Didaktische Reduktion 4

Objektorientierte Modellierung und Programmierung in der Mittelstufe des G8

Inhalt des Vortrags

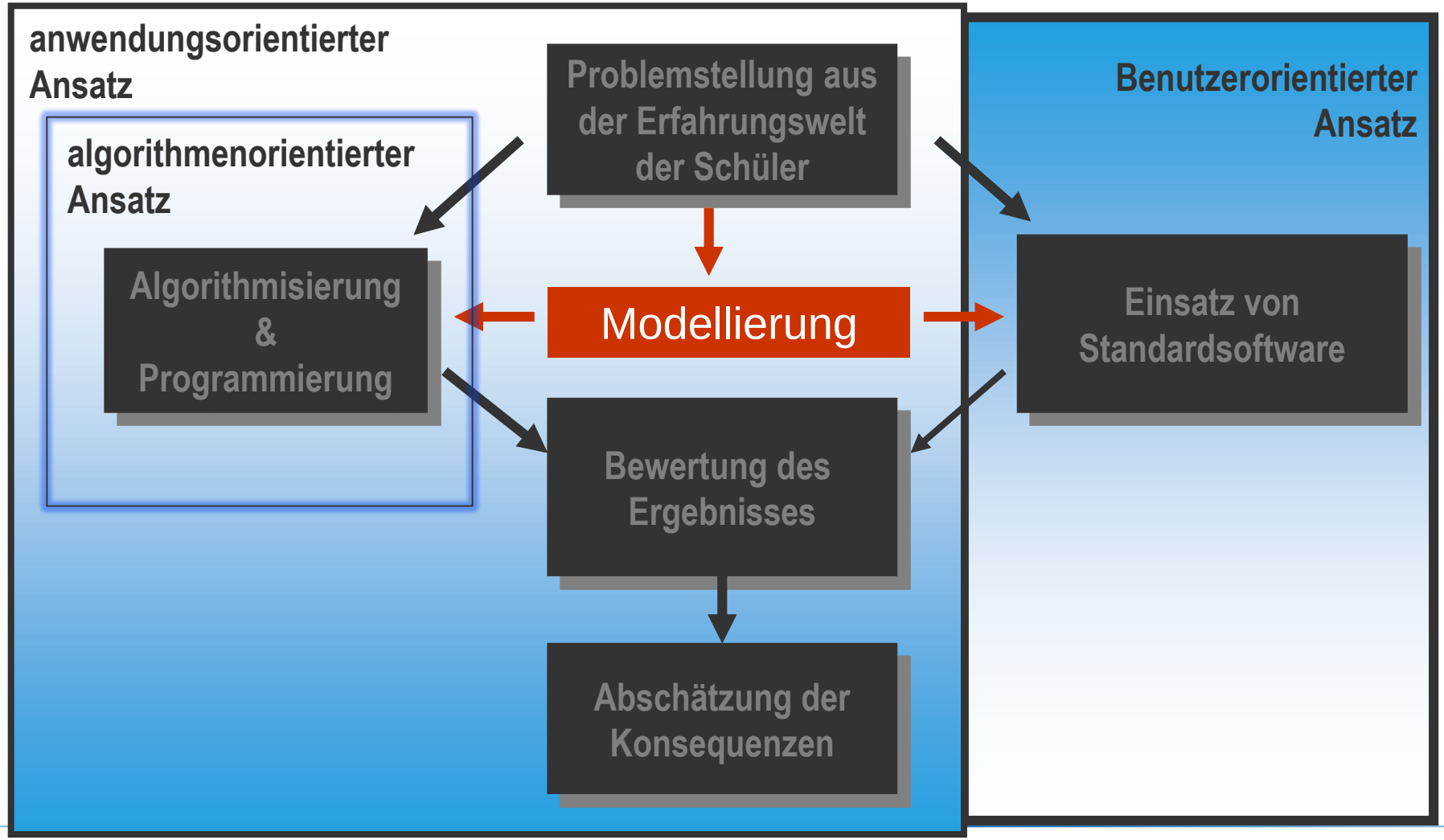
1. Organisatorischer Rahmen und Methodisches Prinzip
2. Was ist Objektorientierung (OO); „Objects first“ – „objects later“
3. Lernzielanalyse des „ersten“ OO-Programms
4. Von OOM zu OOP

