Funktionale und objektorientierte Programmierkonzepte

TECHNISCHE UNIVERSITÄT DARMSTADT

Präsenz-Sprechstunde B

Simon Hock, Nhan Huynh, Daniel Mangold

Überblick

TECHNISCHE UNIVERSITÄT DARMSTADT

Fehlerbehandlung [20min]
- Throwable
- Error
  - AssertionError
- Exception
- RuntimeException
- Exception werfen
- Exception weiterreichen
- Exception fangen

Arbeitsphase [80min]

Fehlerbehandlung

TECHNISCHE UNIVERSITÄT DARMSTADT

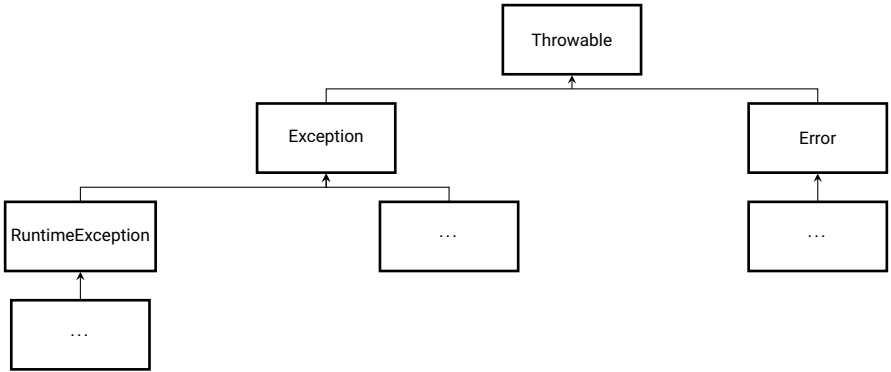

Abbildung: Java Exception Hierarchy

Throwable

TECHNISCHE UNIVERSITÄT DARMSTADT

- Basisklasse für `Exception` und `Error`
- Bietet `try-catch` Konstrukt an, um Fehlerzustände zu behandeln
  - Funktioniert für `Throwable` und alle direkt oder indirekt abgeleiteten Klssen von `Throwable`

```
1 int a = 5, b = 0;
2 try {
3   int c = a / b;
4   System.out.println(c);
5 } catch (ArithmeticException e) {
6   System.err.println("Divisor cannot be zero!");
7 }
```

Error

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wird typischerweise vom Laufzeitsystem geworfen
- Fälle: Fehler welche nach menschlichen Ermessen so gewichtig ist, dass wohl keine sinvolle Fehlerbehandlung möglich ist, sondern der Programmabbruch die beste Lösung zu sein scheint.
- `Error`-Klassen müssen nicht mit `try-catch`-Block gefangen werden
- Formaler Aufbau (Werfen eines Errors):

```
throw new Errorklassenname(Parameter);
```

- Subklassennamen enden mit `Error`

AssertionError

TECHNISCHE UNIVERSITÄT DARMSTADT

- Spezielle abgeleitete Error-Klasse
- Java bietet eine sehr bequeme Sonderform mit AssertionError umzugehen
- Schlüsselwort: `assert`
  - Müssen aktiviert werden!
- Formaler Aufbau:

```
1 if (condition) {
2   throw new AssertionError(message);
3 }
4 // Short form
5 assert !condition:message;
```

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 if (n < 0 || n % 2==1) {
2   throw new AssertionError("Bad n!");
3 }
4 // Short form
5 assert n >= 0 && n % 2==0:"Bad n!";
```

Exception

TECHNISCHE UNIVERSITÄT DARMSTADT

- „Checked Exception“
- Wird vom Compiler entdeckt und muss
  - behandelt werden oder
  - weitergereicht werden
- Formaler Aufbau:

```
throw new Exceptionklassenname(Parameter);
```

- Subklassennamen enden mit `Exception`

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

- `FileReader` Konstruktor wirft eine `Exception`, falls die Datei nicht existiert

```
1 File file = new File("path-to-file");
2 try {
3   BufferedReader reader = new BufferedReader(new FileReader(file));
4 } catch (FileNotFoundException e) {
5   System.err.println("File could not be found!");
6 }
```

Wieso eigene Exceptionklassen definieren?

TECHNISCHE UNIVERSITÄT DARMSTADT

Frage: Ist es sinnvoll, eigene Exceptionklassen zu definieren? Welche Vorteile ergeben sich hieraus?

Wieso eigene Exceptionklassen definieren?

TECHNISCHE UNIVERSITÄT DARMSTADT

Frage: Ist es sinnvoll, eigene Exceptionklassen zu definieren? Welche Vorteile ergeben sich hieraus?

Antwort: Bei eigenen Exception-Klassen können Informationen an den Konstruktor übergeben werden, was die Fehlerbehandlung erleichtern kann und zum Beispiel für eine detaillierte Ausgabe der Probleme sorgt.

RuntimeException

TECHNISCHE UNIVERSITÄT DARMSTADT

- Fehler, die erst zur Laufzeit auftreten und dann zum Abbruch der Ausführung des Programms führen
- Kann nicht vom Compiler entdeckt werden
- Laufzeitsystem bricht Ablauf ab
- Call-Stack mit Zeilennummern wird in der Fehlermeldung angezeigt
  - Man sieht sofort, wo der Fehler aufgetreten ist

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 public class Main {
2
3     public static void main(String[] args) {
4         int a = 5, b = 0;
5         int c = a / b;
6     }
7 }
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Main.main(Main.java:5)
```

