

Sommersemester 2023

# Objektorientierte Programmierung I

## mit Java

#01, 06.04.2023, Leipzig

Prof. (FH) Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

DSPOOPI01

```
// set up the listener
webEngine.getLoadWorker().stateProperty().addListener((observable, oldValue, newValue) -> {
    if (Worker.State.SUCCEEDED == newValue) {
        // set an interface object named 'javaConnector' in the web engine's page
        JSObject window = (JSObject) webEngine.executeScript("window");
        window.setMember("name", "javaConnector", javaConnector);

        // get the Javascript connector object.
        javascriptConnector = (JSObject) webEngine.executeScript("getConnector()");
    }
});

Scene scene = new Scene(webview, width: 300, height: 150);
primaryStage.setScene(scene);
primaryStage.show();
```

# Agenda

<b>1</b>	<b>Vorstellungsrunde</b>	09:00 – 09:30
<b>2</b>	<b>Eckdaten des Moduls</b>	09:30 – 09:40
<b>3</b>	<b>Erfolgsfaktoren für die Teilnahme</b>	09:40 – 09:50
<b>4</b>	<b>Installation von Java</b>	09:50 – 10:30
<b>5</b>	<b>Pause</b>	10:30 – 10:45
<b>6</b>	<b>Erste Programmierübungen</b>	10:45 – 12:15
<b>7</b>	<b>Einstieg in Java</b> (zur Nacharbeit)	



```
// set up the listener
webEngine.getLoadWorker().stateProperty().addListener(observable, oldValue, newValue) -> {
    if (Worker.State.SUCCESS == newValue) {
        // set an interface object named 'javaConnector' in the web engine's page
        JSONObject window = (JSONObject) webEngine.executeScript("window");
        window.setMember("javaConnector", javaConnection);

        // get the Javascript connector object.
        javascriptConnector = (JSObject) webEngine.executeScript("getJsConnector()");
    }
}
}

Scene scene = new Scene(webView, width: 300, height: 150);
primaryStage.setScene(scene);
primaryStage.show();
```

# Agenda

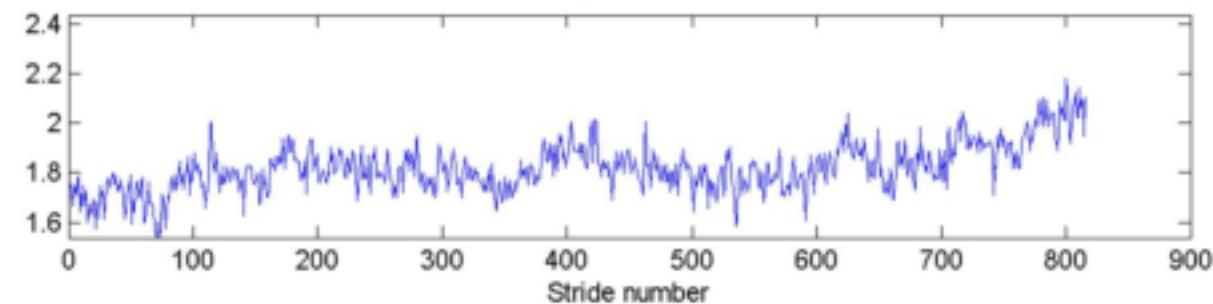
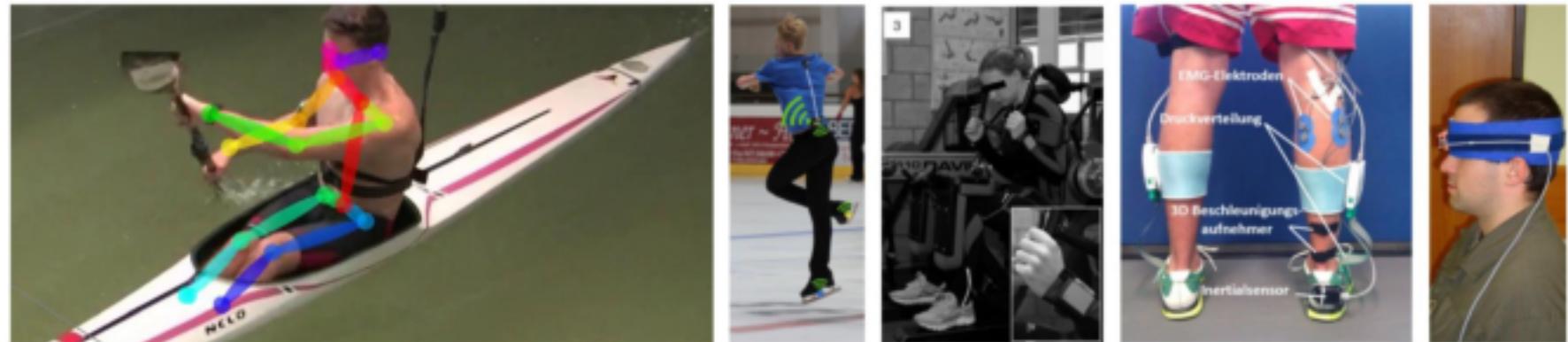
<b>1</b>	<b>Vorstellungsrunde</b>	09:00 – 09:30
<b>2</b>	<b>Eckdaten des Moduls</b>	09:30 – 09:40
<b>3</b>	<b>Erfolgsfaktoren für die Teilnahme</b>	09:40 – 09:50
<b>4</b>	<b>Installation von Java</b>	09:50 – 10:30
<b>5</b>	<b>Pause</b>	10:30 – 10:45
<b>6</b>	<b>Diskussionen über das Programmieren und erste Übungen mit Java</b>	10:45 – 12:15
<b>7</b>	<b>Einstieg in Java</b> (zur Nacharbeit)	

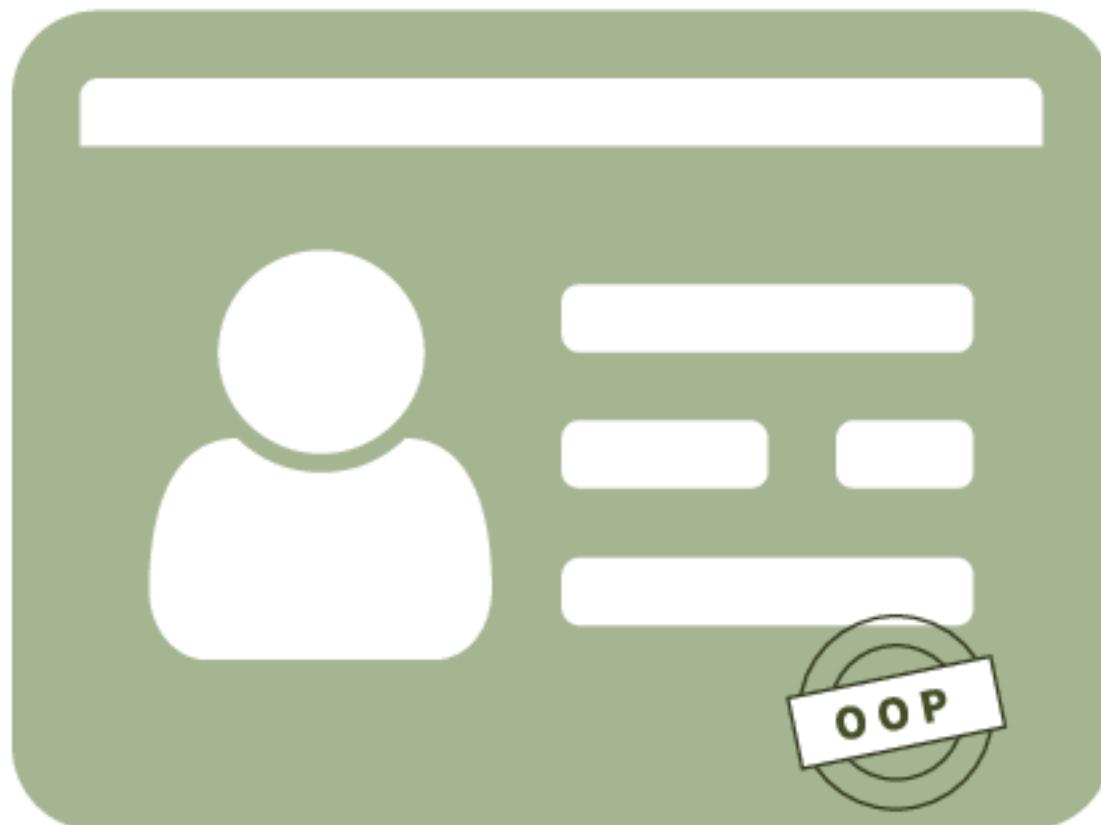
- Zum Vernetzen gerne unter:

- <https://www.linkedin.com/in/sebastianbichler/>
- <https://www.researchgate.net/profile/Sebastian-Bichler>

- Themen:

- Wearable Sensor Technology
- Entwicklung von
  - Mess- und Informationssystemen
  - Feedback-Systemen im Sport
  - Web-Apps, ...
- Datenanalyse
  - Herzfrequenzvariabilität
    - Regeneration
    - Ermüdung
    - EPOC
    - ...
  - biomechanisch
  - physiologisch
  - sportmedizinisch





Der Duke

- Name: Umbenennung von **Oak** nach **Java** (Entscheidung fiel angeblich in einem Coffeeshop)
- Geboren namentlich 1994
- Vater: James Gosling
- Erst-Familie: Sun Microsystems (Green-Dream-Team)
- Erwartungen: zunächst Software für Konsumelektronik,
- Werdegang: Anlehnung an MESA (Xerox PARC), Web-Applets, *Netscape* erhält Lizenz, Web-Backend, ...
- Fun facts: Geburt gilt eher als Betriebsunfall (C++), trotz wechselnder Besitzer (Oracle), Strategieänderungen

# Java

## A brief history

- „Nach 20 Jahren hat sich Java als Plattform endgültig etabliert. Millionen Softwareentwickler verdienen weltweit mit der Sprache ihre Brötchen, Milliarden Geräte führen Java-Programme aus, z. B. alle Blu-ray-Player. Es gibt Millionen Downloads von Oracles Laufzeitumgebung in jeder Woche.“ [Ullenboom2021]
- In den 1970ern will Bill Joy, Mitbegründer von *Sun Microsystems*, eine an *MESA* angelehnte Programmiersprache entwickeln. Der *Xerox Alto* wurde mit *MESA* programmiert. *Xerox Alto* hatte als erster Computer moderne Benutzerschnittstellen (GUI, Maus, ...). In den 1990ern beschrieb er wie eine moderne objektorientierte Sprache angelegt sein müsste. Zunächst sollte sie auf *C++* aufbauen. Ihm wurde bewusst das *C++* nicht geeignet sei.
- James Gosling konnte seine Entwicklungsvorhaben nicht mit *C++* befriedigend umsetzen, so entwickelte er *Oak*.
- Ende 1990 begann das Green-Projekt mit dem Ziel Software für interaktives TV und andere Geräte der Konsumelektronik zu entwickeln.
- Entwicklung wurde letztlich in Richtung *Web* getrieben. U.a. sollte Programmcode empfangen und ausgeführt werden – ohne Sicherheitsrisiken. *C++* schied aus, aufgrund der *Zeigerproblematik*.
- Erster Java-Webbrowser HotJava, für den *Netscape* (glücklicherweise) Java lizenzierte.
- 1996 JDK 1.0: Web-Applets und Java-Apps. ...

# Agenda

1	Vorstellungsrunde	09:00 – 09:30
2	Eckdaten des Moduls	09:30 – 09:40
3	Erfolgsfaktoren für die Teilnahme	09:40 – 09:50
4	Installation von Java	09:50 – 10:30
5	Pause	10:30 – 10:45
6	Diskussionen über das Programmieren und erste Übungen mit Java	10:45 – 12:15
7	Einstieg in Java (zur Nacharbeit)	

# Termine

April							Mai							Juni							Juli						
Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So
27	28	29	30	31	01	02	01	02	03	04	05	06	07	29	30	31	01	02	03	04	26	27	28	29	30	01	02
03	04	05	06	07	08	09	08	09	10	11	12	13	14	05	06	07	08	09	10	11	03	04	05	06	07	08	09
10	11	12	13	14	15	16	15	16	17	18	19	20	21	12	13	14	15	16	17	18	10	11	12	13	14	15	16
17	18	19	20	21	22	23	22	23	24	25	26	27	28	19	20	21	22	23	24	25	17	18	19	20	21	22	23
24	25	26	27	28	29	30	29	30	31	01	02	03	04	26	27	28	29	30	01	02	24	25	26	27	28	29	30
01	02	03	04	05	06	07	05	06	07	08	09	10	11	03	04	05	06	07	08	09	31	01	02	03	04	05	06

O  
O  
P  
I

# Termine und Themen

<b>1</b>	<b>Kick-off (Programmierumgebung)</b>		06.04.2023	09:00 – 12:15
<b>2</b>	<b>Programmierkonzepte</b>		13.04.2023	09:00 – 12:15
<b>3</b>	<b>Thema</b>		20.04.2023	09:00 – 12:15
<b>4</b>	<b>Thema</b>	Themen werden demnächst ergänzt bzw. ergeben sich peau à peau.	27.04.2023	09:00 – 12:15
<b>5</b>	<b>Thema</b>		04.05.2023	09:00 – 12:15
<b>6</b>	<b>Thema</b>		11.05.2023	09:00 – 12:15
<b>7</b>	<b>Thema</b>		25.05.2023	09:00 – 12:15
<b>8</b>	<b>Thema</b>		01.06.2023	09:00 – 12:15
<b>9</b>	<b>Thema</b>		08.06.2023	09:00 – 12:15
<b>10</b>	<b>Thema</b>		15.06.2023	09:00 – 12:15
<b>11</b>	<b>Thema</b>		22.06.2023	09:00 – 12:15
<b>12</b>	<b>Thema</b>		29.06.2023	09:00 – 12:15
<b>13</b>	<b>Thema</b>		06.07.2023	09:00 – 10:30

# Eckdaten

Zugangsvoraussetzungen	keine
Workload	150 Stunden: – Kontaktzeit: 37,5 Stunden – <b>Selbststudium: 112,5 Stunden</b>
ECTS-Punkte	5 – entsprechend des Workloads
Abschlussprüfung	– Präsenzklausur (unter Berücksichtigung der Mitarbeit / der Programmierung in den „Projekten“) – 90 Minuten



- Grundkonzepte der objektorientierten Modellierung kennen und voneinander abgrenzen
- Grundkonzepte und -elemente der Programmiersprache Java kennen
- Erfahrungen in der Verwendung von Java sammeln
- konkret beschriebene Probleme selbstständig lösen
- *Entwickeln eines kleinen Softwareprojekts*

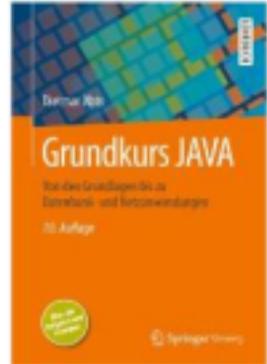
- Kommunikation und Kursmaterialien werden über die Teams-Gruppe abgewickelt:

- <https://teams.microsoft.com/l/channel/19%3akhE7EeMvB-1YNqw0Y-a0-JzlwAYD5vTGfRwGAr4Otd81%40thread.tacv2/Allgemein?groupId=c152c563-4198-4242-8a47-929bd4957280&tenantId=f419c9fe-f7b0-4d87-bee8-e8dfb2190cab>

- Quellcode via *Github* und *Teams* (Information erfolgt noch)

- Literatur

- Abts, D. (2020). *Grundkurs JAVA: Von den Grundlagen bis zu Datenbank- und Netzanwendungen*. Springer Fachmedien Wiesbaden. <https://doi.org/10.1007/978-3-658-30494-2>
- Rayman, & Lahres. (2015). *Objektorientierte Programmierung—Das umfassende Handbuch* (2.). Rheinwerk Computing. [https://openbook.rheinwerk-verlag.de/oop/index.htm#\\_top](https://openbook.rheinwerk-verlag.de/oop/index.htm#_top)
- Ullenboom. (o. J.). *Java ist auch eine Insel*. Rheinwerk Computing. Abgerufen 8. März 2023, von <https://openbook.rheinwerk-verlag.de/javainsel/>



<https://dev.java/>

ORACLE  
DevLive | Level Up

# Java 20 is Here!

Learn more about this milestone release now.

[Learn about Java 20](#)



- Oracle
- Verlinkung auf OpenJDK
- Release cycle: 6 month
- LTS vs Non-LTS

## The Destination for Java Developers

Hundreds of tutorials, news and videos from the experts, all right here.

### Get Started

- [Getting Started with Java](#)
- [Java Language Basics](#)
- [Downloading Java](#)

### Go Deeper

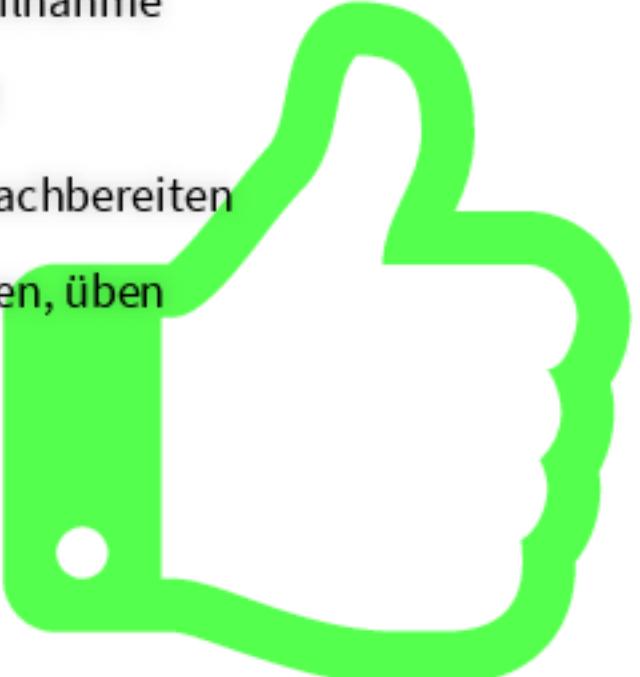
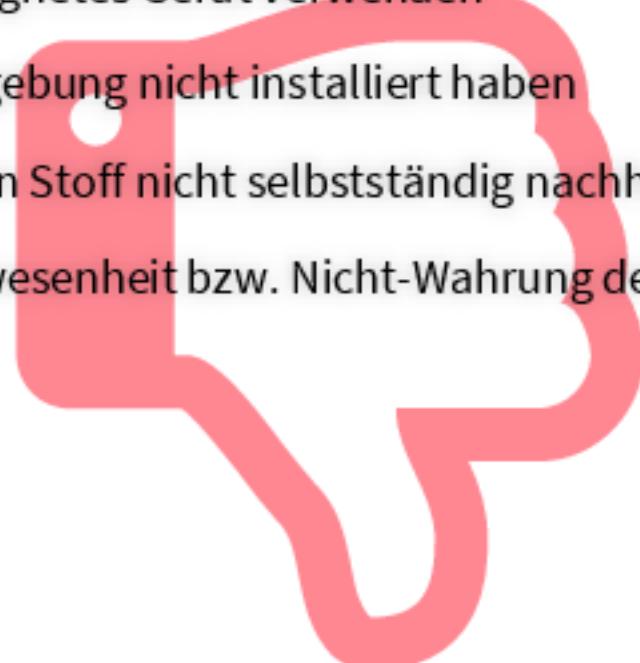
- [Lambda Expressions](#)
- [The Stream API](#)
- [The Collections Framework](#)

### Resources

- [Java News](#)
- [Official Java Podcast and More](#)
- [JavaOne Conference 2022](#)

# Agenda

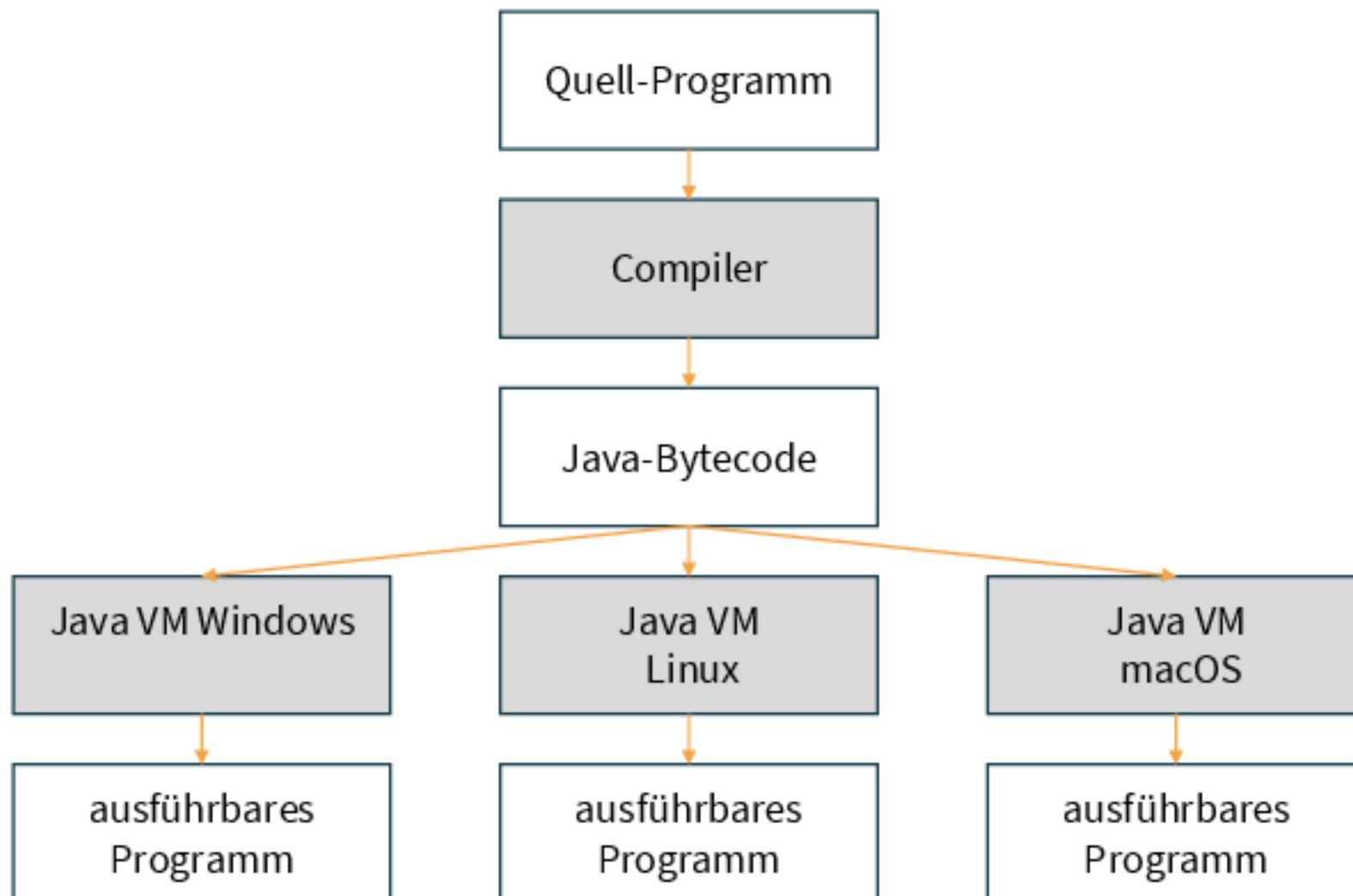
1	Vorstellungsrunde	09:00 – 09:30
2	Eckdaten des Moduls	09:30 – 09:40
3	<b>Erfolgsfaktoren für die Teilnahme</b>	09:40 – 09:50
4	Installation von Java	09:50 – 10:30
5	Pause	10:30 – 10:45
6	Diskussionen über das Programmieren und erste Übungen mit Java	10:45 – 12:15
7	Einstieg in Java (zur Nacharbeit)	

Dos	Don'ts
<ul style="list-style-type: none"><li>• Aktive Teilnahme</li><li>• Mitarbeit</li><li>• Inhalte nachbereiten</li><li>• üben, üben, üben</li></ul> 	<ul style="list-style-type: none"><li>• Nicht geeignetes Gerät verwenden</li><li>• Java-Umgebung nicht installiert haben</li><li>• Verpassten Stoff nicht selbstständig nachholen</li><li>• Nicht-Anwesenheit bzw. Nicht-Wahrung der Regeln</li></ul> 

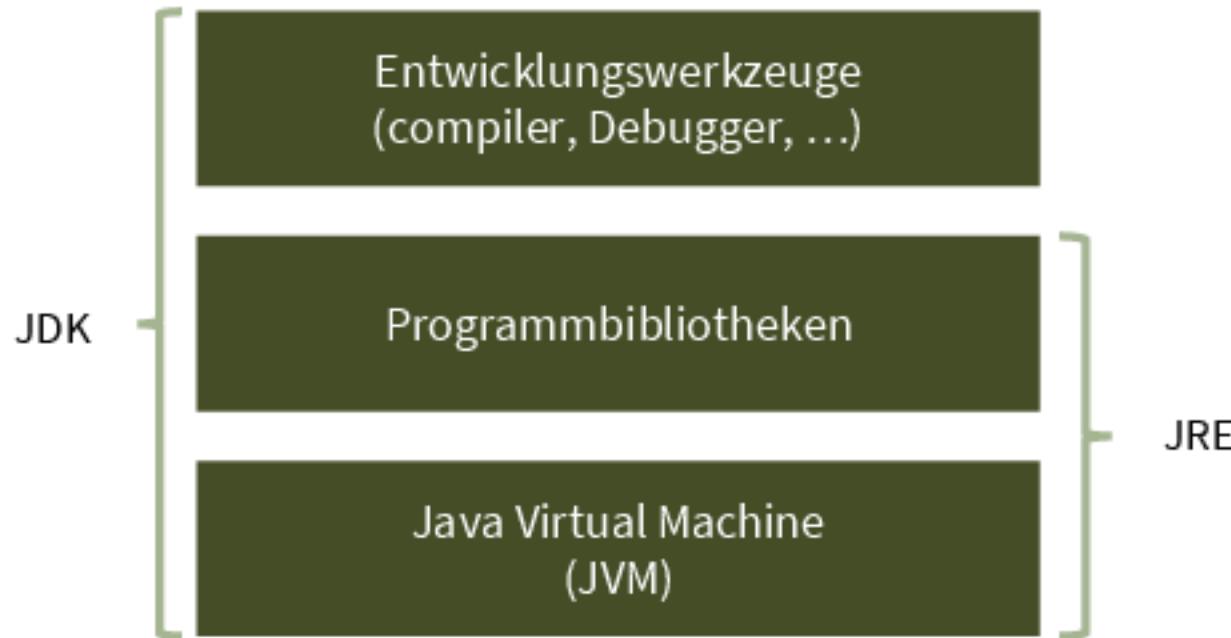
# Agenda

1	<b>Vorstellungsrunde</b>	09:00 – 09:30
2	<b>Eckdaten des Moduls</b>	09:30 – 09:40
3	<b>Erfolgsfaktoren für die Teilnahme</b>	09:40 – 09:50
4	<b>Installation von Java</b>	09:50 – 10:30
5	<b>Pause</b>	10:30 – 10:45
6	<b>Diskussionen über das Programmieren und erste Übungen mit Java</b>	10:45 – 12:15
7	<b>Einstieg in Java</b> (zur Nacharbeit)	

# Plattformunabhängigkeit von Java



# Komponenten und Entwicklungsumgebungen



Bestandteile der Java-Technologie (nach [Abts2020])



<https://whichjdk.com/>

## JDK Project

The goal of this long-running Project is to produce a series of open-source reference implementations of the Java SE Platform, as specified by JSRs in the Java Community Process. The Project ships a feature release every six months according to a strict, time-based model, as proposed.



**What Is this?** The place to collaborate on an open-source implementation of the Java Platform, Standard Edition, and related projects.

## Releases

- [21 \(in development\)](#)
- [20 \(GA 2023/03/21\)](#)
- [19 \(GA 2022/09/20\)](#)
- [18 \(GA 2022/03/22\)](#)
- [17 \(GA 2021/09/14\)](#)
- [16 \(GA 2021/03/16\)](#)
- [15 \(GA 2020/09/15\)](#)
- [14 \(GA 2020/03/17\)](#)
- [13 \(GA 2019/09/17\)](#)
- [12 \(GA 2019/03/19\)](#)
- [11 \(GA 2018/09/25\)](#)
- [10 \(GA 2018/03/20\)](#)

## Resources

- Development list: [jdk dev](#)
- Main-line code repository: <https://github.com/openjdk/jdk/>

Oracle Java SE Support Roadmap<sup>\*\*</sup>

Release	GA Date	Premier Support Until	Extended Support Until
7 (T7)	July 2011	July 2019	July 2022****
8 (T8)	March 2014	March 2022	December 2029****
9 (non-LTS)	September 2017	March 2019	Not Available
10 (non-LTS)	March 2018	October 2018	Not Available
11 (T9)	September 2018	September 2025	September 2025
12 (non-LTS)	March 2019	October 2019	Not Available
13 (non-LTS)	September 2019	March 2021	Not Available
14 (non-LTS)	March 2020	October 2020	Not Available
15 (non-LTS)	September 2020	March 2021	Not Available
16 (non-LTS)	March 2021	October 2021	Not Available
17 (T10)	September 2021	December 2029****	September 2029****
18 (non-LTS)	March 2022	October 2022	Not Available
19 (non-LTS)***	September 2023	March 2025	Not Available
20 (non-LTS)***	May 1 2024	October 2025	Not Available
21 (T11)***	September 2025	October 2028	September 2029

<https://www.oracle.com/java/technologies/java-se-support-roadmap.html>

Last update: 2023/03/21 14:20 UTC

# jdk.java.net

**Production and Early-Access OpenJDK Builds, from Oracle**

**Ready for use:** [JDK 20](#), [JDK 19](#), [JavaFX 20](#), [JMC 8](#)

**Early access:** [JDK 21](#), [Generational ZGC](#), [JavaFX 21](#),  
[Jextract](#), [Loom](#), [Metropolis](#), [Panama](#), & [Valhalla](#)

*Looking to learn more about Java? Visit [dev.java](#) for the latest Java developer news and resources.*

*Looking for Oracle JDK builds and information about Oracle's enterprise Java products and services? Visit the [Oracle JDK Download page](#).*



The screenshot shows a product page for JavaFX. At the top, there is a purple header bar with the word "JavaFX". Below it is a large orange "JavaFX" logo. Underneath the logo is a grey horizontal bar labeled "Basisdaten". The page contains the following data in a tabular format:

Entwickler	Oracle
Aktuelle Version	19 (13. September 2022)
Betriebssystem	Windows, macOS, Linux
Programmiersprache	Java
Kategorie	Framework
Lizenz	GPL mit GPL linking exception

At the bottom of the page, there is a blue button labeled "OpenJFX auf [java.net](#) und [openjfx.io](#)". To the right of the table, there is a vertical list of bullet points:

- Ablösung von Swing, AWT
- Anfänglich sehr viel Hoffnung
- Wechselhafte Geschichte
- FXML (CSS)
- SceneBuilder
- Versionssprung von 2 auf 8
- „Mobile first“
- „Web first“

# Vorbereitungen (eine Möglichkeit ohne IDE bzw. ohne Maven/Graddle)

## Java Development Kit (JDK)

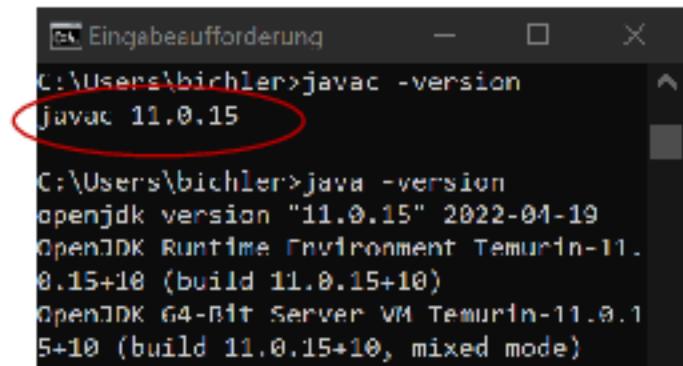
- Verschiedene Alternativen:
  - <https://openjdk.java.net/>
  - <https://www.oracle.com/java/technologies/downloads/>
  - ... siehe <https://whichjdk.com/>
- Lade das für Ihr System geeignete Paket herunter  
Installiere das JDK auf dem Rechner (exe, zip, ...)

## Windows: Path-Variable

- Exe: Ermittle, in welchem Verzeichnis die Java Programme installiert wurden, z.B. unter Pfad:  
C:\Program Files\Java\jdk11.0.15+10\bin
- ZIP: Kopiere das JDK an einen günstigen Ort (direkt ins Projekt oder systemweit für alle Projekte erreichbar)
- Umgebungsvariablen unter Erweiterte Systemeinstellungen öffnen
- JDK-Pfad der Variablen Path hinzufügen (ohne andere Werte zu verändern)

## Test

- Eingabeaufforderung (Windows) oder Terminal (macOS) öffnen
- Eingeben: `javac -version`
- Ergebnis prüfen



```
Eingabeaufforderung
C:\Users\bichler>javac -version
javac 11.0.15

C:\Users\bichler>java -version
openjdk version "11.0.15" 2022-01-19
OpenJDK Runtime Environment Temurin-11.0.15+10 (build 11.0.15+10)
OpenJDK 64-Bit Server VM Temurin-11.0.15+10 (build 11.0.15+10, mixed mode)
```

## Test mit Hilfe eines einfaches Programmes, ob JAVA läuft

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

- Speichern als Datei `HelloWorld.java`
- Im Verzeichnis `javac HelloWorld.java` ausführen
- Danach `java HelloWorld` ausführen
- Es wird `Hello Word` ausgegeben.

## Editor installieren

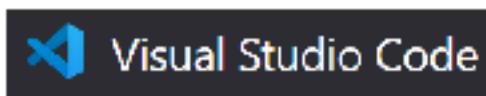
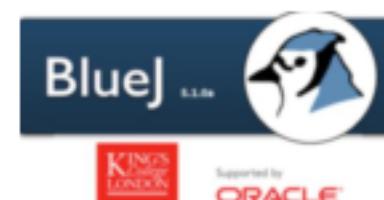
- unter Windows könnte der *Editor* und macOS *TextEdit* verwendet werden
- allerdings nicht komfortabel, da keine Unterstützung durch z.B. farbliche Hervorhebungen
- bessere Alternativen:
  - Windows: <https://notepad-plus-plus.org/>
  - macOS: <https://macromates.com/>
  - ...

## Vertiefende Informationen

- Dokumentation Java 20:  
<https://docs.oracle.com/en/java/javase/20/>
- Dokumentation der Application Programming Interfaces (API) Java 20:  
<https://docs.oracle.com/en/java/javase/20/docs/api/>

# Entwicklungsumgebung mit IDE

- In den ersten Einheiten reicht zunächst eine einfache Umgebung aus:
  - Wie eben vorher beschrieben oder
  - eine Online-Entwicklungsumgebung bzw. Editor und Compiler
    - <https://ideone.com/>
    - <https://www.jdoodle.com/online-java-compiler/>
    - ...
- Für später wird eine **IDE** empfohlen (Projektverwaltung, Repository, Debuggen, Syntax-Highlighting, ... )
  - Auch zunächst lokal, um Entwicklungsschritte nachvollziehen zu können
  - Entwicklung komplett in einer Cloud ist theoretisch auch möglich
  - Beispiele für IDEs



...

1	Vorstellungsrunde	09:00 – 09:30
2	Eckdaten des Moduls	09:30 – 09:40
3	Erfolgsfaktoren für die Teilnahme	09:40 – 09:50
4	Installation von Java	09:50 – 10:30
5	Pause	10:30 – 10:45
6	Diskussionen über das Java-Programmieren und erste Übungen mit Java	10:45 – 12:15
7	Einstieg in Java (zur Nacharbeit)	

# Agenda

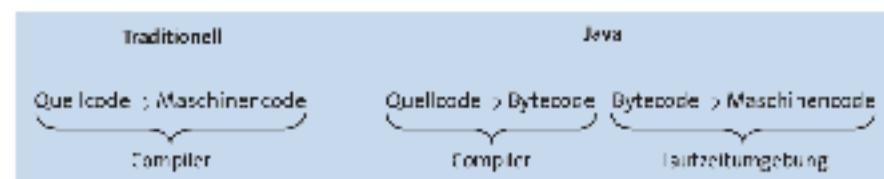
1	<b>Vorstellungsrunde</b>	09:00 – 09:30
2	<b>Eckdaten des Moduls</b>	09:30 – 09:40
3	<b>Erfolgsfaktoren für die Teilnahme</b>	09:40 – 09:50
4	<b>Installation von Java</b>	09:50 – 10:30
5	<b>Pause</b>	10:30 – 10:45
6	<b>Diskussionen über das Java-Programmieren und erste Übungen mit Java</b>	10:45 – 12:15
7	<b>Einstieg in Java</b> (zur Nacharbeit)	

# Eigenschaften von Java

- OO-Sprache, d.h. sie unterstützt alle zentralen Aspekte der Objektorientierung wie Klassen, Objekte, Vererbung und Polymorphie)
- Sprache ist bewusst einfach gehalten (keine expliziten Zeiger und keine Mehrfachvererbung).
- Java ist eine stark typisierte Sprache.
- Java nutzt Unicode.
- Die in einem Programm benutzten Klassen können als Dateien an unterschiedlichen Orten liegen. Sie werden erst zur Laufzeit des Programms bei Bedarf geladen.
- „Garbage Collection“: automatisches Speichermanagement.
- Strukturierte Behandlung von Laufzeit-Fehlern (Exception-Handling).
- Java unterstützt den parallelen Ablauf von eigenständigen Programmabschnitten (Multithreading) und die Synchronisation bei konkurrierenden Datenzugriffen.
- Java-Klassenbibliothek bietet eine einfache Möglichkeit für die Netzwerkkommunikation und stellt viele APIs und Interfaces zur Verfügung.
- ...

