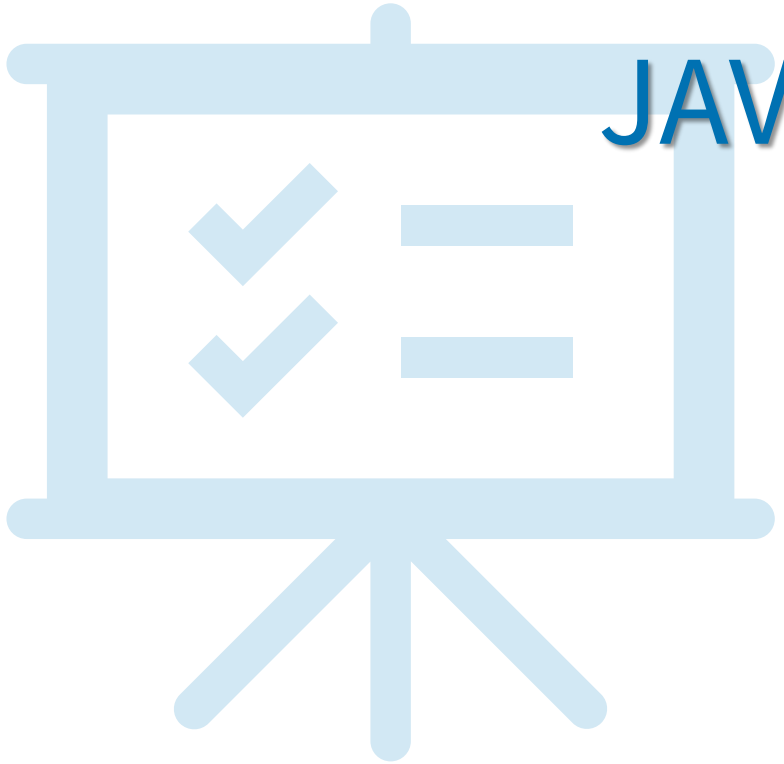
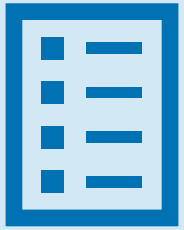


JAVA Programmierung

ECLIPSE & JAVA 8

Themenübersicht



01

Exceptions & Errors

1. Generelles
2. Checked Exceptions
3. Unchecked Exceptions
4. Error
5. Ausnahmen Hierarchie
6. Exceptions abfabgen

01 Exceptions & Errors

1. *Generelles*
2. *Checked Exceptions*
3. *Unchecked Exceptions*
4. *Error*
5. *Ausnahmen Hierarchie*
6. *Exceptions abfangen & behandeln*

1.1 Exceptions & Errors

Generelles

1.1 Exceptions

- **Generelles**

Exceptions (Ausnahmen) sind Probleme welche zur Laufzeit, also während der Ausführung des Programms auftreten. Dies führt unweigerlich zum Abbruch des Programms, sollten diese nicht behandelt werden. Es gibt unterschiedliche Gründe für solche Fehler.
Beispiele:

- ein ungültige Benutzer Eingaben
- Dateien welche geöffnet, aber nicht gefunden werden können
- eine unterbrochene Netzwerkverbindung

Basierend auf diesen Beispielen kann zwischen 3 Kategorien von Ausnahmen unterschieden werden.

- Checked Exceptions (geprüfte Ausnahmen)
- Unchecked Exceptions (ungeprüfte Ausnahmen)
- Errors (Fehler)

1.2 Exceptions & Errors

Checked Exceptions

1.2 Exceptions

- **Checked Exceptions**

Als Checked Exceptions (z.B. FileNotFoundException) werden Ausnahmen bezeichnet, welche zur Kompilierzeit auftreten, also eine Ausnahme welche vom Compiler geprüft wird. Solche Ausnahmen dürfen nicht einfach ignoriert werden. Programmierer **müssen** diese „behandeln“.
Eine geeignete Kontrollstruktur haben Sie bereits kennen gelernt.

=> Try Catch

Ziel ist es beim Auftreten des Fehlers, Sorge zu tragen, dass das Programm nicht abbricht. Mit Hilfe von „Try Catch“ können Sie auf das Auftreten in geeigneter Weise reagieren.

