Mentoriat: Einführung in die Objektorientierte Programmierung

Andreas Paech

Wintersemester 2021

Inhaltsverzeichnis

1 Vorstellung Andreas Paech **2**

2 Aufbau und Termine **2**

3 Zoom **3**

4 Grundlagen Programmierung **3**

4.1 Wissensstand: Mentimeter 3
4.2 Terminal und IDE 4
4.3 Hello World 4
4.4 Aufgabe: Hashes 4
4.5 Schleifen 5
4.6 Aufgabe: Raketen 5
4.7 Klassen, Objekte und Funktionen 6
4.8 Aufgabe: GreaterThanThree 6
4.9 Vordefinierte Klassen 7
4.10 Aufgabe: MaxNumber 7
4.11 Aufgabe: NumberSquare 8

5 OOP / Tic Tac Toe **8**

5.1 Tic Tac Toe 8
5.1.1 Use Case: Neues Spiel starten 8
5.1.2 Use Case: Spielzug 9
5.1.3 Quellcode 10
5.1.4 Use Case: Menü 10
5.1.5 Use Case: Namen eingeben 10
5.1.6 Use Case: Scoreboard anzeigen 10

5.1.7 Use Case: Programm verlassen 10
5.2 Theoretische Inhalte 11

6 AWT / Hangman **11**

7 Ausgewählte Themen / Parallelität und Verteilung **11**
7.1 Grundlagen Parallelität 11
7.2 Exkurs: ISO OSI-Modell 12
7.3 Client-Server und Threads 14
7.4 Remote Procedure Call 14
7.5 Demo und Aufgaben 14

8 Klausurvorbereitung **14**
8.1 Programmierung 15
8.2 Grundlagen Objekte und Vererbung 15
8.3 Typisierung 15
8.4 Frameworks, Bibliotheken, Packages und deren Integration 15
8.5 AWT 15
8.6 Parallelität, Streams und Verteilung 15

1 Vorstellung Andreas Paech

- Studium (Wirtschafts-)Informatik und BWL an der FernUniversität in Hagen
- Laufende Promotion an der Universität Hamburg
- Lehrbeauftragter an mehreren Hochschulen für Software-Entwicklung und IT-Management sowie in Coding Bootcamps für Quereinsteiger
- 11 Jahre Berufserfahrung als Software-Entwickler und Führungskraft

2 Aufbau und Termine

Das Mentoriat findet online über Zoom statt. Der Termine sind Donnerstag abends. Es werden immer unterschiedliche Themen behandelt. Dabei wird sich am Anfang nicht streng an die Reihenfolge der Kurseinheiten orientiert, weil die ersten drei Kurseinheiten die gleichen Inhalte mehrmals betrachten (zunächst theoretisch, dann praktisch, z.T. Vertiefung).

Es gibt nur einen minimalen Vorlesungsanteil in den Mentoriaten. Stattdessen werden vor allem Übungen gemacht (vergleichbar zum immersiven Lernen von Sprachschulen). **Programmierung lernt man nur, wenn man es macht!** Lesen und verstehen

reicht nicht aus. Die Übungsaufgaben werden in kleinen Breakout-Rooms / Kleingruppen bearbeitet und anschließend im Plenum besprochen. In der Klausurvorbereitung (voraussichtlich letzte Veranstaltung) wird eine Auswahl von alten Klausuraufgaben bearbeitet.

Das Ziel des Mentoriats ist das Verständnis der Inhalte und die Vorbereitung auf die Klausur. Dies ist keine Vorlesung, d.h. wir haben Zeit und Raum für Fragen!

Ein paar Worte zur Klausur: diese besteht vor allem aus praktischen Programmieraufgaben! Der Lehrstuhl von Prof. Dr. Steimann stellt die letzten Klausuren inkl. Musterlösung[1] zur Verfügung.

