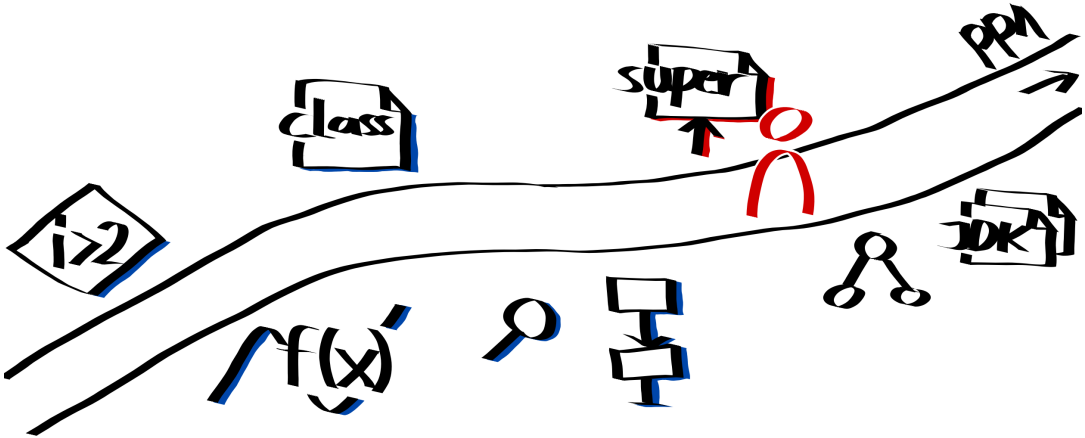
Kapitel 6: Vererbung & Fehlerbehandlung

VL 22: Fehlerbehandlung

Wintersemester 2023/24

Stand 8. Januar 2024, 07:58 Uhr

Wo stehen wir gerade?



Vorlesungsfeedback

Welche Fehler können zur Laufzeit auftreten?

Welche Fehler können zur Laufzeit auftreten?

Heinrich Heine
Universität Düsseldorf

- Division durch 0
- Zugriff auf negativen Array-Index
- Parsen von `"zwei"` zu einem Integer
- Lesen einer Datei, die nicht existiert
- Verbindungsabbruch während Datei-Download

Welche Fehler können zur Laufzeit auftreten?

Heinrich Heine
Universität Düsseldorf

- Division durch 0
- Zugriff auf negativen Array-Index
- Parsen von `"zwei"` zu einem Integer
- Lesen einer Datei, die nicht existiert
- Verbindungsabbruch während Datei-Download

Kann man all diese Fehler schon beim Programmieren verhindern?

Welche Fehler können zur Laufzeit auftreten?

- Division durch 0
- Zugriff auf negativen Array-Index
- Parsen von `"zwei"` zu einem Integer
- Lesen einer Datei, die nicht existiert
- Verbindungsabbruch während Datei-Download

Kann man all diese Fehler schon beim Programmieren verhindern?
Sollte man beim Programmieren bei bestimmten Fehlern gezwungen sein, angemessen zu reagieren?

Erster Lösungsversuch

Erster Lösungsversuch

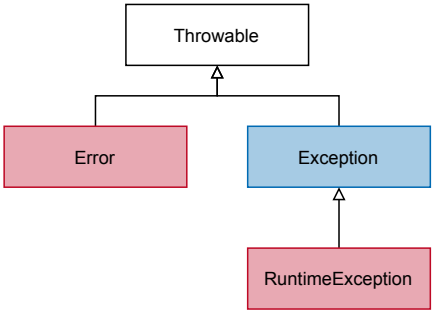
- Spezieller Rückgabewert (z. B. -1) im Fehlerfall
- Probleme:

Erster Lösungsversuch

- Spezieller Rückgabewert (z. B. -1) im Fehlerfall
- Probleme:
  - Verwechslung mit Funktionswerten möglich
  - einfach, Fehlerbehandlung zu vergessen
  - Fehlercodes ggf. durchzureichen
  - schwierig nachzuhalten, welcher Wert für welche Art Fehler steht