Exception werfen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schlüsselwort: `throw`
- Leitet im Methodenkörper eine `Exception` ein bzw. eine `Exception` wird geworfen
- Eine Exception ist ganz konkret ein Objekt der gewählten `Exception`-Klasse und muss daher mit `new` erzeugt werden, bevor sie geworfen werden kann.
- `throw`-`Klausel` beendet die Ausführung einer Methode
  - Im Unterschied zu return wird kein Wert zurückgeliefert, sondern das in der Anweisung erzeugte `Exception`-Objekt wird geworfen.

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Person {

    final String name;

    public Person(String name) {
        if (name==null) {
            throw new NullPointerException("Name cannot be null!");
        }
        this.name = name;
    }
}
```

Alternativ

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 if (name==null) {
2   NullPointerException e = new NullPointerException("Name cannot be null!");
3   throw e;
4 }
```

Exception weiterreichen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schlüsselwort: `throws`
- Im Methodenrumpf: Gibt an, dass diese Methode potenziell eine Exception werfen kann
- Falls mehrere Exceptions weitergereicht werden, können diese mit einem Komma getrennt werden
- Eine Methode muss eine Exception nicht unbedingt fangen, sondern kann diese dem Aufrufer weiterreichen
- Formaler Aufbau:

```
Modifiers Rückgabetyp Methodenname(Parameter)throws Exceptionname, ...
```

- Exceptionname in der `throws`-Klausel muss nicht mit der geworfenen Exception übereinstimmen
  - Geworfene Exception kann eine direkt oder indirekt abgeleitete Exception von der `throws`-Klausel sein

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public static int factorial(int n) throws IllegalArgumentException {
    if (n < 0) {
        throw new IllegalArgumentException("n cannot be negative!");
    }
    int sum = 1;
    for (int i = 1; i <= n; i++) {
        sum *= i;
    }
    return sum;
}
```

Exception fangen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schlüsselwort: `try` und `catch`
- Exception behandeln
- `try`-Block: Codeauschnitt, dass potenziell eine Exception werfen kann
- `catch`-Block: Name der Exception, welche gefangen werden soll
  - Mehrere `catch`-Blöcke werden untereinander mit ihren eigenen Block eingeführt
  - Ein Block kann mehrere Exception fangen und werden mit | (oder) getrennt
  - Wenn eine Exception geworfen wird, dann wird die Ausführung beendet und ein passender `catch`-Block wird ausgewertet.

Formaler Aufbau

TECHNISCHE UNIVERSITÄT DARMSTADT

- Falls es keine passenden `catch`-Blöcke gibt, dann werden alle Blöcke ignoriert
- Auswertung ist von oben nach unten, d.h. wird eine Exception im ersten `catch`-Block behandelt, so werden die Operationen in dieser ausgeführt und die anderen `catch`-Blöcke werden nicht mehr beachtet.

```
1 try {
2   // Code that can potentially throw an exception
3 } catch (Exceptioname1 e) {
4   // Execution of operations if an exception 1 is caught
5 } catch (Exceptioname2 e) {
6   // Execution of operations if an exception 2 is caught
7 }
```

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  int[] a = {1, 2, 3, 4, 5};
2  try {
3    System.out.println(a[6]);
4    for (int n : a) {
5      System.out.println(factorial(n));
6    }
7  } catch (IndexOutOfBoundsException | NullPointerException e) {
8    System.err.println("Invalid task");
9  } catch (IllegalArgumentException e) {
10   throw e;
11 }
```

`try-catch`: Exception vs. RuntimeException

TECHNISCHE UNIVERSITÄT DARMSTADT

Frage: Alle Exceptions, die nicht von der Klasse `RuntimeException` direkt oder indirekt abgeleitet sind, müssen spätestens in der `main`-Methode behandelt werden. Wieso müssen `RuntimeException` nicht unbedingt behandelt werden?

`try-catch`: Exception vs. RuntimeException

TECHNISCHE UNIVERSITÄT DARMSTADT

Frage: Alle Exceptions, die nicht von der Klasse `RuntimeException` direkt oder indirekt abgeleitet sind, müssen spätestens in der `main`-Methode behandelt werden. Wieso müssen `RuntimeException` nicht unbedingt behandelt werden?

Antwort: Müsste man all diese Exception fangen, so wäre der komplette Code mit `try-catch` Blöcken versehen werden und der Code wäre nicht mehr zu lesen. Deshalb müssen sie nicht gefangen oder geworfen werden und werden automatisch geworfen.

```
1 Animal animal = new Dog();
2 Dog dog = (Dog) animal;
3 int a = 10;
4 int b = 42;
5 int c = a / b;
```

```java
Animal animal = new Dog();
try {
  Dog dog = (Dog) animal;
} catch (ClassCastException e) {
  // ...
}
int a = 10;
int b = 42;
try {

  int c = a / b;
} catch (ArithmeticException e) {
  // ..
}
```

Selbstständiges Arbeiten