OOM/OOP in der Mittelstufe

Jgst.	alle Gym.	nat.-techn. Gym	Neuer Lehrplan G9 ab Schuljahr
6	Kombinationsfach „Natur und Technik“ je 1 WS Informatik		2018/19
7			2019/20
8			2020/21
9	Wahlpflichtfach	Pflichtfach	2021/22
10	Wahlpflichtfach	Pflichtfach	2022/23
11	Pflichtfach	Pflichtfach	2023/24
12/13	Kursfach (nat. wiss. Bereich)	Kursfach (nat. wiss. Bereich)	2024/25

Das Unterrichtsprinzip: Synthese didaktischer Ansätze

[Hubwieser, 2001]



Übung 1:
Welche Begriffe verbinden Sie mit Objektorientierung?

Schreiben Sie 6 Fachkonzepte bzw. Begriffe auf je eine Karte, die Sie für besonders wichtig in diesem Zusammenhang halten.

Was ist Objektorientierung (OO)?

[Armstrong 2006]

- Untersuchung von 239 einschlägigen Quellen: „journals, trade magazines, books, and conference proceedings“
- Davon behaupteten 88, dass eine spezielle Menge von Konzepten den OO-Ansatz charakterisieren würden
- Diese wurden als Stichprobe ausgewählt
- Es wurden insgesamt 39 Konzepte erwähnt.
- Rangliste nach Anzahl der Erwähnungen
- Dargestellt sind die 21 Konzepte mit mind. 4 Erwähnungen (4,5%).

Concept	Count	Percentage
Inheritance	71	81%
Object	69	78%
Class	62	71%
Encapsulation	55	63%
Method	50	57%
Message Passing	49	56%
Polymorphism	47	53%
Abstraction	45	51%
Instantiation	31	35%
Attribute	29	33%
Information Hiding	28	32%
Dynamic Binding	13	15%
Relationship	12	14%
Interaction	10	12%
Class Hierarchy	9	10%
Abstract Data Type	7	8%
Object-Identity Independence	6	7%
Collaboration	5	6%
Aggregation	4	5%
Association	4	5%
Object Model	4	5%

Taxonomie der Objektorientierung

[Armstrong 2006]

Construct	Concept	Definition
Structure	Abstraction	Creating classes to simplify aspects of reality using distinctions inherent to the problem.
	Class	A description of the organization and actions shared by one or more similar objects.
	Encapsulation	Designing classes and objects to restrict access to the data and behavior by defining a limited set of messages that an object can receive.
	Inheritance	The data and behavior of one class is included in or used as the basis for another class.
	Object	An individual, identifiable item, either real or abstract, which contains data about itself and the descriptions of its manipulations of the data.
Behavior	Message Passing	An object sends data to another object or asks another object to invoke a method.
	Method	A way to access, set, or manipulate an object's information.
	Polymorphism	Different classes may respond to the same message and each implement it appropriately.

Unsere Sicht: Objektorientierter vs. prozeduraler Programmierstil

	Prozedural (z.B. Pascal)	**Objektorientiert (z. B. Java)**
Prinzip	Zentrale Prozedur steuert alles	Objekte agieren soweit möglich selbstverantwortlich
Datensätze	Records (Verbunde) mit offenen Daten	Objekte mit gekapselten Daten
Operationen	Records werden verändert	Objekte führen Methoden aus
Implementierung von Varianten	über Diskriminatorvariable: IF Art == „Manager“ THEN machMal()	über Unterklassen mit polymorphen Methoden: Das Objekt weiß, zu welcher Unterklasse es gehört und handelt danach

Warum OO im Informatikunterricht?

[Schwill 1995]

„Wir favorisieren für die Einführung in die Informatik den objektorientierten Ansatz vor allem [..]

