# 6
# Exceptions

Jörn Loviscach

Versionsstand: 21. März 2014, 22:58

Die nummerierten Felder sind absichtlich leer, zum Ausfüllen beim Ansehen der Videos:
http://www.j3L7h.de/videos.html



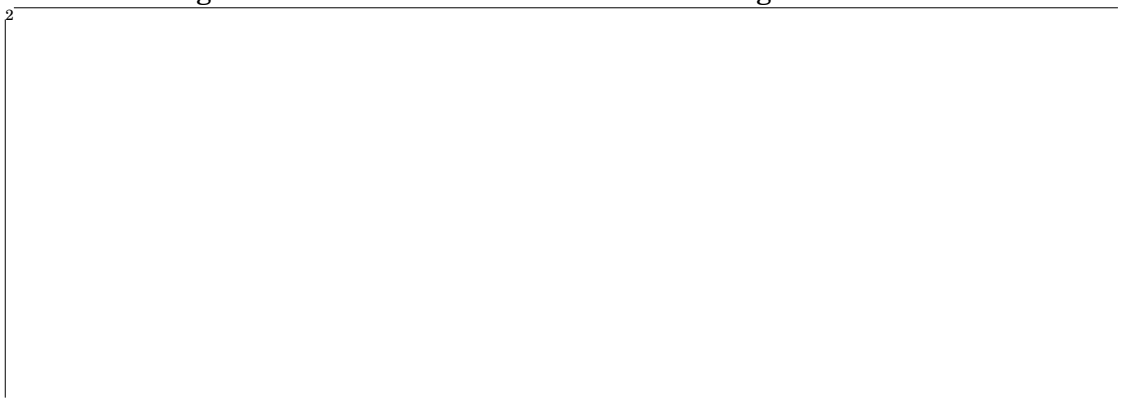
Bitte hier notieren, was beim Bearbeiten unklar geblieben ist

## 1 Idee der Ausnahmebehandlung

Im wahren Leben kann der Ablauf eines Programms an vielen Stellen fehlschlagen. Beispiele:

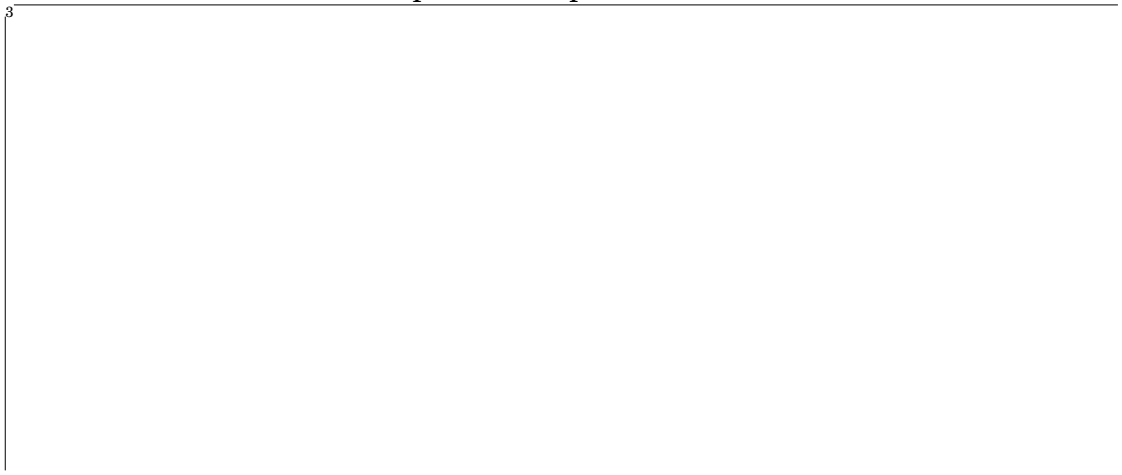
[1]

In C muss man in solchen Situationen nach jedem kritischen Schritt prüfen, ob ein Fehler aufgetreten ist. So sieht das etwas abstrakt geschrieben aus:

2


Ein solches Programm zu schreiben und/oder zu lesen, ist eine Strafarbeit. C++, Java, C# und viele andere Sprachen beherrschen dagegen eine Fehlerbehandlung mit Ausnahmen [exceptions], die weniger Tippen und Denken beim Programmieren verlangt, dafür dem Rechner mehr abverlangt (nicht so gut für ganz simple Microcontroller!).

Man markiert Teile des Programms, die fehlerträchtig sind, und schreibt gebündelt dahinter (statt verstreut dazwischen), was im Fehlerfall passieren soll. Die Funktionen liefern keine Fehlernummern zurück, sondern werfen im Fehlerfall Ausnahmen [to throw an exception]. Beispiel:

3


Die Schweifklammern von `try...catch` wirken wie alle anderen Schweifklammern. Variablen, die man innerhalb der Schweifklammern einführt, sind außen unsichtbar. Deshalb muss in diesem Beispiel das `n` vor dem `try...catch` eingeführt werden, um danach mit einem Wert gefüllt verwendbar zu sein.

# 2 Ausnahmen fangen, Ausnahmen werfen

Exceptions sind in C# Instanzen der Klasse `Exception` – oder von Kindklassen davon. Mit dem Programmcode von eben kann man schon einige verschiedene Klassen an Exceptions sehen. Jede Exception kann man mit `GetType()` nach

ihrem Typ fragen.

4

Man kann an einziges `try` mehrere `catch` anhängen, um Exceptions je nach Art verschieden zu behandeln. Bei der Ausführung geht das Programm in das erste `catch`, in der die Klasse oder eine Vorfahrenklasse der Exception gefangen wird:

5

Nicht nur die Klassen der Bibliothek, sondern auch selbst geschriebene Funktionen können Exceptions werfen, sinnvollerweise von der Klasse `ApplicationException`, noch besser von eigenen Ableitungen davon:

6

Wird eine Exception nicht in der Funktion gefangen, in der sie erzeugt wurde, wird die Funktion abgebrochen und die Exception an die aufrufende Funktion weitergereicht. Wird sie auch dort nicht gefangen, wird auch diese Funktion abgebrochen usw. Wenn dieses „Bubbling“ (Aufsteigen von Bläschen) nicht zum Fangen der Exception führt, endet das Programm hart.

7

Das sofortige Abbrechen der aktuellen Funktion ist nicht immer sinnvoll: Vielleicht möchte man zum Beispiel zuerst noch eine Datei schließen. Dafür gibt es in C# und in Java diese Konstruktion:

[8] 

Was im `finally` steht, wird immer ausgeführt, egal, ob vorher eine Exception aufgetreten ist oder nicht. Randnotizen: In C++ erreicht man denselben Effekt mit Hilfe der Destruktoren von lokalen Variablen. In C# gibt es eine elegante Variante mit `using`.

Aus Sicherheitsgründen muss man in Java anders als in C# beim Schreiben von Methoden für einen Teil der Exceptions angeben, wenn sie geworfen werden können (die „checked exceptions“). Diese müssen dann von allen aufrufenden Methoden gefangen werden – oder ebenfalls deklariert werden. In C++ kann man (muss aber nicht) deklarieren, welche Typen von Exceptions eine Funktion werfen darf. Hält sich die Funktion nicht daran, ist das kein Fehler für den Compiler, sondern führt bei der Ausführung zum Aufruf einer besonderen Funktion, die solche „unerwarteten“ Fälle behandeln soll.