# Eigenschaften von Java (Popularität)

- Plattformunabhängig, Bytecode: auch für ARM
- Java Platform:
  - Auch andere Sprachen verwenden die VM und Java-Bibliotheken (Groovy, Kotlin, Scala,...)
  - Java stellt viele Standards zur Verfügung (Datenstrukturen, Thread-Behandlung,...)
  - Mittlerweile unterstützen bereits bekannte Betriebssysteme diese „leichtgewichtigen Prozesse“, so dass die JVM diese nicht nachbilden muss.
- OOP begrenzt Fehler deutlich gegenüber Sprachen wie C in großen Projekten ein.
- Java ist verbreitet
- Just-in-time-compilation
- Java-Security-Modell
- Anpassungsfähig



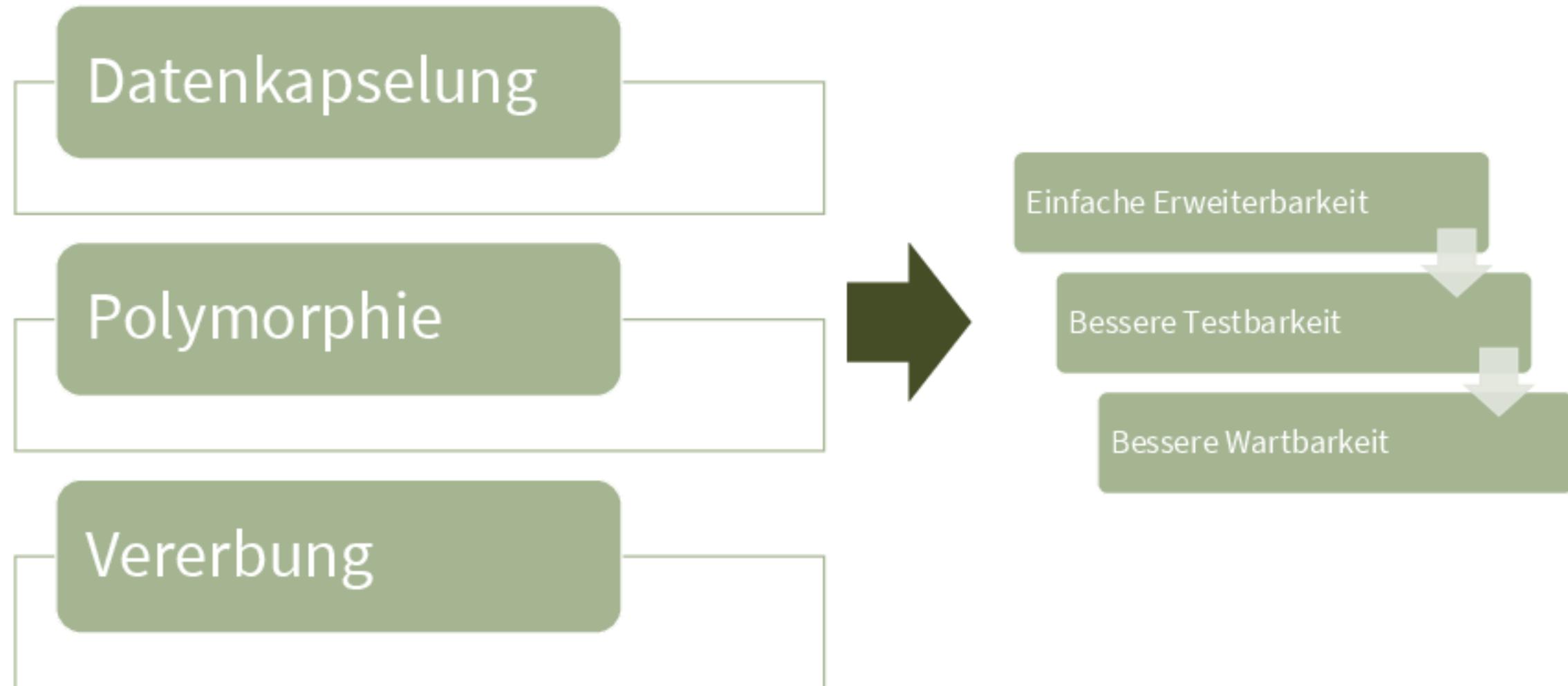
# Wofür eher nicht einsetzen?

- Als „General-Purpose-Language“ ist Java nicht für einige Spezialbereiche geeignet.
- Reines Skripting
- Wenn systemnahe Eigenschaften gebraucht werden (z.B.: Taktfrequenz, Zugriff auf bestimmte Speicherzellen)
- Java ist keine hardwarenahe Programmiersprache.

Aus den genannten Nachteilen, dass Java nicht auf die Hardware zugreifen kann, folgt, dass die Sprache nicht so ohne Weiteres für die Systemprogrammierung eingesetzt werden kann. Treibersoftware, die Grafik-, Sound- oder Netzwerkkarten anspricht, lässt sich in Java nur über Umwege realisieren. Genau das Gleiche gilt für den Zugriff auf die allgemeinen Funktionen des Betriebssystems, die Windows, Linux oder ein anderes System bereitstellt. Typische System-Programmiersprachen sind C(++) oder Go. Da außerdem die JVM eine gewisse Größe hat, ist Java für Microcontroller bisher keine Option.

Aus diesen Beschränkungen ergibt sich, dass Java eine hardwarenahe Sprache nicht ersetzen kann. Doch das muss die Sprache auch nicht! Jede Sprache hat ihr bevorzugtes Terrain, und Java ist eine allgemeine Applikationsprogrammiersprache; C(++) darf immer noch für Hardwaretreiber, eingebettete Systeme und virtuelle Maschinen herhalten. Die Standard-JVM ist (bisher noch) in C++ geschrieben und wird vom GCC-Compiler bzw. Microsoft Visual Studio und XCode übersetzt. C und C++ werden nie verschwinden; diese Sprachen sind wie Mikroben im Vulkangas – sie überdauern alles Leben. Und so wie am Vu kan die Mikroorganismen die Nahrungsgrundlage für andere Organismen bilden, so werden wir auch nicht ohne die systemnahen Sprachen C(++) oder Go auskommen.

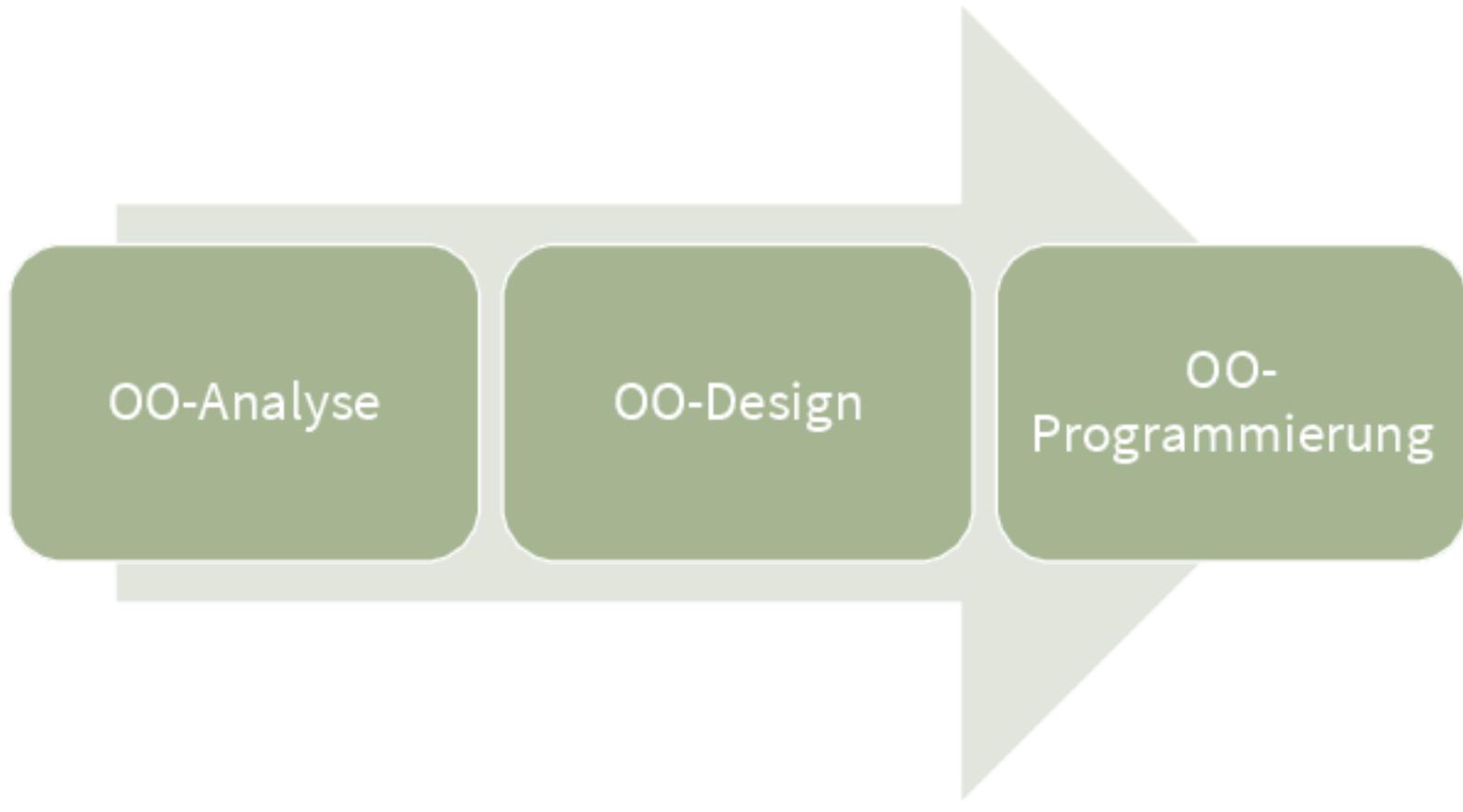
Soll ein Java-Programm trotzdem systemnahe Eigenschaften nutzen – und das kann es mit entsprechenden Bibliotheken ohne Probleme –, bietet sich zum Beispiel der native Aufruf einer Systemfunktion an. Native Methoden sind Unterprogramme, die nicht in Java implementiert werden, sondern in einer anderen Programmiersprache, häufig in C(++) . In manchen Fällen lässt sich auch ein externes Programm aufrufen und so etwa die Windows-Registry manipulieren oder Dateirechte setzen. Es läuft aber immer darauf hinaus, dass die Lösung für jede Plattform immer neu implementiert werden muss.



# Wichtige OOP-Prinzipien

- Einer einzigen Verantwortung
- Trennung der Anliegen
- Wiederholungen vermeiden
- Offen für Erweiterungen, geschlossen für Änderungen
- Trennung der Schnittstelle von der Implementierung
- Umkehr der Abhängigkeiten
- Umkehr des Kontrollflusses
- Mach es testbar

# Phasen der objektorientierten Entwicklung



Software Engineering

## Bis zum nächsten Termin

- Eine Java-Umgebung einrichten mit oder ohne IDE
- Vertraut machen mit Java-Basics, siehe Anhang oder eins der Beginner-Tutorials

# **Einstieg in Java**

## **Code-Beispiele und Aufgaben**

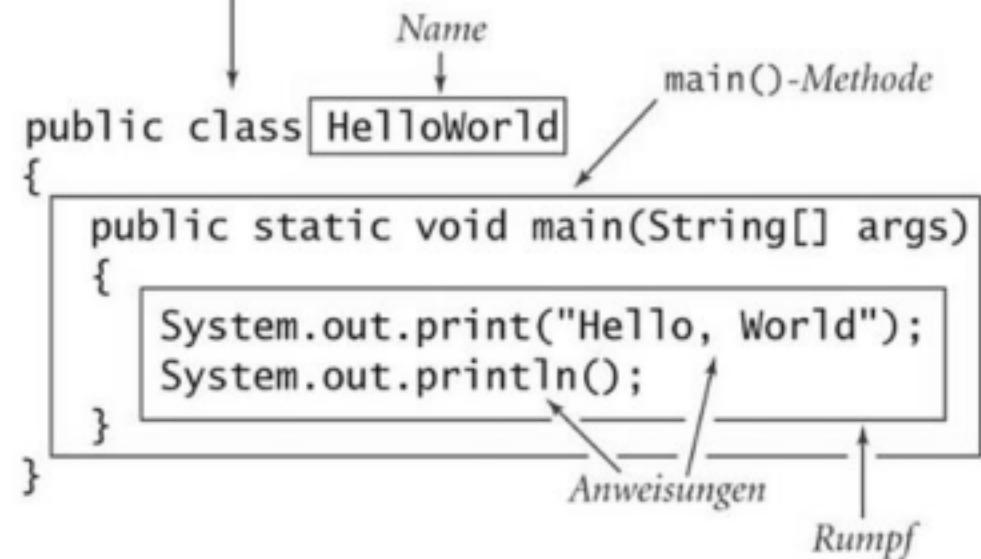
## Programmcode

```
class IU
{
    Public static void main(String[] args)
    {
        System.out.println("IU");
    }
}
```

- Speichern als Datei `IU.java`
- Im Verzeichnis `javac IU.java` ausführen
- Danach `java IU` ausführen

## Java-Syntax

Textdatei namens `HelloWorld.java`



Anatomie eines Programms

## – Klassenname (Schlüsselwort `class`)

- keine Leer- oder Sonderzeichen, beginnend mit Großbuchstaben
- Upper-Camel-Case-Notation üblich (jedes neue Wort Großbuchstabe), z. B. `FirstViewController`
- Dateiname muss mit Klassennamen übereinstimmen

## – Parametername

- keine Leer- oder Sonderzeichen, üblicherweise beginnend mit Kleinbuchstaben

## – Abstände und Einrückungen optional

- Empfehlung: bei jeder geöffneten {-Klammer eine Ebene einrücken

## – Java-Anweisung endet mit Strichpunkt

## – Java-Anweisungen dürfen über mehrere Zeilen reichen

- Zeichenketten dürfen aber nicht über mehrere Zeilen reichen (ggf. mit + verbinden)
- Groß- und Kleinschreibung wird unterschieden
- Konstantenname
  - ausschließlich Großbuchstaben
- Java-Klassenbibliothek
  - in Paketen sind Klassen vordefiniert
  - `java.lang` steht automatisch zur Verfügung
  - bei anderen Import erforderlich (sonst muss immer Paket mit angegeben werden),
    - z.B. `import java.time.LocalDate;`

## Signatur

```
Bibliotheksnname  
public class Math  
  
Signatur . . . Methodenname  
double sqrt(double a)  
Rückgabetypr Argumenttyp  
. . .
```

## Bibliotheksmethode verwenden

```
Bibliotheksnname Methodenname  
double d = Math.sqrt(b*b - 4.0*a*c);  
Rückgabetypr Argument
```

```
import java.time.LocalDate;
Import java.time.format.DateTimeFormatter;
class Programm2 {
    public static void main(String[] args) {
        LocalDatejetzt = LocalDate.now() ;
        DateTimeFormatter meinFormat=DateTimeFormatter.ofPattern(
            "EEEE, d. MMMM yyyy" ) ;
        System.out.println("Heute ist " + meinFormat.format(jetzt) + "!" );
    }
}
```

# Einstieg in Java

## AUFGABE 1

Schreiben Sie ein Programm, das folgende Ausgabe erzeugt:

```
*****
```

Heute ist Dienstag, 29. Juni 2021!

Willkommen zu OOP!

```
*****
```

- einzeilig //
- mehrzeilig von /\* bis \*/
- Javadoc-Kommentare von /\*\* bis \*/
- klare Kommentare sind essenziell!

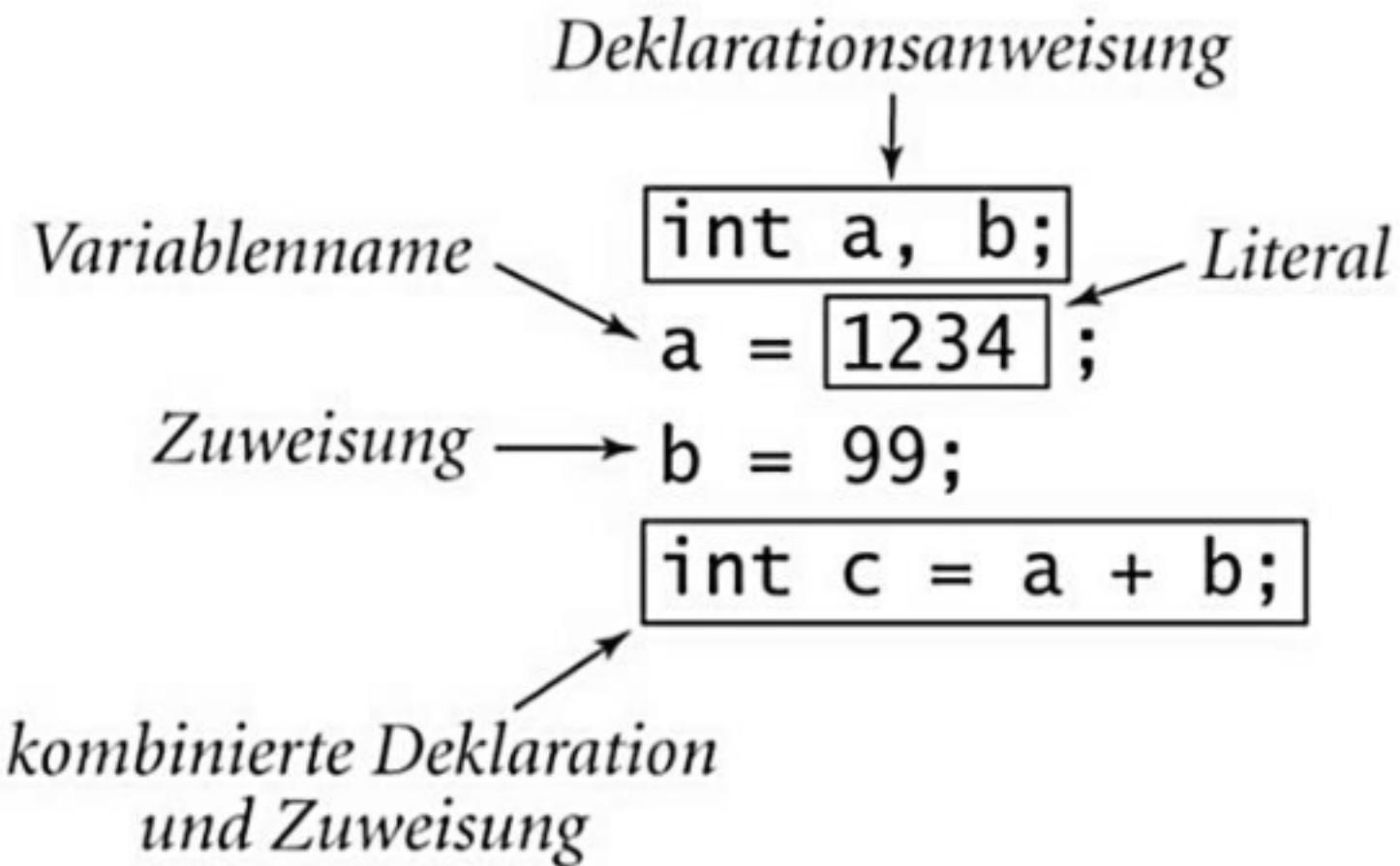
## Primitive Datentypen

- müssen deklariert werden:  
`datentyp varname;`
- müssen initialisiert werden:  
`varname = wert;`
- auch zusammen möglich:  
`datentyp varname = wert;`

DATENTYP	DETAILS
byte	Ganzzahl 1 Byte
short	Ganzzahl 2 Bytes
int	Ganzzahl 4 Bytes
long	Ganzzahl 8 Bytes
boolean	true or false
char	ein Unicode Zeichen
float	Fließkommazahl 4 Bytes
double	Fließkommazahl 8 Bytes

### Datentyp String

- Zeichenfolgen
- im engeren Sinn kein elementarer/primitiver Datentyp, sondern Klasse
- kann zusammen mit primitiven Datentypen als integrierter Datentyp angesehen werden



# Variablen-deklaration ohne Typangabe

## var

- Seit Java 10 ist die Verwendung von **var** erlaubt
- Verfahren heißt: *Local-Variable Type Inference*

```
var i = 10;           // int
var d = 18.3;         // double
var s = "Java";       // String (Zeichenkette)
```

```
// herkömmlicher Code bis Java 9
String[] saz      = new String[3];
ArrayList<String> list = new ArrayList<>();
Stream<String> stream = list.stream();
```

In solchen Fällen erlaubt **var** einen viel klareren Code ohne Redundanz:

```
// moderner Code ab Java 10
var var  = new String[3];
var list = new ArrayList<String>();
```

PRIORITÄT	OPERATOR	BEDEUTUNG
1 →	()	Methodenaufruf
	[]	Zugriff auf Felder
	.	Zugriff auf Objekte, Methoden etc.
2 →	++-	Inkrement/Dekrement (Postfix, z.B. a++)
3 ←	++-	Inkrement/Dekrement (Präfix, z.B. ++a)
	+-	Vorzeichen
	!~	logisches Nicht, binäres Nicht
	new	Objekte erzeugen
	(typ)	explizite Typumwandlung (Casting)
4 →	* / %	Multiplikation, Division, Restwert
5 →	+-	Addition, Subtraktion
	+	Verbindung von Zeichenketten

PRIORITÄT	OPERATOR	BEDEUTUNG
7 →	>=	Vergleich größer, größer-gleich
	<=	Vergleich kleiner, kleiner-gleich
8 →	== !=	Vergleich gleich, ungleich
9 →	&	logisches Und
10 →	^	logisches Exklusiv-Oder
11 →		logisches Oder
12 →	&&	logisches Und (Short-circuit Evaluation)
13 →		logisches Oder (Short-circuit Evaluation)
15 ←	=	Zuweisung
	+-=	Grundrechenarten und Zuweisung

# **Einstieg in Java**

## **AUFGABE 2**

## Aufgabe 2

Welche Werte nehmen die Variablen nach jeder Anweisung an?

	a	b	t
int a, b;			
a=1234;	1234		
b=99;	1234	99	
int t=a;	1234	99	1234
a=b;	99	99	1234
b=t;	99	1234	1234

# **Einstieg in Java**

## **AUFGABE 3**

```
class Programm3 {  
    public static void main (String[] args) {  
        int i = 1000000;  
        i = i * i;  
        System.out.println("Ergebnis ist " + i);  
    }  
}
```

```
class Programm4 {  
    public static void main (String[] args) {  
        double radius = 6.371;  
        double umfang = radius * 2 * 3.14  
        System.out.println("Umfang ist " + umfang);  
    }  
}
```

```
class Programm5 {  
    public static void main (String[] args) {  
        double radius = 6371.0E250;  
        double inhalt = radius * radius * 3.14;  
        System.out.println("Inhalt ist " + inhalt);  
    }  
}
```

# **Einstieg in Java**

## **AUFGABE 4**

## Programmcode Variante 1

```
class Programm 6 {  
    Public static void main(String[] args)  
    {  
        double a = 0.4;  
        double b = 0.6;  
        double c = a + 0.1;  
        double d = b - 0.1;  
        System.out.println(c == d); // weil 0.5 == 0.5  
        System.out.println(c - d);  
    }  
}
```

## Programmcode Variante 2

```
class Programm 7 {  
    Public static void main(String[] args)  
    {  
        double a = 0.7;  
        double b = 0.9;  
        double c = a + 0.1;  
        double d = b - 0.1;  
        System.out.println(c == d);  
        System.out.println(c - d);  
    }  
}
```

### Befehlszeile

- über Befehlszeilenargument (Array): args [0] ... [n]
- Umwandlungsmethoden:
  - Integer.parseInt(args[0])
  - Double.parseDouble(args[0])
  - Long.parseLong(args[0])
- kein Dialog möglich

# **Einstieg in Java**

## **AUFGABE 5**

### Scanner

- über `java.util.Scanner`
- erzeugen:
  - `java.util.Scanner scan = new java.util.Scanner (System.in) ;`
- lesen/scannen, z. B.:
  - `int a = scan.nextInt () ;`
  - `double b = scan.nextDouble () ;`
  - `String s = scan.nextLine () ;`
- schließen:
  - `scan.close () ;`

### Aufgabe 5

Schreiben Sie ein Programm, das den Benutzer auffordert, zwei Zahlen einzugeben und das dann alle mit den vier Grundrechenarten berechenbaren Ergebnisse dieser zwei Zahlen ausgibt.

# Einstieg in Java

## AUFGABE 6

## Aufgabe 6

Schreiben Sie ein Programm, das nach Eingabe eines Temperaturwertes in Grad Celsius den Temperaturwert in Grad Fahrenheit und Kelvin umrechnet und ausgibt.

Hinweis: 0 Grad Celsius entsprechen 273,15 Kelvin

$$\text{Grad}_{\text{Celsius}} = \frac{5}{9} \cdot (\text{Grad}_{\text{Fahrenheit}} - 32)$$

Kofler, M. (2018). Java - Der Grundkurs (2. Auflage). Bonn: Rheinwerk.

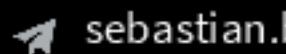
Sedgewick, R. & Wayne, K. (2011). Einführung in die Programmierung mit Java. Hallbergmoos: Pearson.

# Danke

Sebastian Bichler



via Teams



[sebastian.bichler@iu.org](mailto:sebastian.bichler@iu.org)

Sommersemester 2023

DSPOOPI01

# Objektorientierte Programmierung I

## mit Java

#02, 13.04.2023, Leipzig

Prof. (FH) Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

Programmierparadigmen  
OOP-Prinzipien  
Java-Basics

- 1 `lectures.getLast().assess()`
- 2 Java-Projekte planen
- 3 Teams-Aufgabe 01 gemeinsam durchgehen
- 4 Programmierkonzepte/-paradigmen
- 5 OOP-Prinzipien
- 6 Discussion on „How to program simply by not using OOP“
- 7 Praktischer Teil

## Programmierparadigmen OOP-Prinzipien Java-Basics

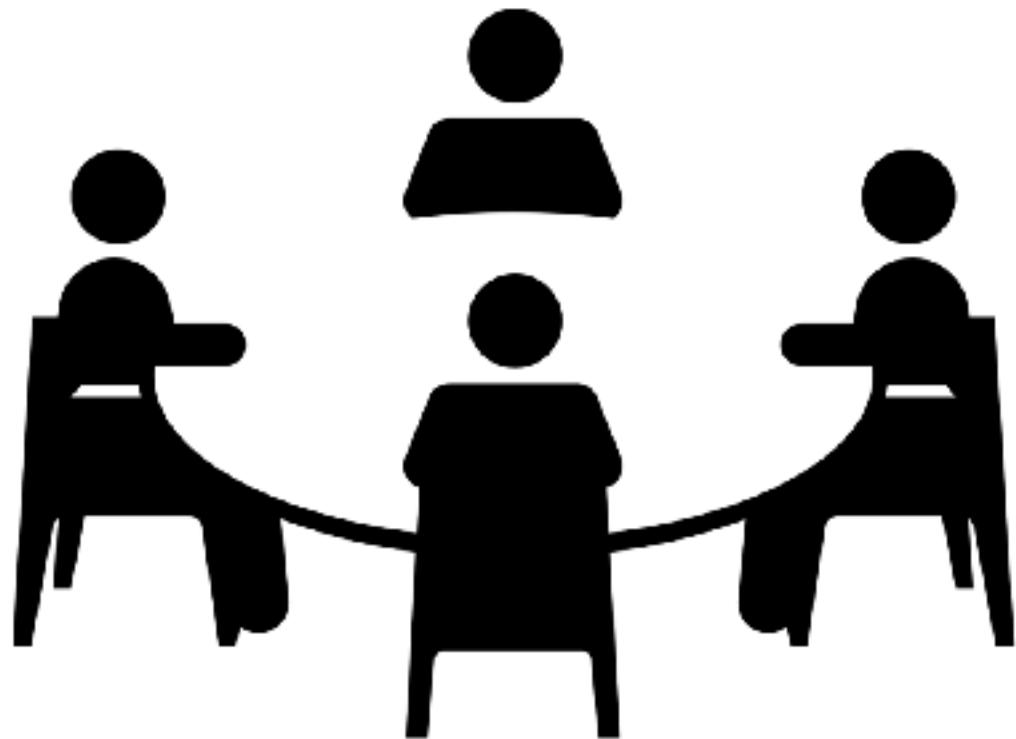
- 1 `lectures.getLast().assess()`
- 2 Java-Projekte planen
- 3 Teams-Aufgabe 01 gemeinsam durchgehen
- 4 Programmierkonzepte/-paradigmen
- 5 OOP-Prinzipien
- 6 Discussion on „How to program simply by not using OOP“
- 7 Praktischer Teil

## Programmierparadigmen

### OOP-Prinzipien

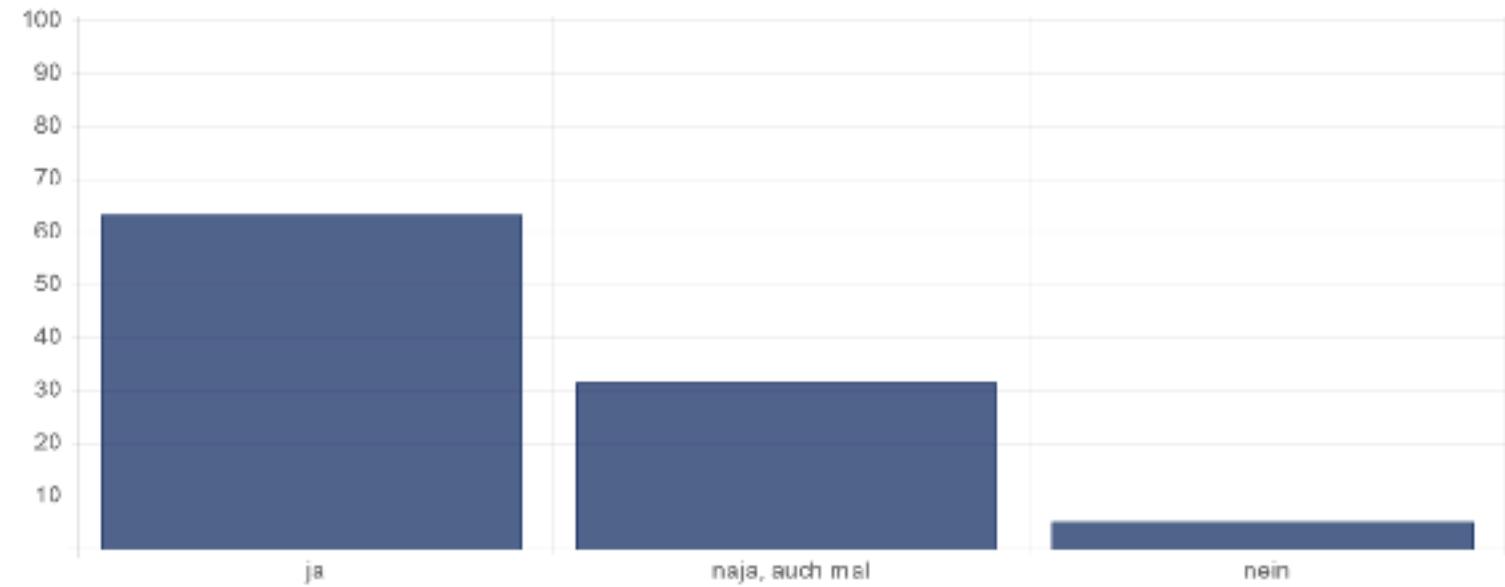
### Java-Basics

## lectures.getLast().assess() vs assess(lastLecture)



Teamwork, Kollaboration, ...

# Programmierst du gerne?



Teilnehmer: 19

Antwortmöglichkeiten:

- 12 63% ja
- 8 32% naja, auch mal
- 1 6% nein

# Setzt du bewusst Programmierparadigmen ein?

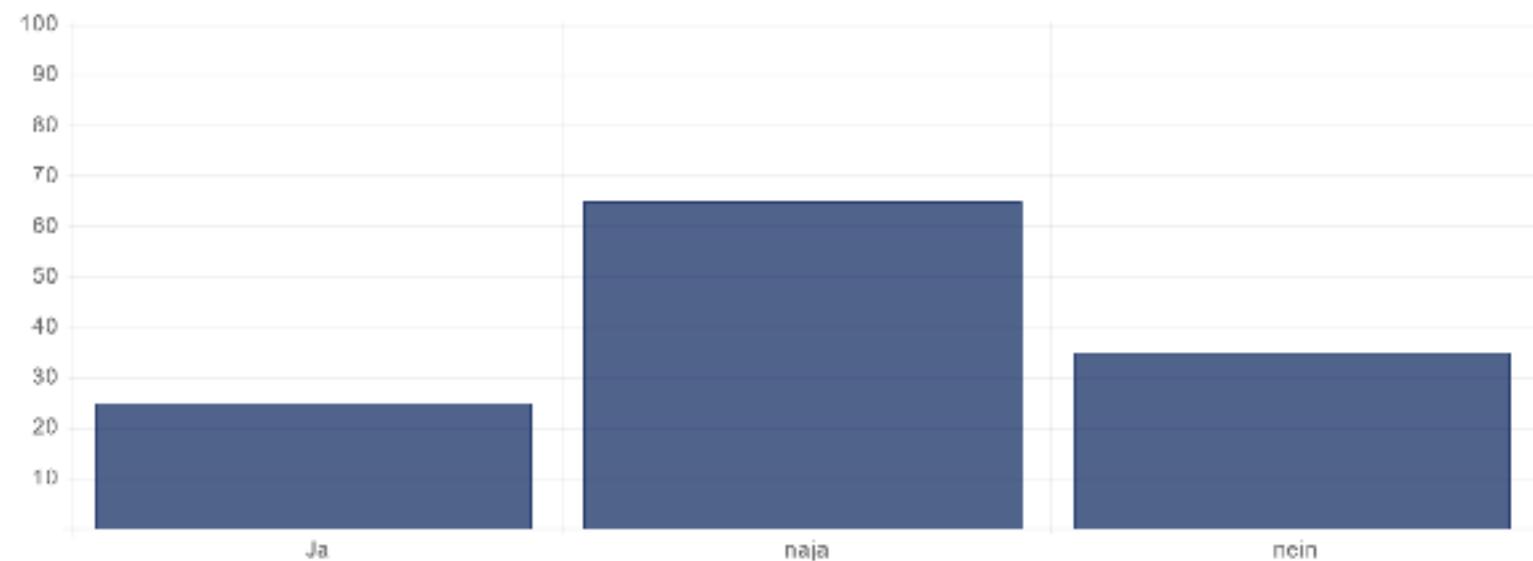


Teilnehmer: 20

Antwortmöglichkeiten:

- 10 50% Ja
- 12 60% Nein

# Programmierst du gerne in Java?

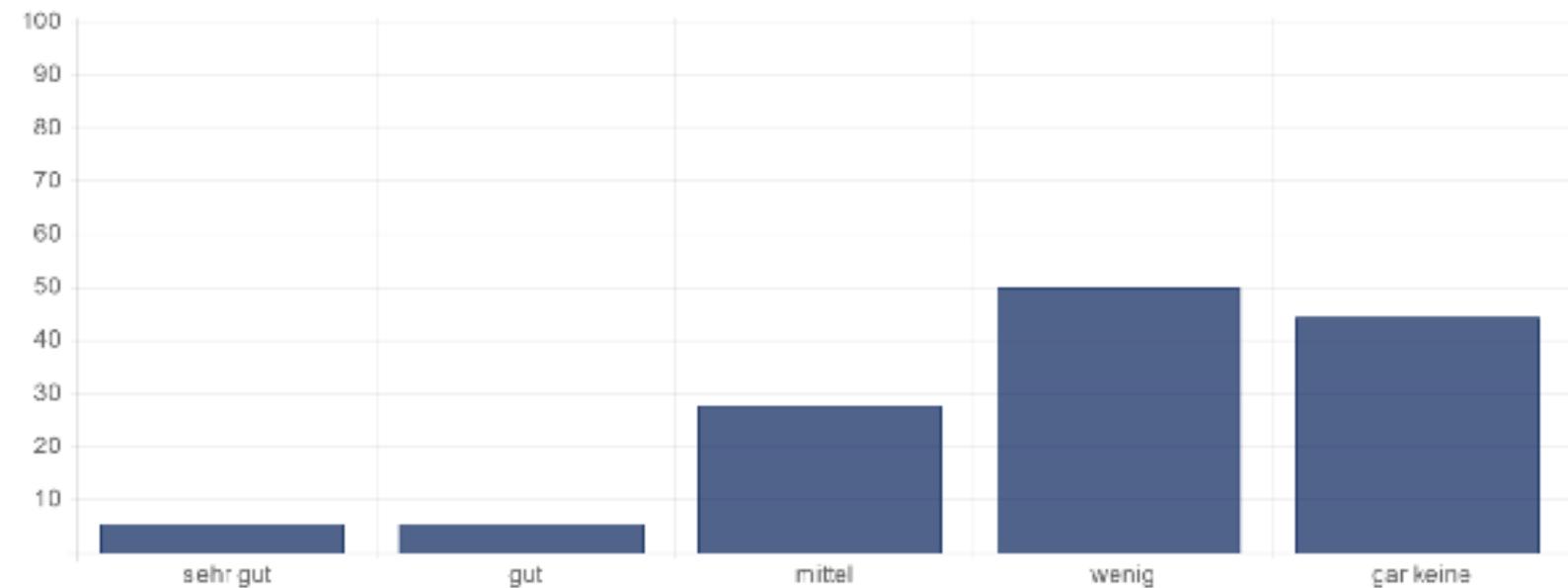


Teilnehmer: 19

Antwortmöglichkeiten:

- 12 63% ja
- 6 32% naja, auch mal
- 1 5% nein

# Wie schätzt Du Deine Java-Programmierkenntnisse ein?



Teilnehmer: 18

Antwortmöglichkeiten:

- 1 6% sehr gut
- 1 6% gut
- 5 28% mittel
- 9 50% wenig
- 8 44% gar keine

# Java-Projekte

## Kriterien, Eigenschaften

Kommunikation
• Datenaustausch
• Datenbank
• XML/JSON
• Sensoren
• Bluetooth
• ...