3 Zoom

Wir nutzen im Online-Semester das Tool Zoom für die Mentoriate. Dabei gibt es ein paar Grundregeln:

- **Kamera an.** Ich sage dazu immer: in der Präsenzveranstaltung haben Sie auch keinen schwarzen Vorhang! Dies ist für mich als Lehrenden und für Ihre Kommilitonen wichtig, damit wir Ihre Gestik und Mimik erleben. Andernfalls fällt die Interpretation des Gesagten wesentlich schwerer (und ist mit ein Grund der „Zoom-Fatique"). Es fehlt auch sonst das Feedback, anhand ich als Lehrender einschätzen kann, ob Sie das Gesagte verstanden haben.
- **Mikrofon aus.** Sie sollen sich immer „muten" außer für einen Redebeitrag, damit die Audioqualität im höchsten ist.
- Nutzen Sie die virtuellen Hintergründe von Zoom, sofern Sie einen neutralen Videohintergrund nutzen möchten.
- Wenn Sie Verbindungsprobleme haben, gilt das zuvor gesagte natürlich nicht! Schalten Sie dann Ihr Video aus und deaktivieren Sie auch, dass Sie Video empfangen. Erfahrungsgemäß betrifft dies jedoch nur einen sehr kleinen Anteil der Teilnehmenden.

4 Grundlagen Programmierung

4.1 Wissensstand: Mentimeter

- Welche Kurseinheiten haben Sie komplett gelesen?
- Welche Einsendeaufgaben haben Sie bearbeitet?

[1] `https://www.fernuni-hagen.de/ps/lehre/lehrveranstaltungen/k01618/klausuren.shtml`

- Wie viel Programmiererfahrung haben Sie? (Links = gar keine, Mitte = komplizierte Excel-Formeln, Rechts = tägliche Programmierpraxis)
- Haben Sie Java up & running?

4.2 Terminal und IDE

Terminal = Old-School-Computer: keine Maus, nur Tastatur und Befehle eingeben, die der Computer direkt ausführt

IDE = Integrated Development Environment, Word für den Programmierer inkl. Terminal-Shortcuts

4.3 Hello World

Zunächst ein kurzer Vergleich kompilierte vs. interpretierte Programmiersprachen und dann das erste Programm!

```
public class Main {
    // main  Methode ist der EINSTIEGSPUNKT
    public static void main(String[] args) {
        // AUSGABE
        System.out.println("Hello World!");
    }
}
```

4.4 Aufgabe: Hashes

Schreiben Sie ein Programm, das folgende Ausgabe erzeugt:

```
#
##
###
####
#####
```

Lösung 1

Lösung 2

4.5 Schleifen

Schleifen dienen dazu, dass der selbe Quellcode **mehrfach** ausgeführt wird. Eine for-Schleife besteht aus einem Schleifenkopf mit

- Start: $inti = 0$
- End(bedingung): $i < 5$
- Schrittweite: $i++$ (Inkrementor, Kurzform von $i = i + 1$)

Der Schleifenrumpf (in den geschweiften Klammern) wir solange ausgeführt, wie die Endbedingung erfüllt ist!

```
for(int i = 1; i <= 5; i++) {
    System.out.println(i)
}
```

```
Ausgabe:
1
2
3
4
5
```

4.6 Aufgabe: Raketen

Schreiben Sie ein Programm, das folgende Ausgabe erzeugt:

```
   /\       /\
  /  \     /  \
 /    \   /    \
+------+ +------+
|      | |      |
+------+ +------+
|Rocket| |Rocket|
|  #1  | |  #2  |
+------+ +------+
|      | |      |
+------+ +------+
   /\       /\
  /  \     /  \
 /    \   /    \
```

Lösung

4.7 Klassen, Objekte und Funktionen

Wir teilen unseren Quellcode in Klassen und Funktionen auf. Dabei besprechen wir

- Objekte als Modellierung der (abstrakten) Realität: Quellcode wird anhand von Klassen strukturiert. Die Struktur richtet sich dabei an die realen oder abstrakten Objekte unserer Domäne (fachlicher Anwendungsbereich), die wir programmieren.
- Klassen sind Baupläne für Objekte.
- Instanziierung von Objekten: man kann von einer einzelnen Klasse mehrere Objekte erzeugen
- Sichtbarkeit: man kann Methoden verstecken. Dann können diese nicht von außen, wie z.B. der main-Methode, aufgerufen werden.

```
import Rakete.Rakete;
// Rakete hat ein Attribut "name" mit getter/setter

public class Main {
    public static void main(String[] args) {
        // erstes Objekt wird erzeugt
        Rakete rakete1 = new Rakete("NAMEN A");
        rakete1.print();
        System.out.println(rakete1.getName());

        // zweites Objekt wird erzeugt - gleiche Klasse.
        // beide Objekte sind UNABHÄNGIG voneinander
        Rakete rakete2 = new Rakete("NAMEN B");
        System.out.println(rakete2.getName());
    }
}
```