1. Erweiterbarkeit, Anpassbarkeit, Rekonfiguration, Vererbbarkeit, Kapselung, evolutionäre Softwareentwicklung Anwendungsorientierung
2. Fortsetzbarkeit: nach Spiralprinzip auf höherem Niveau beliebig ausbaufähig,
3. ordnet sich harmonisch den elementaren kognitiven Prozessen unter, die beim Denken, Erkennen und Problemlösen im menschlichen Gehirn ablaufen (Schemata, Kategorien, funktionale Gebundenheit).“

Unsere Begründung für OO im Unterricht

- OO-Modellierung von Standardsoftware ermöglicht deren Abstraktion auf eine gymnasiale Ebene
- Authentizität (verbreiteter Einsatz in der professionellen Softwareindustrie)
- Nähe zur Erfahrungswelt
- zwangsläufige Betonung der Modellierung
- ausgereifte Modellierungstechniken (UML)
  - viele Sichten
  - ausgereifte Syntax und Semantik
  - viele bewährte Werkzeuge
- ermöglicht Integration aller anderen ausgewählten Themenbereiche

OO als Rahmen für andere Themenbereiche

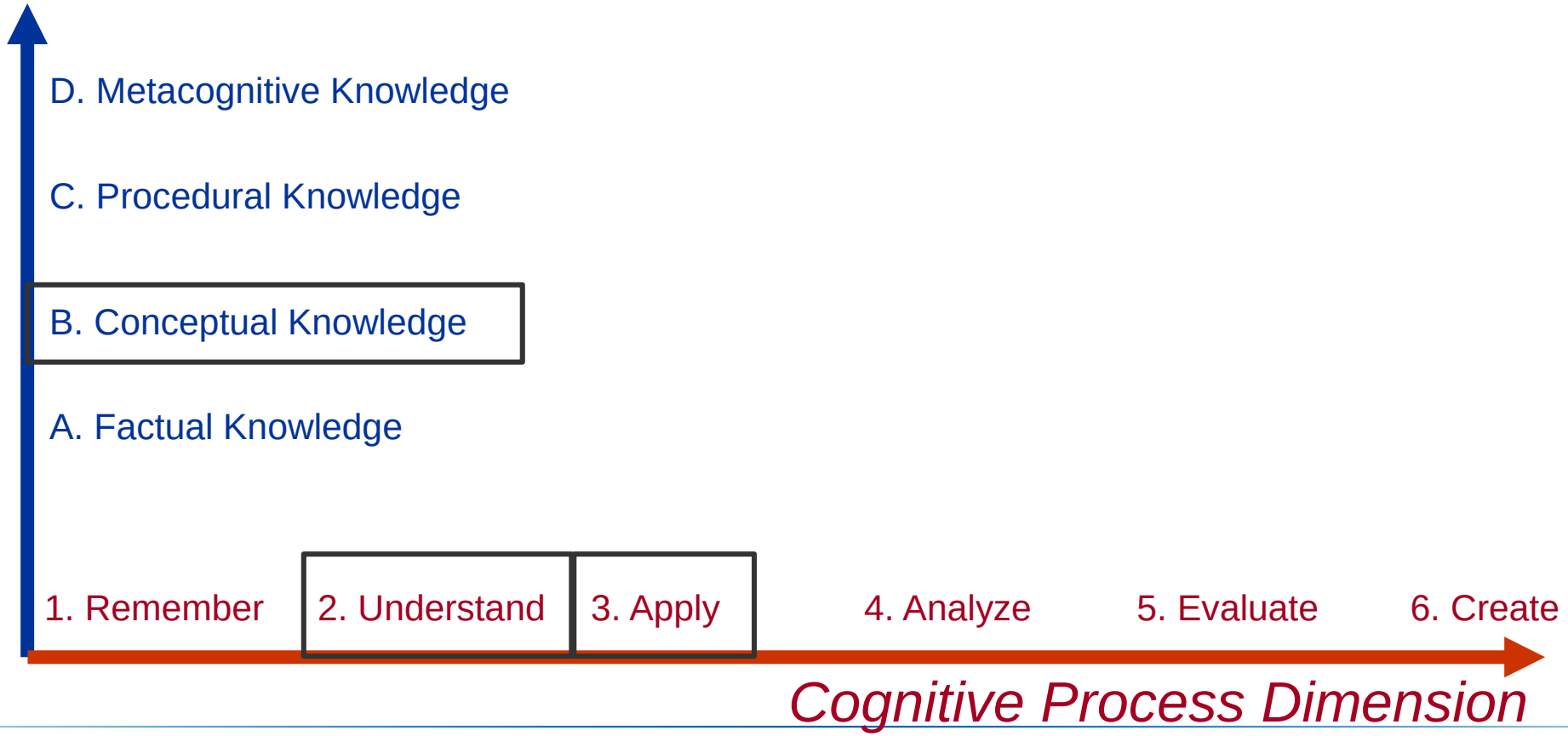
Themenbereich	OO-Sicht	Beispiele, Werkzeuge
Umgang mit Standardsoftware	Klassenmodelle von Datenstrukturen	Grafiken, Texte, Hypertexte, Präsentationen
Algorithmen	Ablaufmodellierung von Methoden (state, activity)	Karol, Scratch, etc.
Funktionen	Funktionale Methoden {.. return <Wert>}	Tabellenkalkulation
Datenbanken	Datensätze ↔ Objekte Tabellen ↔ Klassen	Relationale Datenbanksysteme, SQL
Netzwerktechnik, Internet	Nebenläufige Kooperation von Objekten über Protokolle	E-Mail Versand mit Pufferung, Client-Server
Formale Sprachen	Syntax von Programmiersprachen	Java-Syntax in BNF
Maschinen-programmierung	Maschine als Objekt und Zustandsautomat	Klasse REGISTERMASCHINE