Algorithmus
• Datenstrukturen
• (ADT)
• Such-Alg.
• Sortier-Alg.
• Optimierung
• Graphenbasiert
• ...

GUI
• User-Interaktion



<https://blog.logrocket.com/best-java-and-kotlin-development/>

- 1 `lectures.getLast().assess()`
- 2 **Java-Projekte planen**
- 3 **Teams-Aufgabe 01 gemeinsam durchgehen**
- 4 Programmierkonzepte/-paradigmen
- 5 OOP-Prinzipien
- 6 Discussion on „How to program simply by not using OOP“
- 7 Praktischer Teil

## Programmierparadigmen OOP-Prinzipien Java-Basics

# Java-Projekte

## Gewünschte Themen

- ERP, CRM, CMS, Warehouse, Intranet, Wissensbasis/Suchfunktion
- Simples Spiel
- „Programmieren in Minecraft“

Aufgabe 2: Ein Projekt/Aufgabe konkretisieren/vorschlagen

Browser addons (für liferando)

Sprachenprogramm

Vorwaltungstools für mitarbeiter, zeiterfassung, urlaubs usw

???

Simples spiel

Minecraft hack clients

<3

Spiel



Games

Minecraft mod

Eigene vim config mit lua

App/website mit chatgpt api

Scripts zur automatisierung von perleslung

Minecraft 2

Cs 3

Ein gutes abstimmungstool

Sap

- 1 lectures.getLast().assess()**
- 2 Java-Projekte planen**
- 3 Teams-Aufgabe 01 gemeinsam bearbeiten (siehe (überarbeitete) Folien der ersten Einheit)**
- 4 Programmierparadigmen**
- 5 OOP-Prinzipien**
- 6 Discussion on „How to program simply by not using OOP“**
- 7 Praktischer Teil**

- 1 `lectures.getLast().assess()`
- 2 Java-Projekte planen
- 3 *Teams-Aufgabe 01 gemeinsam bearbeiten*
- 4 Programmierkonzepte/-paradigmen
- 5 OOP-Prinzipien
- 6 Discussion on „How to program simply by not using OOP“
- 7 Praktischer Teil

# Programmierkonzepte/-paradigmen

## Historischer Überblick mit Fokus auf „Divide & Conquer“

Programmierkonzept	Typische Elemente	Einsatzgebiete
Maschinencode	<code>1001001</code> ; 1 und 0 als einzige Ausdrucksmöglichkeit	Nur noch historisch; heute keine Einsatzgebiete mehr
Assemblercode	<code>movb \$0x61</code> ; Prozessorbefehle und direktes Ansprechen von technischen Speicheradressen	Steuerungen für elektronische Geräte (Lüftungen, Motoren, Klimaanlagen), Reaktive Systeme (Sensorsysteme), hardwarenahe Programmierung
Imperative Programmierung	<code>WHILE</code> , <code>FOR</code> , <code>GOTO</code> ; Weiterentwicklung von Assemblercode, erlauben Schleifen und gezielte Sprünge	Kleine Programme zur Lösung einfacher Aufgaben, in der Regel in speziellen Programmiersprachen
Strukturierte Programmierung		
Objektorientierte Programmierung		
Komponentenbasierte Entwicklung		

# Programmierparadigmen bzw. -konzepte

## Historischer Überblick mit Fokus auf „Divide & Conquer“

Programmierkonzept	Typische Elemente	Einsatzgebiete
Strukturierte Programmierung	<i>Prozeduren</i> ; Strukturierung eines Programms in Prozeduren (Funktionen) und Unterprozeduren (Unterfunktionen)	Einfache Webanwendungen, Technische Steuerkomponenten
Objektorientierte Programmierung	<i>Klassen, Objekte, Beziehungen</i> ; ein Programm besteht aus kooperierenden Objekten	Große und komplexe industrielle Softwaresysteme
Komponentenbasierte Entwicklung	<i>Komponenten, Schnittstellen</i> ; Teile eines Systems werden als eine Komponente zusammengefasst, die ganz bestimmte Aufgaben erfüllt	Wiederverwendung von bereits programmierten Funktionen; Einzelne Komponenten sind in der Regel objektorientiert programmiert.

# Programmierparadigma

- Ein **Programmierparadigma** ist ein fundamentaler Programmierstil. Der Programmierung liegen je nach Design der einzelnen Programmiersprache verschiedene Prinzipien zugrunde. Diese sollen den Entwickler bei der Erstellung von ‚gutem Code‘ unterstützen, in manchen Fällen sogar zu einer bestimmten Herangehensweise bei der Lösung von Problemen zwingen.
- Programmierparadigmen unterscheiden sich durch ihre Konzepte für die Repräsentation von statischen (wie beispielsweise Objekte, Methoden, Variablen, Konstanten) und dynamischen (wie beispielsweise Zuweisungen, Kontrollfluss, Datenfluss) Programmelementen.
- Grundlegend für den Entwurf von Programmiersprachen sind die Paradigmen der imperativen und der deklarativen Programmierung. Beim letzteren sind als wichtige Ausprägungen die Paradigmen der funktionalen und der logischen Programmierung zu nennen.
- Die verschiedenen Paradigmen sind, bezogen auf einzelne Computerprogramme, nicht als konkurrierende bzw. alternative Programmierstile zu verstehen. Vielmehr können „viele Programmiersprachen mehrere Paradigmen gleichzeitig unterstützen“

# Programmierparadigma

- Höhere Programmiersprachen sind problemspezifisch.
- Imperativ
  - Berechnung ist eine Folge von Zuständen
  - Zustandsänderungen erfolgen über Zuweisungen
- Funktional
  - Programm entspricht einer Menge von Funktionen
  - Berechnung entspricht dem Ausrechnen eines Ausdrucks
  - $\lambda$  – Kalkül beschreibt die Grundlagen dafür (Theoretische Informatik)
- logisch
  - Programm entspricht einer Menge von Relationen
  - Berechnung entspricht
- constraint-basiert (Weiterentwicklung der logischen Programmierung)
- objektorientiert
  - Quasi-Parallelisierung, Polymorphie, ... mit Vor- und Nachteilen (Debugging, ...)
- Es gibt weitere Paradigma, z.B.:
  - Aspektorientierte Programmierung (AspectJ) behandelt „Cross-Cutting Concerns“)

# Strukturierte Programmierung als Vorläufer der OOP

## [x] Routine

Ein abgegrenzter, separat aufrufbarer Bestandteil eines Programms. Eine Routine kann entweder Parameter haben oder ein Ergebnis zurückgeben. Eine Routine wird auch als Unterprogramm bezeichnet.

Routinen sind das Basiskonstrukt der strukturierten Programmierung. Indem ein Programm in Unterprogramme zerlegt wird, erhält es seine grundsätzliche Struktur.

## [x] Funktion

Eine Funktion ist eine Routine, die einen speziellen Wert zurückliefert, ihren Rückgabewert.

## [x] Prozedur

Eine Prozedur ist eine Routine, die keinen Rückgabewert hat. Eine Übergabe von Ergebnissen an einen Aufrufer kann trotzdem über die Werte der Parameter erfolgen.

- Daten werden nicht als homogener Speicherbereich betrachtet.
- Es werden globale, lokale, statisch und dynamisch allozierte Variablen verwendet.

**„Eine Programmiersprache, die nicht die Art und Weise beeinflusst, in der du über das Programmieren nachdenkst, ist es nicht wert, dass man sie kennt.“**

**Alan Perlis**

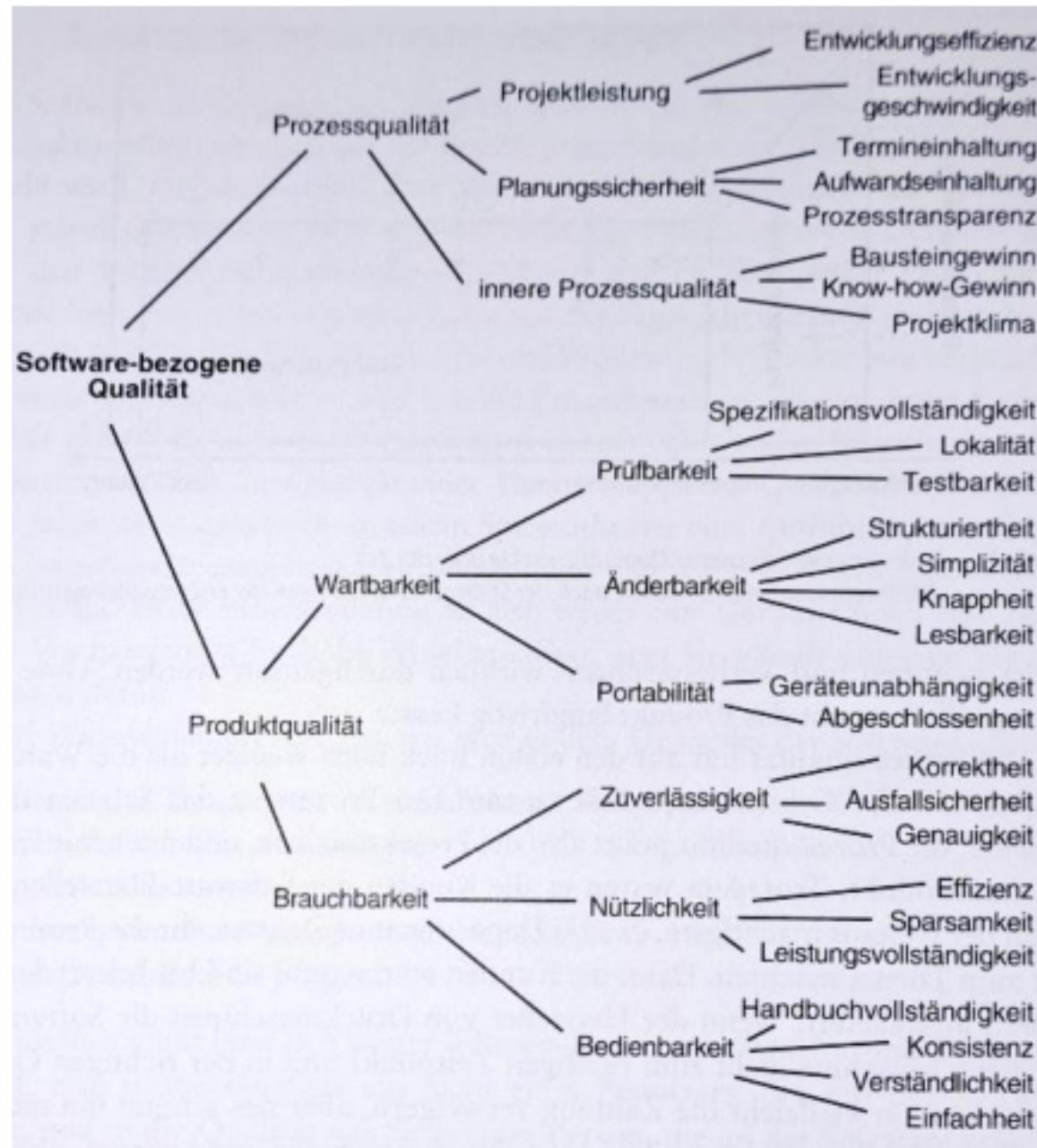


- Entscheidend ist die Denkweise
- und damit die Anwendung der Paradigma und damit verknüpfter Prinzipien

# Aspekte von Programmiersprachen

- Sprachbeschreibung
- Berechnungsmodell, Datentypen, Abstraktionsmöglichkeiten
- Implementierbarkeit
- Anwendbarkeit
- Robustheit
- Portabilität
- Orthogonalität
- ...

„Gute“ Software

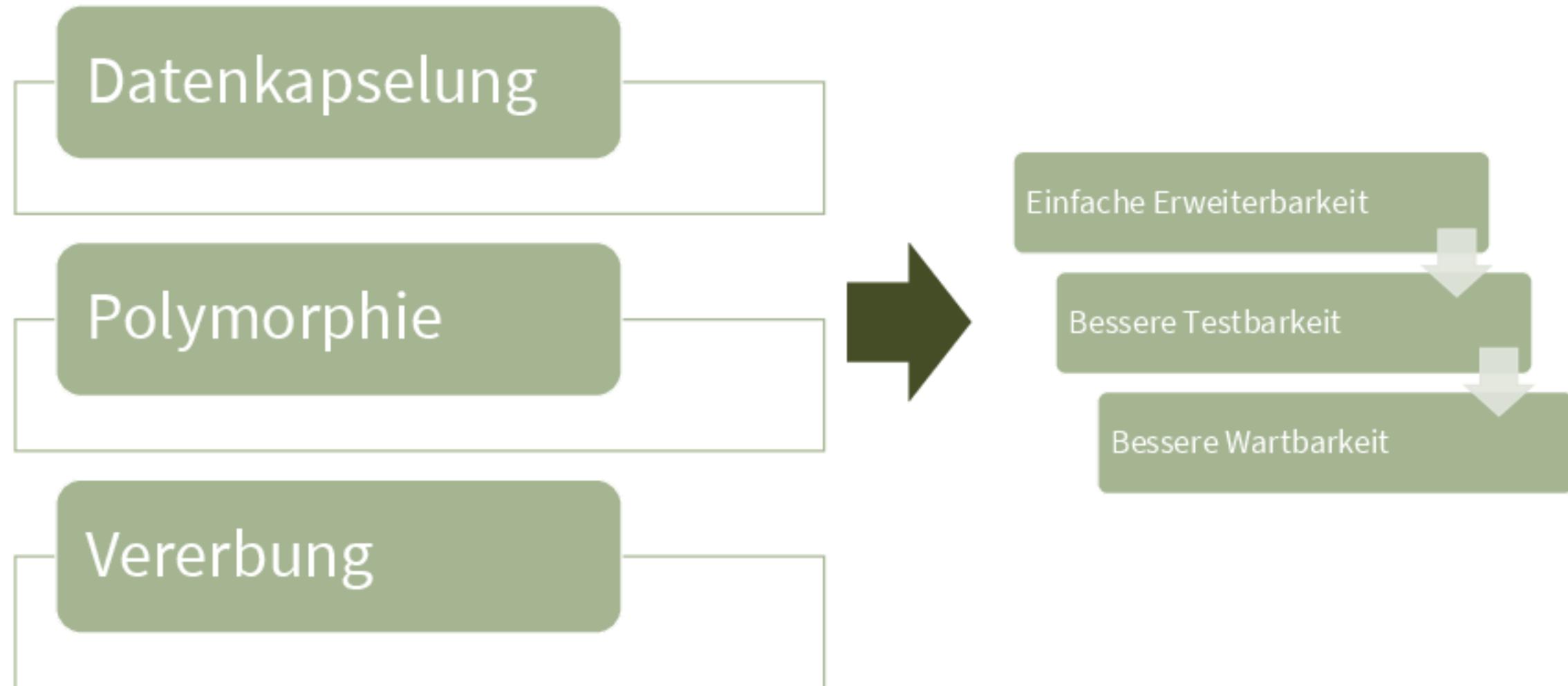


## „Gute“ Software

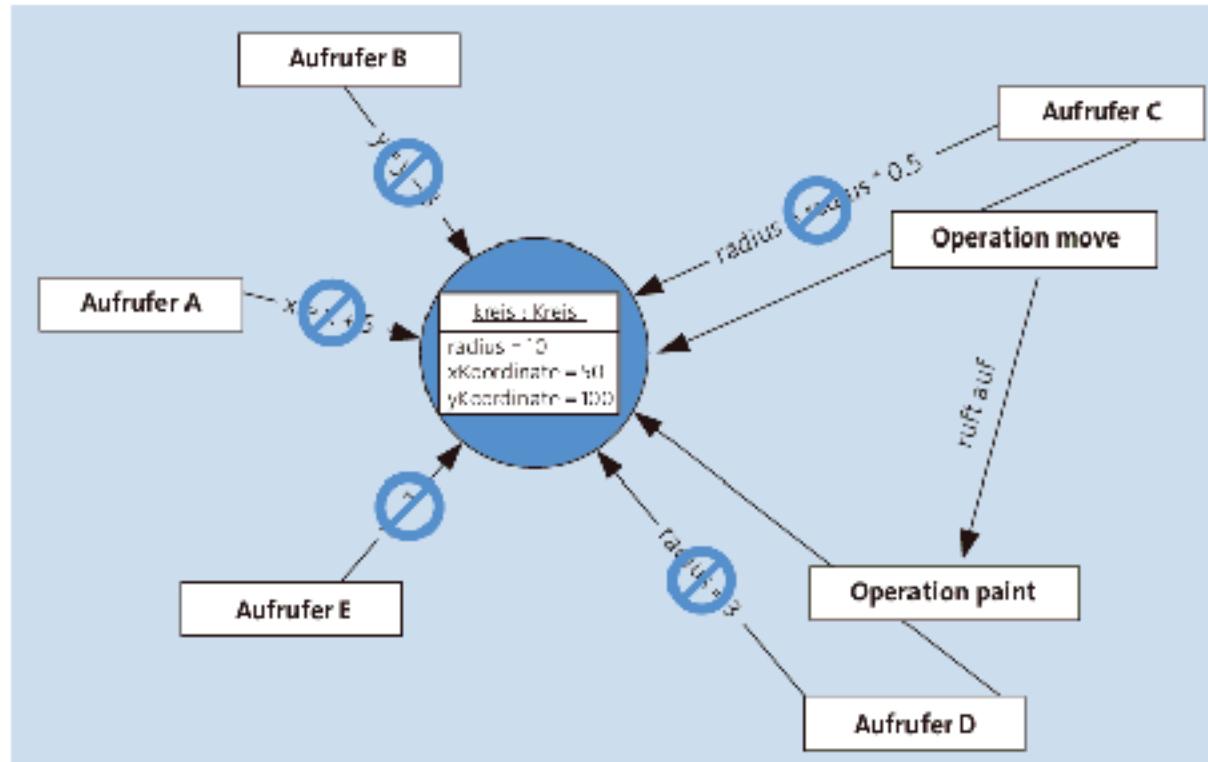
- Korrektheit
- Benutzerfreundlich
- Effizienz
- Wartbarkeit/Anpassungsfähigkeit

- 1 `lectures.getLast().assess()`
- 2 Java-Projekte planen
- 3 *Teams-Aufgabe 01 gemeinsam bearbeiten*
- 4 Programmierkonzepte/-paradigmen
- 5 OOP-Prinzipien
- 6 Discussion on „How to program simply by not using OOP“

Programmierparadigmen  
OOP-Prinzipien  
Java-Basics

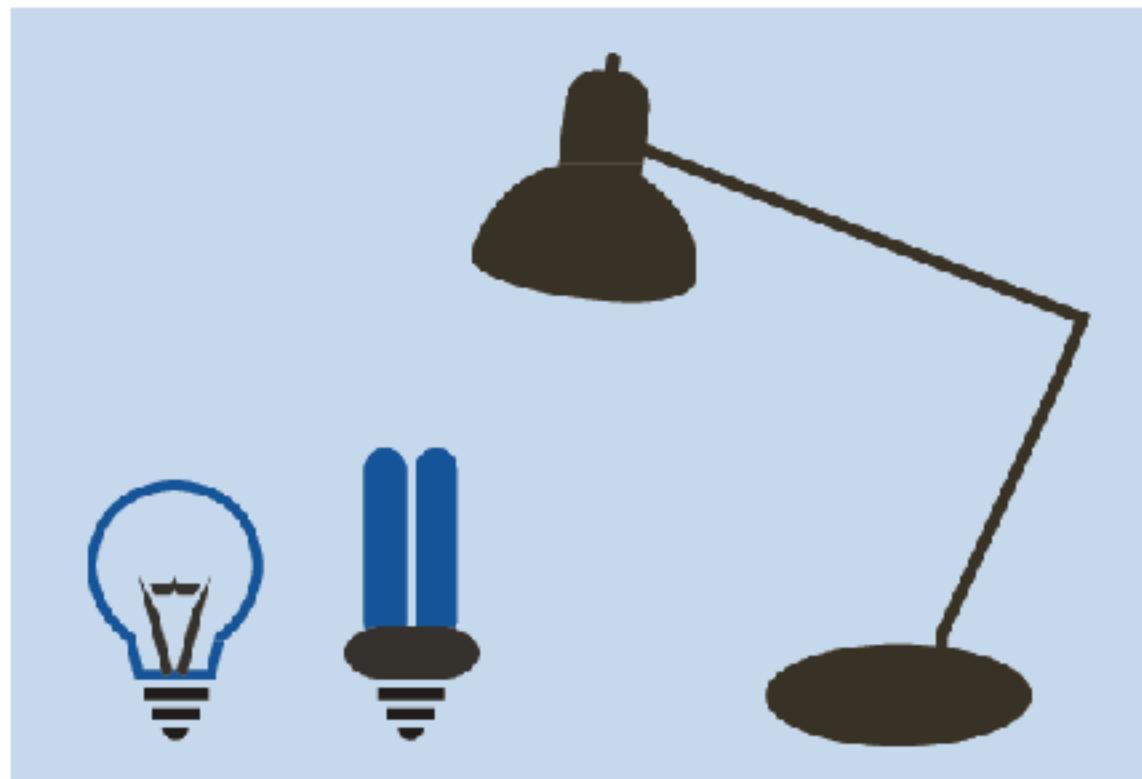


# Datenkapslung



Zugriff auf ein Objekt nur über definierte Routinen

- Objekt ist selbstverantwortlich Zugriffe zu managen.
- Daten gehören explizit zu einem Objekt.
- Möchte ein Aufrufe diese Daten verändern, muss er sich über ein klar definierte Schnittstelle an das Objekt wenden und die Änderung anfordern.
- Konsistenz von Daten wird sichergestellt.
- Das unterstützt die Korrektheit eines Programms.
- Aufwand wird reduziert, da nur die betroffenen Objekte involviert sind.



- Vielgestalt
- Unterstützt aus Austausch(barkeit) und Kompatibilität
- Steigert Wartbarkeit und Flexibilität.

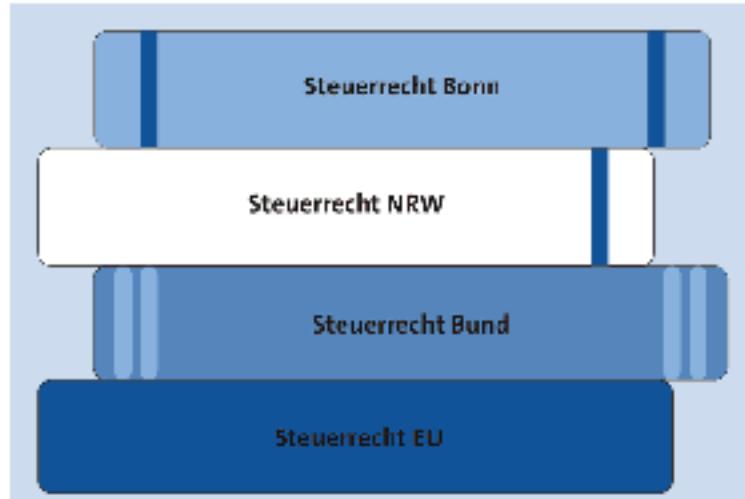


Diese Objekte, obwohl sie unterschiedliche Aufgaben haben, ...

... bieten dieselbe Schnittstelle an.



- Gemeinsamkeiten modellieren durch „Vererbung der Spezifikation“
- Eine abstrakte Spezifikation gibt dabei vor, welche Eigenschaften die Objekte haben müssen, um die Spezifikation zu erfüllen.
- Vererbung ist eng mit der Polymorphie verbunden (Geräte sind aus Sicht der Steckdose austauschbar).



Hierarchie von Regelungen im Steuerrecht

- Vererbung von Umsetzungen (Implementierungen)
- Vererbungen von umgesetzten Verfahren
- Regelungen sind hierarchisch
- Es wird immer detaillierter, spezieller, ...
- Es gelten aber immer die Rahmenbedingungen der höheren Ebenen.

# Liskovsches Substitutionsprinzip

Sie haben hier ein fundamentales Prinzip der objektorientierten Systeme vorliegen: An jeder Stelle, an der ein Exemplar einer Oberklasse erwartet wird, kann auch ein Exemplar einer ihrer Unterklassen verwendet werden. Dieses Prinzip wird *Prinzip der Ersetzbarkeit* genannt, auch als das Liskovsche Substitutionsprinzip bekannt.

## [>] Prinzip der Ersetzbarkeit

Wenn Klasse B eine Unterklasse von Klasse A ist, können in einem Programm alle Exemplare von Klasse A durch Exemplare von Klasse B ersetzt worden sein, und es gelten trotzdem weiterhin alle zugesicherten Eigenschaften von Klasse A.

## Erste Konsequenz des Prinzips der Ersetzbarkeit für Unterklassen

Eine Unterklasse kann die Vorbedingungen für eine Operation, die durch die Oberklasse definiert wird, einhalten oder abschwächen. Sie darf die Vorbedingungen aber nicht verschärfen.

## Zweite Konsequenz des Prinzips der Ersetzbarkeit für Unterklassen

Eine Unterklasse kann die Nachbedingungen für eine Operation, die durch eine Oberklasse definiert werden, einhalten oder verschärfen. Sie darf die Nachbedingungen aber nicht lockern.

## Dritte Konsequenz des Prinzips der Ersetzbarkeit für Unterklassen

Eine Unterklasse muss dafür sorgen, dass die für die Oberklasse definierten Invarianten immer gelten.

- 1. Einer einzigen Verantwortung
  - 2. Trennung der Anliegen
  - 3. Wiederholungen vermeiden
  - 4. Offen für Erweiterungen, geschlossen für Änderungen
  - 5. Trennung der Schnittstelle von der Implementierung
  - 6. Umkehr der Abhängigkeiten (Kontrollflusses')
  - 7. Umkehr des Mach es testbar
- Gruppenarbeit
  - [https://openbook.rheinwerk-verlag.de/oop/oop\\_kapitel\\_03\\_001.htm](https://openbook.rheinwerk-verlag.de/oop/oop_kapitel_03_001.htm)
  - Jede Gruppe stellt ein Prinzip vor.



# (1) Das OOP-Prinzip einer einzigen Verantwortung

- Grundsätzlich gilt: „Teile und herrsche“, denn Software besteht aus Teilen.
- Zunächst allgemeine Betrachtung, d.h. Erwähnung von Modulen anstelle von Objekten.
- Erhöhung von Mehrfach-Verwendung

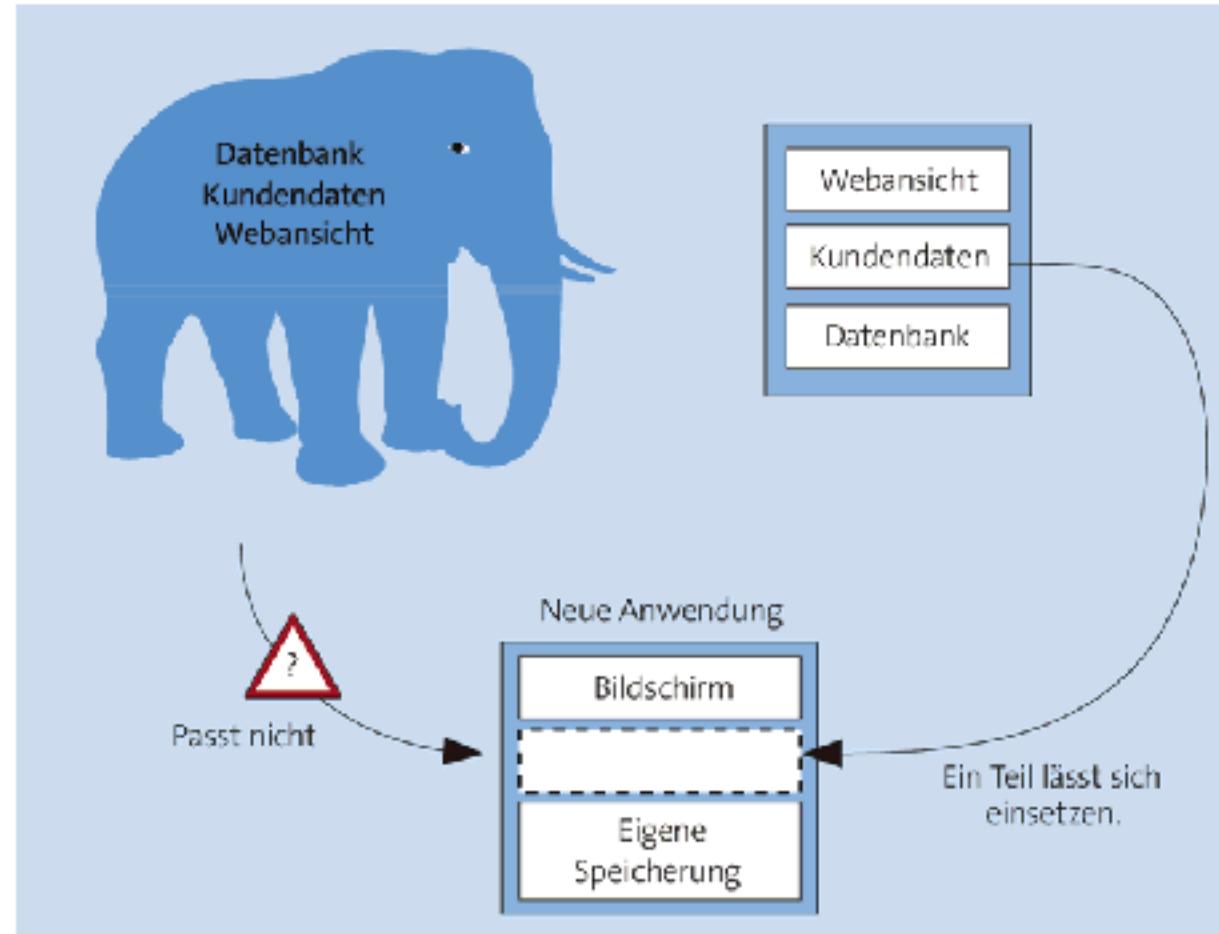
## » Module

Unter einem Modul versteht man einen überschaubaren und eigenständigen Teil einer Anwendung – eine Quelltextdatei, eine Gruppe von Quelltextdateien oder einen Abschnitt in einer Quelltextdatei. Etwas, das ein Programmierer als eine Einheit betrachtet, die als ein Ganzes bearbeitet und verwendet wird. Solch ein Modul hat nun innerhalb einer Anwendung eine ganz bestimmte Aufgabe, für die es die Verantwortung trägt.

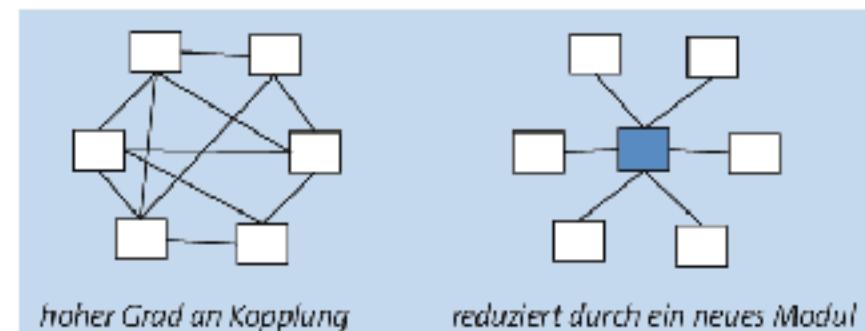
## » Verantwortung (Responsibility) eines Moduls

Ein Modul hat innerhalb eines Softwaresystems eine oder mehrere Aufgaben. Damit hat das Modul die Verantwortung, diese Aufgaben zu erfüllen. Wir sprechen deshalb von einer Verantwortung oder auch mehreren Verantwortungen, die das Modul übernimmt.

# Das OOP-Prinzip einer einzigen Verantwortung



- Erhöhung von Mehrfach-Verwendung
- Kohäsion maximieren
- Kopplung minimieren
- z.B. durch Einführung eines neuen Moduls mit Beachtung weiterer Regeln/Prinzipien

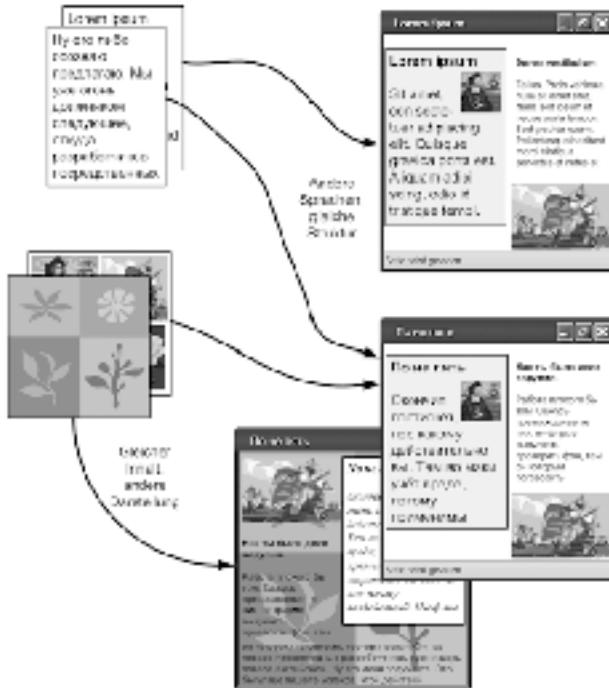


## (2) OOP-Prinzip: Trennung der Anliegen

### [2] Trennung der Anliegen (Separation of Concerns)

Ein in einer Anwendung identifizierbares Anliegen soll durch ein Modul repräsentiert werden. Ein Anliegen soll nicht über mehrere Module verstreut sein.

- keine Stärke von OOP
- eher von AOP



## (3) OOP-Prinzip: Wiederholungen vermeiden

### [x] Wiederholungen vermeiden (Don't repeat yourself)

Eine identifizierbare Funktionalität eines Softwaresystems sollte innerhalb dieses Systems nur einmal umgesetzt sein.