4.8 Aufgabe: GreaterThanThree

Schreiben Sie eine Klasse, die entscheidet, ob eine übergebene Zahl größer, kleiner oder gleich 3 ist. Ein entsprechender Text wird auf der Konsole zurückgegeben.

```
greaterThanThree(5)
> 5 ist größer als 3.

greaterThanThree(3)
> 3 ist gleich 3.

greaterThanThree(1)
> 1 ist kleiner als 3.
```

Verallgemeinern Sie die Klasse. Die Zahl 3 soll nun konfigurierbar sein.

```
GreaterThanX greatherThanFive = new GreaterThanX(5);
greatherThanFive.decide(6);
> 6 is greater than 5.
```

Lösung

4.9 Vordefinierte Klassen

```
import java.util.Scanner;

public class Main {
    public static void main()
    {
        // Using Scanner for Getting Input from User
        Scanner input = new Scanner(System.in);

        System.out.println("Please enter a string");
        String enteredString = input.nextLine();
        System.out.println("You have entered: " + enteredString)
    }
}
```

4.10 Aufgabe: MaxNumber

Schreiben Sie ein Programm, das drei Zahlen als Input erfragt und die größte Zahl anschließend ausgibt.

```
> Erste Zahl?
3
> Zweite Zahl?
5
> Dritte Zahl?
4
> 5 ist die höchste Zahl von (3,5,4).
```

Lösung

4.11 Aufgabe: NumberSquare

Schreiben Sie ein Programm, das folgende Ausgabe erzeugt. Dabei Programm bekommt die Start- und Endzahl übergeben.

```
Start: 2, End: 6


2 3 4 5 6
3 4 5 6 2
4 5 6 2 3
5 6 2 3 4
6 2 3 4 5
```

5 OOP / Tic Tac Toe

5.1 Tic Tac Toe

Der heutige Tag besteht aus einer zusammenhängenden Aufgabe. Der Programmierung des Spieles Tic Tac Toe! Wir werden dazu nur die Konsole als Aus- und Eingabe nutzen. Gleichzeitig werden wir eine Reihe von wichtigen Grundbegriffen der objektorientierten Programmierung sowie Modellierung klären und praktisch anwenden.

5.1.1 Use Case: Neues Spiel starten

Ein Spielbrett (3x3) wird angezeigt.

```
   |   |
---+---+---
   |   |
---+---+---
   |   |
```

5.1.2 Use Case: Spielzug

Die Spieler geben abwechseln eine Zahl 1-9 an für das jeweilige Spielfeld. Im Anschluss wird die Belegung (x/o auf dem Spielfeld) angezeigt. Bereits belegte Felder können nicht doppelt belegt werden. Hat ein Spieler eine gültige Kombination (horizontal, vertikal, diagonal), wird folgende Nachricht ausgegeben: „Du hast gewonnen! Drücke 0 um zum Menü zurückzukehren“.

```
   |   |
---+---+---
   |   |
---+---+---
   |   |

Player 1 ist dran:
1

 x |   |
---+---+---
   |   |
---+---+---
   |   |

Player 2 ist dran:
5

 x |   |
---+---+---
   | o |
---+---+---
   |   |
```

5.1.3 Quellcode

5.1.4 Use Case: Menü

Beim Starten des Programmes soll ein Menü geöffnet werden, das die folgenden Optionen bietet:

```
Herzlich Willkommen zu TicTacToe!


-   Neues Spiel starten (1)
-   Player 1 Namen eingeben (2)
-   Player 2 Namen eingeben (3)
-   Scoreboard anzeigen (4)
-   Programm verlassen (9)
```

5.1.5 Use Case: Namen eingeben

Beide Spieler können Ihren eigenen Namen eingeben.

```
Player 1 Namen eingeben
-----------------------
Bisheriger Name: Player1
Neuer Name:
_Eingabe_

Der Name von Player1 wurde erfolgreich in "_Eingabe_" geändert.
```

5.1.6 Use Case: Scoreboard anzeigen

Das Programm merkt sich, wie viele Spiele jeder Spieler gewonnen / verloren hat und zeigt dann folgende Ausgabe:

```
| Spieler | Gewonnen | Verloren |
| ------- | -------- | -------- |
| Player1 | 1        | 5        |
| Player2 | 4        | 1        |
```

5.1.7 Use Case: Programm verlassen

Das Programm wird beendet. Alle eingegeben Daten gehen verloren.