Didaktisches Dilemma des ersten OO-Programms

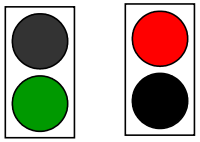
[Hubwieser 2007]

- EINERSEITS verlangen moderne konstruktivistische Lernansätze den Einstieg:
  - über Aufgabenstellungen aus authentischen Anwendungsbereichen mit realistischer Komplexität
  - mit Bezug zur Erfahrungswelt der Schülerinnen und Schüler
  - die zeigen, dass die zu lernenden Konzepte tatsächlich Relevanz im Alltag haben
  - **also keine „Hello World“-Programme**!
- ANDERERSEITS verlangt ein solcher Einstieg, dass die Schülerinnen und Schüler
  - viele schwierige Konzepte
  - zur gleichen Zeit lernen müssen!

Lernzieltaxonomie von Anderson/Krathwohl 2001



„Einfache Klasse“: 2-Licht-Ampel



```
1     import java.awt.*;
2
3     public class Traffic2Lights {
4        private Rectang body;
5        private Circle topLight;
6
7        …
8        public Traffic2Lights() {
9        }
10       public void draw() {
11            body = new Rectang();
12            body.changeSize(50, 105);
13            body.changeColor("black");
14            body.makeVisible();
15            topLight = new Circle(); …
16       }
⋮        ⋮
40    }
```

Übung 2:
Lernziele eines „relevanten“ OO-Programms

Sie haben eine Liste mit Befehlen eines OO-Programms vor sich.

1. Teilen Sie die Zeilen dieses Programms gleichmäßig untereinander auf.
2. Stellen Sie fest, welche Lernziele der Struktur <*Apply – Conceptual Knowledge*> Ihre Schülerinnen und Schüler erreicht haben müssen, um die Ihnen zugeteilten Programmzeilen verstehen und implementieren zu können.
3. Notieren Sie alle Wissenselemente dieser Lernziele in der jeweiligen Zeile des Programms

Weitere Lernziele für die ganze Einheit

Zusätzlich zu den o.g. offensichtlichen Lernzielen ist für das Verständnis von Algorithmen und die Arbeit mit Programmen generell noch die Erreichung einiger weiterer Lernziele notwendig, die auch für viele ähnliche Aufgabenstellungen vorausgesetzt werden müssen (daher “global”):

- G1: Algorithmuskonzept: Darstellung, Ein- und Ausgabe, Strukturelemente
- G2: Benutzung der Programmierumgebung
- G3: Zustände eines Programms: *codiert – übersetzt – ablaufend*
- G4: Beschreibungstechniken für die Syntax einer künstlichen Sprache, z.B. Syntaxdiagramme oder BNF
- G5: Zustandssemantik imperativer Programme

Implizite Lernziele

Drittens gibt es Lernziele, die als implizite Voraussetzung für die o.g. angesehen werden müssen, z.B.:

- I1: Um das Referenzkonzept zu verstehen (L8), muss man wissen, dass Programme zusammen mit Daten im Arbeitsspeicher des Rechners liegen (“von Neumann Prinzip”).
- I2: Für die Arbeit mit booleschen Ausdrücken (L12), z.B. in Bedingungen (L11), müssen Grundlagen der Aussagenlogik vorausgesetzt werden, z.B. die UND/ODER-Operatoren.