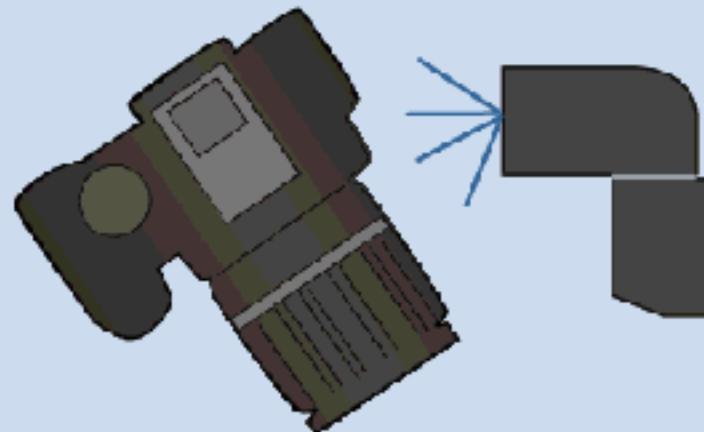
- Gewisse Redundanz an Code nicht vermeidbar
- Ursache auch „Quick & Dirty“
- Verwendung von Konstanten in verschiedenen Modulen mit eindeutiger Bezeichnung
- Implizite Abhängigkeiten in Modulen auslagern, die von allen betroffenen (abhängigen) Modulen verwendet wird → Fehler werden eher gefunden

## (4) OOP-Prinzip: Offen für Erweiterung, geschlossen für Änderung



Eine Kompaktkamera ist für einen Einsatzbereich konstruiert und optimiert. Sie ist einfach, aber nicht erweiterbar.

Eine Spiegelreflexkamera besitzt eingebaute Erweiterungspunkte, an denen man → bestimmte Komponenten anschließen kann.



Eine kompakte und eine erweiterbare Kamera

- Grundsätzliche Kompatibilität erhalten
- Möglichkeiten zur Verfeinerung anbieten
- Ausnahmen (spezielle Umarbeiten am Modul) nur in Ausnahmefällen ☺ (Erweiterungspunkte)
- Unnötige Komplexität vermeiden.

## (4) OOP-Prinzip: Offen für Erweiterung, geschlossen für Änderung

### [»] Offen für Erweiterung, geschlossen für Änderung (Open-Closed-Principle)

#### **Ein Modul soll für Erweiterungen offen sein.**

Das bedeutet, dass sich durch die Verwendung des Moduls zusammen mit Erweiterungsmodulen die ursprüngliche Funktionalität des Moduls anpassen lässt. Dabei enthalten die Erweiterungsmodule nur die Abweichungen der neu gewünschten von der ursprünglichen Funktionalität.

#### **Ein Modul soll für Änderungen geschlossen sein.**

Das bedeutet, dass keine Änderungen des Moduls nötig sind, um es erweitern zu können. Das Modul soll also definierte Erweiterungspunkte bieten, an die sich die Erweiterungsmodule anknüpfen lassen.

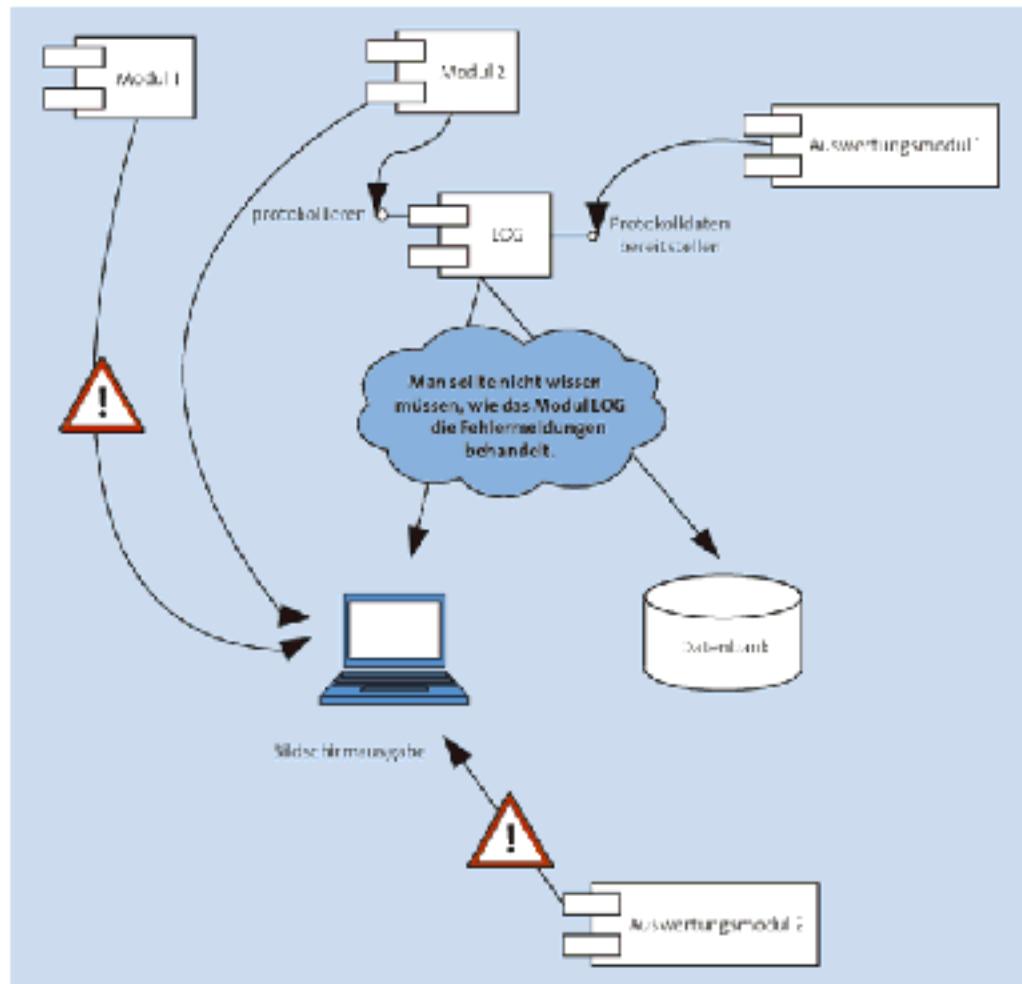
## (5) OOP-Prinzip: Trennung der Schnittstelle von der Implementierung

### [x] Trennung der Schnittstelle von der Implementierung (Program to Interfaces)

Jede Abhängigkeit zwischen zwei Modulen sollte explizit formuliert und dokumentiert sein. Ein Modul sollte immer von einer klar definierten Schnittstelle des anderen Moduls abhängig sein und nicht von der Art der Implementierung der Schnittstelle. Die Schnittstelle eines Moduls sollte getrennt von der Implementierung betrachtet werden können.

- ... , um Module austauschen zu können.
- „Programmiere gegen Schnittstellen, nicht gegen Implementierungen“
- Probleme sind in der Regel darauf zurückzuführen, dass sich Programme auf bestimmte Implementierungen verlassen
- Verwendung von „Interfaces“ alleine genügt nicht: Keine Annahmen über konkrete Implementierung machen!

## (5) OOP-Prinzip: Trennung der Schnittstelle von der Implementierung



## (6) OOP-Prinzip: Umkehr der Abhängigkeiten

- Da im Allg. „Teile und herrsche“ angewendet wird, kommt eine Top-Down-Entwurf zum Einsatz, der u.U. nicht ideal sein kann.
- Um Abhängigkeiten zwischen Modulen gering zu halten, sollten Abstraktionen verwendet werden.

### [»] Abstraktion

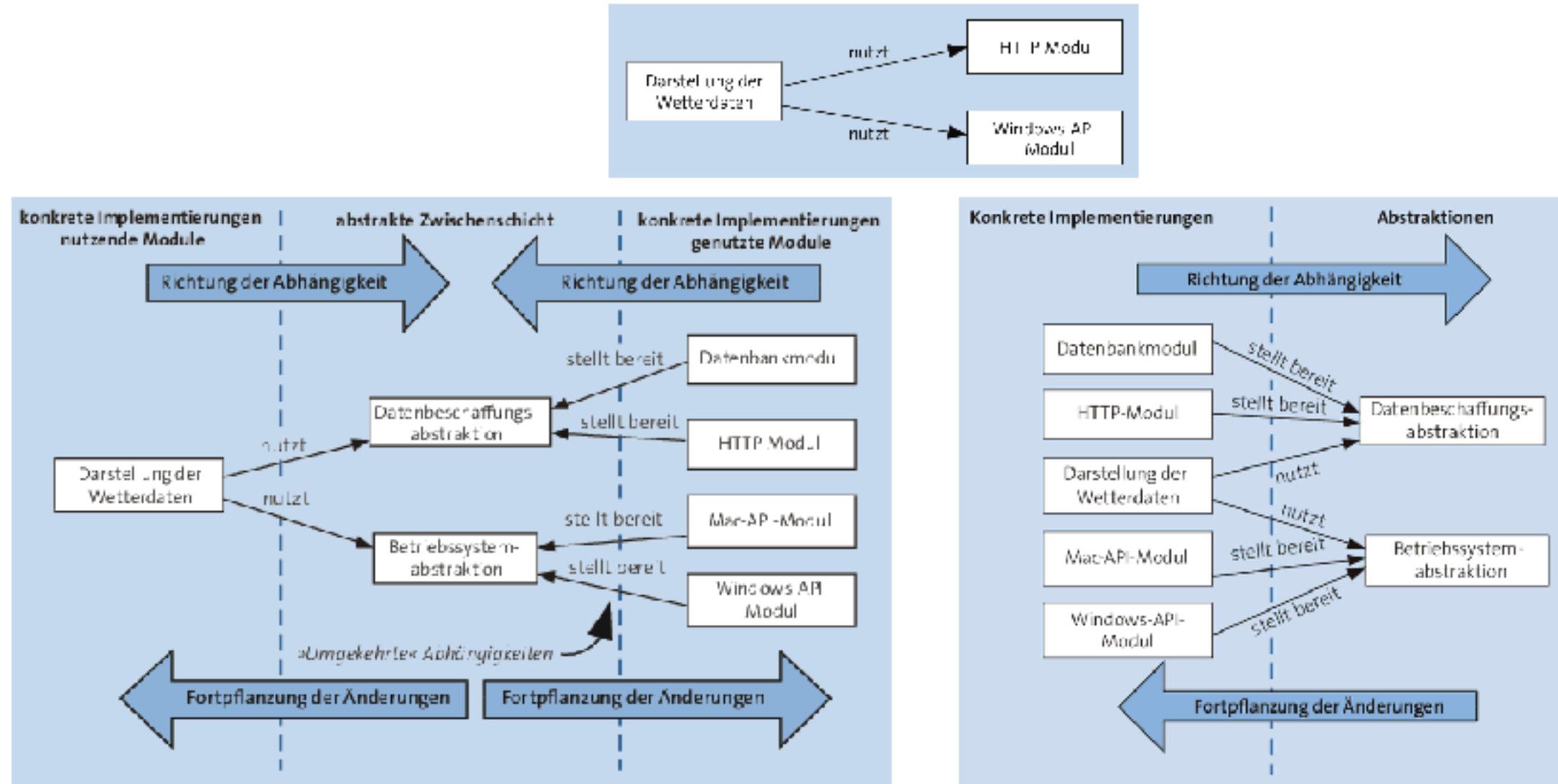
Eine Abstraktion beschreibt das in einem gewählten Kontext Wesentliche eines Gegenstands oder eines Begriffs. Durch eine Abstraktion werden die Details ausgeblendet, die für eine bestimmte Betrachtungsweise nicht relevant sind. Abstraktionen ermöglichen es, unterschiedliche Elemente zusammenzufassen, die unter einem bestimmten Gesichtspunkt gleich sind.

So lassen sich zum Beispiel die gemeinsamen Eigenschaften von verschiedenen Betriebssystemen als Abstraktion betrachten: Wir lassen die Details der spezifischen Umsetzungen und spezielle Fähigkeiten der einzelnen Systeme weg und konzentrieren uns auf die gemeinsamen Fähigkeiten der Systeme. Eine solche Abstraktion beschreibt die Gemeinsamkeiten von konkreten Betriebssystemen wie Windows, Linux oder macOS.

### [»] Umkehr der Abhängigkeiten (Dependency Inversion Principle)

Unser Entwurf soll sich auf Abstraktionen stützen – nicht auf Spezialisierungen.

## (6) OOP-Prinzip: Umkehr der Abhängigkeiten

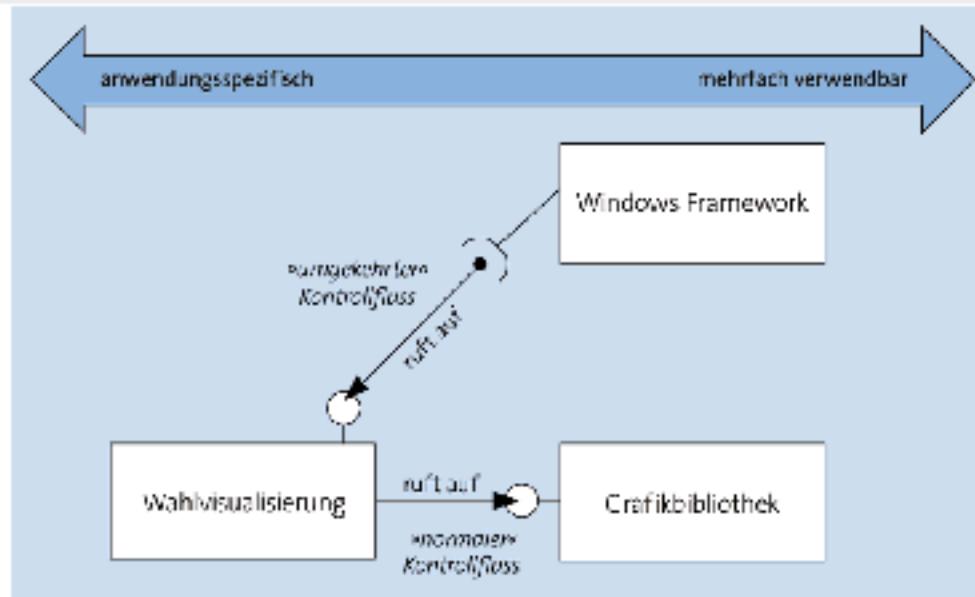


# (6) OOP-Prinzip: Umkehr der Abhängigkeiten/ des Kontrollflusses

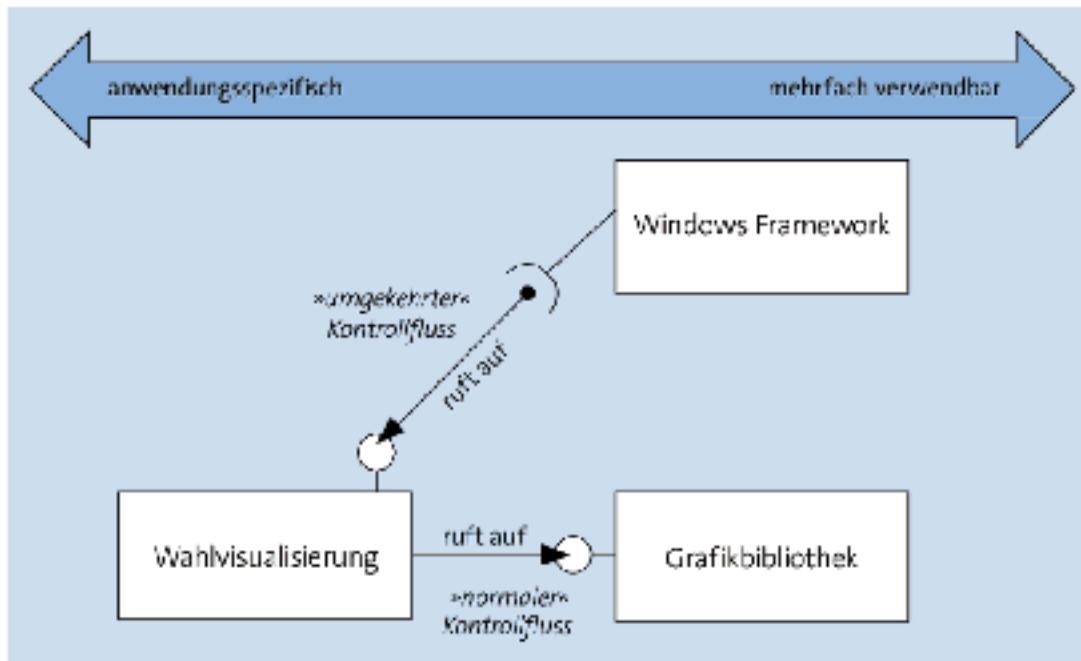
## [1] Umkehrung des Kontrollflusses (Inversion of Control)

Als die Umkehrung des Kontrollflusses wird ein Vorgehen bezeichnet, bei dem ein spezifisches Modul von einem mehrfach verwendbaren Modul aufgerufen wird. Die Umkehrung des Kontrollflusses wird auch Hollywood-Prinzip genannt: »Don't call us, we'll call you!«

Die Umkehrung des Kontrollflusses wird eingesetzt, wenn die Behandlung von Ereignissen in einem mehrfach verwendbaren Modul bereitgestellt werden soll. Das mehrfach verwendbare Modul übernimmt die Aufgabe, die anwendungsspezifischen Module aufzurufen, wenn bestimmte Ereignisse stattfinden. Die spezifischen Module rufen also die mehrfach verwendbaren Module nicht auf, sie werden stattdessen von Ihnen aufgerufen.



## (6) OOP-Prinzip: Umkehr der Abhängigkeiten/ des Kontrollflusses



- „Inversion of control“
- Hollywood“-Prinzip: „Don't call us, we'll call you“
- Dependency Injection

## (7) OOP-Prinzip: Mach es testbar

### [x] Unit-Tests

Ein Unit-Test ist ein Stück eines Testprogramms, das die Umsetzung einer Anforderung an eine Softwarekomponente überprüft. Die Unit-Tests können automatisiert beim Bauen von Software laufen und so helfen, Fehler schnell zu erkennen.

- Single Responsibility (SRP)**
- Open Closed Principle (OCP)**
- Liskov Substitution Principle (LSP)**
- Interface Segregation Principle (ISP)**
- Dependency Inversion Principle (DIP)**

- 1 `lectures.getLast().assess()`
- 2 Java-Projekte planen
- 3 *Teams-Aufgabe 01 gemeinsam bearbeiten*
- 4 Programmierparadigmen
- 5 OOP-Prinzipien
- 6 Discussion on „How to program simply by not using OOP“
- 7 Praktischer Teil

Programmierparadigmen  
OOP-Prinzipien  
Java-Basics

# Discussion on „How to program simply by not using OOP“

<https://levelup.gitconnected.com/how-to-program-simply-by-not-using-object-oriented-programming-a68de601fc26>

- “OOP is great, but it silently makes your simple programs complex”
- Give arguments of the author
- Are we on the right track? If ... why?

## 1. Technologies Will Change, but Theories Won't

Unfortunately, most of the time, modern programmers learn technologies, not theories. For example, web developers typically enter software engineering by mastering a specific frontend framework. Technologies get changed frequently, and existing technologies get outdated faster than you think. If your software engineering knowledge solely depends on a particular framework, language, or library, your knowledge also gets outdated.

Think for a moment — what strategy do developers follow if they need to build a library for multiple frontend frameworks? They usually write a library-agnostic core package and create light-weight wrappers for each frontend library (i.e., see the modular design of [TanStack Query](#)). Similarly, we can place our technical skillset on a foundation built with theoretical computer science aspects. Then, we have negligible damage whenever we move to a new technology stack.

<https://levelup.gitconnected.com/8-lessons-i've-learned-from-studying-software-engineering-for-15-years-a68de601fc26>

## To-Dos

- Themen für Java-Projekte

- 1 `lectures.getLast().assess()`
- 2 Java-Projekte planen
- 3 *Teams-Aufgabe 01 gemeinsam bearbeiten*
- 4 Programmierparadigmen
- 5 OOP-Prinzipien
- 6 Discussion on „How to program simply by not using OOP“
- 7 Praktischer Teil

Programmierparadigmen  
OOP-Prinzipien  
Java-Basics

abstract	do	implements	private	this
assert	double	import	protected	throw
boolean	else	instanceof	public	throws
break	enum	int	record●	transient
byte	extends	interface	return	true
case	false	long	sealed●	try
catch	final	native	short	var●
char	finally	new	static	void
class	float	non-sealed●	strictfp	volatile
const	for	null	super	while
continue	if	package	switch	yield●
default	goto	permits	synchronized	

**Einstieg in Java**

**Algorithmen, Visualisierung, Kontrollstrukturen**

## Struktogramm (Nassi-Shneidermann)

### Wertzuweisung und Sequenz

- Wertzuweisung

```
x := x + 5
```

- Folge (Sequenz)

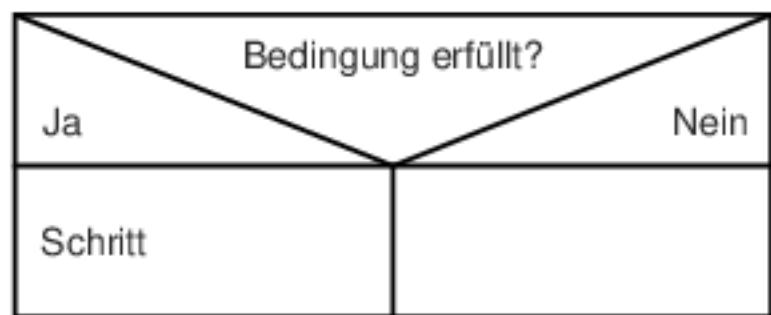
```
z := x
```

```
x := y
```

```
y := z
```

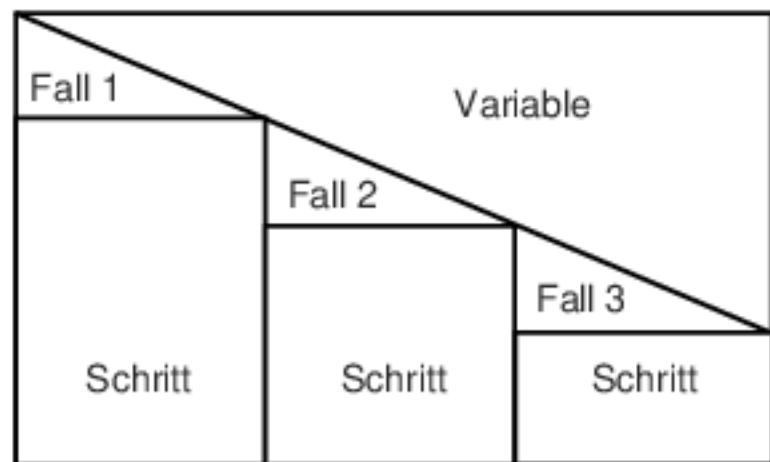
## Struktogramm (Nassi-Shneidermann)

### „Auswahl“



## Struktogramm (Nassi-Shneidermann)

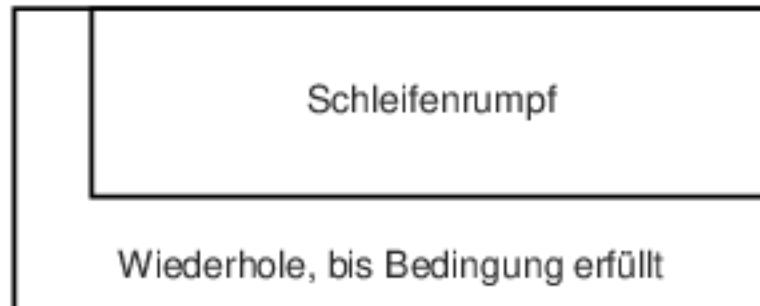
### Mehrfachverzweigung



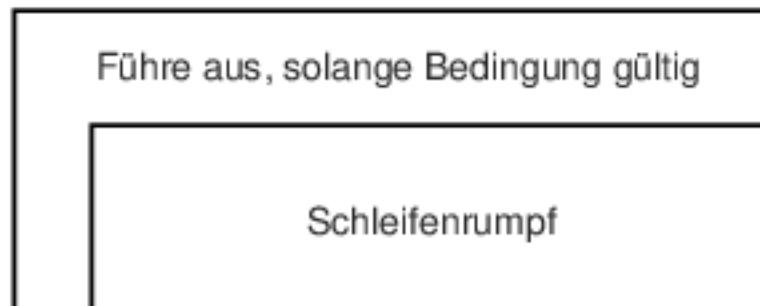
## Struktogramm (Nassi-Shneidermann)

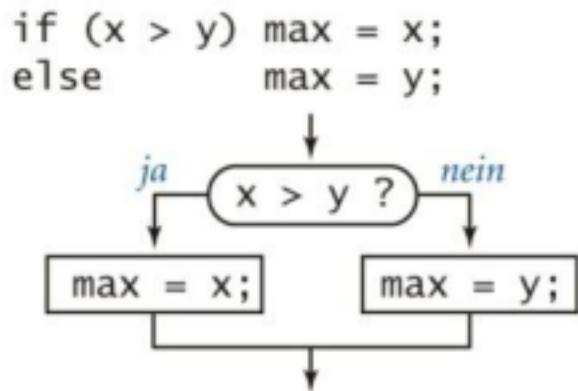
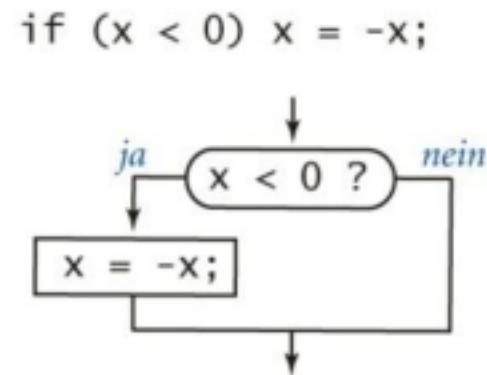
### Die „Wiederholung“

- Wiederholung (Iteration) mit Test der Laufbedingung am Ende



- mit Test der Laufbedingung am Anfang





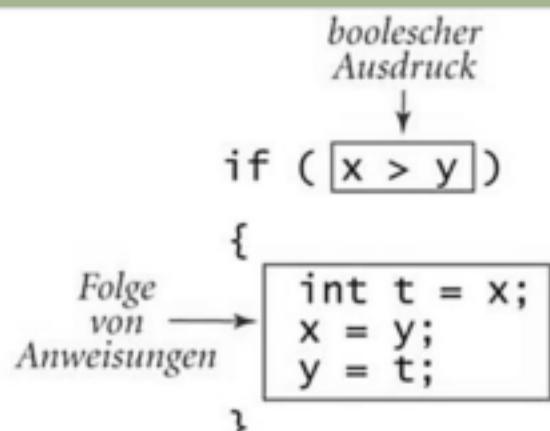
Flussdiagramm-Beispiele (if-Anweisungen)

## ALLGEMEIN

- Treffen von Entscheidungen
- if
- switch
- ternärer Operator

## IF-VERZWEIGUNG

- Klammern und Einrücken entscheidend für sauberen Code
- else und else if



## SWITCH-VERZWEIGUNG

- Alternative zu verschachtelten if-Verzweigungen

```
switch (ausdruck) {  
    case constante1:  
        anweisungen;  
        break;  
    case constante2:  
        anweisungen;  
        break;  
    default:  
        anweisungen;  
}
```

## The new „switch“

```
int value = switch (greeding) {  
    case "hi" -> {  
        System.out.println("I am not just yielding!");  
        yield 1;  
    }  
    case "hello" -> {  
        System.out.println("Me too.");  
        yield 2;  
    }  
    default -> {  
        System.out.println("OK");  
        yield -1;  
    }  
}
```

```
String month = "März";  
int days =  
    switch(month) {  
        case "Januar", "März", "Mai", "Juli", "August",  
            "Oktober", "Dezember" -> 31;  
        case "April", "Juni", "September", "November" -> 30;  
        case "Februar" -> 28; // oder 29, wenn Schaltjahr  
        default -> 0;  
    };  
  
if(days != 0)  
    System.out.println("Der " + month + " hat " + days + " Tage.");
```

# **Einstieg in Java**

## **AUFGABE 7**

## Aufgabe 7

Erstellen Sie ein Struktogramm für folgenden Algorithmus: Die größte der drei eingegebenen Zahlen a, b, c soll in der Variablen max gespeichert und ausgegeben werden!

Schreiben Sie ein entsprechendes Java-Programm.

# **Einstieg in Java**

## **AUFGABE 8**

Erstellen Sie ein Struktogramm nach Nassi-Shneidermann für die Umrechnung einer ganzzahligen Dezimalzahl in eine Binärzahl.

# **Einstieg in Java**

## **AUFGABE 9**

### Aufgabe 9

Ein Jahr ist ein Schaltjahr, wenn es ganzzahlig durch 4 teilbar ist. Eine Ausnahme besteht, wenn das Jahr ganzzahlig durch 100, aber nicht durch 400 teilbar ist, dann ist das Jahr kein Schaltjahr.

- Stellen Sie den Algorithmus grafisch (z. B. in einem Struktogramm nach Nassi Shneidermann) dar.
- Setzen Sie den Algorithmus als Programm um.

### Allgemein

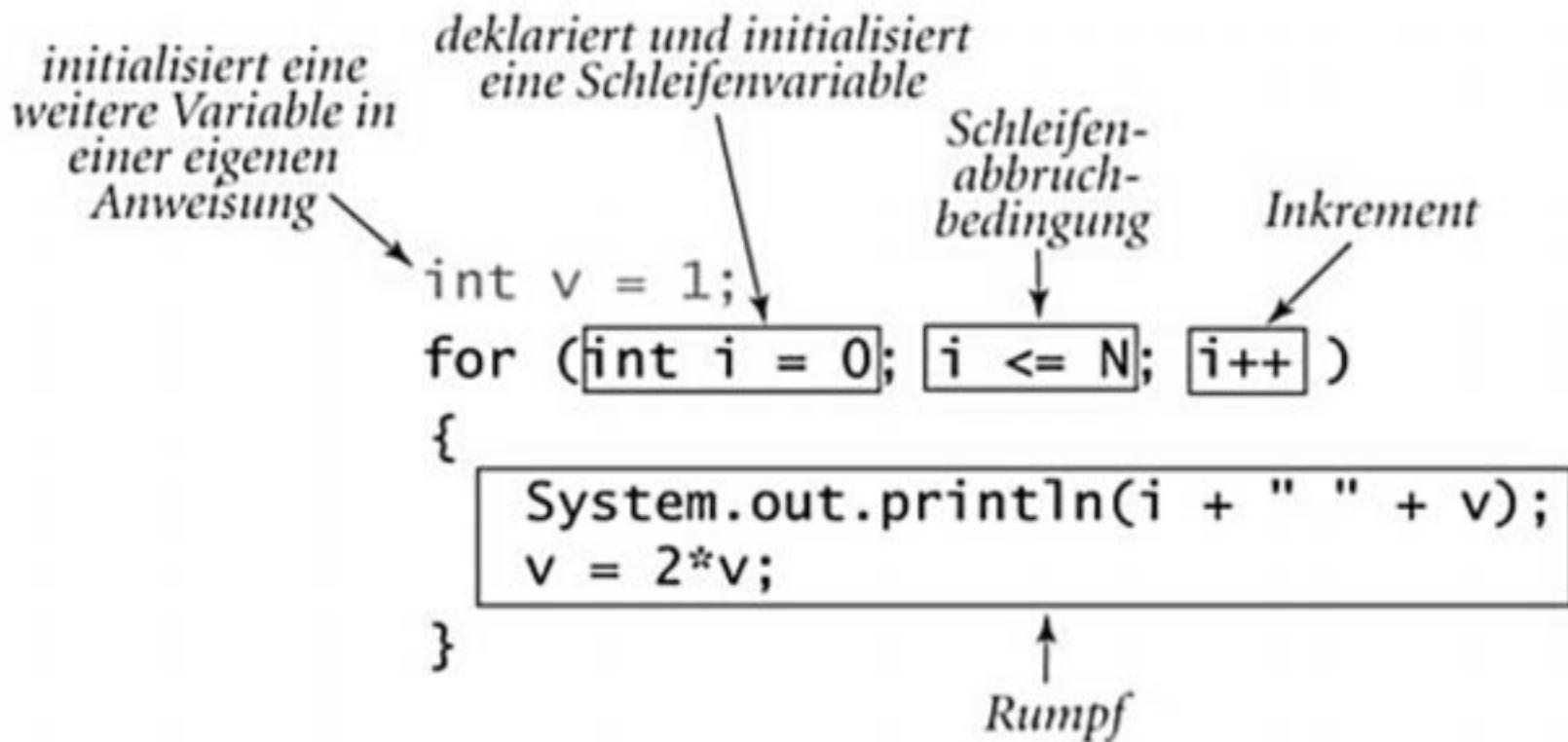
- Wiederholtes Ausführen von Programmcode
- for
- while
- do while



# For-loop

## Simple

```
for (initialization; Boolean-expression; step)
    statement;
```



Anatomie einer for-Schleife (die Zweierpotenzen ausgibt)

```
int i;  
for(i=0; i<10; i++) {  
    System.out.println(i);  
}
```

Beachten Sie, dass die beiden folgenden Codes *nicht* gleichwertig sind:

```
// explizite Deklaration // Deklaration von 'i' in for  
int i;  
for(i=0; i<10; i++) {           for(int i=0; i<10; i++) {  
    doSomething(i);             doSomething(i);  
}  
}
```

## Labeled (break, continue)

- break: Verlassen des aktuellen Loops
- continue: Abbruch der aktuellen Iteration des Loops
- Labels für Wechsel der Iterationsebenen

```
aa: for (int i = 1; i <= 3; i++) {  
    if (i == 1)  
        continue;  
    bb: for (int j = 1; j <= 3; j++) {  
        if (i == 2 && j == 2) {  
            break aa;  
        }  
        System.out.println(i + " " + j);  
    }  
}
```

- Vor- und Nachteile ...

```
for(Type item : items)
    statement;
```

```
int[] intArr = { 0,1,2,3,4 };
for (int num : intArr) {
    System.out.println("Enhanced for-each loop: i = " + num);
}
```

```
for (String item : list) {
    System.out.println(item);
}
```

*List<String>*

```
for (Entry<String, Integer> entry : map.entrySet()) {
    System.out.println(
        "Key: " + entry.getKey() +
        " - " +
        "Value: " + entry.getValue());
}
```

*Map<String, Integer>*

# For-loop

## Iterable.forEach()

- Vor- und Nachteile ...

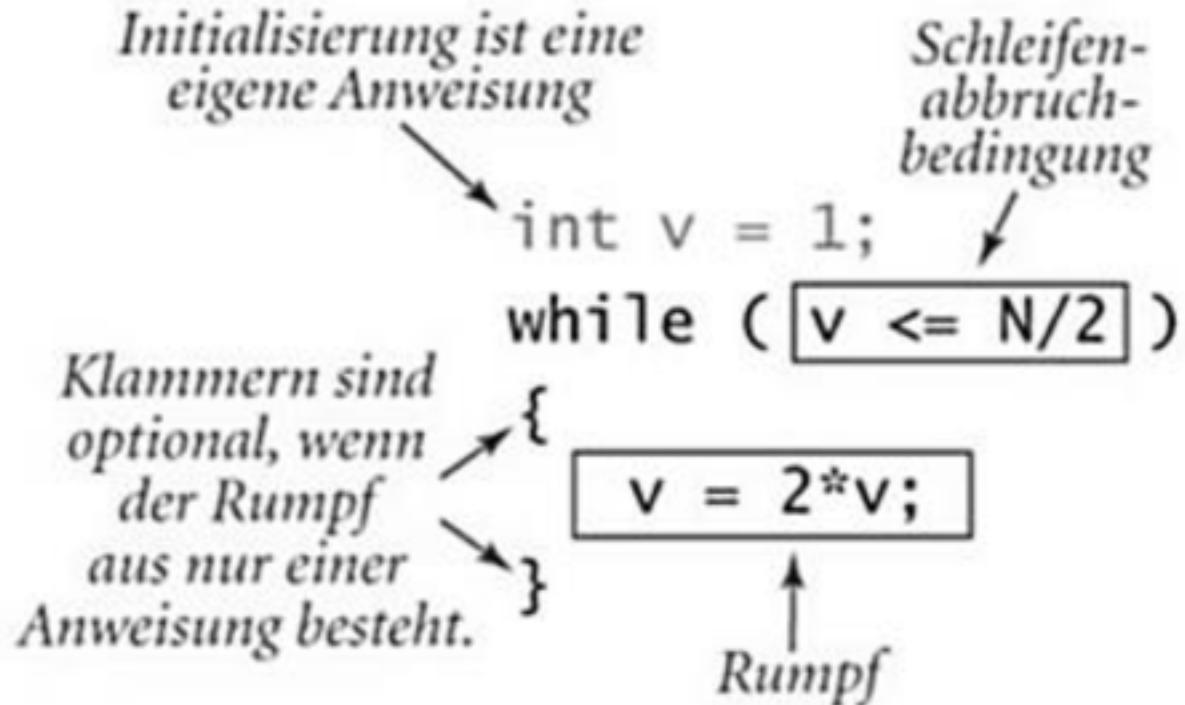
```
List<String> names = new ArrayList<>();  
names.add("Larry");  
names.add("Steve");  
names.add("James");  
names.add("Conan");  
names.add("Ellen");  
  
names.forEach(name -> System.out.println(name));
```

```
// Make a collection
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");

// Get the iterator
Iterator<String> it = cars.iterator();

// Print the first item
System.out.println(it.next());
```

- Vor- und Nachteile ...
- Iterator vs forEach()
- Modify a collection → iterator



## Anatomie einer while-Anweisung

# **Einstieg in Java**

## **AUFGABE 10**

### Aufgabe 10

Ein Programm soll nach Eingabe einer Ganzzahl (im Beispiel 5) folgende Ausgabe erzeugen:

$$1 \times 1 = 1$$

$$1 \times 2 = 2$$

$$1 \times 3 = 3$$

$$1 \times 4 = 4$$

$$1 \times 5 = 5$$

$$2 \times 1 = 2$$

$$2 \times 2 = 4$$

$$2 \times 3 = 6$$

$$2 \times 4 = 8$$

$$2 \times 5 = 10$$

...

$$5 \times 1 = 5$$

$$5 \times 2 = 10$$

$$5 \times 3 = 15$$

$$5 \times 4 = 20$$

$$5 \times 5 = 25$$

- Erstellen Sie ein Struktogramm für das Programm.
- Erstellen Sie auf Basis des Struktogramms das Programm.