Exceptions-Hierarchie

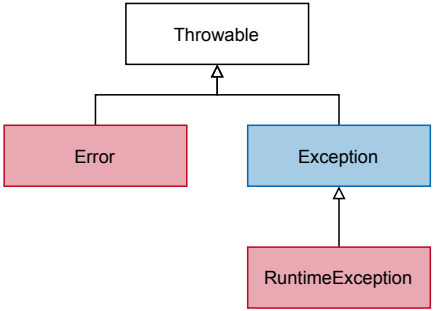
- `RuntimeException`
  - i. A. Hinweis auf Programmierfehler
  - ArrayIndexOutOfBoundsException, ArithmeticException, NullPointerException, NumberFormatException . . .
- `Error`
  - schwerwiegende Fehler, von denen sich Programm i. A. nicht erholen kann
  - OutOfMemoryError, StackOverflowError, AssertionError, . . .
- `Exception` (die keine RuntimeException ist)
  - i. A. Fehlerfälle, die zur Laufzeit immer bedacht werden sollten
  - NoSuchFileException, AccessDeniedException, . . .



Checked und Unchecked Exceptions

- Unchecked Exceptions: Errors, RuntimeExcpetions
  - keine Behandlung erzwungen
- Checked Exceptions: Exceptions, die keine RuntimeExceptions sind
  - Behandlung durch Compiler erzwungen

  → abfangen oder explizit weiterreichen



Was passiert hier?

```
5 String number = "two";
6 Integer.parseInt(number);
7
8 System.out.println("Programm läuft weiter ?");
```

Was passiert hier?

Heinrich Heine Universität Düsseldorf

```
5 String number = "two";
6 Integer.parseInt(number);
7
8 System.out.println("Programm läuft weiter ?");
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string:
  "two"
        at java.base/java.lang.NumberFormatException
           .forInputString(NumberFormatException.java:68)
        at java.base/java.lang.Integer.parseInt(Integer.java:658)
        at java.base/java.lang.Integer.parseInt(Integer.java:776)
        at NFException.main(NFException.java:6)
```

`try-catch`: Exceptions abfangen

- Exception innerhalb von try-Block abfangbar
  - falls nirgendwo abgefangen: Programmabsturz (präziser: aktueller Thread sofort beendet)
- catch-Block im Fehlerfall ausgeführt
  - catch fängt Exceptions der angegebenen Klasse und aller Unterklassen ab

```
1 String number = "two";
2
3 try {
4     Integer.parseInt(number);
5 } catch(NumberFormatException exception) {
6     System.out.println("Error while parsing " + number);
7 }
8
9 System.out.println("Programm läuft weiter ...");
```

`try-catch`: Exceptions abfangen

- Exception innerhalb von try-Block abfangbar
  - falls nirgendwo abgefangen: Programmabsturz (präziser: aktueller Thread sofort beendet)
- catch-Block im Fehlerfall ausgeführt
  - catch fängt Exceptions der angegebenen Klasse und aller Unterklassen ab

```java
String number = "two";

try {
    Integer.parseInt(number);
} catch(NumberFormatException exception) {
    System.out.println("Error while parsing " + number);
}

System.out.println("Programm läuft weiter ...");
```

```
Error while parsing two
Programm läuft weiter ...
```

Exception-Details

- Mehr Details über Throwable-Objekte u. a. mit
  - `Throwable.getMessage()`
  - `Throwable.printStackTrace()`

```
1  String number = "two";
2
3  try {
4      Integer.parseInt(number);
5      System.out.println(number);
6  } catch(NumberFormatException e) {
7      System.out.println(e.getMessage());
8      e.printStackTrace();
9  }
10
11 System.out.println("Programm läuft weiter ...");
```

Exception-Details

- Mehr Details über Throwable-Objekte u. a. mit
  - `Throwable.getMessage()`
  - `Throwable.printStackTrace()`

```
For input string: "two"
java.lang.NumberFormatException: For input string: "two"
        at java.base/java.lang.NumberFormatException
           .forInputString(NumberFormatException.java:68)
        at java.base/java.lang.Integer.parseInt(Integer.java:658)
        at java.base/java.lang.Integer.parseInt(Integer.java:776)
        at DetailsNFException.main(DetailsNFException.java:8)
Programm läuft weiter ...
```

Ist das hier eine gute Idee?