Lernzielbilanz

- für die Klasse 2-Ampel:
  - explizite offensichtliche Lernziele: L1 .. L19
  - globale Lernziele: G1 .. G5
  - implizite Lernziele: I1 .. I2

  => Mehr als 26 Lernziele für das Verständnis eines Programms!

- Wie kann man so viele Lernziele auf einmal erreichen?

  GAR NICHT!

- Lösung: Wir bauen auf der Vorarbeit von drei Schuljahren (Jgst. 6, 7, 9) auf!

Lösung: Schrittweise Vorbereitung von OOP

Jgst.	Thema	Locale LZ	Globale LZ
6	Vektorgrafik	L2, L3, L7, L9, L10	
	Texte	L4	
7	Hypertexte	L8	
	Roboter und Algorithmen	L6, L11, L14, L18	G1, G4, G5
9/10	Funktionale Modellierung und Tabellenkalkulationen	L12, L13	
9/10	OOM/OOP	L1, L5, L16, L17, L19	G2, G3, G5

Dennoch bleibt der Weg zum ersten OO-Programm lang und schwierig!

Das “objects first“-Konzept

[Lewis 2000]

A distinction must quickly be made between

- initially *writing classes that define objects,*
- *and using objects* defined by preexisting classes.

It is sometimes suggested that if students do not write multiple classes and methods initially, they are not being indoctrinated into an object oriented approach. Most educators agree, however, that **using objects is the appropriate first step**.

„Objects first“ – „objects later“

[Ehlert/Schulte 2009]

OOP-first	OOP-later
1. Classes and objects 2. Attributes (incl. data types) 3. Methods (including control structures) 4. Inheritance 5. Association	1. Data types (incl. Variables, constants) 2. Control structures 3. Procedures 4. Classes and objects 5. Inheritance and association

Figure 1: Typical teaching sequences for OOP-first and OOP-later

Die Stufen unseres OO-Konzeptes

[Hubwieser 2006]

1. OO-Modellierung von Datenstrukturen, Hard- bzw. Softwarekomponenten; Simulation mit Standardsoftware

2. Modellierung von „realen“ Systemen aus der Erfahrungswelt der Schülerinnen und Schüler; Simulation mit Tabellenkalkulationen bzw. Datenbanksystemen

3. OO-Programmierung; Implementierung mit OO-Programmiersprache (meist Java)

1. Stufe:

OO-Modellierung von Datenstrukturen, Hard- bzw. Softwarekomponenten; Simulation mit Standardsoftware

6./7. Jahrgangsstufe

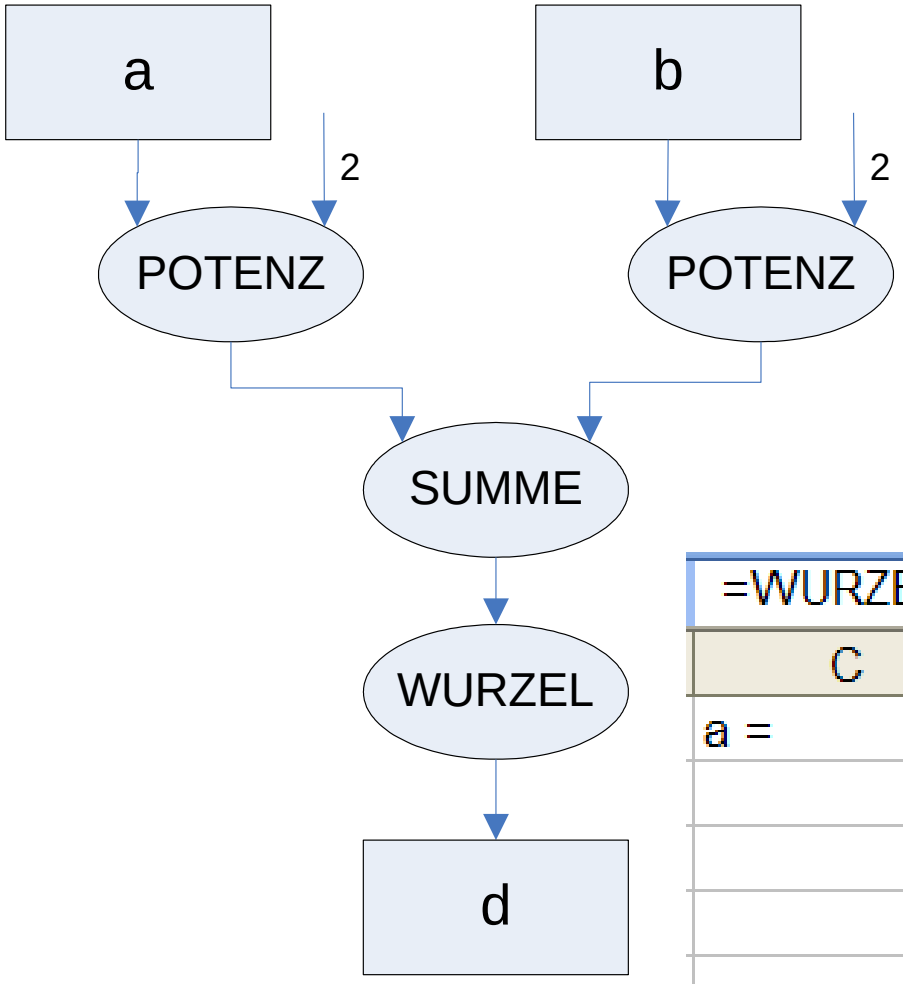
Siehe Vortrag Didaktische Reduktion, Teil 1: Objektorientierte Modellierung von Standardsoftware und Dokumenten

