Karlsruher Institut für Technologie

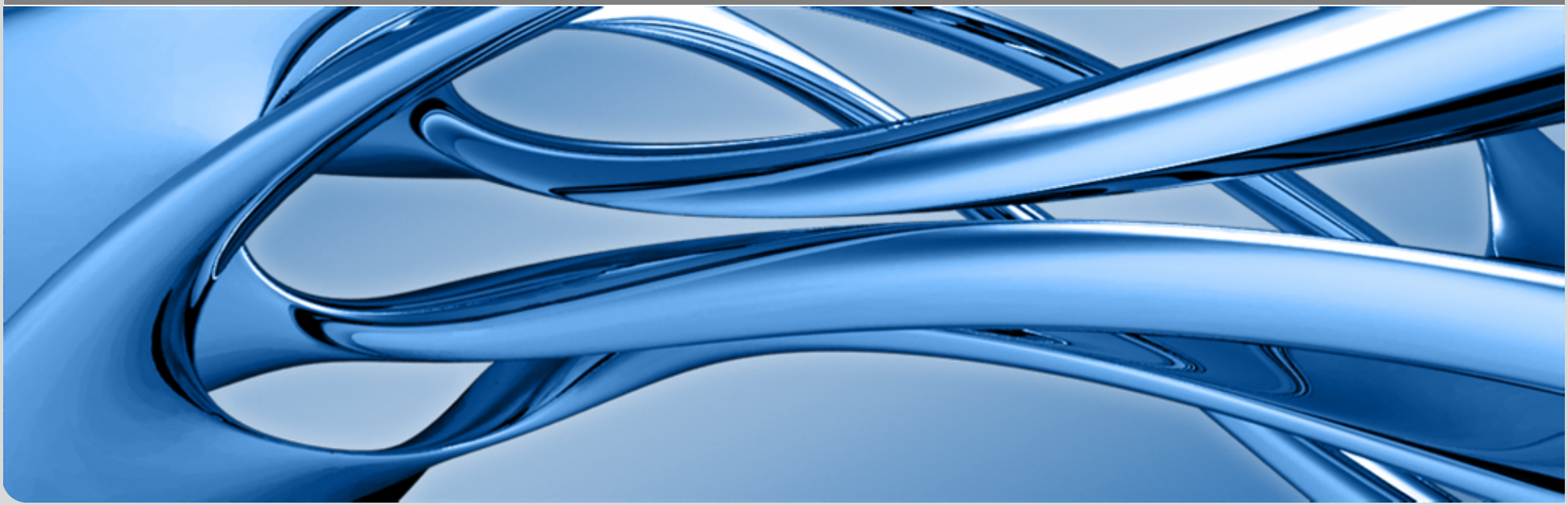
Software Design and Quality

Tutorium 9

Exceptions und Java API

Christian Zielke | 15. Januar 2018

CHAIR FOR SOFTWARE DESIGN AND QUALITY



Gliederung

1. Allgemeines
   - Übungsblatt 3
   - Präsenzübung
2. Ausnahmebehandlung (Exceptions)
   - Konzept
   - Hierarchie
   - Exception-Klasse
   - Eigene Exceptions
   - Syntax
   - Beispiel
   - finally
   - No-Go's

Gliederung

Übungsblatt 3

Aufgabe A

- Optional Checkstyle Fehler gab immer einen Punkt Abzug
- Keine Terminalaufrufe in List Klasse!
- NullPointerException

