

PROGRAMMIEREN 1

Objekt Orientierte Programmierung

Java / IntelliJ / Git

2021

Basisdokument (Simon Plakolb – Danke!)

ÜBER MICH

- ◆ Stephan Kochauf
- ◆ Studium: IT & IT-Marketing (Campus02)
- ◆ 15 Jahre Erfahrung Softwareentwicklung in diversen Rollen
- ◆ Seit 2014 Nebenberuflicher Lektor am Campus02

DISTANCE LEARNING

 Vorteile

 Nachteile

VORTEILE DISTANCE LEARNING

- Wir sitzen zu Hause (eventuell noch im Pyjama)
- Gewöhntes Umfeld
- Zeitliche Ersparnis (kein Anfahrtsweg)
- Es könnten nur Computerviren übertragen werden (keine Chance für Covid 😊), aber auch keine Chance für Computerviren

NACHTEILE DISTANCE LEARNING

- Spreche mit Bildschirm
- Sehe kein Feedback anhand von Gestik, Gesichtsausdrücken, etc.
- Sehe keine rauchenden Köpfe

ABLAUF

- Theorie
- Live-Coding (mitschreiben oder mitmachen)
- Eigenständiges Coding

MEINE BITTE AN SIE!

- Bin ich zu schnell / zu langsam => **Feedback**
- Verstehen Sie etwas nicht / **unterbrechen Sie mich, schreiben Sie mir**
- Brauchen Sie bei Übungen mehr Zeit / **Feedback**
- Technische Probleme? / **Feedback**
- Aktiv dabei sein & Mitdiskutieren
- Pünktlich sein

ÜBER DIESE LV

- Einführung in die Objekt-orientierte Programmierung (**OOP**)
 - ▶ Konzept und Verständnis
- Programmiersprache: **Java**
 - ▶ Grundlagen
- Werkzeuge: **IntelliJ** und **Git**

ZUR BENOTUNG

- Die Note ergibt sich aus der **Endklausur**
 - ▶ 19.04.2021
- Positiver Abschluss bei $> 50\%$ der Punkte
 - ▶ Selbstständiges Programmieren
 - ▶ Übungsklausuren werden zuvor bereitgestellt
- Wiederholung bei der **Nachklausur**
 - ▶ Termine werden bekanntgegeben
 - ▶ Kommissionell schriftlich **und** mündlich

LV PLAN

- **Einführung**
- Werkzeuge
- Datentypen
- Kontrollfluss
- Rekursion
- Objekte und Klassen
- Packages
- Collections

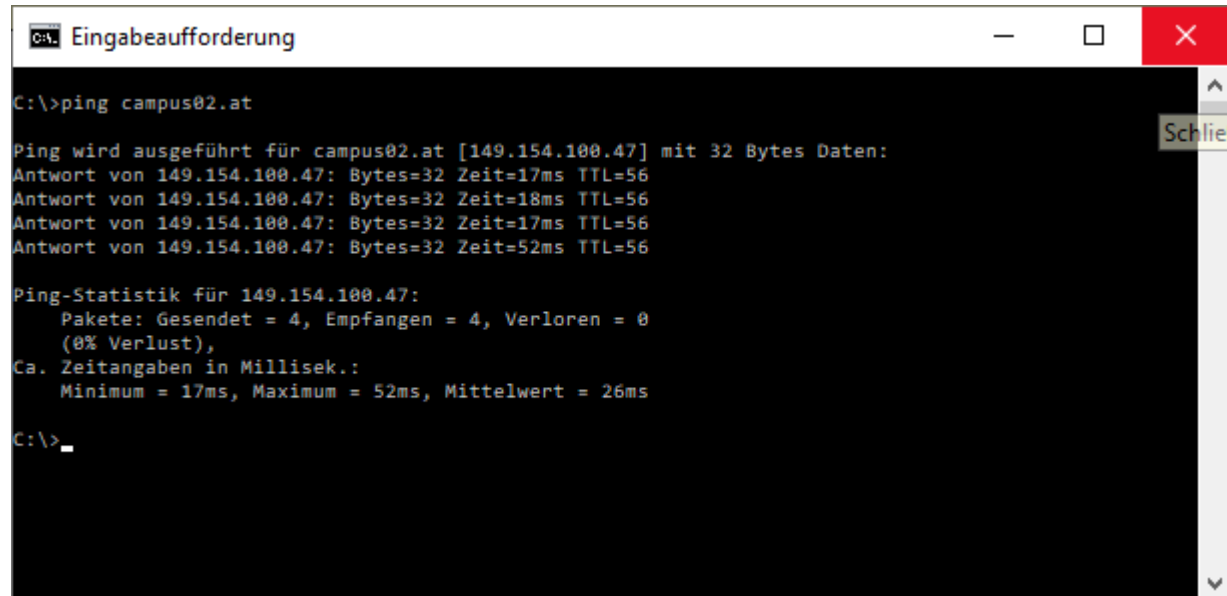
BASICS

- Unterschiedliche Typen von Programmen
 - Desktopanwendungen (Grafische Benutzerschnittstellen => Word, Excel, Chrome)
 - Serveranwendungen (laufen im Hintergrund)
 - Webanwendungen (oft eigene Programmiersprachen, laufen im Browser)
 - Konsolenanwendungen (so hat alles begonnen ;-))
Kommunikation über Konsole