OPTIONALE VERTIEFUNG: Die Daten sind auch bei einem Neustart vorhanden (Stichwort: Serialisierung).

5.2 Theoretische Inhalte

- Exception
- Überladen
- Static vs Object
- Inheritance
- optionale Vertiefung
  - Dependency Injection
  - Single Responsbility & Separations of Concerns
  - Fluent Interface & Method Chaining
  - Law of Demeter

6 AWT / Hangman

Wir haben AWT bzw. Swing verwendet, um das Spiel Hangman zu programmieren. Der Quellcode ist aufgrund des Umfanges angehängt.

7 Ausgewählte Themen / Parallelität und Verteilung

7.1 Grundlagen Parallelität

Parallelität durch Zeitscheiben: die CPU wird mehreren Prozessen gleichzeitig verfügbar gemacht. Scheduling (Teil des Betriebssystems) entscheidet über Priorisierung, welcher Prozess wann wie viele Zeitscheiben erhält. Kommunikation zwischen den Prozessen über den Arbeitsspeicher.

- Prozesse (Single- / Multi-Core CPU, Betriebssystem)
- Thread (Intra-Prozess, Programmiersprache macht Switch)

Einen Thread kann man auch als Mini-Unterprozess ansehen. Der Hauptprozess (Hauptthread) und die (Unter-)Threads werden vom Betriebssystem als EIN Prozess wahrgenommen und behandelt.

Alternative sind **verteilte Systeme**, die sich auf mehrere Computer beziehen. Die Prozesse laufen dabei getrennt, in verschiedenen Computers (Betriebssystemen) und kommunizieren dann über Netzwerke (Internet!).

7.2 Exkurs: ISO OSI-Modell

Standard für die Netzwerkkommunikation über sieben Schichten. Auf jeder Ebene gibt es ein Protokoll, das genutzt wird. Manchmal überspannt ein Protokoll auch mehrere (theoretische) Schichten. Als Software-Entwickler werden i.d.R. die oberen drei Schichten zusammen betrachtet.