# **Einstieg in Java**

## **AUFGABE 11**

Verwenden Sie `for`, `while` und `do-while`, um auf drei unterschiedlichen Arten eine Schleife zu bilden, die in Fünferschritten von 100 nach 50 runterzählt.

# Einstieg in Java

## AUFGABE 12

Ein Programm soll eine positive ganze Zahl einlesen und die Ziffern als Text ausgeben (Beispiel: 3571 wird ausgegeben als „drei fünf sieben eins“).

- Verwenden Sie dazu eine `while`-Schleife und eine `switch`-Anweisung.
- Erstellen Sie vorab ein Struktogramm.

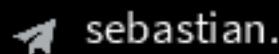
Sedgewick, R. & Wayne, K. (2011). Einführung in die Programmierung mit Java. Hallbergmoos: Pearson.

# Danke

Sebastian Bichler



via Teams



[sebastian.bicher@iu.org](mailto:sebastian.bicher@iu.org)

Sommersemester 2023

DSBOOPI01

# Objektorientierte Programmierung I

## mit Java

#03, 13.04.2023,

Prof. (FH) Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

Java-Basics  
Objekte – Klassen  
Attribute, Methoden,  
Beziehungen (UML)

- 1 Eigene Java-Projekte / „Mobile First“ && „Web First“**
- 2 Java-Basics (Einheiten: 1, 2) bearbeiten**
- 3 Einführung in die objektorientierte Systementwicklung**
- 4 Einführung in die objektorientierte Modellierung**

Java-Basics  
Objekte – Klassen  
Attribute, Methoden,  
Beziehungen (UML)

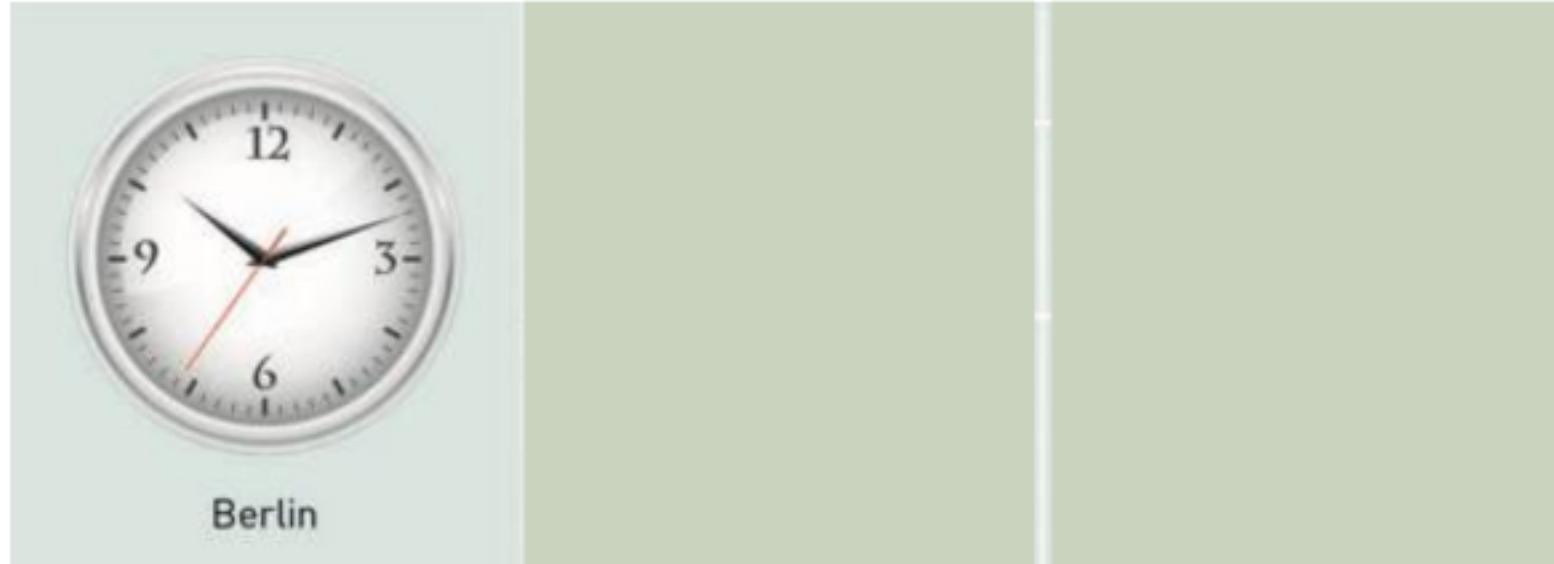
# Eigene Java-Projekte / „Mobile First“ & „Web First“

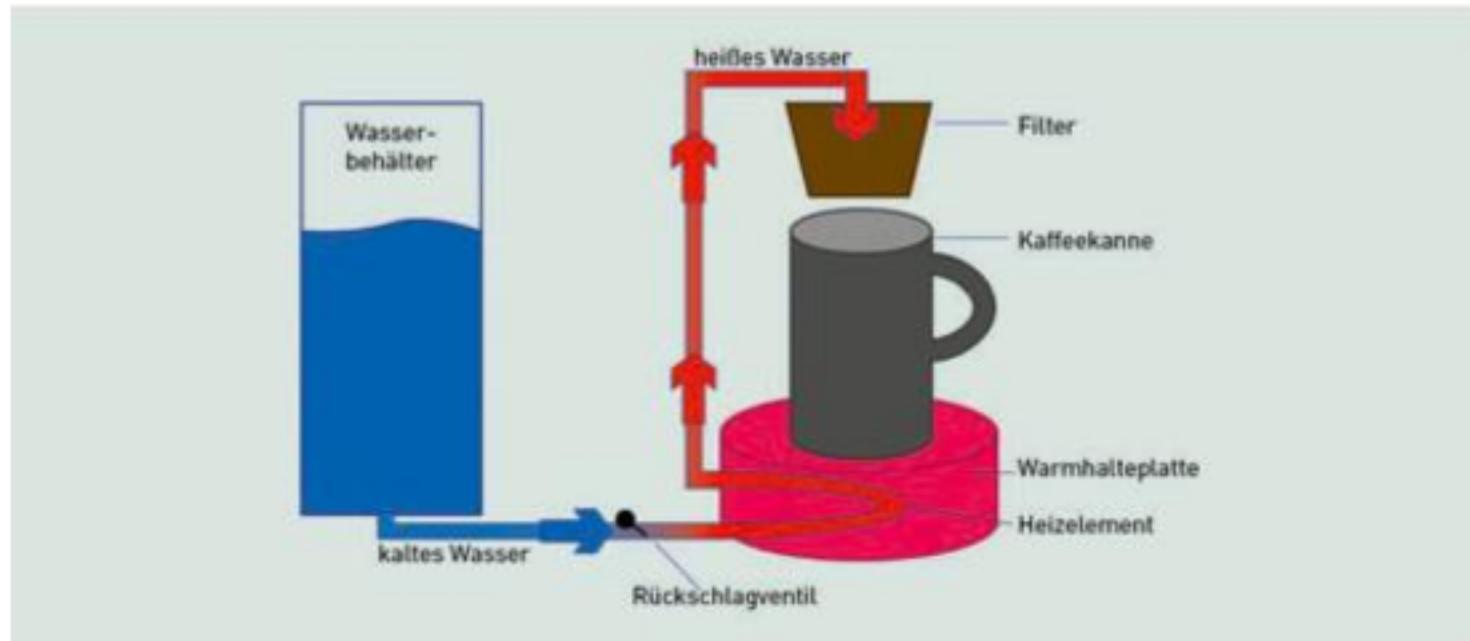
## Vorschlag: Zentrale Ausleihe am IAT mitgestalten

- Inventar
- Ausleihe
- Auftragsverwaltung
  - Objekte/Zubehör → Sets
  - Akkuverwaltung, Updates → Timings, ...
- (Mitarbeiterverwaltung)
- Kalender
- QR- und Barcode
- ...

- 1 Eigene Java-Projekte / „Mobile First“ && „Web First“**
- 2 Java-Basics (Einheiten: 1, 2) bearbeiten**
- 3 Einführung in die objektorientierte Systementwicklung**
- 4 Einführung in die objektorientierte Modellierung**
- 5 Praxis: Playing with JavaFX**

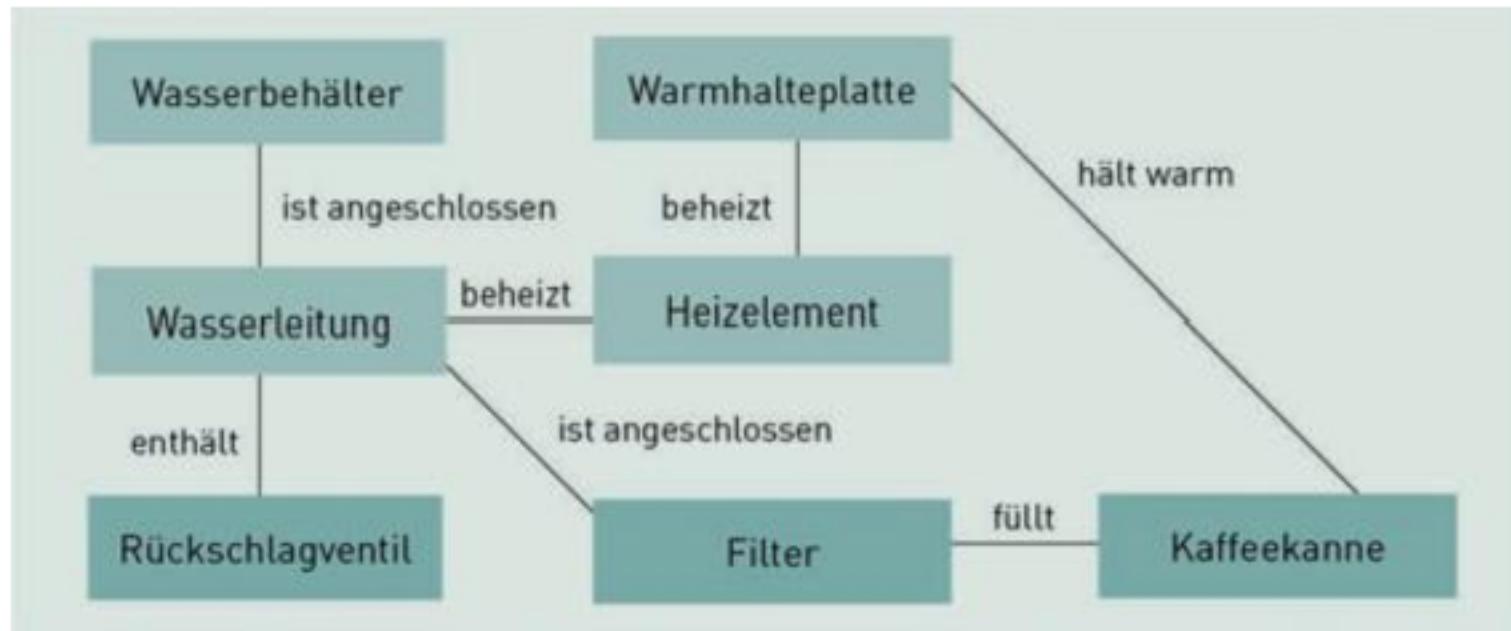
**Java-Basics**  
**Objekte – Klassen**  
**Attribute, Methoden,**  
**Beziehungen (UML)**



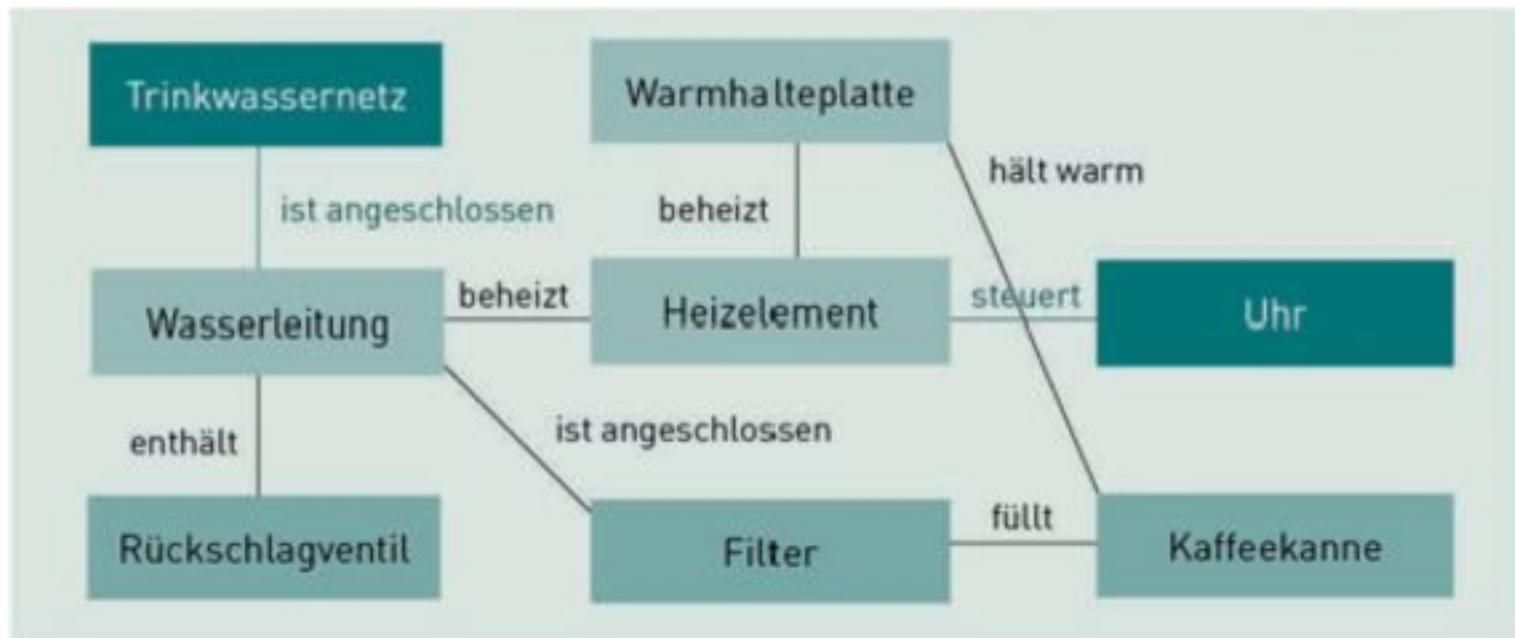


Beispielszenario einer einfachen Kaffeemaschine

- OOP (theoretisch) nicht notwendig
- Erfahrungen der letzten 20 Jahre Software Entwicklung zeigen aber, dass es weniger Probleme mit der Skalierung, Stabilität und Erweiterung von IT-Systemen bei Einsatz von OOP gibt.
- OOP kann als Prinzip und Mittel verstanden werden, um die Komplexität der Software in den Griff zu bekommen.



Zusammenspiel von Objekten einer Kaffeemaschine



Erweiterte Kaffeemaschine

Objekte	Zuständigkeiten/Funktion
Wasserbehälter, später ersetzt durch Trinkwassernetz	Wasserversorgung
Wasserleitung	Transportieren von Wasser
Rückschlagventil	Strömungsrichtung des Wassers regeln
Heizelement	Elektrische Energie in Wärme umwandeln
Warmhalteplatte	Kaffeekanne wärmen
Filter	Kaffeepulver zurückhalten und Wasser durchströmen lassen
Uhr	Bei Erreichen der eingestellten Uhrzeit Signal auslösen

- Kapselung

## Objekte einer Kaffeemaschine

# Einführung in die objektorientierte Systementwicklung

Durch die klare **Kapselung** von Zuständigkeiten in ganz bestimmte Objekte und die klare Definition von Schnittstellen, können Objekte später ausgetauscht (in unserem Beispiel wurde der Wasserbehälter durch das Trinkwassernetz ausgetauscht) oder erweitert (in unserem Beispiel die Erweiterung der Kaffeemaschine um eine Uhr zur Zeitsteuerung) werden, ohne die Stabilität und Funktionalität des gesamten Systems zu beeinflussen. Das erleichtert wiederum die Maßnahmen zur Qualitätssicherung, da nicht das ganze System komplett neu getestet werden muss, sondern nur die betroffenen Teile des Systems.

**Kapselung**  
Dabei werden  
Objekte gezielt für  
abgeschlossene  
Funktionen und Auf-  
gaben erstellt. Über  
Schnittstellen kön-  
nen andere Objekte  
diese konkreten  
Funktionen und Auf-  
gaben aufrufen.

# Einführung in die objektorientierte Systementwicklung

## Aufgabe: Kaffeemaschine

- **Entwickelt ein „besseres“/eigenes OO-Modell von einer Kaffeemaschine!**
- In dieser und der nächsten Einheit liegt der Fokus auf der **OOA**.
- OOD und OOI folgen im weiteren Verlauf.



# Einführung in die objektorientierte Systementwicklung

Die Objektorientierung ist eine Sichtweise auf komplexe Systeme, bei der das System durch das Zusammenspiel kooperierender Objekte beschrieben wird.

Bei der objektorientierten Systementwicklung werden Dinge der „realen“ Welt durch Objekte der „digitalen“ Welt nachgebildet. Dabei werden nur die für den Zweck des Systems relevanten Werte und Funktionen berücksichtigt. Die digital nachgebildeten Objekte speichern Werte in ihren Attributen und können in ihren Methoden Werte berechnen. Objektorientierte Programme bestehen aus kooperierenden „digitalen“ Objekten. Die Kooperation der Objekte erfolgt durch das gegenseitige Aufrufen von Methoden.

Der objektorientierte Entwicklungsprozess besteht aus den Phasen objektorientierte Analyse, objektorientiertes Design und objektorientierte Implementierung. Ziel des Entwicklungsprozesses ist ein System, das den Anforderungen des Auftraggebers in Funktionalität, Qualität und Aufwand entspricht.

Durch die Aufteilung von Systemen in einzelne Objekte mit ganz bestimmten Zuständigkeiten hilft die Objektorientierung bei der Entwicklung, Weiterentwicklung und der Qualitätssicherung von komplexen industriellen Softwaresystemen.

# Einführung in die objektorientierte Systementwicklung

## [»] Objekte in der objektorientierten Programmierung

Ein Objekt ist ein Bestandteil eines Programms, der Zustände enthalten kann. Diese Zustände werden vom Objekt vor einem Zugriff von außen versteckt und damit geschützt.

Außerdem stellt ein Objekt anderen Objekten Operationen zur Verfügung. Von außen kann dabei auf das Objekt ausschließlich zugegriffen werden, indem eine Operation auf dem Objekt aufgerufen wird.

Ein Objekt legt dabei selbst fest, wie es auf den Aufruf einer Operation reagiert. Die Reaktion kann in Änderungen des eigenen Zustands oder dem Aufruf von Operationen auf weiteren Objekten bestehen.

- Ein Objekt ist eine für sich abgeschlossene, handlungsfähige Einheit.

## [»] Daten eines Objekts

Ein Objekt kann Werte zugeordnet haben, die nur von ihm selbst verändert werden können. Diese Werte sind die Daten, die das Objekt besitzt. Von außen sind die Daten des Objekts nicht sichtbar und nicht zugreifbar.

## [»] Eigenschaften (Attribute) von Objekten

Objekte haben Eigenschaften, die von außen erfragt werden können. Dabei kann von außen nicht unterschieden werden, ob eine Eigenschaft direkt auf Daten des Objekts basiert oder ob die Eigenschaft auf der Grundlage von Daten berechnet wird. Eigenschaften, die nicht direkt auf Daten basieren, werden abgeleitete Eigenschaften genannt.

# Einführung in die objektorientierte Systementwicklung

## Darstellung eines Objekts in UML

In der Sprache UML bildet man ein Objekt einfach als ein Rechteck ab. In [Abbildung 4.3](#) ist ein Beispiel dafür zu sehen. Oben in dem Rechteck steht der unterstrichene Name des Objekts, und darunter in getrennten Abteilungen befinden sich die Daten und die Operationen bzw. die Methoden des Objekts. Laut der UML-Spezifikation bezeichnet man abgeleitete Eigenschaften eines Objekts mit einem Schrägstrich.

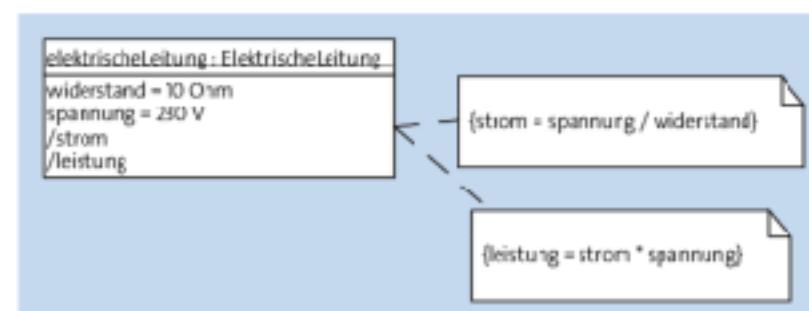
- UML: siehe Kursmaterialien/UML
- Wird detaillierter später behandelt.

Die UML-Diagramme können verwendet werden,

- um die nach außen sichtbare Schnittstelle darzustellen und
- um sie für die Darstellung der gekapselten Struktur der Implementierung zu nutzen.



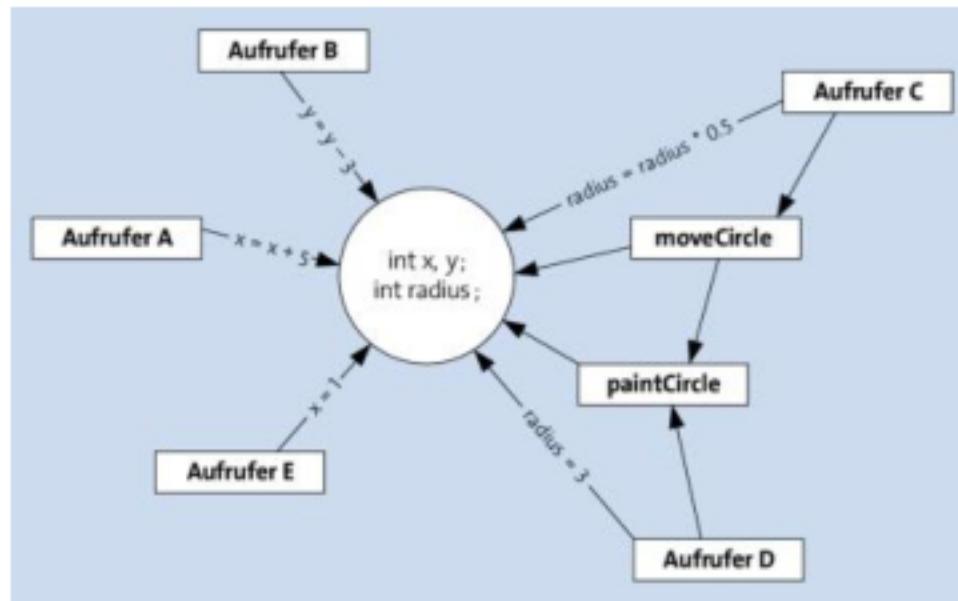
Darstellung der äußeren Struktur eines Objekts



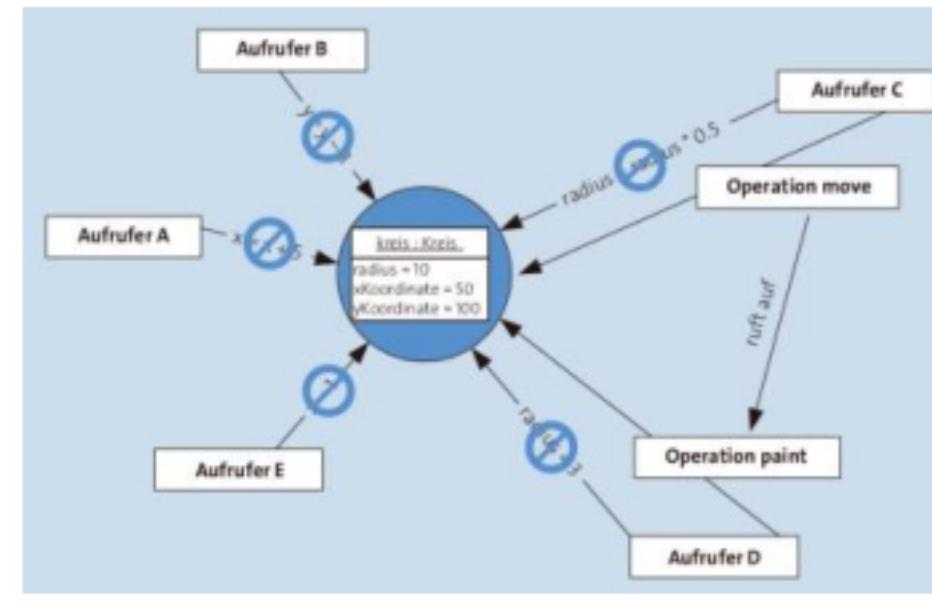
Implementierungssicht auf ein Objekt

# Einführung in die objektorientierte Systementwicklung

## Warum ist Datenkapselung wichtig?



Unregulierter Zugriff eines Kreises



Kein direkter Zugriff auf Daten des Objekts „Kreis“

## [x] Operationen von Objekten

Operationen spezifizieren, welche Funktionalität ein Objekt bereitstellt. Unterstützt ein Objekt eine bestimmte Operation, sichert es einem Aufrufer damit zu, dass es bei einem Aufruf die Operation ausführen wird. Durch die Operation wird dabei zum einen die Syntax des Aufrufs vorgegeben, also zum Beispiel für welche Parameter Werte eines bestimmten Typs zusammen mit dem Aufruf übergeben werden müssen.

Zum anderen werden dadurch auch Zusicherungen darüber gemacht, welche Resultate die Operation haben wird.

## [»] Schnittstelle eines Objekts

Die Schnittstelle eines Objekts ist die Menge der Operationen, die das Objekt unterstützt. Die Schnittstelle sagt nichts über die konkrete Realisierung der Operationen aus.

## [»] Methoden

Methoden von Objekten sind die konkreten Umsetzungen von Operationen. Während Operationen die Funktionalität nur abstrakt definieren, sind Methoden für die Realisierung dieser Funktionalität zuständig.

# Einführung in die objektorientierte Systementwicklung

## [»] Kontrakt (Vertrag) bezüglich einer Operation

Objekte gehen einen Kontrakt ein, der die Rahmenbedingungen beim Aufruf einer Operation regelt.

Eine Operation hat *Vorbedingungen*, für deren Einhaltung der Aufrufer zu sorgen hat. Außerdem hat die Operation *Nachbedingungen*, für deren Einhaltung das aufgerufene Objekt verantwortlich ist. Zusätzlich können für ein Objekt *Invarianten* definiert werden. Das sind unveränderliche Bedingungen, die für das Objekt immer gelten sollen.

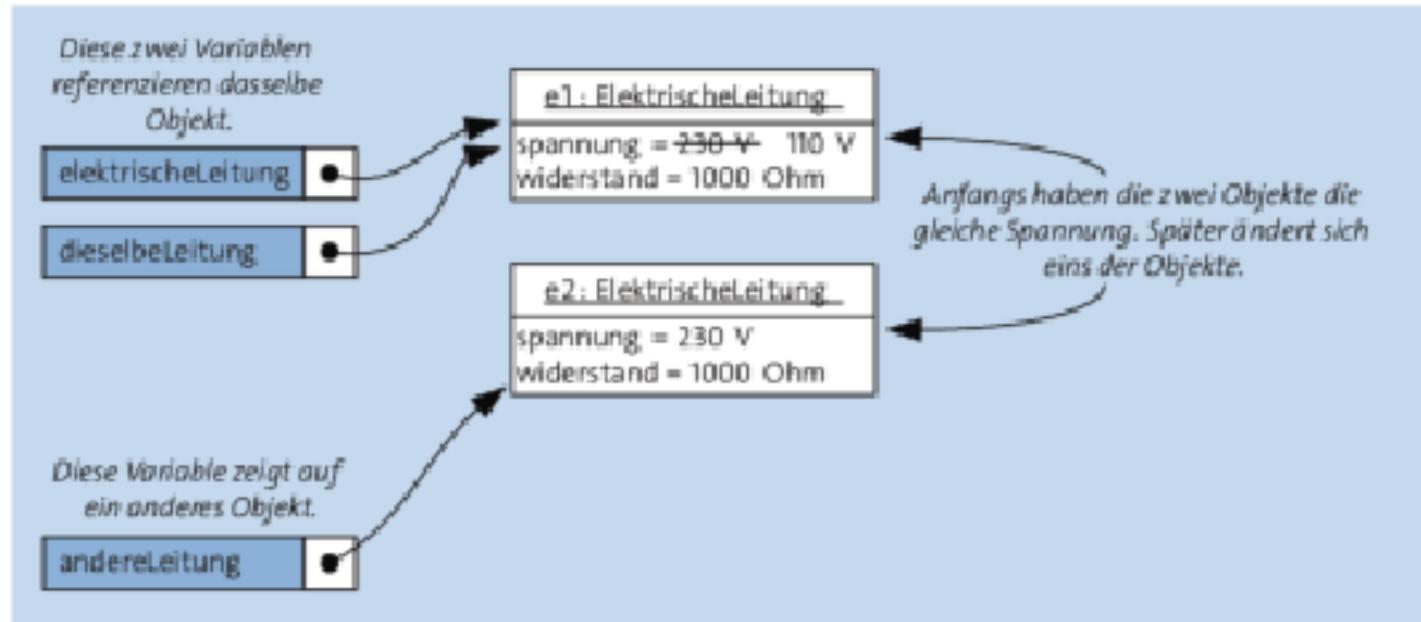
Mehr zu Vorbedingungen, Nachbedingungen und Invarianten finden Sie in [Abschnitt 4.2.2, »Kontrakte: die Spezifikation einer Klasse«](#).

## [»] Identität von Objekten

Objekte haben stets eine eigene Identität. Zwei Objekte können dadurch immer unterschieden werden, auch wenn sie zu einem Zeitpunkt exakt den gleichen Zustand aufweisen. Das Kriterium, nach dem Objekte grundsätzlich unterschieden werden können, wird Identitätskriterium genannt.

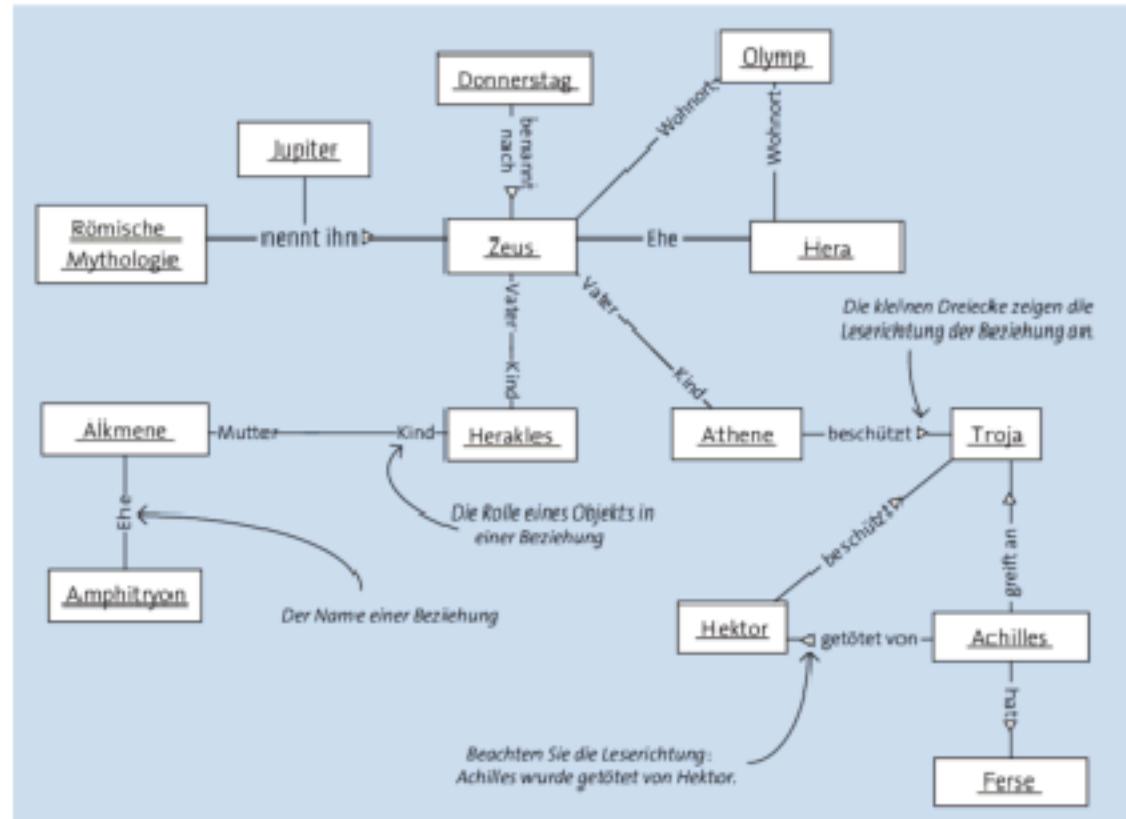
In vielen Programmiersprachen ist die Adresse des Objekts im Speicherbereich das generell verfügbare Identitätskriterium. In objektorientierten Anwendungen können auch andere Kriterien verwendet werden, wie zum Beispiel die Übereinstimmung einer eindeutigen Kennung.

# Einführung in die objektorientierte Systementwicklung



- Objekte vs „Werte“

Mehrere Variablen referenzieren dasselbe Objekt



Objekte haben Beziehungen: Beziehungen zwischen den Objekten der antiken Mythologie

Löwen

ein glücklicher Löwe

Clarence

Alex

Tiere

Gonzo

Hunde

- Welche Ordnung der Objekte würdet ihr vornehmen?
- Hängt von Modellierungszweck ab ...
- Dennoch Vorschlag! Wo sind Gemeinsamkeiten/Unterschiede?