Heinrich Heine Universität Düsseldorf

```java
int[] numbers = new int[10];
String userInput = "4";

try {
    numbers[10] = Integer.parseInt(userInput);
} catch(Exception e) {
    System.out.println("invalid user input");
}
```

Ist das hier eine gute Idee?

Heinrich Heine Universität Düsseldorf

```java
int[] numbers = new int[10];
String userInput = "4";

try {
    numbers[10] = Integer.parseInt(userInput);
} catch(Exception e) {
    System.out.println("invalid user input");
}
```

```
invalid user input
```

Exceptions so genau wie möglich fangen

Heinrich Heine Universität Düsseldorf

→ verhindert versehentliches Fangen anderer Fehler

```java
int[] numbers = new int[10];
String userInput = "4";

try {
    numbers[10] = Integer.parseInt("4");
} catch(NumberFormatException e) {
    System.out.println("invalid user input");
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 out
  of bounds for length 10
       at CatchExceptionPrecise.main(CatchExceptionPrecise.java:9)
```

Ist das eine gute Idee?

Heinrich Heine Universität Düsseldorf

```
1  public static void main(String[] args) {
2
3      int number = 0;
4      try {
5          number = Integer.parseInt(args[0]);
6      } catch (NumberFormatException exception) {
7      }
```

Ist das eine gute Idee?

Heinrich Heine Universität Düsseldorf

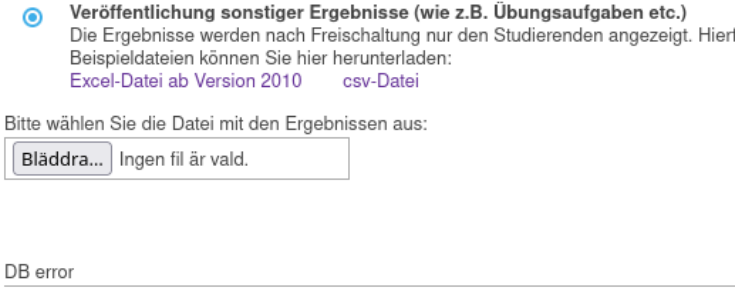
```java
public static void main(String[] args) {

    int number = 0;
    try {
        number = Integer.parseInt(args[0]);
    } catch (NumberFormatException exception) {
    }
```

Keine angemessene Fehlerbehandlung

- mindestens sprechende Fehlermeldung ausgeben
- sicherstellen, dass `number` sinnvoll gesetzt
  - falls nicht möglich: Programm beenden

Geht so Fehlerbehandlung?



Welche Variante ist schöner?

Heinrich Heine Universität Düsseldorf

```java
int[] numbers = {1, 2, 3, 4};
int index = 4;

if(index >= 0 && index < numbers.length) {
    System.out.println(numbers[index]);
} else {
    System.out.println("Index ungültig");
}
```

```java
int[] numbers = {1, 2, 3, 4};
int index = 4;

try {
    System.out.println(numbers[index]);
} catch(ArrayIndexOutOfBoundsException e) {
    System.out.println("Index ungültig");
}
```

Welche Variante ist schöner?

Heinrich Heine Universität Düsseldorf

```java
int[] numbers = {1, 2, 3, 4};
int index = 4;

if(index >= 0 && index < numbers.length) {
    System.out.println(numbers[index]);
} else {
    System.out.println("Index ungültig");
}
```

```java
int[] numbers = {1, 2, 3, 4};
int index = 4;

try {
    System.out.println(numbers[index]);
} catch(ArrayIndexOutOfBoundsException e) {
    System.out.println("Index ungültig");
}
```

Faustregel: Keine RuntimeExceptions abfangen. (Ausnahme: NumberFormatException)

Fangen mehrerer Fehler

- `NoSuchFileException` ist Unterklasse von `IOException`

⇒ `NoSuchFileException` muss zuerst behandelt werden (sonst: Compilerfehler)

```
1 try {
2     Files.delete(Paths.get("/tmp/progra"));
3 } catch (NoSuchFileException e) {
4     System.out.println("/tmp/progra does not exist");
5 } catch (IOException e) {
6     System.out.println("error while deleting /tmp/progra");
7     System.out.println(e.getMessage());
8 }
```

(Achtung: löscht ohne Rückfrage!)

