sidion

SIDION

Zuhören. Analysieren. Beraten.

ÜBER MICH

Matthias Koch, sidion GmbH

matthias.koch@sidion.de

Expert Software Developer

Java

Clean Code

Funktionale Programmierung

VERMEIDUNG VON NULL POINTER EXCEPTIONS IN JAVA – THE COMPLETE GUIDE

Matthias Koch I Java Forum Stuttgart I 10.07.2025

sidion

URSPRUNG

Sir Charles Antony Richard Hoare (2009):

“I call it my billion-dollar mistake. It was the invention of the null reference in 1965. […]But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

DAS PROBLEM

```
Customer customer = new Customer("#4711", null);

Order order = new Order(customer,
          List.of(new Item("An old teddy bear", BigDecimal.TEN)));

String street = order.customer().address().street();
```

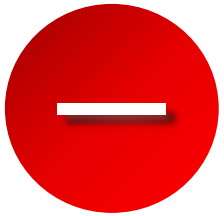
EINE SIMPLE LÖSUNG: ELVIS-OPERATOR

```
Customer customer = new Customer("#4711", null);

Order order = new Order(customer,
        List.of(new Item("An old teddy bear", BigDecimal.TEN)));

String street = order?.customer()?.address()?.street();
```

sidion

KOTLIN FÜR FEHLERANFÄLLIGE KLASSEN

+ Einfache Umstellung
+ Compiler erlaubt (in reinem Kotlin) keine NPE!!!
+ Restliche Anwendung kann so bleiben, wie sie ist
+ Ideal für Daten aus 3rd Party Application z.B. in Anti Corruption Layer



- Vorgaben können das nicht erlauben
- Konvertierte Klassen brauchen Nacharbeit
- Kotlin-Code ggf. schwer verständlich
- Fallstricke bei Mix Java-/Kotlin-Code
- Annotation-Processing (z.B. Lombok) funktioniert nicht

Defensives Programmmieren

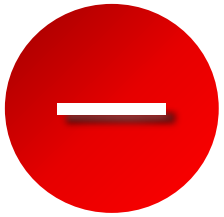
sidion

DEFENSIVES PROGRAMMMIEREN

```
String street = "Downingstreet 10";

if(order != null && order.customer() != null
                 && order.customer().address() != null) {
    street = order.customer().address().street();
}
```

DEFENSIVES PROGRAMMMIEREN

+ Funktioniert „out of the box“



- keine Compilerunterstüzung
- keine IDE-Unterstützung
- Code wird durch viele Null-Checks schwer lesbar

sidion

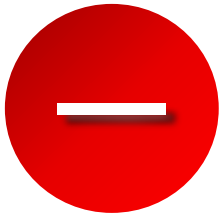
Verwendung von Optional

VERWENDUNG VON OPTIONAL

```
String street = order.customer()
                     .address()
                     .map(Address::street)
                     .orElse("Bakerstreet 221B");
```

VERWENDUNG VON OPTIONAL

+ (meist) gut lesbarer Code



- keine Compilerunterstützung
- keine komplette IDE-Unterstützung
- Fallstricke in der API
- Optionals wurden nicht zur Vermeidung von NPEs eingeführt

Null-Objekt Pattern

sidion

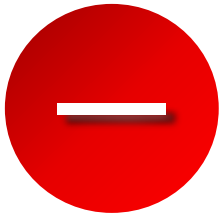
NULL-OBJEKT PATTERN

```
public sealed interface Address permits ConcreteAddress, NullAddress {
    String street();
    String city();
}
public record NullAddress(String city, String street) implements Address {
    @Override
    public String street() {
        return "SesameStreet 42";
    }
}
public record Customer(String customerNumber, Address address) {
    public Customer(String customerNumber, Address address) {
        this.address = (address != null) ? address : new NullAddress(null, null);
        this.customerNumber = customerNumber;
    }
}
```

NULL-OBJEKT PATTERN

Besser als Verwendung von Optional!

+ gut lesbarer Code



- keine Compilerunterstützung
- keine komplette IDE-Unterstützung
- mehr Klassen + Interfaces

LIFTING

Lift in funktionaler Programmierung:

- Erweitert eine Funktion auf einen allgemeineren Context

Mathematisch:

- Macht aus einer partiellen Funktion eine totale Funktion
  z.B. x/y ist für y = 0 undefiniert

Lift bewirkt:

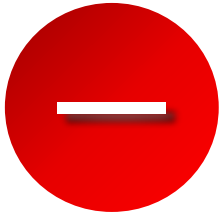
- `10/2 = Optional.of(5)`
  `10/0 = Optional.empty()`

LIFTING

```
Function<Order, String> processOrder = OrderService::processOrder;

Function<Order, Optional<String>> safeProcessOrder = SaveFunction.lift(processOrder);

log.info(safeProcessOrder.apply(order));
```

LIFTING

+ Leicht zu implementieren
+ zaubert NPEs und andere Exceptions zuverlässig weg
+ Ideal für Daten aus 3rd Party Application z.B. in Anti Corruption Layer
+ Typsicherheit
+ deklaratives Fehlerhandling
+ Composability
+ einfaches Testing

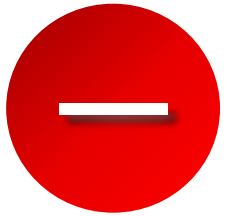


- Performance
- Overhead
- Kein fein granulares Exception Handling

Checker-Framework

CHECKER-FRAMEWORK

+ Kann NPEs zur Compilezeit erkennen!!!
+ Sauberer Code



- Konfigurationsaufwand
- Ist mit Lombok schwierig

WEITERE TOOLS

- Error Prone von Google
- Null Away von UBER
- SonarQube
- PMD
- FindBugs wird nicht weiterentwickelt
- SpotBugs

VALUE TYPES

- Projekt Valhalla
- „codes like a class, behaves like an int“
- Keine Identity
- Immutable
- Nicht nullable!!!
- Perfomanceverbesserung

VALUE TYPES

```
primitive class Address {
    private final String street;
    private final String city;

    Address(String street, String city) {
        this.street = street;
        this.city = city;
    }

    public Address move(String street, String city) {
        return new Address(street, city);
    }
}
```

SUMMARY

- Kein Königsweg
- Viele verschiedene Lösungsmöglichkeiten
- Ausblick Value Types

FRAGEN

Mitdenken

Mitgestalten

Mitarbeiten

bei sidion

sidion

QUELLENANGABEN

- https://checkerframework.org/
- Talk von Stuart Marks über Optionals: https://www.youtube.com/watch?v=Ej0sss6cq14
- https://errorprone.info/
- https://github.com/uber/NullAway
- https://www.sonarsource.com
- https://pmd.github.io/
- https://findbugs.sourceforge.net/
- https://spotbugs.github.io/

sidion

VIELEN DANK