Reptilien

Ka

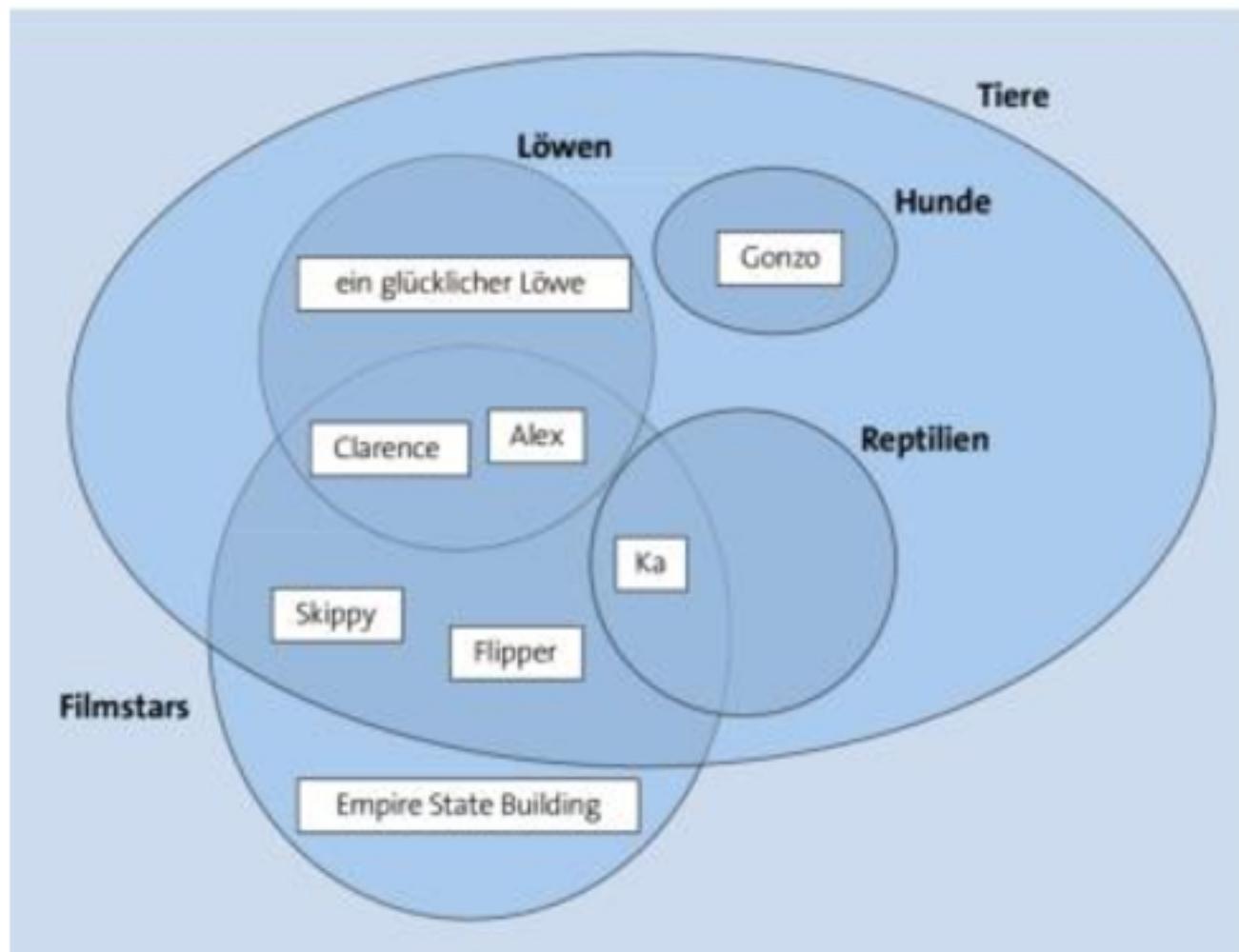
Skippy

Flipper

Filmstars

Empire State Building

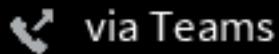
# Einführung in die objektorientierte Systementwicklung



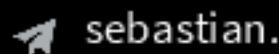
- Mengendarstellung für Klassen
- Mehrfach- vs Einfach-Vererbung
- Die meisten objektorientierten Sprachen sind eher „klassenorientiert“

# Danke

Sebastian Bichler



via Teams



[sebastian.bicher@iu.org](mailto:sebastian.bicher@iu.org)

Sommersemester 2023

DSBOOPI01

# Objektorientierte Programmierung I

## mit Java

#04, 27.04.2023, Leipzig

Prof. (FH) Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

Java-Basics  
Objekte – Klassen  
Attribute, Methoden,  
Beziehungen (UML)

- 1 Wiederholung bzw. Fortsetzung: „Einstieg in Java“**
- 2 Erste einfache Modellierung einer Kaffeemaschine (Gruppenarbeit)**
- 3 Einführung in die objektorientierte Systementwicklung**
- 4 Einführung in die objektorientierte Modellierung**

Java-Basics  
Objekte – Klassen  
Attribute, Methoden,  
Beziehungen (UML)

**Einstieg in Java**

**Algorithmen, Visualisierung, Kontrollstrukturen**

## Ziel dieses Abschnitts

- Modellierung (nur kurz Hinweise: DSPOOPI01\_Java\_Overview.pdf)
    - Flussdiagramm
    - Struktogramm
    - UML
- 
- Aufgaben: 7, 8, **9**, 10, 11, 12

## Aufgabe 7

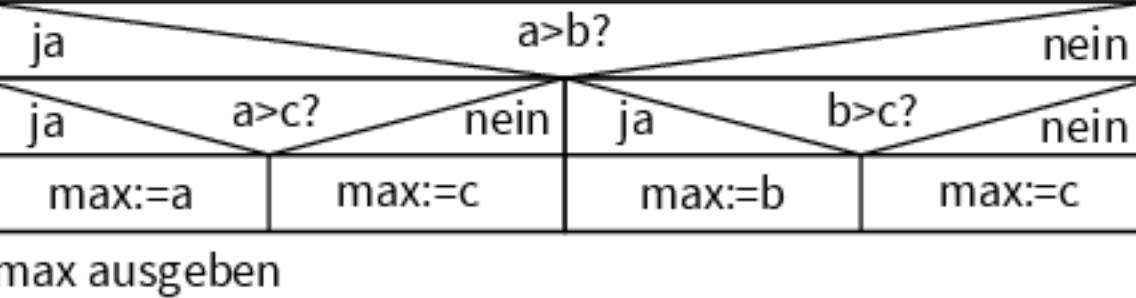
## Aufgabe 7

Variablen a, b, c, max deklarieren

a abfragen

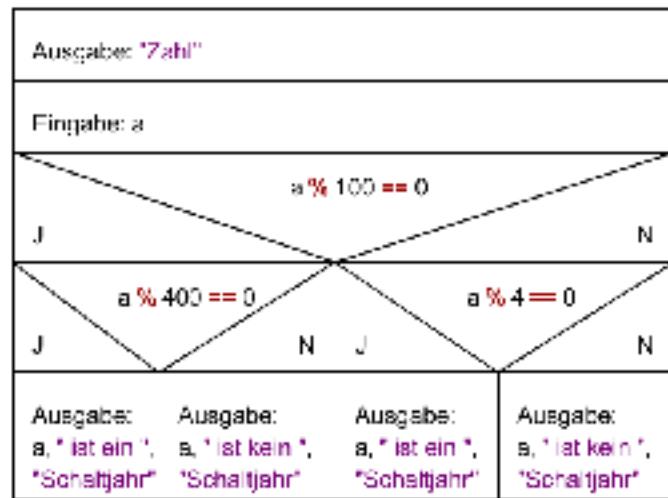
b abfragen

### c abfragen

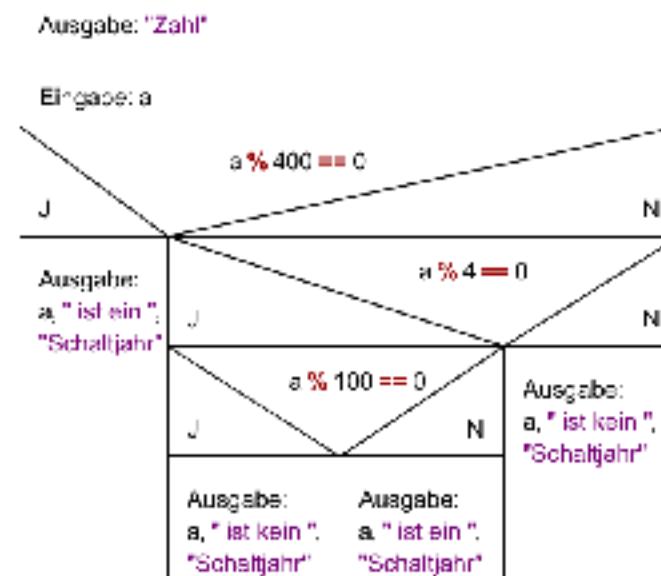


# Aufgabe 9

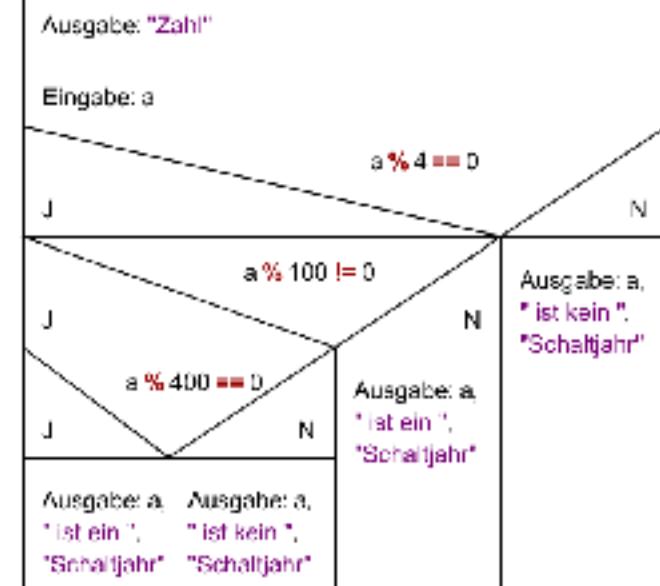
Schaltjahr (Variante 1)



Schaltjahr (Variante 2)



Schaltjahr (Variante 3)



# Aufgabe 9

```
public class Schaltjahr {  
    public static void main(String[] args){  
        java.util.Scanner scan = new java.util.Scanner(System.in);  
        System.out.print("Geben Sie eine Jahreszahl an: ");  
        int Jahr = scan.nextInt();  
        if (Jahr%4==0){  
            if (Jahr%100==0){  
                if (Jahr%400==0){  
                    System.out.println(Jahr + " ist Schaltjahr");  
                }  
                else{  
                    System.out.println(Jahr + " ist kein Schaltjahr");  
                }  
            }  
            else{  
                System.out.println(Jahr + " ist Schaltjahr");  
            }  
        }  
        else{  
            System.out.println(Jahr + " ist kein Schaltjahr");  
        }  
        scan.close();  
    }  
}
```

```
public class LeapYear  
{  
    public static void main(String[] args)  
    {  
        int year = Integer.parseInt(args[0]);  
        boolean isLeapYear;  
        isLeapYear = (year % 4 == 0);  
        isLeapYear = isLeapYear && (year % 100 != 0);  
        isLeapYear = isLeapYear || (year % 400 == 0);  
        System.out.println(isLeapYear);  
    }  
}
```

Dieses Programm prüft, ob ein Integer nach dem Gregorianischen Kalender einem Schaltjahr entspricht. Ein Jahr ist ein Schaltjahr, wenn es durch 4 teilbar ist (2004). Ausgenommen hiervon sind die Integer, die durch 100 teilbar sind (1900) – dann ist es kein Schaltjahr, es sei denn die Zahl ist durch 400 teilbar (2000) – dann ist es wieder ein Schaltjahr.

```
% javac LeapYear.java  
% java LeapYear 2004  
true  
% java LeapYear 1900  
false  
% java LeapYear 2000  
true
```

## Aufgabe 9

```
public class Aufgabe_09 {  
    public static void main(String[] args){  
        int jahr = 2020;  
        boolean schaltjahr;  
  
        if (jahr % 4 == 0 && (jahr % 100 != 0 || jahr % 400 == 0))  
            schaltjahr = true;  
        else  
            schaltjahr = false;  
  
        if (schaltjahr)  
            System.out.println(jahr + " ist ein Schaltjahr.");  
        else  
            System.out.println(jahr + " ist kein Schaltjahr.");  
    }  
}
```

```
public class LeapYear  
{  
    public static void main(String[] args)  
    {  
        int year = Integer.parseInt(args[0]);  
        boolean isLeapYear;  
        isLeapYear = (year % 4 == 0);  
        isLeapYear = isLeapYear && (year % 100 != 0);  
        isLeapYear = isLeapYear || (year % 400 == 0);  
        System.out.println(isLeapYear);  
    }  
}
```

Dieses Programm prüft, ob ein Integer nach dem Gregorianischen Kalender einem Schaltjahr entspricht. Ein Jahr ist ein Schaltjahr, wenn es durch 4 teilbar ist (2004). Ausgenommen hiervon sind die Integer, die durch 100 teilbar sind (1900) – dann ist es kein Schaltjahr, es sei denn die Zahl ist durch 400 teilbar (2000) – dann ist es wieder ein Schaltjahr.

```
% javac LeapYear.java  
% java LeapYear 2004  
true  
% java LeapYear 1900  
false  
% java LeapYear 2000  
true
```

## Aufgabe 10, 11, 12

- bitte selbstständig oder in der Gruppe durchführen (Selbststudium) und
- in die Teams-Gruppe hochladen (im Ordner „Aufgaben“ unter dem eigenem Namen oder der Gruppe)

- 1 Wiederholung bzw. Fortsetzung: „Einstieg in Java“**
- 2 Erste einfache Modellierung einer Kaffeemaschine (Gruppenarbeit)**
- 3 Einführung in die objektorientierte Systementwicklung**
- 4 Einführung in die objektorientierte Modellierung**

Java-Basics  
Objekte – Klassen  
Attribute, Methoden,  
Beziehungen (UML)

## Mini-Requirements Engineering

- Produktbeschreibung mit wesentlichen Anforderungen
- Statische Analyse: Aufbau, Komponenten
- Dynamische Analyse: Funktionsweise



- UML-Modellierung (der „tatsächlichen“ Objekte)
- Gruppenarbeit ...

- 1 Wiederholung bzw. Fortsetzung: „Einstieg in Java“**
- 2 Erste einfache Modellierung einer Kaffeemaschine (Gruppenarbeit)**
- 3 Einführung in die objektorientierte Systementwicklung**
- 4 Einführung in die objektorientierte Modellierung**

## Java-Basics

Objekte – Klassen  
Attribute, Methoden,  
Beziehungen (UML)

## » Klassifizierung

Die Zuordnung von Objekten zu Klassen heißt Klassifizierung. Dabei werden relevante Gemeinsamkeiten von Objekten identifiziert und Objekte mit diesen Gemeinsamkeiten derselben Klasse zugeordnet.

- Exemplar vs Instanz
- Instanz oft verwendet, aber nicht zu empfehlen

## » Exemplare einer Klasse

Objekte sind Exemplare der Klassen, zu denen sie gehören. Dabei kann eine Klasse mehrere Exemplare haben, und Objekte können auch Exemplare von mehreren Klassen sein.

In den meisten Programmiersprachen ist ein Objekt immer nur ein direktes Exemplar einer einzigen Klasse. Über Beziehungen zwischen Klassen, die wir in [Kapitel 5, »Vererbung und Polymorphie«](#), vorstellen werden, kann ein Objekt aber auch indirekt Exemplar von weiteren Klassen sein. Im weiteren Verlauf werden wir generell den Begriff Exemplar verwenden. Zwischen direkten und indirekten Exemplaren werden wir nur unterscheiden, wenn diese Differenzierung für eine Problemstellung relevant ist.

## [»] Einfache und mehrfache Klassifizierung

Wenn ein Objekt immer nur ein direktes Exemplar einer einzigen Klasse sein kann, spricht man von der *einfachen Klassifizierung*. Die meisten objektorientierten Programmiersprachen unterstützen nur die einfache Klassifizierung. Kann ein Objekt direkt mehreren konkreten Klassen zugeordnet werden, spricht man von *mehrfacher Klassifizierung*.

Bei der Klassifizierung gibt es generell zwei verschiedene Ebenen, auf denen Objekte zu Klassen gruppiert werden:

- das konzeptionelle Modell (oder Analysemodell)
- das Implementierungsmodell (oder Designmodell)

# Einführung in die objektorientierte Systementwicklung

## [»] Konzeptionelles Modell (Analysemodell)

In einem **konzeptionellen Modell** beschreibt eine Klasse die konzeptionellen Gemeinsamkeiten von bestimmten Objekten, deren Rollen, deren Verwendung und deren Verantwortlichkeiten. Die erstellten Konzepte bleiben unabhängig von der Technologie, in der Software realisiert werden soll. Sie beschreiben die Fachdomäne und nicht die Software.

Klassen dienen im konzeptionellen Modell dazu, Begriffe, Beziehungen und Prozesse der Fachdomäne zu kategorisieren und zu strukturieren. Die Aufgabe, das konzeptionelle Modell zu erstellen, wird als *Analysephase* bezeichnet. Sie wird häufig nicht von Softwareentwicklern, sondern von separaten Teams in Abstimmung mit den fachlichen Anforderern durchgeführt.

## [»] Das Implementierungsmodell

Ein **Implementierungsmodell** legt die konkrete Umsetzung des konzeptionellen Modells fest. Manche Klassen aus dem konzeptionellen Modell können Klassen im Implementierungsmodell direkt entsprechen. Häufig wird es keinen direkten Bezug zu Klassen der Implementierung geben. Abhängig von gewählter Architektur und Programmiersprache werden bestimmte Konzepte unterschiedlich komplex realisiert. Wir werden im Folgenden die Klassen immer auf der Ebene des Implementierungsmodells betrachten.

# Einführung in die objektorientierte Systementwicklung

Klassen haben in einem Implementierungsmodell zwei unterschiedliche Funktionen:

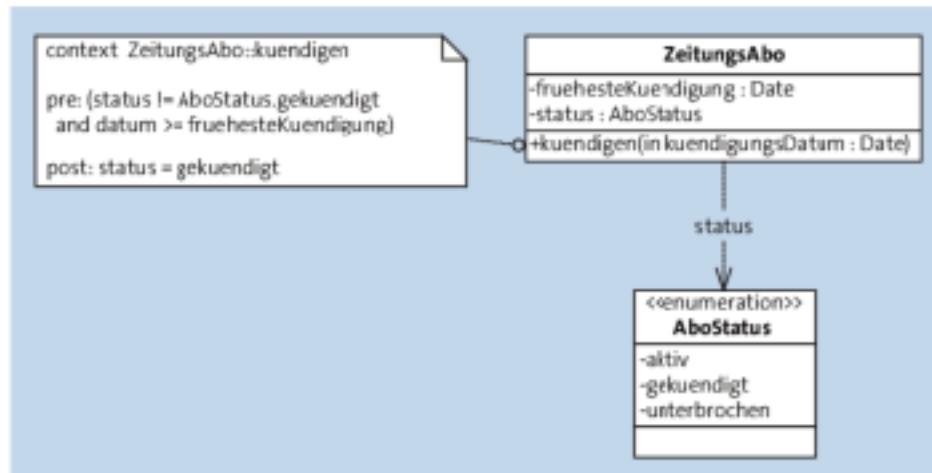
- Zum einen können Sie eine Spezifikation auf Basis der Klassen erstellen. Dabei entspricht die Klasse einem abstrakten Typ: Sie beschreibt exakt die Schnittstelle für alle Exemplare der Klasse. Diese Rolle der Klassen werden wir im folgenden [Abschnitt 4.2.2](#) genauer erläutern.
- Zum anderen ist einer Klasse aber auch eine Implementierung zugeordnet. Sie kann damit als Modul verwendet werden, das eine bestimmte Umsetzung ihrer Spezifikation kapselt. In [Abschnitt 4.2.4](#) werden wir auf diese Sichtweise genauer eingehen.

# Einführung in die objektorientierte Systementwicklung

## [»] Schnittstelle einer Klasse

Klassen legen für alle ihre Exemplare fest, welche Eigenschaften und Operationen diese Exemplare besitzen. Die Schnittstelle einer Klasse besteht damit aus allen durch die Klasse definierten Eigenschaften und Operationen. Außerdem werden durch die Schnittstelle die Vor- und Nachbedingungen für die Operationen beschrieben sowie geltende Invarianten für Beziehungen zwischen den Eigenschaften von Exemplaren der Klasse.

Einige Programmiersprachen bieten Konstrukte an, die explizit sogenannte Schnittstellenklassen definieren. Es ist aber zu beachten, dass jede Klasse eine Schnittstelle definiert. Reine Schnittstellenklassen haben dabei lediglich die Restriktion, dass sie nicht gleichzeitig eine Implementierung der Schnittstelle anbieten können.



## [»] Spezifikation einer Klasse

Die Spezifikation einer Klasse beschreibt für alle Operationen dieser Klasse deren Vor- und Nachbedingungen. Zusätzlich enthält die Spezifikation eine Beschreibung der Invarianten, die für alle Exemplare der Klasse gelten. Damit lässt sich die Spezifikation einer Klasse in Form von Vorbedingungen, Nachbedingungen und Invarianten formulieren.

# Einführung in die objektorientierte Systementwicklung

- Klassen sind Datentypen

## [»] Typsysteme der Programmiersprachen

Ein Typsystem ist ein Bestandteil der Umsetzung einer Programmiersprache oder deren Laufzeitumgebung, der die im Programm verwendeten Datentypen zur Übersetzungszeit oder Laufzeit überprüft. Dabei wird jedem Ausdruck ein Typ zugeordnet. Das Typsystem stellt dann fest, ob sich der Ausdruck in einer bestimmten Verwendung mit den Regeln des Typsystems verträgt.

Zu diesen Prüfungen gehört zum Beispiel auch, ob eine Operation auf einem Objekt durchgeführt oder ob ein Objekt einer Variablen zugeordnet werden kann.

Je nachdem, wann programmtechnisch diese Prüfung stattfindet, unterscheiden wir die Typsysteme der Programmiersprachen:

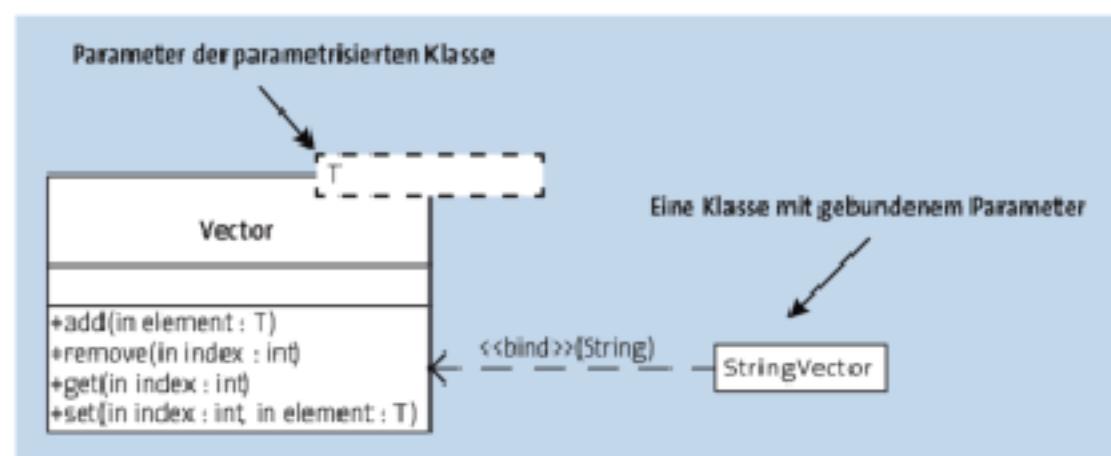
- Beim statischen Typsystem erfolgt die Überprüfung zur Übersetzungszeit des Programms.
- Beim dynamischen Typsystem erfolgt die Prüfung zur Laufzeit des Programms.

## ■ Parametisierte Klassen

### [»] Parametisierte Klassen

Bei der Deklaration von parametrisierten Klassen können ein oder mehrere Typparameter angegeben werden. Wird ein Exemplar einer solchen Klasse erstellt, muss für diesen Parameter ein konkreter Typ angegeben werden. Zur Übersetzungszeit wird der Parameter dann durch den konkret angegebenen Typ ersetzt.

Eine Klassendeklaration, die mit einem Typparameter versehen ist, deklariert nicht nur eine Klasse, sondern eine Menge von Klassen, die sich durch den konkreten Wert der Typparameter unterscheiden.



## [>] Stark und schwach typisierte Sprachen

Eine stark typisierte Sprache überwacht das Erstellen und den Zugriff auf alle Objekte so, dass sichergestellt ist, dass Variablen immer auf Objekte verweisen, die auch die Spezifikation des Typs erfüllen, der für die Variable deklariert ist.

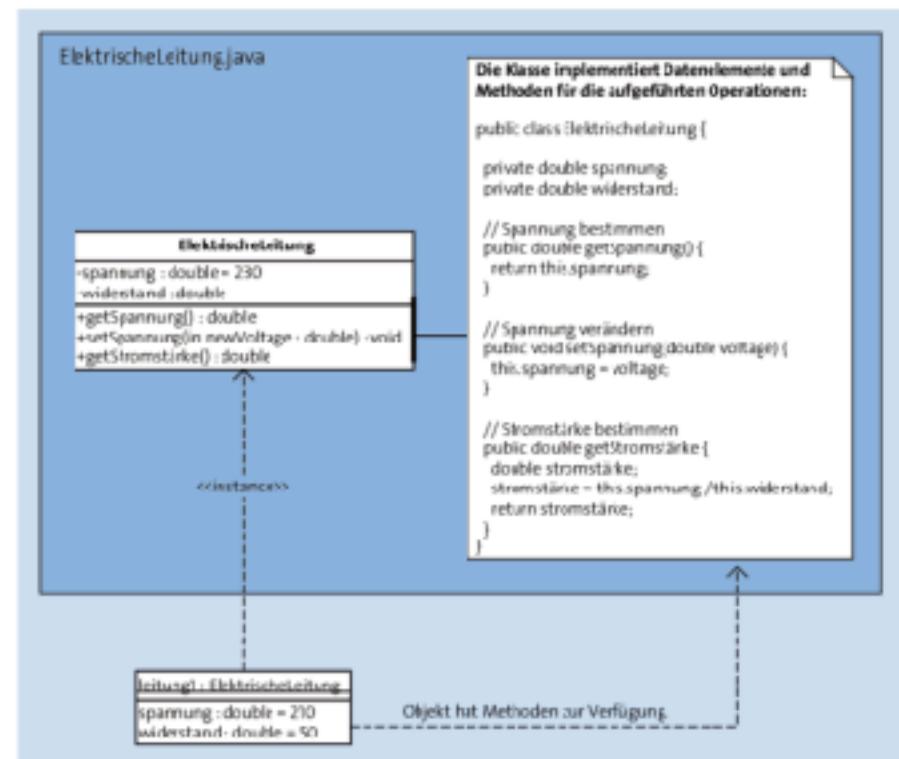
Schwach typisierte Sprachen haben diese Restriktion nicht. In solchen Sprachen ist es möglich, ein Objekt einer Variablen zuzuordnen, ohne dass das Objekt notwendigerweise die Spezifikation des Typs der Variablen erfüllt.

## ■ Klassen sind Module

Eine Klasse übernimmt also die Aufgabe eines Moduls und stellt häufig die folgenden Elemente zur Verfügung:

- Sie deklariert die Methoden der Klasse.
- Sie enthält die Implementierung der deklarierten Methoden.
- Sie deklariert die Datenelemente ihrer Exemplare.<sup>[17]</sup>

- Klasse spezifiziert nicht nur eine Schnittstelle, sondern stellt oft eine Implementierung der Schnittstelle zur Verfügung.
- Klasse muss nicht für alle spezifizierten Operationen wirklich eine Methode umsetzen (abstrakte Klassen, ...)
- Programmiersprachen unterscheiden sich bezüglich der Konzepte zur Modularisierung, ...

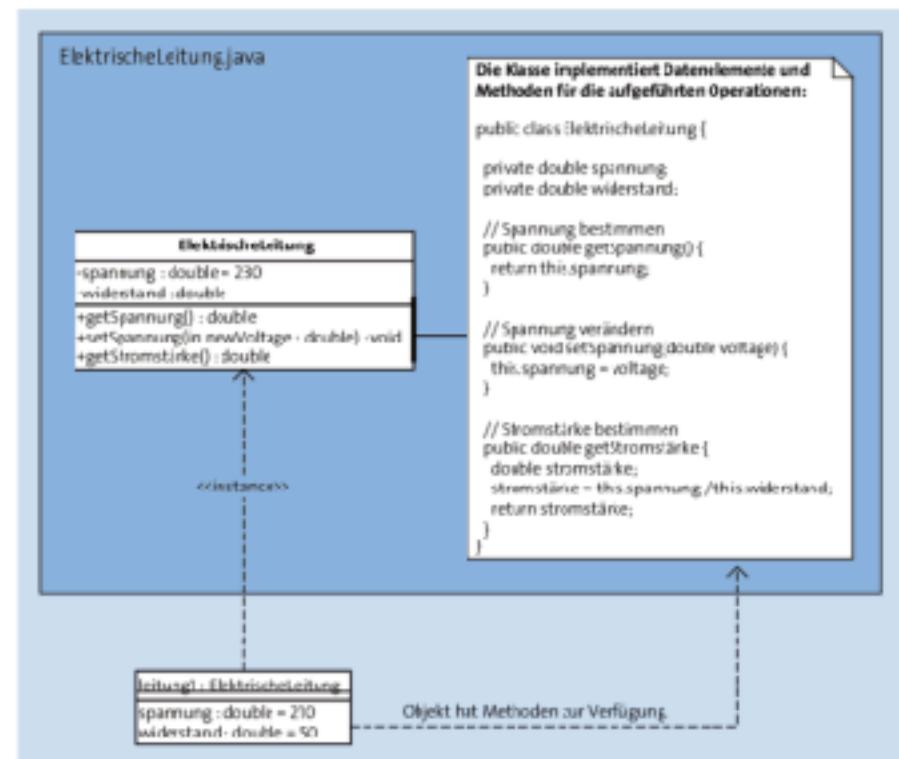


## ■ Klassen sind Module

Eine Klasse übernimmt also die Aufgabe eines Moduls und stellt häufig die folgenden Elemente zur Verfügung:

- Sie deklariert die Methoden der Klasse.
- Sie enthält die Implementierung der deklarierten Methoden.
- Sie deklariert die Datenelemente ihrer Exemplare.<sup>[17]</sup>

- Klasse spezifiziert nicht nur eine Schnittstelle, sondern stellt oft eine Implementierung der Schnittstelle zur Verfügung.
- Klasse muss nicht für alle spezifizierten Operationen wirklich eine Methode umsetzen (abstrakte Klassen, ...)
- Programmiersprachen unterscheiden sich bezüglich der Konzepte zur Modularisierung, ...



- 1 Eigene Java-Projekte / „Mobile First“ && „Web First“**
- 2 Java-Basics (Einheiten: 1, 2) bearbeiten**
- 3 Einführung in die objektorientierte Systementwicklung**
- 4 Einführung in die objektorientierte Modellierung**

## Java-Basics

### Objekte – Klassen

Attribute, Methoden,  
Beziehungen (UML)

- Eine Klasse liefert in der Objektorientierung die Struktur, die zur Bildung eines „digitalen Objektes“ erforderlich ist. Auf Basis einer Klasse können beliebig viele Objekte erzeugt werden. Alle aus einer Klasse erzeugten Objekte haben die gleiche Struktur, also dieselben Attribute und Methoden.
- Bei der objektorientierten Entwicklung werden Objekte aus der physischen (realen) Welt in Objekte der Programmiersprache nachgebildet. So gibt es dann zum „echten“ Kunden „Hans Meier“ ein digitales Gegenstück in Form eines digitalen Objektes „Hans Meier“. Der real existierende Kunde (Herr Hans Meier) wird in Form eines „digitalen Objektes“ der jeweiligen Programmiersprache des IT-Systems nachgebildet. Bei dieser digitalen Nachbildung werden aber nur die wichtigsten Eigenschaften wie Name, Vorname, Geburtsdatum oder Geschlecht übernommen. Eigenschaften wie Haarfarbe, Augenfarbe oder Farbe der Kleidung interessieren in den meisten Fällen nicht, da sie für die Aufgaben des Systems nicht benötigt werden.

- Ein objektorientiertes System erzeugt dann bei Bedarf die entsprechenden Objekte, zum Beispiel aus Datenbankeinträgen oder Benutzereingaben. Durch die Beschreibung der Klassen wird gewährleistet, dass sich alle Objekte, die auf Basis dieser Klassen erzeugt werden, gleich verhalten. Alle Objekte einer Klasse haben dieselben Attribute und verfügen über die gleichen Funktionen und Routinen (die sogenannten Methoden).
- Damit können alle aus einer Klasse erzeugten Objekte von den Funktionen des Systems in gleicher Weise verarbeitet werden.

# Einführung in die objektorientierte Modellierung

## Identifizieren von Klassen

- Um bei der Analyse zu bestimmen, was ein System tun soll, und dann darauf aufbauend später beim Design festzulegen, aus welchen Klassen das System bestehen soll, wird in einem ersten Schritt ein Analysemodell des Systems entwickelt:
- Aus der Aufgabenstellung beziehungsweise den Anforderungen an das System müssen Kandidaten für Klassen identifiziert und notiert werden.
- Analysemodell: Das ist das Ergebnis der objektorientierten Analyse und dient zur Kommunikation zwischen den Entwicklern und dem Auftraggeber bzw. den Anwendern des Systems.

# Einführung in die objektorientierte Modellierung

## Identifizieren von Klassen

Eine etablierte Vorgehensweise zur Identifikation von Klassen kann wie folgt beschrieben werden:

1. Alle Hauptwörter (Substantive) der Aufgabenstellung werden als Kandidaten für Klassen markiert.
2. Dann wird geprüft, ob die markierten Hauptwörter durch weitere Hauptwörter beschrieben werden können oder ob sie Beziehungen oder Abhängigkeiten zu anderen Hauptwörtern haben. Trifft eine der beiden Aussagen zu, wird das Hauptwort als Klasse modelliert.
3. Wird das Hauptwort verwendet, um ein anderes Hauptwort zu detaillieren, wird es als Attribut einer Klasse modelliert.
4. Alle verbleibenden Wörter, die weder Klasse noch Attribut sind, werden auf Relevanz überprüft und ggf. von der Liste der Kandidaten für Klassen gestrichen.

# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop

Die in einem Online-Shop zu verkaufenden Artikel sollen Medien aller Art sein, insbesondere **Bücher**, **Musikartikel**, **Filme** und **Spiele**. Jeder Artikel hat einen Hersteller, einen **Titel** und eine **Artikelnummer**. *Bücher* und *Spiele* haben einen **Autor**, *Filme* einen **Regisseur** und *Musikartikel* einen **Interpreten**.