Windows: cmd / „Eingabeaufforderung“

KONSOLENANWENDUNG

- Kennt man aus „Hacking-Szenen“
- Über „Konsole“ / Tastatur können Programme gestartet werden
- Programm kommuniziert über Text mit User



```
C:\>ping campus02.at

Ping wird ausgeführt für campus02.at [149.154.100.47] mit 32 Bytes Daten:
Antwort von 149.154.100.47: Bytes=32 Zeit=17ms TTL=56
Antwort von 149.154.100.47: Bytes=32 Zeit=18ms TTL=56
Antwort von 149.154.100.47: Bytes=32 Zeit=17ms TTL=56
Antwort von 149.154.100.47: Bytes=32 Zeit=52ms TTL=56

Ping-Statistik für 149.154.100.47:
    Pakete: Gesendet = 4, Empfangen = 4, Verloren = 0
    (0% Verlust),
    Ca. Zeitangaben in Millisek.:
        Minimum = 17ms, Maximum = 52ms, Mittelwert = 26ms

C:\>_
```

PROBLEME FÜR COMPUTER

- Digitale
Bildbearbeitung
- Lastprofile berechnen
- ...
- Kommunikation
- Große Datenmengen
verarbeiten

ALGORITHMEN

- Geben eine **genaue Definition der Berechnung**
- Beschreiben **Abfolge** und **Art** der Berechnungs-Schritte
- In Form von **Sprache** und **Mathematik**

ÜBUNGSBEISPIEL

- ❖ Schreiben sie einen Algorithmus:
- ❖ Zum **Finden** der höchsten Karte in einem Kartenstapel
- ❖ Zum **Sortieren** eines Kartenstapels

LÖSUNG

- ❖ Oft gibt es **mehr als eine** Lösung
- ❖ Zum Beispiel gibt es viele Sortieralgorithmen
 - ▶ Quicksort
 - ▶ Heapsort
 - ▶ Mergesort

<https://www.youtube.com/watch?v=kPRA0W1kECg>

DER COMPUTER

- Ein Computer ist **nicht intelligenter** als wir
- Er ist aber bedeutend **schneller**!
- Wie aber funktioniert ein Computer?

DER COMPUTER - BAUSTEINE

Prozessor (CPU)

- ▶ Steuerwerk
- ▶ Rechenwerk

Arbeitsspeicher (RAM)

- ▶ Programm
- ▶ Daten

DER COMPUTER - FUNKTION

- ❖ Computer stellen Daten sowie Programme in Form von **binären Zahlen** dar
 - ▶ Sie kennen also nur Abfolgen von **0 und 1**
- ❖ Zum Beispiel können wir uns den Befehl zum **Addieren** eines Intel 8008 Prozessors

DER COMPUTER - FUNKTION

Der Befehl 1 0 0 0 0 s s s sagt dem Intel 8008, dass die (binäre) Zahl s s s zum Wert im Register zu **addieren** ist.

Um die Zahl 3 zu addieren sähe der Befehl also so aus:

1 0 0 0 0 0 1 1 **(011 entspricht dem Wert 3)**

Nur in 0 und 1 zu Programmieren ist für Menschen jedoch **schwierig** und **kaum leserlich**, **macht keinen Spaß**, **dauert lange**,
...

DER ASSEMBLER

Wir können den Befehl auch in seiner **Bestandteile** zerlegen

Instruktion

1 0 0 0 0

Daten

0 1 1

.. um diesen Teilen dann **bedeutungsvolle Namen** zu geben

Instruktion

ADD

Daten

3

DER ASSEMBLER

- Ein **Assembler** übersetzt dann von den für **Menschen** leserlichen Instruktionen zurück in binäre, für den **Computer** verständliche Werte
- Der Assembler übersetzt also ein **Vokabular**

PROGRAMMIERSPRACHEN

- Assembly ist bereits eine **rudimentäre** Programmiersprachen
 - ▶ **Nicht** für alle Prozessoren **standardisiert**
 - ▶ **Aufwändig** und unübersichtlich
- Um es uns leichter zu machen müssen wir die **Grammatik** anpassen
- Ranking: <https://www.tiobe.com/tiobe-index/>

ASSEMBLER – HELLO WORLD

```
.MODEL Small
.STACK 100h
.CONST
    HW      DB      "Hallo Welt!$"
.CODE

start:
    MOV AX,@data
    MOV DS,AX
    MOV DX, OFFSET DGROUP:HW
    MOV AH, 09H
    INT 21H
    MOV AH, 4Ch
    INT 21H
end start
```


DER COMPILER

- Ein **Compiler** übersetzt nicht nur “Vokabeln” sondern auch die “Grammatik”

So kann man zum Beispiel in der Programmiersprache **C** in einer Zeile die Zahl 3 zu 0 addieren und das Ergebnis speichern:

```
int a = 0 + 3;
```

OBJEKT-ORIENTIERTE PROGR.