Präsenzübung

- Donnerstag 17:30-17:50 bzw. 18:05 - 18:25
- Keine Hilfsmittel erlaubt

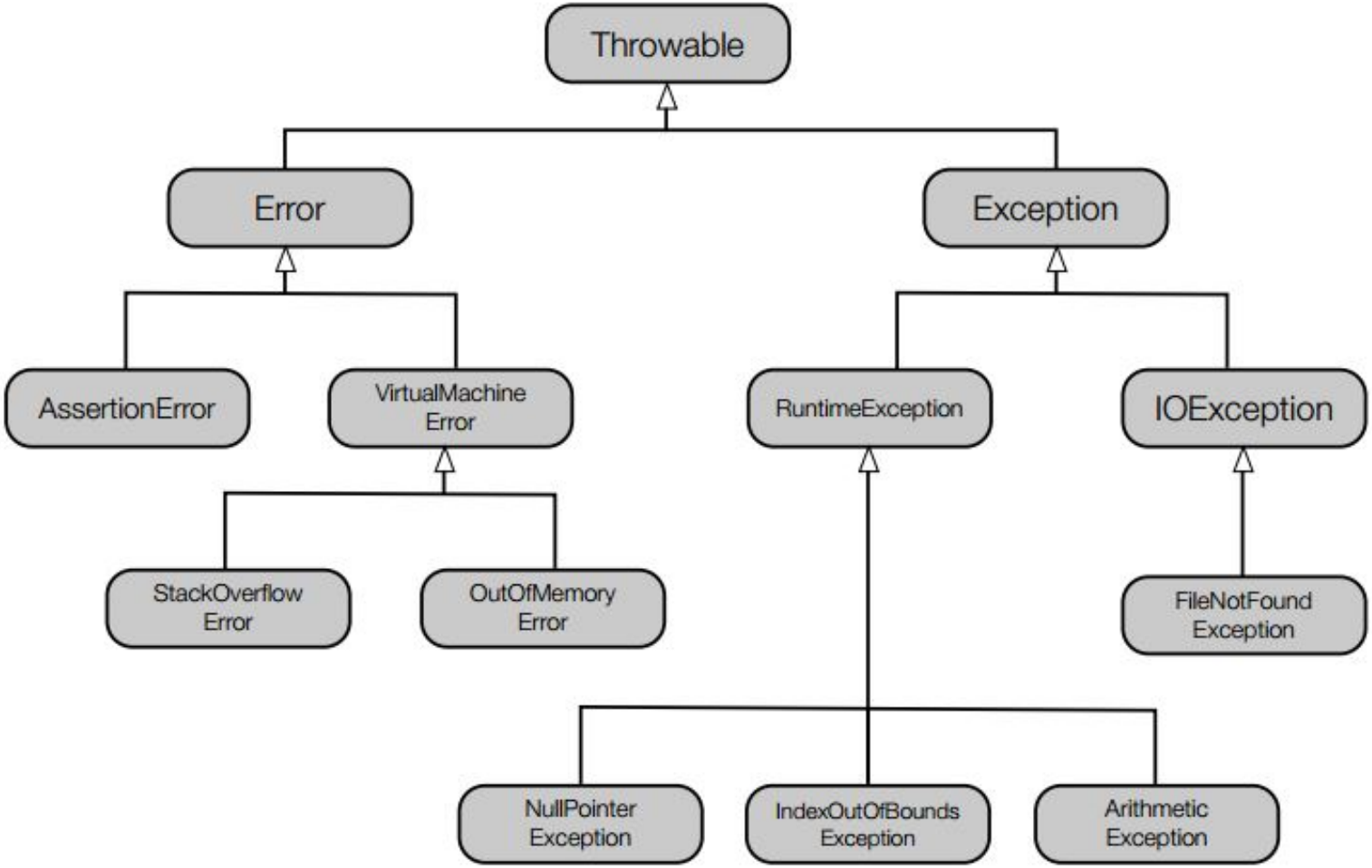
Wichtig

- Pünktlich sein!
- Ausweise mitbringen!

Exceptions - Konzept

- Exceptions dienen der Fehlerbehandlung
- Brechen den Programmfluss ab (ähnlich wie `return`)
- Trennen Algorithmik von Benutzerinteraktion
- Ermöglichen es, Fehler an anderer Stelle zu behandeln
- Können direkt behandelt oder “weitergereicht” werden
- Sind normale Objekte

Hierarchie


Throwable
Error
Exception
AssertionError
VirtualMachine Error
RuntimeException
IOException
StackOverflow Error
OutOfMemory Error
FileNotFound Exception
NullPointer Exception
IndexOutOfBounds Exception
Arithmetic Exception

Exception-Klasse

- Oberklasse von allen Exceptions
- Wichtige Methoden:
  - `String getMessage()`
    → Gibt die Fehlermeldung der Exception zurück
  - `void printStackTrace()`
    → Gibt den Aufrufstack aus
  - `StrackTraceElement[] getStackTrace()`
    → Gibt den Aufrufstack als Array von einzelnen Elementen zurück

Eigene Exceptions

- Da man vorhandene Exceptions nur für dafür vorgesehene Zwecke (siehe Java-API) verwenden soll, kann man eigene, aussagekräftige Exceptions implementieren
- Erben von `Exception` oder `RuntimeException`
- Implementieren mindestens die beiden Kontruktoren:
  - *`ExceptionName`*`()`
  - *`ExceptionName`*`(String message)`
- Bei Bedarf: Überschreiben von Methoden wie z.B. `getMessage()`
- Aufbauen von Exception-Hierarchien möglich

Syntax

- Erzeugung:

```
new ExceptionName(message);
```

- Auslösung:

```
throw ExceptionObject;
```

- Weitergabe:

```
public void method() throws ExceptionName {}
```

- Abfangen:

```
try {
    Dangerous Code
} catch (ExceptionName e){
    Exception Handling
} catch (SecondPossibleException e) {...}
```