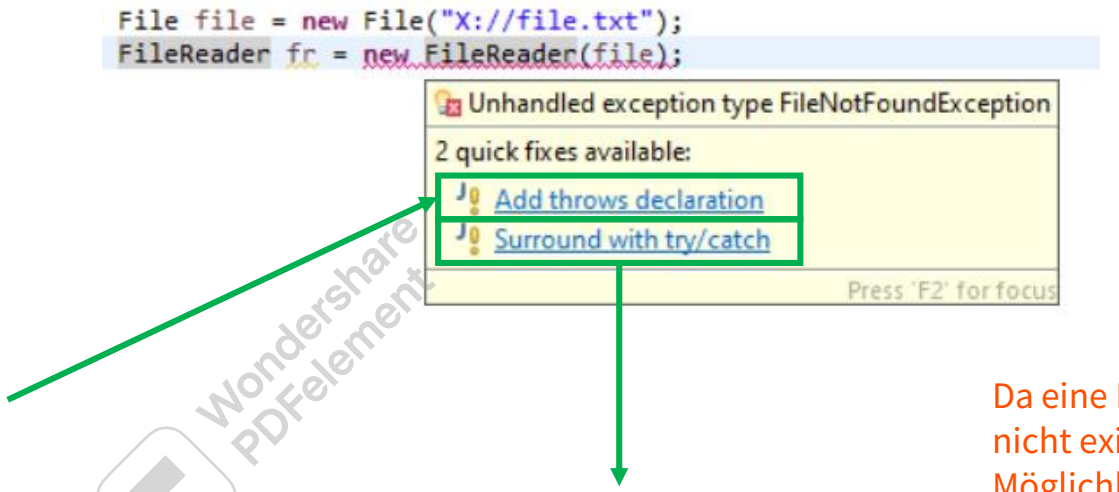
Wie Sie darauf reagieren wird im Kapitel „Exceptions abfangen & behandeln“ genauer erklärt.

1.2 Exceptions

- **Checked Exceptions**

Eclipse fordert Sie in einigen Fällen direkt auf, mögliche Risiken innerhalb des Try Catch abzufangen.

Auf die Möglichkeit „Add throws declaration“ kommen wir im späteren Verlauf des Kurses nochmal zu sprechen.




```
File file = new File("X://file.txt");
try {
    FileReader fr = new FileReader(file);
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Da eine Datei eventuell nicht existiert, muss diese Möglichkeit in Betracht gezogen und behandelt werden !!!!

1.3 Exceptions & Errors

Unchecked Exceptions

1.3 Exceptions

- **Unchecked Exceptions**

Als „Unchecked Exceptions“ werden Ausnahmen bezeichnet, welche zur Laufzeit auftreten, also Ausnahmen, welche **nicht** vom Compiler geprüft wird. Solche Ausnahmen sind in der Regel Programmierfehler, Logikfehler oder die fehlerhafte Nutzung von Funktionalitäten (z.B. Schnittstellen oder APIs).
Ein klassisches Beispiel ist der Zugriff auf Elemente eines Arrays, welche nicht existieren.
Dieser Fehler tritt erst beim Ausführen des Programms auf.