- In Assembly und nur wenig abstrakteren Sprachen wird **prozedural** programmiert
- Befehlsabläufe werden als **Prozeduren** zusammengefasst und aufgerufen
- **Objekt-orientierte Programmierung (OOP)** ist eine Abstraktionsstufe höher

OBJEKT-ORIENTIERTE PROGR.

- Um uns das Programmieren zu erleichtern orientiert sich OOP daran
wie wir Menschen die Welt wahrnehmen
- Es gibt **Objekte**
 - ▶ Mit Eigenschaften (**Attributen**)
 - ▶ Und Funktionen (**Methoden**)

DAS OBJEKT – BEISPIEL: HAUS

Attribute

- Höhe
- Länge
- Breite
- Einwohner

Methoden

- Volumen berechnen
- Person hinzufügen
- Auszug von Person

ÜBUNGSaufGABE - OOP

■ Beschreibe **Attribute** und **Methoden** für:

- ▶ Ein Rechteck
- ▶ Einen Tisch
- ▶ Bonus: Haben Tisch und Rechteck etwas gemeinsam?

JAVA

- Objekt-orientierte Programmiersprache
- Läuft in einer **virtuellen Maschine**
 - Java Virtual Machine - **JVM**
- Buch: Java ist auch eine Insel
 - <http://openbook.rheinwerk-verlag.de/javainsel/>

JAVA BYTECODE

- Die **JVM** läuft auf allen modernen Betriebssystemen
- Der Java Compiler übersetzt Java-Code in -Bytecode
- Der Bytecode ist die **Maschinensprache** für die JVM

DAS JDK

- Die JVM ist Teil des Java Runtime Environments
 - **JRE**
- Der Compiler sowie alle **Programmbibliotheken** von Java sind enthalten im
 - **JDK**
 - <https://openjdk.java.net/>

JAVA “VOKABELN”

- Sogenannte **Keywords** sind festgelegte Begriffe mit definierter Bedeutung
 - ▶ Müssen nicht auswendig gelernt werden
- QUIZ: https://www.sporcle.com/games/robv/java_keywords

JAVA “GRAMMATIK”

C-like

- ▶ Zeilen werden mit einem **Semikolon (;)** beendet
- ▶ Code-Blöcke werden in **geschwungenen Klammern** umschlossen ({ .. })
- ▶ Funktionen werden mit **runden Klammern** aufgerufen ((..))

LV PLAN

- Einführung
- **Werkzeuge**
- Datentypen
- Kontrollfluss
- Rekursion
- Objekte und Klassen
- Packages
- Collections

GIT UND GITHUB

- Git eignet sich um verschiedene **Versionen** und Fortschritte zu **speichern und verwalten**
 - ▶ <https://git-scm.com/>
- Richtet man Git für ein Projekt ein, spricht man von einem Git-**Repository**
- GitHub dient als **Cloud-Speicher** für Git-Repositoryen
 - ▶ <https://github.com/>

GIT BEFEHLE

- Um ein neues Repository anzulegen:
 - ▶ `git init`
- Um eine Datei vorzubereiten:
 - ▶ `git add [Dateiname]`
- Alle vorbereiteten Dateien zur aktuellen Version hinzufügen:
 - ▶ `git commit -m "Beschreibung"`
- Um Änderungen zum letzten commit anzuzeigen
 - ▶ `git diff`
- Git-CheatSheet:
 - ▶ <https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf>

GIT BEFEHLE

- Um einen neuen Zweig zu eröffnen:
 - ▶ `git branch [Branch-Name]`
- Alle Zweige anzeigen
 - ▶ `git branch`
- Zu einem Zweig wechseln
 - ▶ `git checkout [Branch-Name]`
- Einen Zweig mit dem derzeitigen zusammenführen
 - ▶ `git merge [Branch-Name]`
- Git-CheatSheet:
 - ▶ <https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf>

INTELLI-J

- JetBrains IntelliJ ist eine sogenannte **IDE**
 - ▶ Integrated Development Environment
- Übernimmt viele **redundante Aufgaben**
- Kann selbstständig **Fehler finden**
- Integrierte Git – Versionskontrolle