Beispiel

```
1   int fac(int n) {
2     if (n < 0) {
3       throw new IllegalArgumentException("faculty for values less
              then zero is not defined!");
4     }
5     ...
6   }
```

→ Bei `fac(-1)` stürzt das Programm ab

```
1   int fac(int n) throws IllegalArgumentException {
2     if (n < 0) {
3       throw new IllegalArgumentException("faculty for values less
              then zero is not defined!");
4     }
5     ...
6   }
```

→ Compiler verlangt Abfangen per `try-catch` Block

Beispiel

```
1   public static void main(String[] args) {
2     if (args.length != 1) {
3       System.out.println("Wrong argument count!");
4     }
5     try {
6       System.out.println(fac(Integer.parseInt(args[0])));
7     } catch (IllegalArgumentException e){
8       System.out.println(e.getMessage());
9     } catch (NumberFormatException e) {
10      System.out.println(e.getMessage());
11    }
12  }
```

→ Wenn Exception auftritt, wird diese abgefangen und der passende `catch`-Block aufgerufen

finally

`finally` ermöglicht es, Code unabhängig davon auszuführen, ob etwas erfolgreich war oder nicht

```
1   public static void main(String[] args) {
2     try {
3       System.out.println(fac(-1));
4     } catch (IllegalArgumentException e){
5       System.out.println(e.getMessage());
6     } finally {
7       System.out.println("About to exit");
8     }
9   }
```

No-Go's: try-catch

```
1   try {
2     //Whole code
3   } catch (ExceptionName e) {
4     //Nothing
5   }
```

- Kein `try` um das **gesamte Programm**!
- Keine **leeren `catch`-Blöcke**!

No-Go's: catch everything!

- Fängt jede beliebige Art von Exception
- Nicht möglich zu erkennen, wo der Fehler liegt
  $\rightarrow$ Keine Behandlung möglich!
- Das Programm läuft weiter
  $\rightarrow$ Unvorhersehbare Konsequenzen
- **Grottiger Stil - NIEMALS!**

No-Go's: catch nothing!

- Das Schlimmste, was man tun kann:
  Fehler nicht behandeln!
- Macht Programm völlig unbenutzbar
- Exceptions, die nicht behandelt werden können, werden weiter "nach oben" geworfen

Java-API

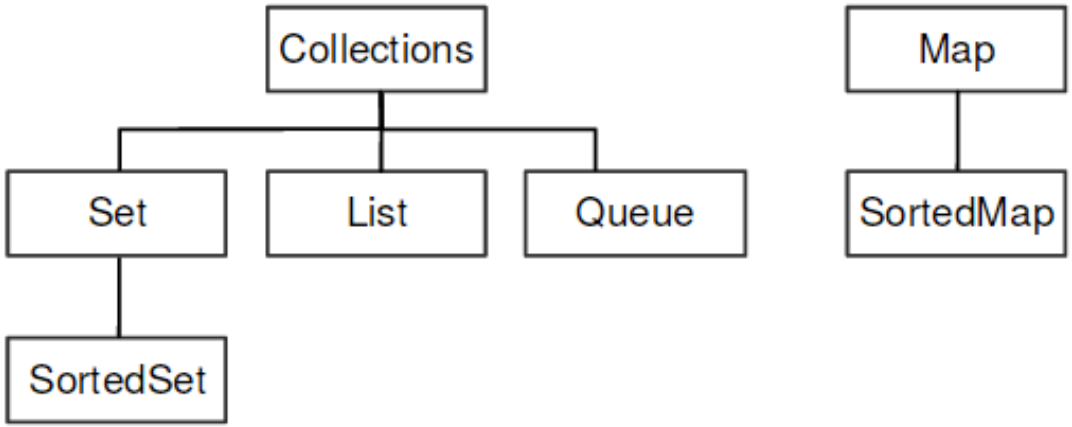
- Häufig benötigte Klassen und Pakete
- z.B. Listen muss nicht jeder selbst schreiben
- Wichtige Pakete:
  - *java.lang*
  - *java.util*
  - *java.io*

java.lang

- Grundfunktionalität:
  - Object
  - String
  - Enum
  - ...
- Math:
  - *sqrt(Number, Number)*
  - *pow(Number, Number)*
  - *abs(Number)*
  - ...
- Interface *Iterable<T>* - Voraussetzung für foreach Schleife
- ...

java.util

- Zeit- und Datumsfunktionen
- Java Collections Framwork
  - 14 Interfaces mit mehreren Implementierungen



Übungsaufgaben

FileReader

- implementiere die vorgegebenen Methoden
- erstelle eine eigene Exception FileSizeException
- teste ob dein ExceptionHandling korrekt funktioniert