```
int nummmern[] = { 1, 2, 3, 4 };
System.out.println(nummmern[5]);
```

```
<terminated> test [Java Application] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (11.04.2022, 22:17:33)
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at test.main(test.java:27)
```

Das Vermeiden solcher Fehler ist nur durch Testen und gewissenhaftes Programmieren möglich, da Sie nicht jede Eventualität abfangen können.

1.4 Exceptions & Errors

Error

1.4 Exceptions

- **Errors**

Der Error (Fehler) ist eine Sonderform der Ausnahme (Unchecked Exception) und tritt ausschließlich zur Laufzeit auf. Errors können nicht während der Programmierung von Entwickler abgefangen werden (z.B. durch Try Catch). Erst nach ihrem Auftreten kann eine Fehleranalyse im Quellcode Aufschluss darüber geben, wo der Fehler genau herkommt.

Als klassisches Beispiel kann hier der Stapelüberlauf Fehler (StackOverflowError) genannt werden.

Das soll natürlich nicht heißen, dass der Entwickler während des Code-Schreibens nicht bereits dafür sorgen kann, dass der Fehler nicht auftritt. Er bekommt dabei einfach nur keine Unterstützung durch die IDE.

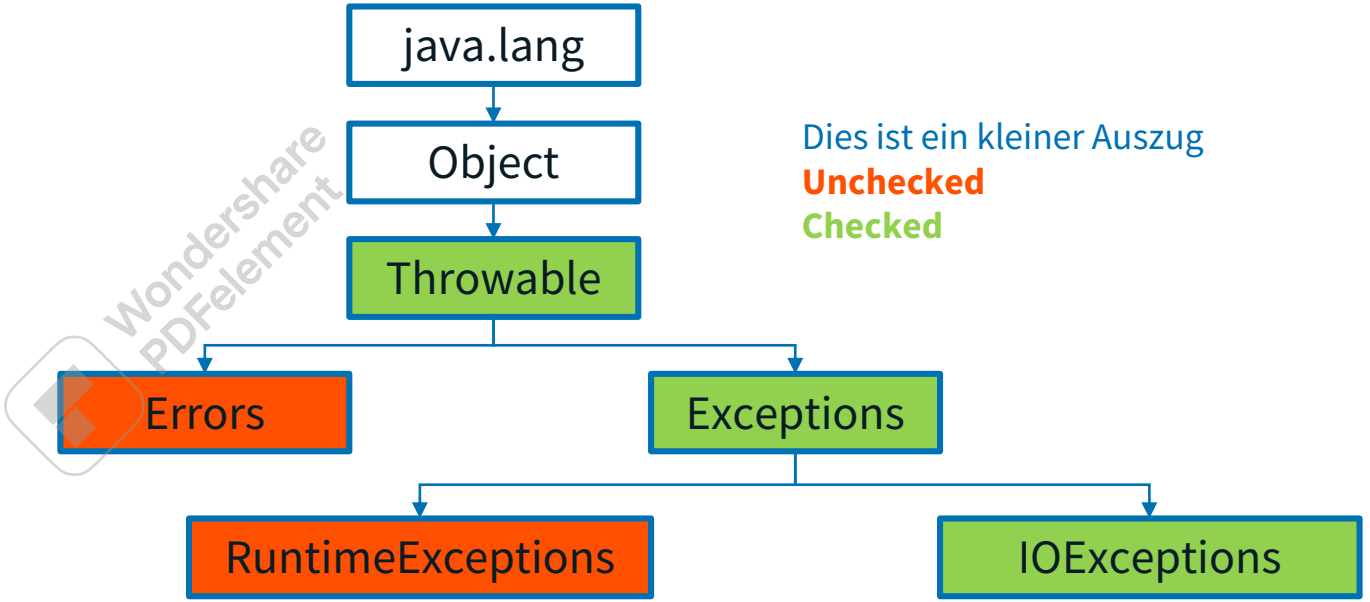
1.5 Exceptions & Errors

Ausnahmen Hierarchie

1.5 Exceptions

- **Ausnahmen Hierarchie**

Alle Exceptions und Errors unterliegen einer Hierarchie. Von Throwable leiten sich Errors sowie Exceptions ab. Trotz der Ableitung von Throwable, welches eine „Checked Exception“ ist, müssen nicht alle unterliegenden Exceptions ebenfalls „Checked Exceptions“ sein.



1.6 Exceptions & Errors

Exceptions abfangen & behandeln

1.6 Exceptions

- **Exceptions abfangen & behandeln**

Die Kontrollstruktur zum Abfangen von Exceptions ist das „try catch“. Innerhalb des Try Blocks steht der eventuell fehleranfällige Code, welcher eine Exception auslösen KÖNNTE (aber nicht zwingend muss). Sollte die Exception ausgelöst werden, wird sie (sofern eine passende Exception angegeben wurde) im Catch-Block abgefangen und das Programm kann ohne Abzubrechen fortfahren.
Für mehrere Catch-Anweisungen ist die Hierarchie der Exceptions zu beachten. Diese muss immer aufsteigend deklariert werden.

Aufsteigende Hierarchie der Exceptions:
FileNotFoundException
IST TEIL VON
IOException

```
try {
    FileInputStream file = new FileInputStream("Test.txt");
    byte x = (byte) file.read();
} catch (FileNotFoundException ex) {
    ex.printStackTrace();
} catch (IOException ex1) {
    ex1.printStackTrace();
}finally {
    System.out.println("finally");
}
```

VIELEN DANK!