<https://www.jetbrains.com/idea/>

NEUES PROJEKT ANLEGEN

- Der Code im Projekt befindet sich immer im Ordner '**src**' (*Source*)
- Intelli-J erstellt diese Ordnerstruktur **automatisch**

TEXTEDITOR

- ❖ **Rote Markierungen** zeigen **Fehler** an
- ❖ Doppelklick auf Name für **Vollbild**

PROJECT

- ❖ Dient zur **Übersicht** über die **Projektstruktur**
- ❖ Bildet die **Ordner-** und **Dateistruktur** ab

STRUCTURE

- Dient zur **Übersicht über den Code** im Projekt
- Verschiedene Gruppierungen können **ein-** und **ausgeblendet** werden

TODO

- ❖ Code-Kommentare die mit 'TODO:' anfangen, werden hier gesammelt
- ❖ In Java fängt ein Kommentar mit `//` an
- ❖ Ein Block-Kommentar fängt mit `/*` an und hört mit `*/` auf

VERSION CONTROL

- ❖ Hier ist **Git** in INTELLI-J integriert
- ❖ Grafische Oberfläche **ersetzt die Git-Befehle!**
- ❖ Farbliche Markierung der Dateien nach Erfassungsstand

ÜBUNGSaufgabe – JAVA 1

- Legen Sie ein neues Projekt an
 - ▶ Name: “FirstSteps”
- Erstellen Sie die Klasse “Hello” mit folgendem Inhalt (Code-Nr: 1):

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello world");  
    }  
}
```








RUN

- Hier wird ein Programm ausgeführt
- Daher erfolgt auch hier die Ausgabe
- Zum **Ausführen** eines Programms: Play-Knopf klicken

ÜBUNGSaufgabe – JAVA 2

- Legen Sie ein neues Projekt an
 - ▶ Name: “FirstProject”
- Erstellen Sie die Klasse “HelloCampus”
 - ▶ Diese Klasse soll beim Ausführen “Hello Campus” ausgeben (Code-Nr.: 1)

DER DEBUGGER

-  Mit dem Debugger lässt sich ein Programm **schrittweise** durchlaufen
-  Wird mit dem **grünen Käfer** Icon gestartet
-  An sogenannten **Breakpoints** hält der Debugger an bis wir fortfahren
 -  Diese lassen sich rechts neben der Zeilennummerierung per Mausklick setzen
 -  *Tastaturkombination:* Strg + F8
-  Ein Breakpoint kann auch am **Anfang einer Methode oder Klasse** stehen
 -  Dort wird gehalten, wenn ein Objekt aufgerufen wird

REFACTORING

- Bei Änderungen am Code ist **Vorsicht** geboten
- Durch **Abhängigkeiten** können unabsichtlich **Fehler** verursacht werden
- Daher gibt es das **Refactoring**
 - ▶ Die **IDE** übernimmt für uns die Aufgabe nach möglichen Folgefehlern zu suchen

REFACTORING

Ablauf

- ▶ Auf den Text der umbenannt werden soll klicken
- ▶ Refactor -> Rename
 - Entweder [Shift]+[F6] -> Refactor -> Do Refactor (unten)
 - Oder Umbenennen + [Enter]

LOCAL HISTORY

- Zu jeder Datei gibt es eine **local history**
 - ▶ Durch Rechtsklick auf die Datei
 - ▶ Oder im **VCS** Menü (*Version Control System*)
- Es können alte Zustände der Datei wiederhergestellt werden.
 - ▶ Coden ist wie in der Sandkiste spielen, man kann nicht viel kaputt machen!

STACKOVERFLOW.COM

- Fast alle Probleme die beim Programmieren auftreten wurden bereits gelöst
- Durch **Googlen** der Fehlermeldung findet man diese **Lösungen** auf Stackoverflow

SELBSTSTÄNDIGKEIT

- Wie jede andere Sprache lernt man auch Programmiersprachen nur durch **regelmäßiges und intensives Anwenden**
- Nehmen Sie sich Zeit zu Hause, um mit dem Erlernten zu „spielen“
 - ▶ Aus Fehlern kann man lernen (Mit Stackoverflow)
 - ▶ Es kann nichts kaputt gehen (Dank local history)

ÜBUNGSaufgabe

- Erweitern Sie ihren Code indem sie **mehrere** Ausgaben hintereinander ausführen. (Code-Nr.: 1)
 - ▶ Wann wird welche Ausgabe ausgeführt?
 - ▶ Überprüfen Sie Ihre Annahme mit dem Debugger!
- Geben Sie in einer der Ausgaben den **Namen** der Klasse aus
- Schreiben sie einen Kommentar, welcher den Namen der Klasse beinhaltet
 - ▶ zB ein TODO
- Ändern Sie den Namen der Klasse mittels **Refactoring** so, dass er sich auch in der Ausgabe und im Kommentar ändert