`finally`: Aufräumarbeiten

- finally-Block immer ausgeführt, unabhängig vom Auftreten einer (gefangenen oder nicht gefangenen) Exception
- typische Aufgaben: Schließen geöffneter Dateien, Datenbankverbindungen . . .

```
1  String line = "";
2  BufferedReader reader = null;
3  try {
4      reader = new BufferedReader(new FileReader(path));
5      line = reader.readLine();
6  } catch(IOException e) {
7      System.out.println("Error ;)");
8  } finally {
9      if(reader != null) {
10         reader.close();
11     }
12 }
```

Es gibt noch mehr zu entdecken . . .

- try-with-resources: Schließt Ressourcen automatisch

```
String line = "";
try(BufferedReader reader = new BufferedReader(new FileReader(path))) {
    line = reader.readLine();
} catch(IOException e) {
    System.out.println("Error ;) (don't do it like this)");
}
```

- gleiche Behandlung verschiedener Exceptions

```
} catch (NoSuchFileException | DirectoryNotEmptyException e) {
    System.out.println("/tmp/progra does not exist or is not empty");
}
```

`throws`: Weiterreichen von Exceptions

Heinrich Heine Universität Düsseldorf

- falls Behandlung einer Checked Exception in Methode nicht sinnvoll machbar: explizit weiterreichen

```
1 public static void removeFolder(String path) throws IOException {
2     Files.delete(Paths.get(path));
3 }
```

`throw`: selbst Exceptions werfen

Heinrich Heine Universität Düsseldorf

- wirft Exception-Objekt

```
1 public T first() {
2     if(head == null) {
3         throw new java.util.NoSuchElementException();
4     }
5     return head.element;
6 }
```

Typische Exceptions aus dem JDK

Auswahl „fertiger“ Exceptions, die man selbst verwenden kann[1]:

- ArithmeticException
- IndexOutOfBoundsException
- IllegalArgumentException
- java.util.NoSuchElementException

[1] Vollständige Liste: `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Exception.html`

Eigener Fehlertext

Heinrich Heine Universität Düsseldorf

- viele Exception-Klassen nehmen Fehlerbeschreibung als Konstruktor-Argument

```java
public T getFirst() {
    if(head == null) {
        throw new java.util.NoSuchElementException("list is empty");
    }
    return head.element;
}
```

```
Exception in thread "main" java.util.NoSuchElementException: list is empty
        at List.getFirst(List.java:24)
        at List.main(List.java:31)
```

Eigene Exception-Klassen

- Erstellen eigener Exceptions durch Erben von einer Excpetion-Klasse
- nur notwendig, wenn passende Exception nicht schon in JDK vorhanden

```java
public class EmptyListException extends RuntimeException {

    public EmptyListException(String message) {
        super(message);
    }

}
```

Assertions

- Sicherstellen/Dokumentation von Bedingungen, die innerhalb einer Klasse garantiert gelten **müssen**
  - z. B. Invarianten von Algorithmen
  - ! nicht zur Prüfung von Parametern von öffentlichen Methoden
- Entwicklungswerkzeug, standardmäßig Assertions nicht geprüft
  - Aktivierung beim java-Aufruf mit `-ea`
- Verletzte Assertion wirft AssertionError

```
1    public static int ggt(int a, int b) {
2        int result = 0;
3        // ...
4        assert a % result == 0;
5        assert b % result == 0;
6        return result;
7    }
```

Lernziele

Heinrich Heine
Universität Düsseldorf

Sie können am Ende der Woche ...

- eine Exception **abfangen**.
- Exceptions **werfen**.

Wichtige Begriffe

Exception try catch

throw new IllegalArgumentException()

Übungsblatt

- BugHunt: Generische Liste
- Weitere Widgets
- Matrixmultiplikation

Vortrag: Qubits, QPUs und Quantencomputer

Heinrich Heine
Universität Düsseldorf

- Montag, 15.01.24, 14:30 Uhr, Hörsaal 5D
- Grundprinzipien des Quantencomputings, Quantengatter und Quantenalgorithmen (u. a. zur Verschlüsselung)
- explizit auch für Informatik-Erstsemester geeignet