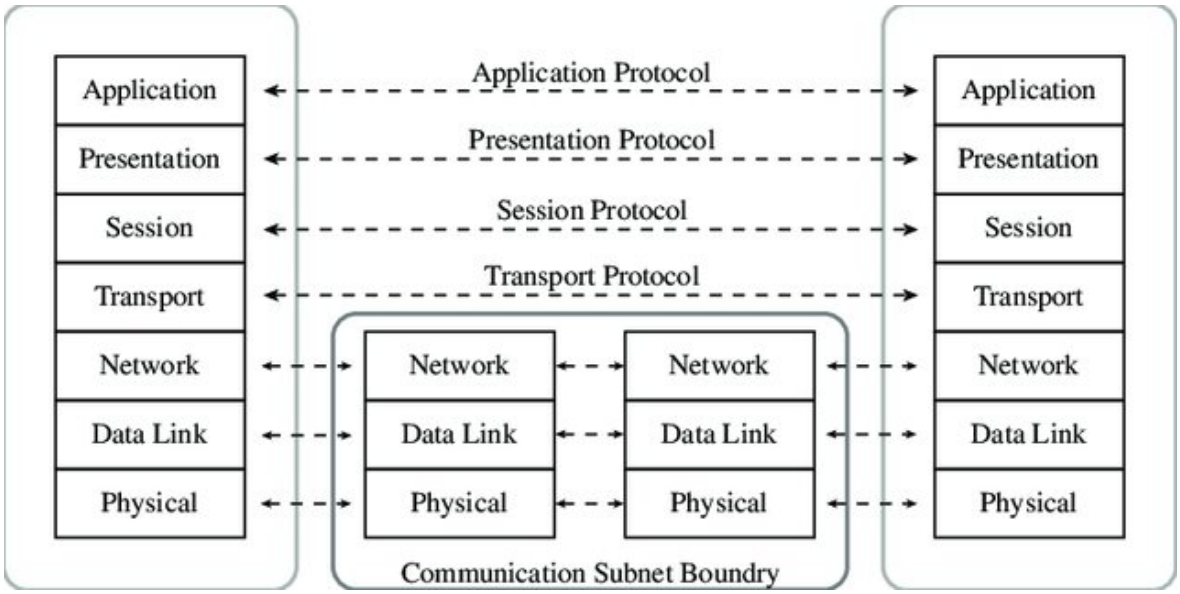

Abbildung 1: OSI-Modell mit Sender und Empfänger
Quelle: Pablo Soldati (Dissertation)

Für jede Schicht gibt es verschiedene Protokolle, so werden die unteren Schichten durch die Hardware bestimmt (z.B. WLAN vs LAN) und die oberen durch die Anwendungen. Das Internet bzw. World Wide Web nutzt: Browser, HTML (HyperText Markup Language), HTTP (HyperText Transfer Protocol), TCP* (Transmission Control Protocol), IP (Internet Protocol, *globale Adressierung von Computers, IP-Adress = Analogie zur Telefonnummern von Computern*).

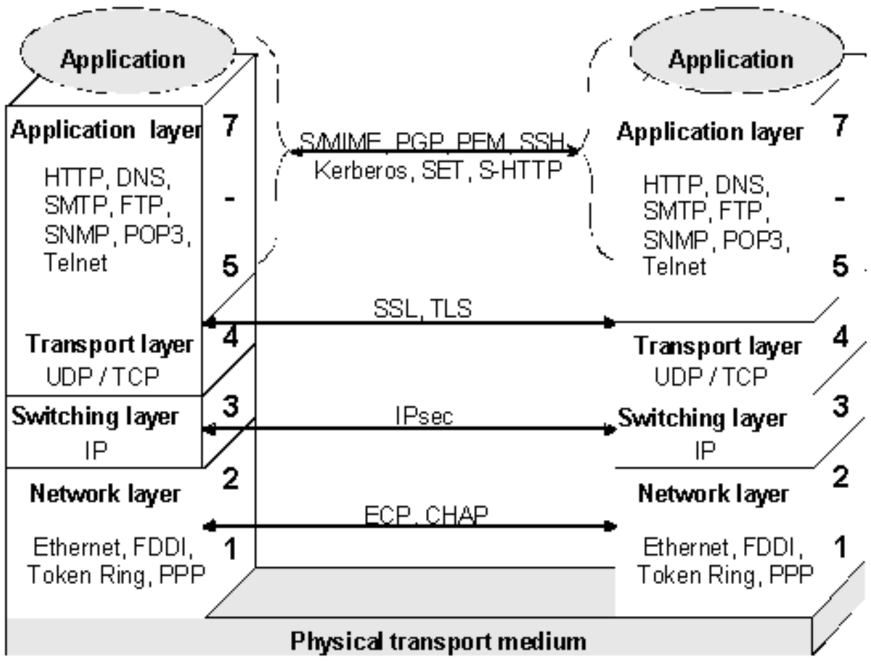

Abbildung 2: OSI-Modell: Protokolle
Quelle: blacksheepnetworks.com

Die Datenübertragung verläuft in Paketen, d.h. die Nutzdaten („eigentliche Daten“) werden nicht alle zusammenhängend übertragen, sondern kleingeschnitten (*Analogie zu Parallelität mit den CPU-Zeitscheiben*). Ein 5GB-Video wird also in n Paketen übertragen. IP-Paket (Internet Protokoll, Schicht 3) enthält mindestens 576 Byte und maximal 65.535 Bytes an Nutzdaten.

Zur der Datenübertragung werden die Daten in zwei Bereiche geteilt: Header und Nutzdaten. Jede Schicht (bzw. jedes Protokoll) fügt seinen Header hinzu und packt Meta-Informationen in den Header. Im IP-Paket ist eine Header-Information z.B. die Ziel-IP-Adresse.

