnormale Zustände repräsentieren, die aus einer Fehlfunktion der Java-Umgebung resultieren, und deshalb nicht gefangen werden sollten. In den meisten Fällen handelt es sich dabei um fatale Fehler, bei denen die Programmausführung sofort beendet werden muss. Ein Beispiel hierfür ist der HeapOverflowError, die von der Java-Umgebung augelöst werden, wenn kein freier Speicher mehr vorhanden ist, was meistens auf Programmierfehler zurückzuführen ist.

RuntimeExceptions sind ebenfalls unchecked sollten in den meisten Fällen nicht gefangen werden, da sie im normalen Programmablauf nicht auftreten sollten. Anders als bei Errors arbeitet die Java-Umgebung hier jedoch korrekt, der Fehler liegt im Programmcode. Ein Beispiel für eine RuntimeException ist die NullPointerException, die ausgelöst wird, wenn versucht wird, auf ein nicht instantiiertes Objekt zuzugreifen. RuntimeExceptions sollten vom Programmierer behoben, nicht gefangen werden.

2.6 Vor- und Nachteile des Exceptionmodells in Java [JLS 11.2]

Einer der großen Vorteile von checked Exceptions ist es, dass der Programmierer nicht vergessen kann, bestimmte Exceptions zu behandeln, da der Programmcode ohne eine entsprechende Behandlung nicht kompiliert. Dadurch erhält das Programm eine geringere Fehleranfälligkeit, sofern die Fehlerbehandlung richtig implementiert wird.

Ein Nachteil dieses Modells ist, dass bei strenger Einhaltung des Exceptionmodells die throws-Deklarationen von Methoden sehr groß werden, da aufgerufene Methoden viele verschiedene Exceptions erzeugen können, sämtliche auftretenden Fehler jedoch in einer Methode, die in der Aufrufhierarchie weit oben steht, etwa der Benutzeroberfläche, abgefangen und dem Benutzer gemeldet werden sollen. In diesem Fall bietet es sich an, RuntimeExceptions zu verwenden.
Hierfür werden auch nichtkritische Fehler im Programm in eine RuntimeException (oder eine davon abgeleitete Exception) “verpackt”, von einer passenden Methode, die die Fehlerbehandlung durchführen kann, etwa einer Methode der Benutzeroberfläche, abgefangen und an eine Fehlerbehandlungsmethode weitergegeben, die zum Beispiel eine Fehlermeldung anzeigt.

3 Logging [Insel S. 1385 ff.]

Um Fehlerquellen nachzuvollziehen benutzen Programmierer oft die Methode System.out.println(), die eine Textmeldung auf die Konsole schreibt. Diese Methode stellt die einfachste Form des sogenannten “Loggings” dar. Eine solche Ausgabe auf die Konsole ist jedoch vor allem bei größeren Projekten unpraktisch, da eine spätere Umstellung des Ausgabeziels (etwa in eine Datei), eine Änderung der Formatierung oder eine Filterung der Informationen aufwändig ist. Ohne geeignete Filterung wird immer jede Meldung ausgegeben, selbst wenn sie eine extrem detaillierte Ausgabe ist, die der Programmierer einmal zur Fehlersuche benötigt hat und eine Ausgabe auf der Konsole wird spätestens dann unpraktisch, wenn das Programm außerhalb der Entwicklungsumgebung benutzt werden soll, da die Konsole dann meistens nicht sichtbar ist.

Um solche Probleme zu vermeiden, gibt es Logger-Klassen, die diesen Vorgang automatisieren und ein größeres Maß an Flexibilität bieten. Die Vorteile solcher Logger sind dabei:

- Leichte Formatierung (z.B. hinzufügen der Zeit)
- Flexibles Ausgabeziel (Konsole, Datei...)
- Filterung (Verschiedene Detailstufen von Informationen wählbar, je nach Einstellung werden Nachrichten ab einem bestimmten Detailgrad verworfen)

Detaillierte und präzise Ausgaben durch Logger an geeigneten Stellen im Programm können die Fehlersuche deutlich vereinfachen. Bei einer Ausgabe in eine Datei ist zum Beispiel eine Funktion möglich, die bei einem Fehler nach Rückfrage beim Benutzer die Log-Datei an den Programmierer schickt, wie sie zum Beispiel beim Mozilla Firefox realisiert ist.

A Literaturverzeichnis

- Insel: Ullenboom, Christian. Java ist auch eine Insel - Das umfassende Handbuch, 7. Auflage. In: Galileo Computing, ISBN 978-3-8362-1146-8
- API: Java-API. http://java.sun.com/javase/6/docs/api/. Stand: 18. Dezember 2007
- JLS: Java Language Specification, Third Edition, http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html. Stand: 18. Dezember 2007