Aufgaben:

- Überprüft nun diese Wörter und identifiziert sie als Klassen ...
- Stelle die Klassen grafisch dar (mit oder ohne Beziehung)



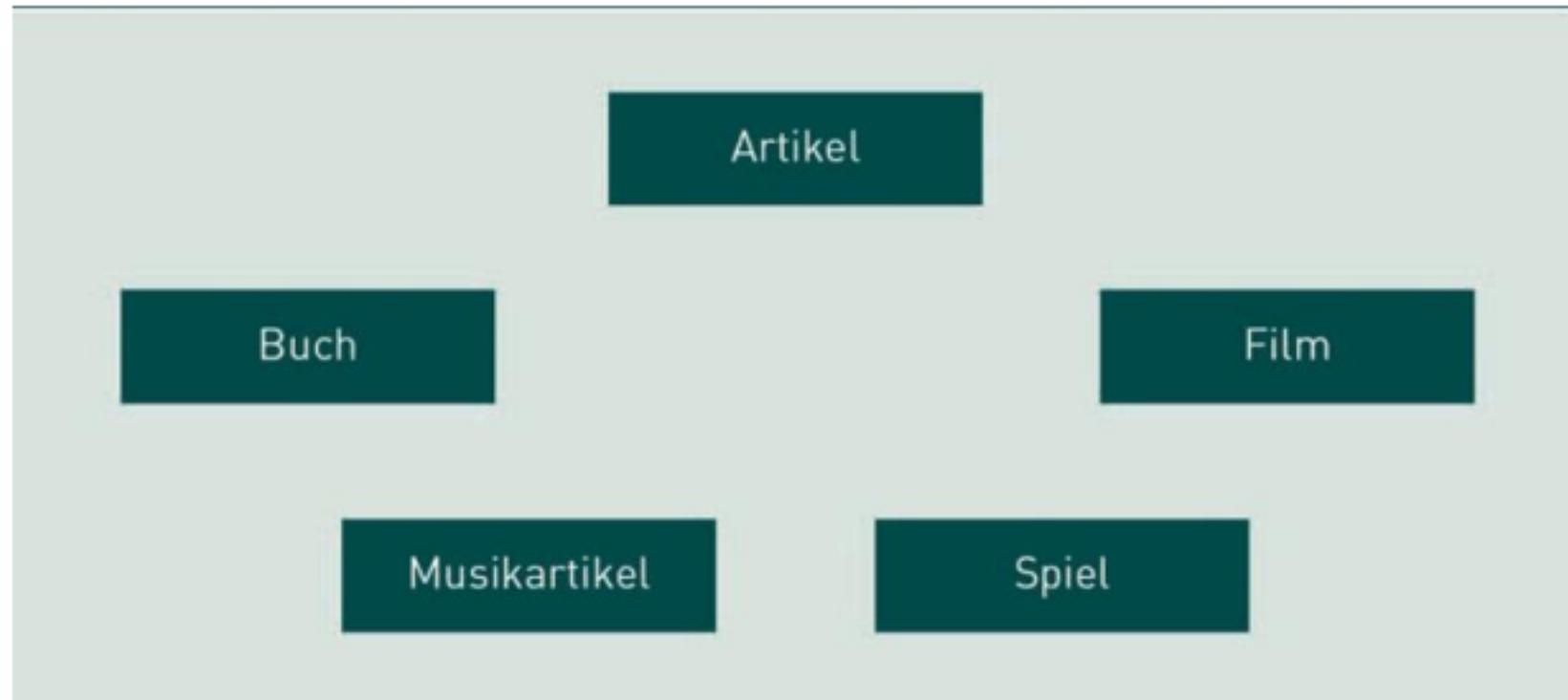
# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop

- Schritt 1: Nach dem Markieren aller Hauptwörter besteht die Liste aus folgenden Einträgen (dabei werden die Hauptwörter nur in Einzahl notiert):
  - Online-Shop, Artikel, Medien, Art, Buch, Musikartikel, Film, Spiel, Hersteller, Titel, Artikelnummer, Autor, Regisseur, Interpret
- Schritt 2–4: Überprüfung auf Kandidat für Klasse, Attribut und generell auf Relevanz:
  - Online-Shop: nicht relevant, weil es das System als Ganzes bezeichnet
  - Artikel: als Klasse relevant
  - Medien: nicht relevant, da nur Beschreibung zu Artikel
  - Art: nicht relevant, da nur Beschreibung zu Artikel
  - Buch: als Klasse relevant, da durch Attribute verfeinert
  - Musikartikel: als Klasse relevant, da durch Attribute verfeinert
  - Film: als Klasse relevant, da durch Attribute verfeinert
  - Spiel: als Klasse relevant, da durch Attribute verfeinert
  - Hersteller: als Attribut relevant, da Beschreibung zu Klassen
  - Titel: als Attribut relevant, da Beschreibung zu Klassen
  - Artikelnummer: als Attribut relevant, da Beschreibung zu Klassen
  - Autor: als Attribut relevant, da Beschreibung zu Klassen
  - Regisseur: als Attribut relevant, da Beschreibung zu Klassen
  - Interpret: als Attribut relevant, da Beschreibung zu Klassen

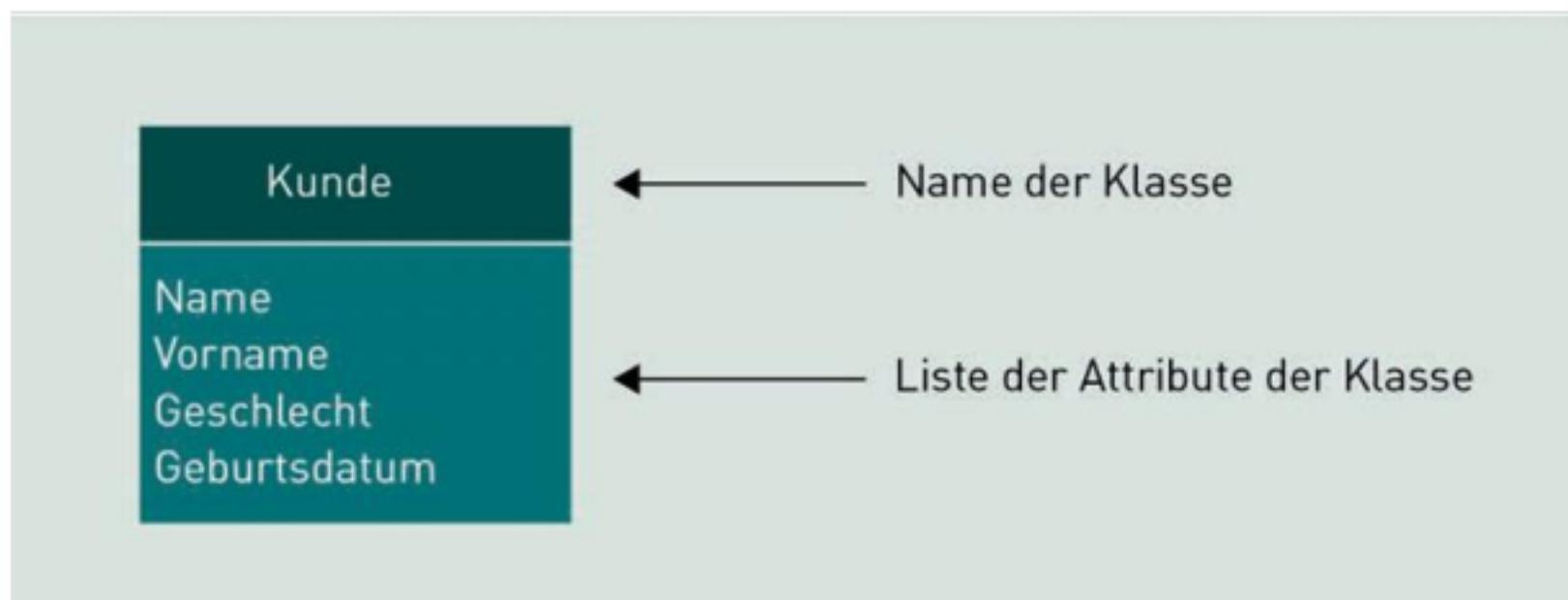
# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Klassen



# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Attribute



# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Attribute

Eigenschaften eines Attributs	Beschreibung	Beispiel
Name	Name des Attributs	Vorname
Datentyp	Legt fest, wie die Werte zu dem Attribut aussehen, also ob es z. B. eine Zahl, eine Zeichenkette oder ein Datum ist	Date (Datum) String (Zeichenkette) Integer (Ganze Zahl)
Konstante (ja/nein)	Legt fest, ob sich der Wert des Attributs ändern darf oder nicht	Gerundete Kreiszahl Pi: 3,1415
Defaultwert	Legt den voreingestellten Wert des Attributes fest	2010-01-01

# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Attribute

Nach der Identifikation möglicher Klassen müssen nun die Attribute zu den Klassen identifiziert werden. Gute Kandidaten für Attribute sind Hauptwörter, die zur Beschreibung oder Detaillierung anderer Hauptwörter genutzt werden. In der folgenden Abbildung sind die im Beispielszenario Online-Shop identifizierten Attribute ergänzt:

Artikel			
Buch	Musikartikel	Spiel	Film
Hersteller	Hersteller	Hersteller	Hersteller
Titel	Titel	Titel	Titel
Artikelnummer	Artikelnummer	Artikelnummer	Artikelnummer
Autor	Interpret	Autor	Regisseur

# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Methoden

- Methoden (auch: Funktionen, Operationen) sind dynamische Elemente von Klassen.
- Sie enthalten Algorithmen, Anweisungen und Abarbeitungsvorschriften, mit denen Werte erstellt, berechnet, verändert und gelöscht werden können.
- Die Ergebnisse der Methoden können in Attribute der Klasse oder in neu erzeugte Objekte gespeichert werden.
- Mit Methoden wird das Verhalten von Objekten beschrieben, also was ein Objekt macht.
- So können mit Methoden Berechnungen von Werten und das Anwenden von Geschäftsregeln oder Vorschriften zum Erzeugen und Löschen von Objekten programmiert werden.

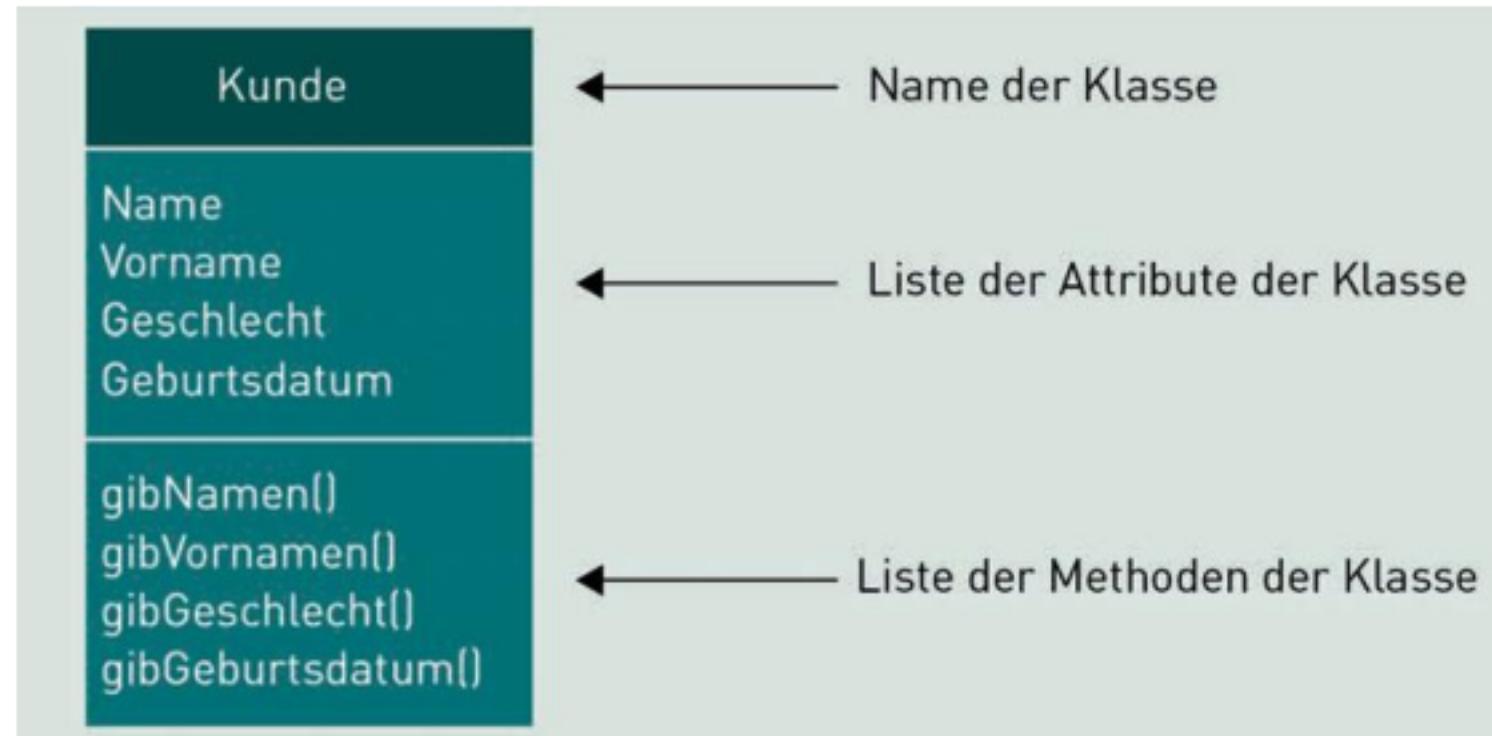
# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Methoden

- Zur bereits mit Attributen versehenen Klasse „Kunde“ sollen nun die Methoden festgelegt werden.
- Wichtig ist in jedem Fall, dass die in den Attributen gespeicherten Werte wieder ausgelesen werden können.
- Wie bereits beschrieben, darf von außen auf Attribute nur über Methoden zugegriffen werden und niemals direkt. (Setter)
- Daher muss die Klasse „Kunde“ für jedes ihrer Attribute eine Methode bereitstellen, die den aktuellen Wert des Attributs zurückgibt (Getter)

# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Klasse mit Attribut und Methoden



# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Beziehungen zwischen Klassen

- Alle Elemente eines objektorientierten Systems werden in Form von Klassen programmiert.
- Ein Zusammenspiel verschiedener Klassen ist nur möglich, wenn während der Phasen des Entwicklungsprozesses Beziehungen (auch: Assoziationen) zwischen Klassen definiert werden.
- Durch die Festlegung von Beziehungen zwischen Klassen ist eine Kooperation zwischen Objekten überhaupt erst möglich.

# Einführung in die objektorientierte Modellierung

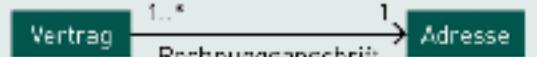
## Beispiel Online-Shop: Beziehungen zwischen Klassen

Beziehungs- typen	Beschreibung	Beispiel
„hat/kennt“	Dieser Beziehungstyp drückt aus, dass eine Klasse eine andere Klasse „hat“ oder „kennt“.	Ein Versicherungsnenmer hat Kinder. Ein Vertrag hat Versicherungsbedingungen. Ein Kalender hat Monate. Ein Verkäufer kennt seine Kunden.
„besteht aus“	Dieser Beziehungstyp drückt aus, dass eine Klasse ein Bestandteil einer anderen Klasse ist. Er wird genutzt, wenn eine Klasse ein großes Konstrukt ist, dessen Elemente nicht durch einfache Attribute beschrieben werden können.	Ein Auto besteht aus einem Motor, 4 Radern, 3 Türen, 1 Getriebe und 2 Sitzen. Ein Haus besteht aus einem Dach, 23 Fenstern, 2 Türen, 6 Räumen und 1 Treppenhaus.

Beziehungs- typen	Beschreibung	Beispiel
„ist ein“	Dieser Beziehungstyp drückt aus, dass eine Klasse A von der Art her eine Klasse B ist, aber eine spezifischere Bedeutung hat und sich ggf. um bestimmte Attribute und Methoden von Klasse B unterscheidet.	Ein Pkw ist ein Auto. Ein Lkw ist ein Auto. Ein Kunde ist eine Person. Ein Buch ist ein Artikel. Ein Igel ist ein Säugetier.

# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Darstellung von Beziehungen

Darstellung der Beziehung	Bedeutung	Darstellung der Beziehung	Bedeutung
 Durchgezogene Linie, ohne Pfeilspitze und ohne Beschriftung	Vertrag und Adresse stehen in einer nicht näher beschriebenen Beziehung zueinander.	 An den Enden der Beziehung werden Multiplizitäten modelliert und damit Aussagen über die Anzahl der assoziierten Objekte gemacht.	Ein Vertrag hat genau eine Rechnungsanschrift, eine Adresse kann jedoch die Rechnungsanschrift zu mindestens 1 aber maximal beliebig vielen Verträgen sein. (Detaillierte Erklärung dazu siehe unten.)
 Durchgezogene Linie, Beschriftung der Linie mit einem Namen für die Beziehung	Vertrag und Adresse sind über eine benannte Assoziation verbunden; die beiden Klassen sind verbunden durch die Beziehung „Rechnungsanschrift“.		
 Durchgezogene Linie mit einer Pfeilspitze, ggf. Benennung der Beziehung	Durch die Pfeilspitze wird eine Navigationsrichtung vorgegeben: vom Vertrag zur Adresse. Das bedeutet, dass der Vertrag eine Adresse kennt und man sich vom Vertrag zur Adresse durchhangeln kann. Jedoch nicht von der Adresse zum Vertrag: Ein Vertrag kennt seine Adresse, aber die Adresse weiß nichts von und über die Existenz des Vertrags.		

# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Darstellung von Beziehungen

- Zu Beziehungen können mit Multiplizitäten (auch: Kardinalitäten) Mengenangaben festgelegt werden.
- Diese Angaben werden jeweils an das Ende und die Spitze einer Beziehung notiert:
- Links von „..“ steht die Untergrenze und rechts von „..“ die Obergrenze der Mengenangaben.

# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Darstellung von Beziehungen

Notation	Erklärung	Beispiel
0..1	Optionale Assoziation	 <p>Zu einem Auto gehören 0..1 Anhänger. Zu einem Anhänger gehören 0..1 Autos.</p>
1	Obligatorische Assoziation	 <p>Zu einem Auto gehört genau 1 Fahrer. Ein Fahrer gehört zu genau 1 Auto.</p>
0..*	Optional beliebig	 <p>Ein Student kann 0..* Kurse belegen. Ein Kurs kann von 1..* Studenten belegt sein.</p>
1..*	Beliebig, aber mindestens 1	 <p>Ein Tutor kann 1..* Kurse machen. Ein Kurs wird von genau 1 Tutor durchgeführt.</p>

Notation	Erklärung	Beispiel
n..m	Mindestens n und maximal m	 <p>Ein Auto hat mindestens 3 und maximal 4 Türen. Eine Tür gehört zu genau 1 Auto.</p>
	Keine Angabe entspricht 1 (sollte aber vermieden werden, um Missverständnisse zu vermeiden)	 <p>Ein Auto hat genau 1 Motor. Ein Motor gehört zu genau 1 Auto.</p>

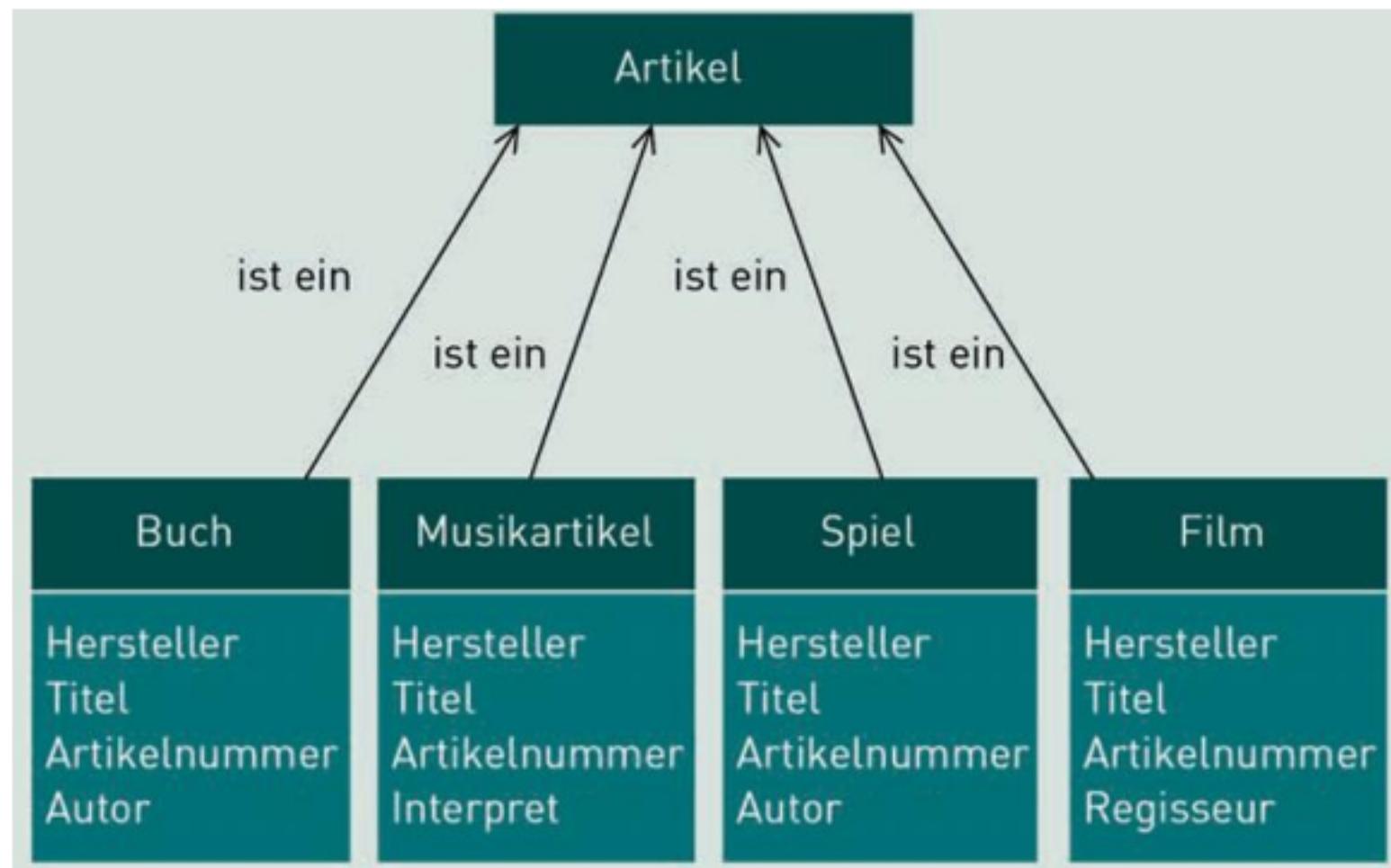
# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Darstellung von Beziehungen

- Um mögliche Beziehungen für das Beispielszenario „Online-Shop“ zu identifizieren und zu ergänzen, müssen wir uns erneut den ursprünglichen Text vomehmen:
  - „Die im Online-Shop zu verkaufenden Artikel sollen Medien aller Art sein, also insbesondere Bücher, Musikartikel, Filme und Spiele.“
  - Jeder Artikel hat einen Hersteller, einen Titel und eine Artikelnummer.
  - Bücher und Spiele haben einen Autor, Filme einen Regisseur und Musikartikel einen Interpreten.“
- Folgende Abbildung zeigt eine entsprechende Darstellung:
  - Außer den „ist ein“-Beziehungen der Klasse „Artikel“ zu den Klassen „Buch“, „Musikartikel“, „Spiel“ und „Film“ sind vorerst keine weiteren Beziehungen erforderlich.

# Einführung in die objektorientierte Modellierung

## Beispiel Online-Shop: Darstellung von Beziehungen



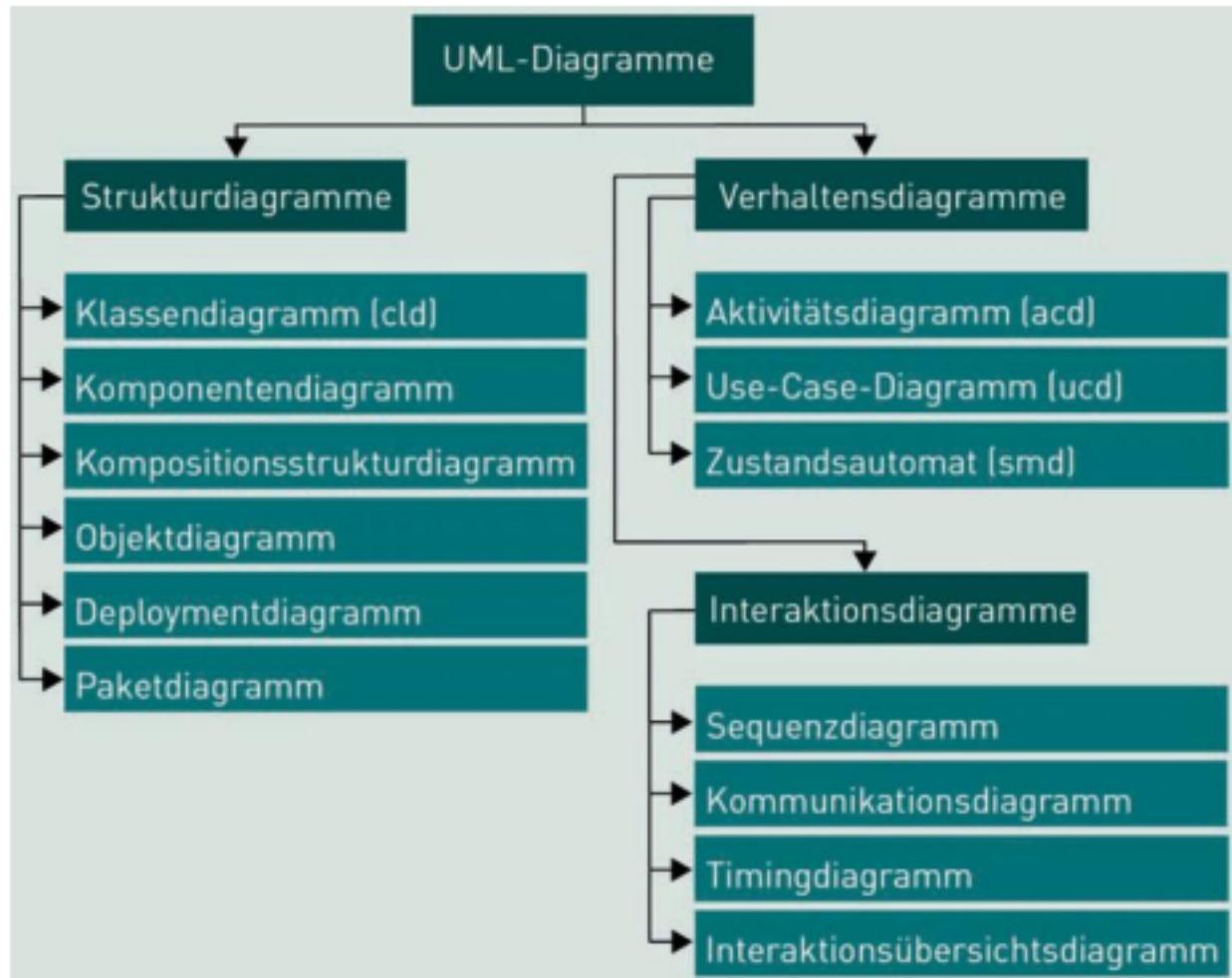
# Einführung in die objektorientierte Modellierung

## UML

- Klassen mit ihren Attributen, Methoden und Beziehungen werden mit dem UML-Klassendiagramm modelliert.
- Alle vorherigen Beispiele sind gültige UML- Klassendiagramme.
- Das UML Klassendiagramm ist weltweit eine der am häufigsten genutzten Dokumentationsform bei der objektorientierten Systementwicklung.
- Alle anderen Strukturdiagramme der UML bauen mehr oder weniger auf den Modellierungskonzepten des Klassendiagramms auf.

# Einführung in die objektorientierte Modellierung

## UML



# Einführung in die objektorientierte Modellierung

## UML

- Objektdiagramme sind eine Spezialform der Klassendiagramme.
- Mit ihnen kann man ganz konkrete Ausprägungen von Klassen darstellen.
- Dazu werden Objekte modelliert, deren Attribute Werte enthalten.
- Die folgende Abbildung stellt die Klasse „Kunde“ zwei konkreten Objekten der Klasse „Kunde“ gegenüber.
- Objekte unterscheiden sich von Klassen dahingehend, dass
  1. jedes Objekt mit einer eindeutigen ID identifizierbar ist (hier „id1“ und „id2“), notiert links vom Klassennamen gefolgt von einem Doppelpunkt „::“.
  2. zu jedem Attribut eines Objekts ein konkreter Wert dargestellt ist. Alle Objekte einer Klasse haben dabei die gleichen Attribute, nur deren Werte können sich unterscheiden.

# Einführung in die objektorientierte Modellierung

## UML



# Einführung in die objektorientierte Modellierung

## UML

Vor der Programmierung wird in den Phasen objektorientierte Analyse (OOA) und objektorientiertes Design (OOD) sowohl das Problem als auch das Design des objektorientierten Systems modelliert. Dafür werden Klassen mit ihren Attributen und Methoden sowie Beziehungen zwischen Klassen mit UML Klassendiagrammen modelliert.

Durch die Analyse der Problemerstellung nach relevanten Substantiven werden mögliche Kandidaten für Klassen identifiziert. Anschließend werden diese Klassen um Attribute und Methoden erweitert, je nachdem wie es die Problemstellung erfordert.

Nach der Identifikation von Attributen können diese durch einen Namen, einen Datentyp, die Information, ob es sich um eine Konstante handelt, und ggf. einen Defaultwert genauer charakterisiert werden. Methoden hingegen werden durch ihren Namen, benötigte Parameter und ihren Rückgabewert beschrieben.

## Weiterführende Ziele

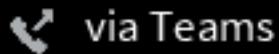
- Weiteres Arbeiten an der Modellierung der **Kaffeemaschine**
- Nutzung der OOA um die nächsten Schritte zu machen
- OOD
- OOI



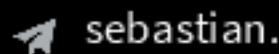
Commandline  
JavaFX

# Danke

Sebastian Bichler



via Teams



[sebastian.bicher@iu.org](mailto:sebastian.bicher@iu.org)

Sommersemester 2023

DSPOOPI01

# Objektorientierte Programmierung I

## mit Java

#05, 04.05.2023, Leipzig

Prof. (FH) Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

Analyse  
Design  
Implementierung

# Termine und Themen

<b>1</b>	<b>Kick-off (Programmierumgebung)</b>	06.04.2023	09:00 – 12:15
<b>2</b>	<b>Programmierkonzepte</b>	13.04.2023	09:00 – 12:15
<b>3</b>	<b>Einführung in die OO-Systementwicklung und -modellierung</b>	20.04.2023	09:00 – 12:15
<b>4</b>	<b>Aufgaben 7, 8 9 (Flussdiagramm, Struktogramm, UML)</b>	27.04.2023	09:00 – 12:15
<b>5</b>	<b>UML, Von OOA zu OOD und OOP: Kaffeemaschine modellieren</b>	04.05.2023	09:00 – 12:15
<b>6</b>	<b>Von UML zum Java-Code</b>	11.05.2023	09:00 – 12:15
<b>7</b>	<b>Vererbung, Schnittstellen, Generics</b>	25.5.2023	09:00 – 12:15
<b>8</b>	<b>Thema</b>	01.06.2023	09:00 – 12:15
<b>9</b>	<b>Thema</b>	08.06.2023	09:00 – 12:15
<b>10</b>	<b>Thema</b>	15.06.2023	09:00 – 12:15
<b>11</b>	<b>Thema</b>	22.06.2023	09:00 – 12:15
<b>12</b>	<b>Thema</b>	29.06.2023	09:00 – 12:15
<b>13</b>	<b>Thema</b>	06.07.2023	09:00 – 10:30

Themen werden ergeben sich peau a peau.

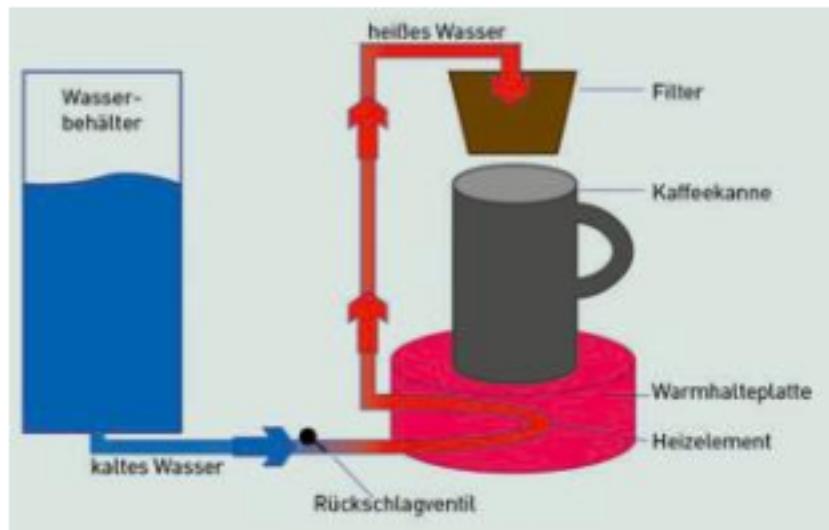
1 Warming-Up: Software-Themen, Fragen, offene Aufgaben, ...

2 Analyse

Design

Implementierung

Analyse  
Design  
Implementierung



# Warming-Up

## Themen

- Ausleihe
- Video mit zeitgesteuerter Anzeigen (Storybook)
- Erstellung eines Volumenmodells (STL: Mesh → Körper)

## Offenes (Aufgaben, Probleme, ...)

- Aufgaben 10, 11, 12
- EVA → MVC
- Literale
- Andere Fragen

1 Warming-Up: Software-Themen, Fragen, offene Aufgaben, ...

2 Analyse

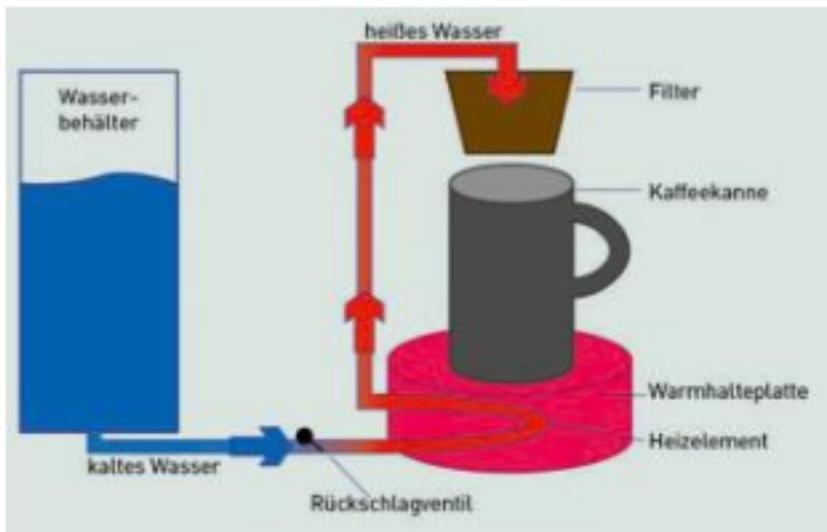
3 Design

4 Implementierung

## Analyse

## Design

## Implementierung



Commandline  
JavaFX

## Analyse

### Entwicklung einer Kaffeemaschine/Nachbildung (Aufgabe 3)

- Fortsetzung und Erweiterung der Aufgabe aus der letzten Einheit
- Ermittelt in euren Gruppen den Aufbau und Funktionsweise einer selbstgewählten Kaffeemaschinenart. Prüft Möglichkeiten, wie sich mit Vererbung und Polymorphie eventuell weitere Arten von Kaffeemaschinenarten im Modell einbeziehen lassen.
- Führt eine Mini-Anforderungsanalyse durch, um das Produkt zu beschreiben. Wodurch zeichnet sich die Kaffeemaschine aus.
- Diese OO-Analyse soll unabhängig von der Implementierung als Software geschehen.

Sommersemester 2023

DSPOOPI01

# Objektorientierte Programmierung I

## mit Java

#06, 11.05.2023, Leipzig

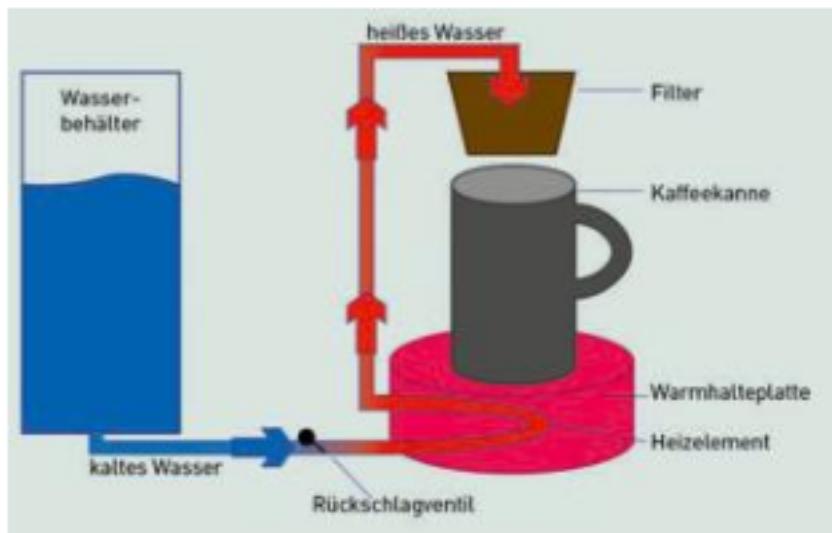
Prof. (FH) Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

Analyse  
Design  
Implementierung

- 1 Analyse: Vorstellung und Besprechung der UML-Modelle**
- 2 Design**
- 3 Implementierung/OOPs-Realisierung in Java**



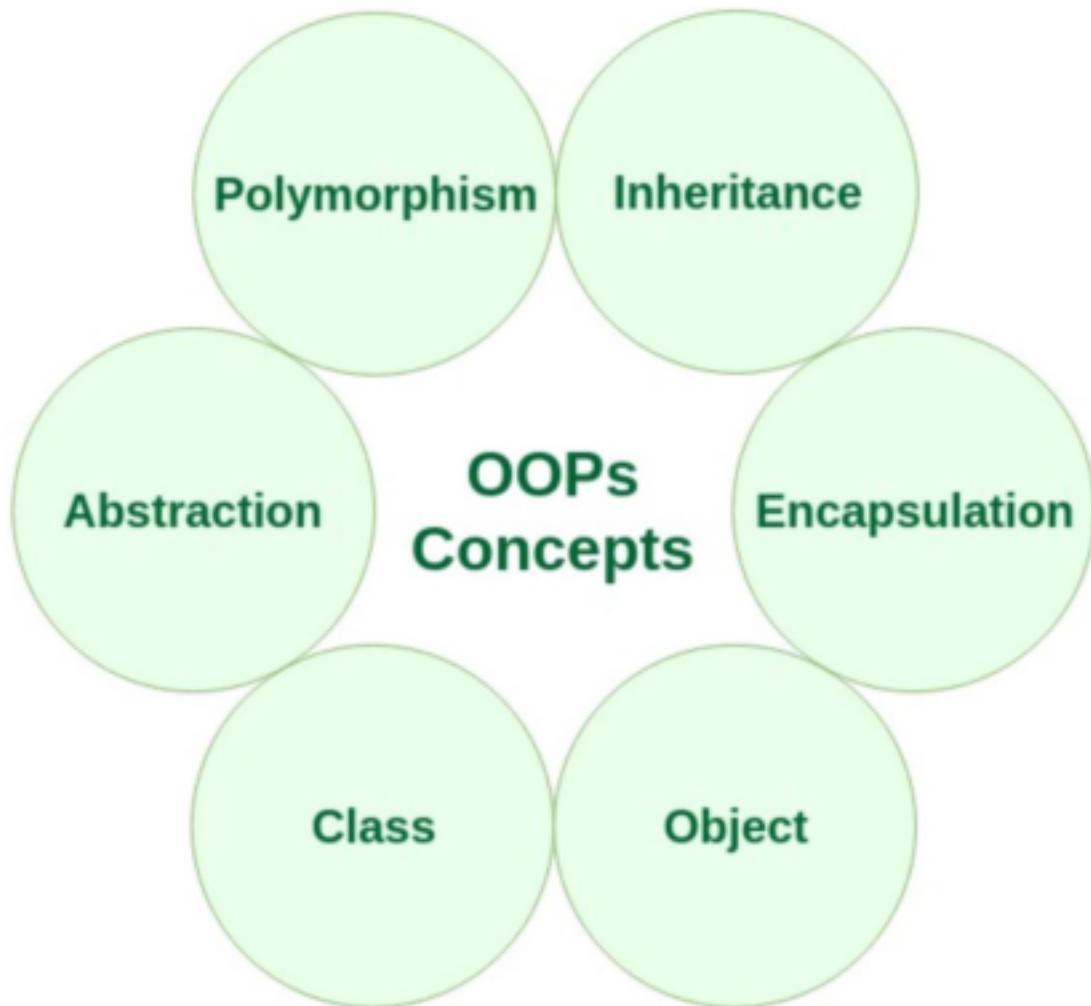
Analyse  
Design  
Implementierung



Commandline  
JavaFX

# Analyse

## Vorstellung und Besprechung der UML-Modelle (konzeptionell)



Modelmerkmale:

- Abbildungsmerkmal
- Verkürzungsmerkmal
- Pragmatisches Merkmal
- minimal?
- OOP-Prinzipien (Java) ?