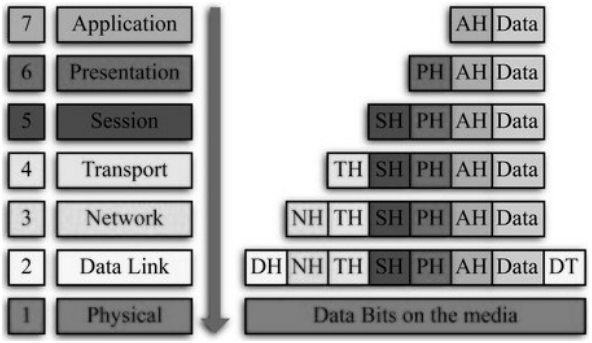

Abbildung 3: OSI-Modell: Daten
Quelle: netzwerke.com

7.3 Client-Server und Threads

Client-Server ist ein Architektur-Pattern zur Verteilung, das eng mit (Rechner-)Netzen und Kommunikation verbunden ist. Es gibt zwei Prozesse: Client und Server. Die Prozesse können dabei auf dem selben (z.B. als Software-Entwickler während der Programmierung) oder unterschiedlichen Rechnern laufen (i.d.R. der Produktivbetrieb).

Umgangssprachlich wird beim Begriff *Server* nicht zwischen Prozess und Hardware unterschieden. Als Beispiel „*der* Datenbankserver“: dies bezieht sich zum einen auf den Server als Computerhardware und zum anderen auf den Datenbank-Prozess, der auf dieser Hardware läuft. Es können auf mehrere (Prozess-)Server auf einem (Hardware-)Server laufen: z.B. ein Firmenserver (Hardware) auf dem ein Fileserver, ein Mailserver, ein Webserver, etc. laufen.

Der Client macht eine Anfrage (engl. *Request*) an den Server und bekommt eine Antwort (engl. *Response*) zurück. Dabei wird das gesamte OSI-Modell mit konkreten Protokollen durchlaufen.

Als Programmierer werden wir den Server (der als Prozess läuft) i.d.R. aus Performance-Gründen parallel programmieren. Zum Beispiel wird für jeden ankommenden Request ein Thread geöffnet, der die Response erstellt und zurückgibt. Dann schließt sich der Thread. Sonst müsste jede neue Request warten, bis alle vorhergegangenen Requests abgearbeitet sind.

7.4 Remote Procedure Call

Anwendung von Client-Server-Prinzip und OSI-Kommunikation auf Programmiersprachen-Ebene.

7.5 Demo und Aufgaben

1. Demo: Server.java: ein Server-Programm, das Threads nutzt und als Client die Konsole (STDIN)
2. Demo: Web.java: ein Web-Server (mit Hilfe der Sun-Bibliothek)
3. Aufgabe 1: Multi-Thread Web Server nach Lab-Anleitung Teil A (Anhang)
4. Aufgabe 2: Multi-Thread Web Server nach Lab-Anleitung Teil B (Anhang)

8 Klausurvorbereitung

Wir werden dazu alte Klausuren besprechen. Anbei eine inhaltliche Übersicht der möglichen Themen:

8.1 Programmierung

- Prozedurale vs. objektorientierte Programmierung
- Werte, Typen und Variablen (Zuweisungen)
- Arrays
- Kontrollstrukturen: Bedingungen und Schleifen
- Ausnahmebehandlung (Exceptions)

8.2 Grundlagen Objekte und Vererbung

- Objekte, Klassen, Methoden, Konstruktoren
- Vererbung/Spezialisierung
- Überladen und Überschreiben
- Liskov'sches Substitutionsprinzip
- Dynamisches Binden
- Static: Klassemethoden und -attribute
- Parametrisierung (Generics)

8.3 Typisierung

- Klassen
- Vererbung / Subtyping
- Abstrakte Klassen
- Interfaces und Aufzählungstypen
- Polymorphie

8.4 Frameworks, Bibliotheken, Packages und deren Integration

- Framework
- Bibliotheken
- Packages
- Lokale Klassen
- Anonyme Klassen
- Lambda-Ausdrücke

8.5 AWT

8.6 Parallelität, Streams und Verteilung

- Parallelität

- Threads
- Remote Method Invokation
- Stub-Skeleton
- Sockets
- Input- und OutputStream