# Analyse

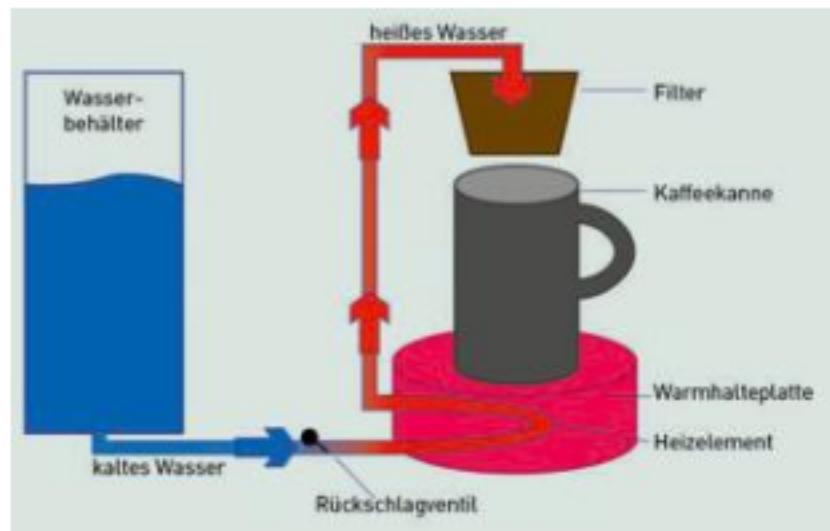
## Vorstellung und Besprechung der UML-Modelle



1 Analyse: Vorstellung und Besprechung der UML-Modelle

2 Design

3 Implementierung/OOPs-Realisierung in Java



Analyse  
Design  
Implementierung



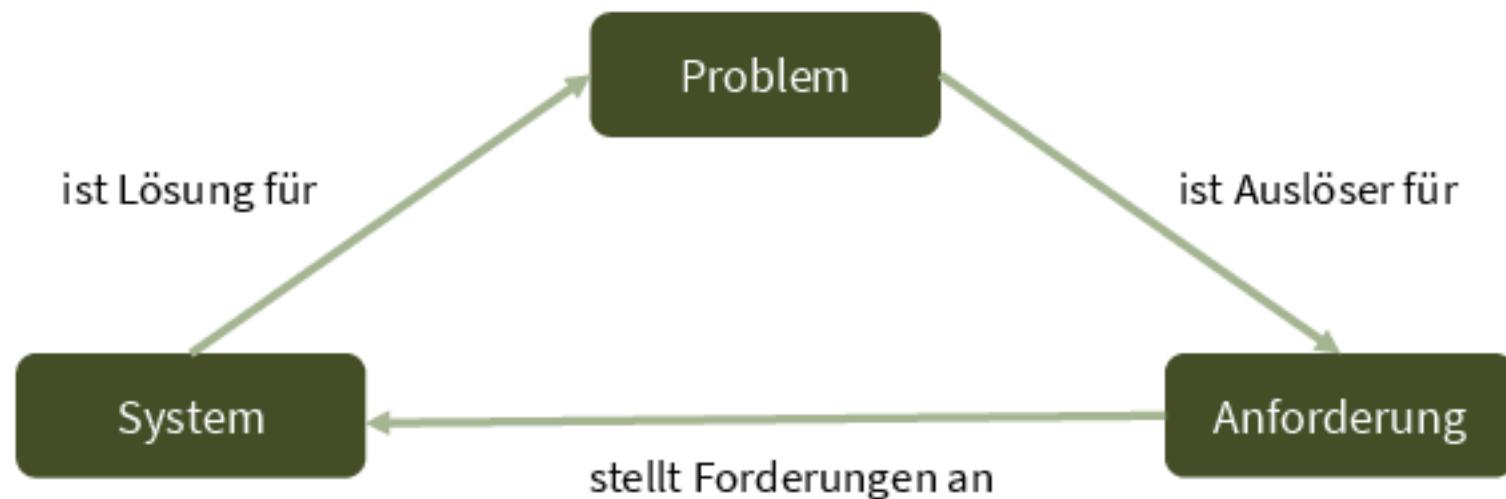
Commandline  
JavaFX

# Design

## Vorstufe der Implementierung

- Ziel/Zweck noch einmal verdeutlichen
- Probleme/Aufgaben konkretisieren und
- die daraus erwachsenen Anforderungen prüfen und
- eine dementsprechende Realisierung entwickeln/prüfen, ...

- Auszug aus Modul „Requirements Engineering“
- Achtung: Gefahr besteht sich von bestehenden Implementierungen bzw. Ideen von einem System verleiten zu lassen.
- Das *Problem* sollte zunächst erkannt werden.
- Hierfür eignen sich Kreativtechniken ...



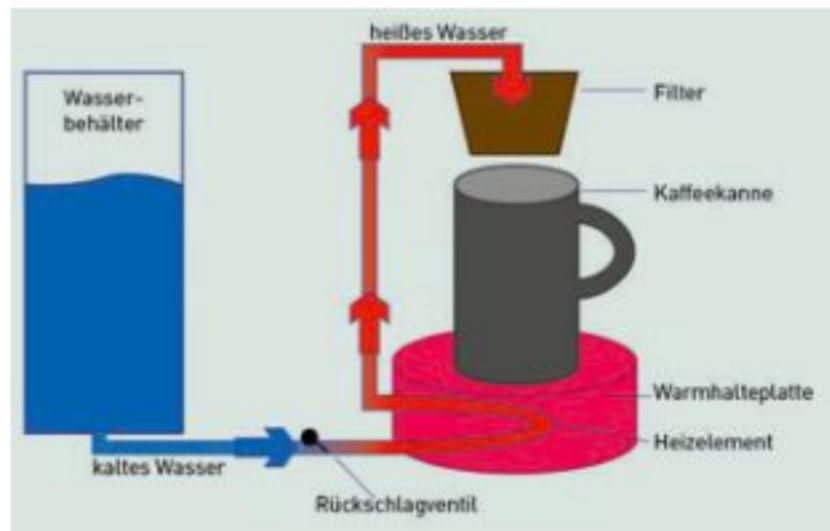
- Technische Möglichkeiten ausloten, Vor- und Nachteile in Betracht ziehen
- Adaptation vorsehen (Einschränkungen, Besonderheiten, ...)
- Entwicklung eines Modells, welches in der ausgewählten Umgebung implementiert wird
  - UML-Klassendiagramm
  - UML-Zustandsdiagramm
  - ...

# Agenda

1 Analyse: Vorstellung und Besprechung der UML-Modelle

2 Design

3 Implementierung/OOPs-Realisierung in Java



Analyse  
Design  
Implementierung



Commandline  
JavaFX

# SOLID-Prinzipien, weitere und davon abgeleitete „Design Patterns“

## Single Responsibility Principle (SRP)

Prinzip einer einzigen Verantwortung

## Open Closed Principle (OCP)

Offen für Erweiterungen, geschlossen für Veränderungen

## Liskov Substitution Principle (LSP)

Repräsentation von Objekten durch abgeleitete

## Interface Segregation Principle (ISP)

Trennung der Schnittstellen von der Implementierung

## Dependency Inversion Principle (DIP)

Umkehr der Abhängigkeiten / des Kontrollfluss

## Separation of Concerns

Trennung der Anliegen

## Don't repeat yourself

Wiederholungen vermeiden

## Principles of Testing

z.B.: F.I.R.S.T, Test Driven Development

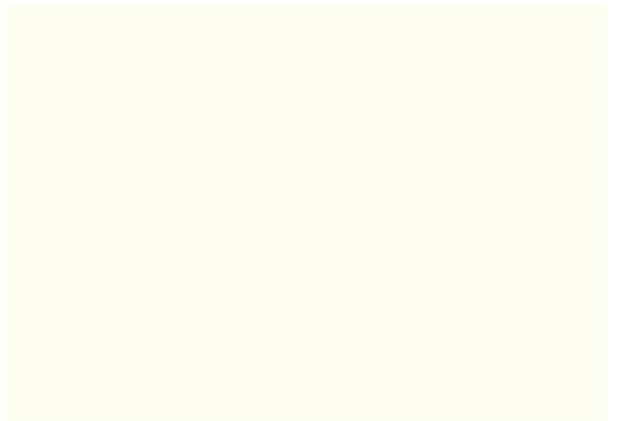
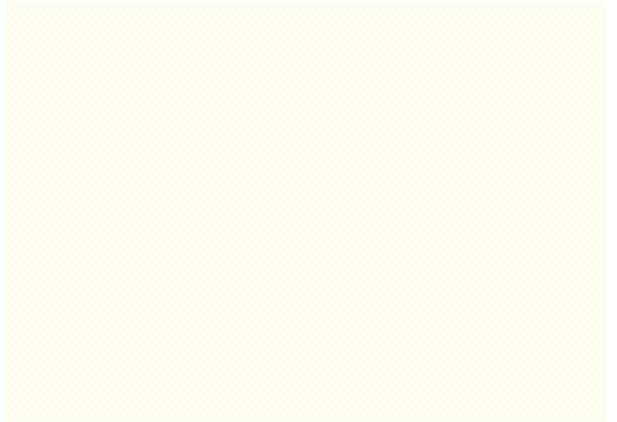
KISS, ...

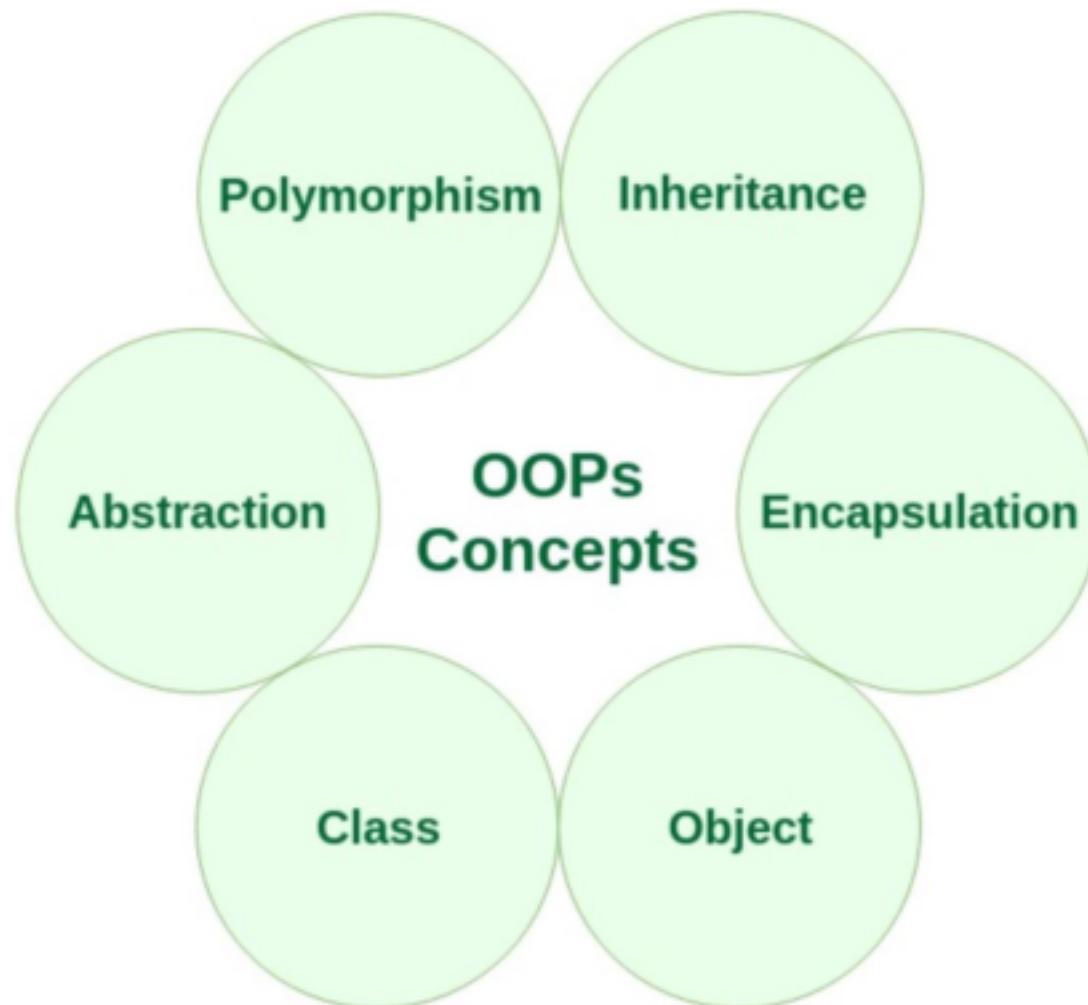
## Design Patterns

Model-View-Control, EVA, Singleton, Decorator, Factory, Observer, Strategy, ...

# Implementierung

- Zur Validierung und Verifizierung ist es vorteilhaft, wenn beide „nahe beieinander liegen“.
  - Implementierungsmodell
  - Implementierung





<https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/>

- Abstraction
  - abstract classes
  - interfaces
- Inheritance
  - Super-class
  - Sub-class: Verwendung aller Methoden von den Super-Klassen
- Polymorphismus
  - Over-Loading
  - Over-Riding
- Encapsulation / Data-hiding
  - Zugriffsteuerung: **public**, **private**, **protected**, Getter/Setter

Sommersemester 2023

DSBOOPI01

# Objektorientierte Programmierung I

## mit Java

#07, 24.05.2023, Leipzig

Prof. (FH) Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

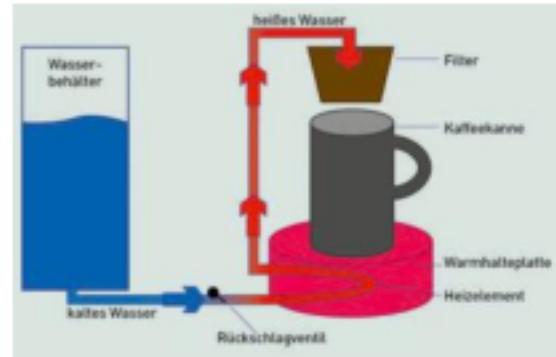
LE-BA-WIN-WiSe-22GTW

Analyse  
Design  
**Implementierung**

# Termine und Themen

1	<b>Kick-off (Programmierumgebung)</b>	06.04.2023	09:00 – 12:15
2	<b>Programmierkonzepte</b>	13.04.2023	09:00 – 12:15
3	<b>Einführung in die OO-Systementwicklung und -modellierung</b>	20.04.2023	09:00 – 12:15
4	<b>Aufgaben 7, 8 9 (Flussdiagramm, Struktogramm, UML)</b>	27.04.2023	09:00 – 12:15
5	<b>UML, Von OOA zu OOD und OOP: Kaffeemaschine modellieren</b>	04.05.2023	09:00 – 12:15
6	<b>Von UML zum Java-Code</b>	11.05.2023	09:00 – 12:15
7	<b>Software Development:</b> Vererbung, Schnittstellen, Generics, ...	25.05.2023	09:00 – 12:15
8	<b>Software-Development:</b> Stacks, Queues, ..., Properties, Bindings, Optionals, ...	01.06.2023	09:00 – 12:15
9	<b>Software-Development:</b> GUI	08.06.2023	09:00 – 12:15
10	<b>Software-Development:</b> Collections, Streams, ...	15.06.2023	09:00 – 12:15
11	<b>Software-Development:</b> Exceptions, ...	22.06.2023	09:00 – 12:15
12	<b>Software-Development:</b> (Design Patterns: Creational, Structural, Behavioral)	29.06.2023	09:00 – 12:15
13	<b>Konsultation</b>	06.07.2023	09:00 – 10:30

- UML-Class-Diagramm (Mini-Version einer Kaffeemaschine)
- Implementierung
- Lauffähiges Programm (Commandline)
  - wenn möglich mit austauschbaren Komponenten



## Implementierung

Abstraction	Inheritance (extend)	Polymorphismus	Encapsulation/ Data-hiding
<ul style="list-style-type: none"><li>• abstract classes</li><li>• interfaces</li></ul>	<ul style="list-style-type: none"><li>• <b>Superclass</b></li><li>• <b>Subclass</b></li></ul>	<ul style="list-style-type: none"><li>• Over-Loading</li><li>• Over-Riding</li></ul>	<ul style="list-style-type: none"><li>• Access</li><li>• Getter/Setter</li></ul>

SOLID: SRP – OCP – LSP – ISP – DIP ...

Sommersemester 2023

DSPOOPI01

# Objektorientierte Programmierung I

## mit Java

#08, 25.05.2023, Leipzig

Prof. Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

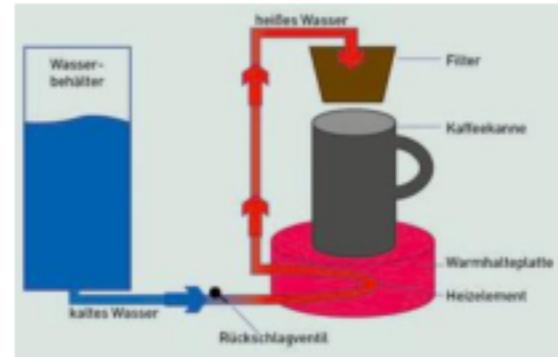
LE-BA-WIN-WiSe-22GTW

Analyse  
Design  
**Implementierung**

# Termine und Themen

<b>1</b>	<b>Kick-off (Programmierumgebung)</b>	06.04.2023	09:00 – 12:15
<b>2</b>	<b>Programmierkonzepte</b>	13.04.2023	09:00 – 12:15
<b>3</b>	<b>Einführung in die OO-Systementwicklung und -modellierung</b>	20.04.2023	09:00 – 12:15
<b>4</b>	<b>Aufgaben 7, 8 9 (Flussdiagramm, Struktogramm, UML)</b>	27.04.2023	09:00 – 12:15
<b>5</b>	<b>UML, Von OOA zu OOD und OOP: Kaffeemaschine modellieren</b>	04.05.2023	09:00 – 12:15
<b>6</b>	<b>Von UML zum Java-Code</b>	11.05.2023	09:00 – 12:15
<b>7</b>	<b>Software Development:</b> Vererbung, Schnittstellen, ...	25.05.2023	09:00 – 12:15
<b>8</b>	<b>Software-Development:</b> GUI	01.06.2023	09:00 – 12:15
<b>9</b>	<b>Software-Development:</b> Properties, Bindings, Optionals, ...	08.06.2023	09:00 – 12:15
<b>10</b>	<b>Software-Development:</b> Stacks, Queues, ..., Collections, Streams, ...	15.06.2023	09:00 – 12:15
<b>11</b>	<b>Software-Development:</b> Generics, Exceptions, ...	22.06.2023	09:00 – 12:15
<b>12</b>	<b>Software-Development:</b> (Design Patterns: Creational, Structural, Behavioral)	29.06.2023	09:00 – 12:15
<b>13</b>	<b>Konsultation</b>	06.07.2023	09:00 – 10:30

- Mini-Beispiele
- Vorstellung der Kaffeemaschinen
  - Anwenden von Prinzipen (dafür und dagegen) und Erweiterungen/Anpassungen probieren



## Implementierung

Abstraction
• abstract classes
• interfaces

Inheritance (extend)
• <b>Superclass</b>
• <b>Subclass</b>

Polymorphismus
• Over-Loading
• Over-Riding

Encapsulation/ Data-hiding
• Access
• Getter/Setter

SOLID: SRP – OCP – LSP – ISP – DIP ...

Sommersemester 2023

DSPOOPI01

# Objektorientierte Programmierung I

## mit Java

#09, 01.06.2023, Leipzig

Prof. Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

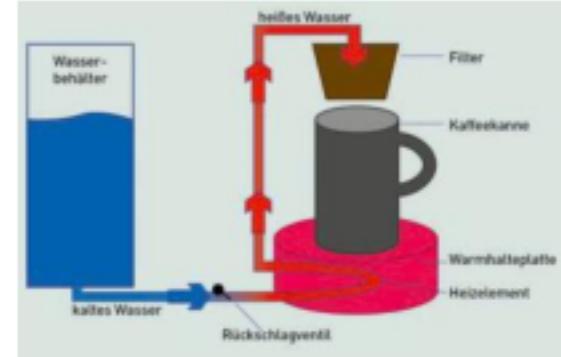
Implementierung  
**GUI**

# Termine und Themen

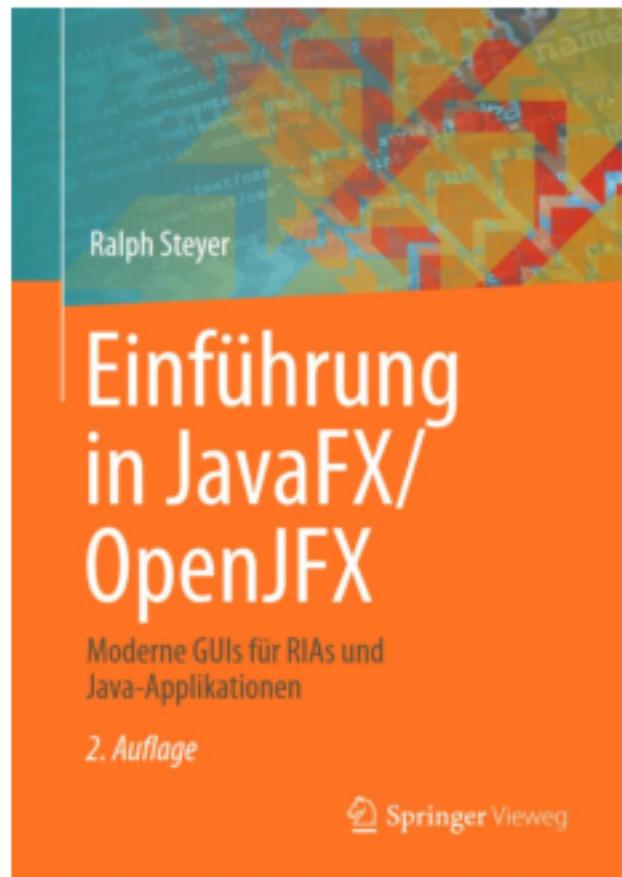
1	<b>Kick-off (Programmierumgebung)</b>	06.04.2023	09:00 – 12:15
2	<b>Programmierkonzepte</b>	13.04.2023	09:00 – 12:15
3	<b>Einführung in die OO-Systementwicklung und -modellierung</b>	20.04.2023	09:00 – 12:15
4	<b>Aufgaben 7, 8 9 (Flussdiagramm, Struktogramm, UML)</b>	27.04.2023	09:00 – 12:15
5	<b>UML, Von OOA zu OOD und OOP: Kaffeemaschine modellieren</b>	04.05.2023	09:00 – 12:15
6	<b>Von UML zum Java-Code</b>	11.05.2023	09:00 – 12:15
7	<b>Software Development:</b> Vererbung, Schnittstellen, ...	25.05.2023	09:00 – 12:15
8	<b>Software-Development:</b> GUI	01.06.2023	09:00 – 12:15
9	<b>Software-Development:</b> Properties, Bindings, Optionals, ...	08.06.2023	09:00 – 12:15
10	<b>Software-Development:</b> Stacks, Queues, ..., Collections, Streams, ...	15.06.2023	09:00 – 12:15
11	<b>Software-Development:</b> Generics, Exceptions, ...	22.06.2023	09:00 – 12:15
12	<b>Software-Development:</b> (Design Patterns: Creational, Structural, Behavioral)	29.06.2023	09:00 – 12:15
13	<b>Konsultation</b>	06.07.2023	09:00 – 10:30

# Ziele für heute

- **Kaffeemaschine** → mit Commandline und GUI-UserInterface
  - siehe Listenings
- JavaFX



**Implementierung**



Steyer, R. (2022). *Einführung in JavaFX/OpenJFX: Moderne GUIs für RIAs und Java-Applikationen*. Springer Fachmedien Wiesbaden.  
<https://doi.org/10.1007/978-3-658-35539-5>

in IU-Bibliothek



Epple, A. (2016). *JavaFX 8: Grundlagen und fortgeschrittene Techniken* (1., korrigierter Nachdruck). dpunkt-verlag.

Sommersemester 2023

DSPOOPI01

# Objektorientierte Programmierung I

## mit Java

#10, 15.06.2023, Leipzig

Prof. Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

Implementierung  
**GUI**

# Termine und Themen

<b>1</b>	<b>Kick-off (Programmierumgebung)</b>	06.04.2023	09:00 – 12:15
<b>2</b>	<b>Programmierkonzepte</b>	13.04.2023	09:00 – 12:15
<b>3</b>	<b>Einführung in die OO-Systementwicklung und -modellierung</b>	20.04.2023	09:00 – 12:15
<b>4</b>	<b>Aufgaben 7, 8 9 (Flussdiagramm, Struktogramm, UML)</b>	27.04.2023	09:00 – 12:15
<b>5</b>	<b>UML, Von OOA zu OOD und OOP: Kaffeemaschine modellieren</b>	04.05.2023	09:00 – 12:15
<b>6</b>	<b>Von UML zum Java-Code</b>	11.05.2023	09:00 – 12:15
<b>7</b>	<b>Vererbung, Schnittstellen, ...</b>	25.5.2023	09:00 – 12:15
<b>8</b>	<b>Wiederholung der Prinzipien und Implementierung</b>	01.06.2023	09:00 – 12:15
<b>9</b>	„Design Principles & Design Pattern“ Teil I	08.06.2023	09:00 – 12:15
<b>10</b>	„Design Principles & Design Pattern“ Teil 2, Mini-Beispiele	15.06.2023	09:00 – 12:15
<b>11</b>	Mini-Beispiele, Wichtige Design Patterns	22.06.2023	09:00 – 12:15
<b>12</b>	Wiederholungen, Übungen	29.06.2023	09:00 – 12:15
<b>13</b>	<b>Konsultation</b>	06.07.2023	09:00 – 10:30

Sommersemester 2023

DSPOOPI01

# Objektorientierte Programmierung I

## mit Java

#11, 22.06.2023, Leipzig

Prof. Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

Implementierung

# Termine und Themen

1	<b>Kick-off (Programmierumgebung)</b>	06.04.2023	09:00 – 12:15
2	<b>Programmierkonzepte</b>	13.04.2023	09:00 – 12:15
3	<b>Einführung in die OO-Systementwicklung und -modellierung</b>	20.04.2023	09:00 – 12:15
4	<b>Aufgaben 7, 8 9 (Flussdiagramm, Struktogramm, UML)</b>	27.04.2023	09:00 – 12:15
5	<b>UML, Von OOA zu OOD und OOP: Kaffeemaschine modellieren</b>	04.05.2023	09:00 – 12:15
6	<b>Von UML zum Java-Code</b>	11.05.2023	09:00 – 12:15
7	<b>Vererbung, Schnittstellen, ...</b>	25.5.2023	09:00 – 12:15
8	<b>Wiederholung der Prinzipien und Implementierung</b>	01.06.2023	09:00 – 12:15
9	<b>„Design Principles &amp; Design Pattern“ Teil I</b>	08.06.2023	09:00 – 12:15
10	<b>„Design Principles &amp; Design Pattern“ Teil 2</b>	15.06.2023	09:00 – 12:15
11	<b>Generics, Design Patterns, allgemeines Feedback (Klausur)</b>	22.06.2023	09:00 – 12:15
12	<b>Wiederholungen, Übungen</b>	29.06.2023	09:00 – 12:15
13	<b>Konsultation</b>	06.07.2023	09:00 – 10:30

# Agenda

- Durchsehen der letzten Versionen der Kaffeemaschinen mit Überarbeitung durch *Interfaces* und *abstrakten Klassen*
- Weitere Beispiele in Java
  - Generics, ...
  - Design Patterns
- Statusabfrage/Feedback
- Fragen und Diskussionen

# Überarbeitete Ansätze

- Kommunikation der Klassen
- Austausch von Informationen
- Wie
- Gemeinsame Variablen, Zustände, ...
  - Power ist in allen Modulen enthalten:
- Lösungen mit Vor- und Nachteilen:
  - Gemeinsame Variable in Hauptklasse (PowerInterface triggert)
  - Wrapper für alle Klassen, um eine gemeinsame Variable zu verwalten, z.B. mit gemeinsamer abstrakter Klasse
  - Bus-system mit Optionals, Properties, Bindings, ...

# Generics

## Generische Typen, Methoden, Klassen und Interfaces

- „Klassen, Interfaces und Methoden können mit Hilfe von formalen Typparametern (Platzhaltern) implementiert werden.
- Der Typparameter repräsentiert zum Zeitpunkt der Implementierung noch einen unbekannten Typ. Man definiert also Schablonen, die erst durch Angabe von konkreten Typen bei ihrer Verwendung zu normalen Klassen, Interfaces bzw. Methoden ausgeprägt werden.
- Diese Möglichkeit nennt man Generizität [bzw. Generics].
- Generics werden in Java ausschließlich vom Compiler verarbeitet. Das Laufzeitsystem (JVM) arbeitet weiterhin mit „normalen“ Klassen und Interfaces.“

# Generics

## Generische Typen, Methoden, Klassen, Interfaces

Beispiele:

- class A<**T**> oder class A<**T, U**>,
- class A<**T**> implements ExampleInterface<**T**>
- **T** attribute;
- void **T** method() { ... }
- class A <**T extends AnyClass**>
  - Typ-Bound/Kompatibilität: T is subclass of AnyClass, d.h. class A<**T extends Object**> ist == class A<**T**>
  - Mehrfache Typebounds sind möglich: T extends Type1 & Type2 & Type3 & ...
- Raw Types (Abwärtskompatibilität)
- Wildcards:
  - class A<?**>**
  - class A<? **extends B**>
  - class A<? **extends K1 & I1 & I2**> (gilt für Klassen und Interfaces)

- TODO1: Realisiere Methoden zur Berechnung bspw. von Addition, Subtraktion, Multiplikation und Division – zunächst für einige Datentypen wie Integer und Double.
- TODO2: Übertrage TODO1 in eine generalisierte bzw. allgemeine Form → Generics.

# Generics

## Generische Typen, Methoden, Klassen, Interfaces

### Invarianz

Aus "A ist kompatibel zu B" folgt *nicht* "C<A> ist kompatibel zu C<B>".

Die Kompatibilität der Typargumente überträgt sich also *nicht* auf die parametrisierten Typen (*Invarianz*).

So ist beispielsweise die Zuweisung

```
Box<Number> box = new Box<Integer>();
```

*nicht* möglich.

Damit wird verhindert, dass Objekte, die nicht vom Typ Integer sind, durch bloße "Umetikettierung" der Box hinzugefügt werden können.

Bei Arrays ist das Verhalten jedoch anders:

Integer ist kompatibel zu Number, ebenso ist ein Integer-Array kompatibel zu einem Number-Array.

Beispiel:

```
Number[] a = new Integer[1];  
a[0] = 3.14;
```

Dieser Code lässt sich fehlerfrei compilieren. Erst zur Laufzeit wird die Ausnahme ArrayStoreException ausgelöst. Arrays überprüfen die Elementtypen zur Laufzeit.

# Design Patterns

## Creational

- Singleton
- Factory
- Builder

## Structural

- Adapter
- Decorator
- Composite

## Behavioral

- Observable
- Iterator
- Strategy

■ Dies ist nur ein Auszug.

- „Ein Singleton ist eine Klasse, von der es während der Laufzeit der Anwendung nur eine einzige Instanz gibt.“
- Singletons werden benutzt, um beispielsweise teure Ressourcen garantiert nur einmal zu erzeugen.“
- Das Wesentliche:
  - Der Konstruktor eines Singletons muss private sein, damit von außen keine Instanzen erzeugt werden können.
  - Die einzige Instanz des Singletons ist in einem öffentlichen und statischen Attribut gespeichert, um den Zugriff von außen zu gewähren.

- Konstruktion eines bestimmten Objekts nicht über den spezielle Konstruktor,
- sondern über eine Methode, die entscheidet, welche konkrete dafür verwendet wird.
- Ziel: Erzeugungslogik verbergen.
- Einfachste Variante ist ein Switch- bzw. If-Anweisung ...
- Die *Factory* entscheidet und leitet den konkreten Typen anhand der gegebenen Umstände ab.

- Programmiere eine „bessere“ Methode, um anhand von bspw. Enumerations Typen von Kaffeemaschinen mit bestimmten Konfigurationen anzulegen.

# **Statusabfrage/Feedback**

# Statusabfrage/Feedback

- Java Basics ?
- Programmierkonzepte/-paradigmen?
- OOP Basics ?
- OOP Anwendung ? (interfaces, abstract classes, ...)?

# Statusabfrage/Feedback

## Offene Themen bzw. für OOPII

- Collections
- Streams
- Lambda Ausdrücke/Anonyme Funktionen
- Optionals
- Dateien, Ordner
- Datenströme
- Serialisierung
- Threads/Nebenläufigkeit
- Process-API
- GUI

Sommersemester 2023

DSPOOPI01

# Objektorientierte Programmierung I

## mit Java

#12, 29.06.2023, Leipzig

Prof. Dr. Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

Recap

# Termine und Themen

<b>1</b>	<b>Kick-off (Programmierumgebung)</b>	06.04.2023	09:00 – 12:15
<b>2</b>	<b>Programmierkonzepte</b>	13.04.2023	09:00 – 12:15
<b>3</b>	<b>Einführung in die OO-Systementwicklung und -modellierung</b>	20.04.2023	09:00 – 12:15
<b>4</b>	<b>Aufgaben 7, 8 9 (Flussdiagramm, Struktogramm, UML)</b>	27.04.2023	09:00 – 12:15
<b>5</b>	<b>UML, Von OOA zu OOD und OOP: Kaffeemaschine modellieren</b>	04.05.2023	09:00 – 12:15
<b>6</b>	<b>Von UML zum Java-Code</b>	11.05.2023	09:00 – 12:15
<b>7</b>	<b>Vererbung, Schnittstellen, ...</b>	25.5.2023	09:00 – 12:15
<b>8</b>	<b>Wiederholung der Prinzipien und Implementierung</b>	01.06.2023	09:00 – 12:15
<b>9</b>	<b>„Design Principles &amp; Design Pattern“ Teil I</b>	08.06.2023	09:00 – 12:15
<b>10</b>	<b>„Design Principles &amp; Design Pattern“ Teil 2</b>	15.06.2023	09:00 – 12:15
<b>11</b>	<b>Generics, Design Patterns, allgemeines Feedback (Klausur)</b>	22.06.2023	09:00 – 12:15
<b>12</b>	<b>Wiederholungen, Übungen</b>	29.06.2023	09:00 – 12:15
<b>13</b>	<b>Konsultation</b>	06.07.2023	09:00 – 10:30

# Agenda

- Statusabfrage
- Design Pattern: Factory
- Übungen: SOLID

# Statusabfrage

■ ...

# Design Pattern: Factory

- Programmiere eine „bessere“ Methode, um anhand von bspw. Enumerations Typen von Kaffeemaschinen mit bestimmten Konfigurationen anzulegen ...

**SOLID**

**Abts - Grundlagen der Javaprog. Kap. 32**