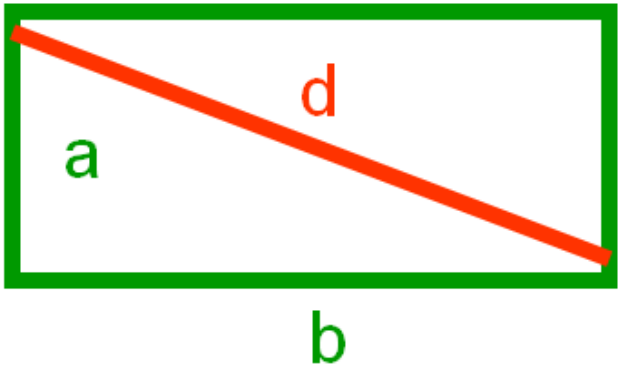
2. Stufe:

Modellierung von „realen“ Systemen aus der Erfahrungswelt

9.-10. Jahrgangsstufe

Vorarbeit: Funktionen in Rechenblättern, Inf9 Lernbereich 1

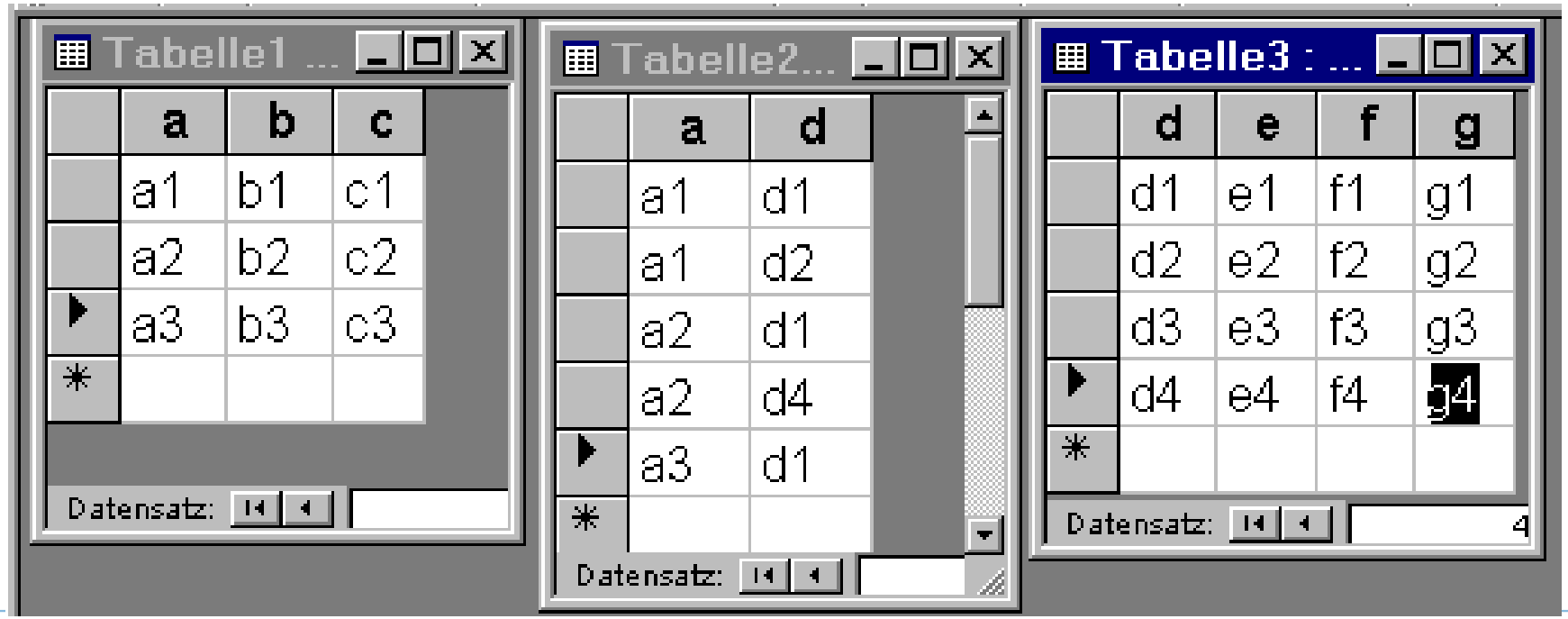




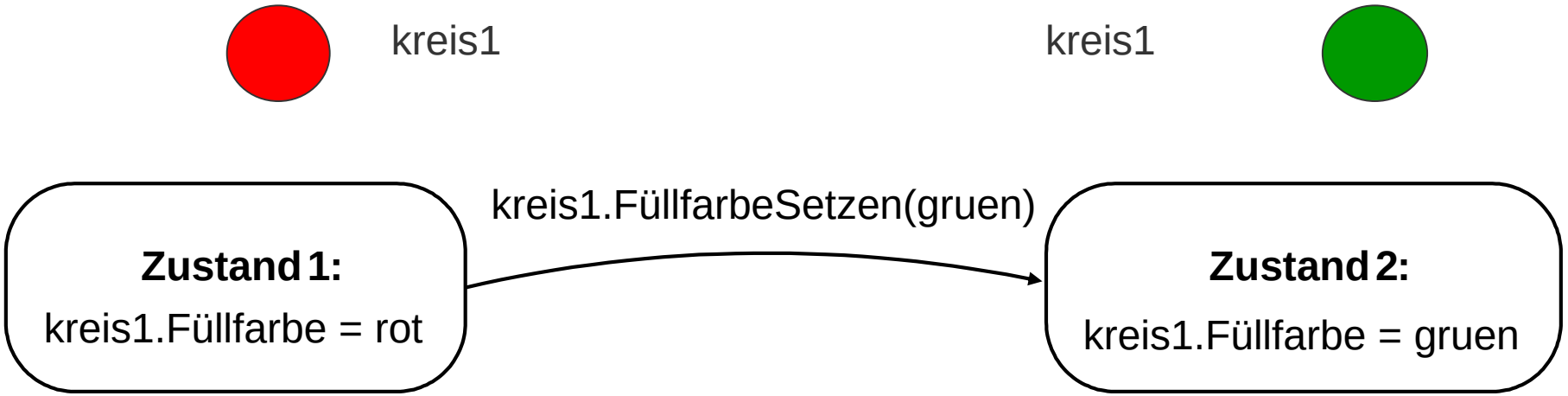
=WURZEL(SUMME(POTENZ(C2;2);POTENZ(D2;2)))

C	D	E	F
a =	b =		
3	4		
	d=		
	5		

Vorarbeit: Objekte, Klassen und Beziehungen in Datenbanktabellen in Inf10 Lernbereich 1



Inf9 Lernbereich 3: Zustände von Objekten



- Durch die Veränderung des Wertes eines Attributes kann man den Zustand eines Objektes ändern.
- Diese Zustandsänderung kann
  - durch unmittelbare Zuweisung eines neuen Wertes an das Attribut (mittels einer „Setzen“-Methode) oder
  - durch indirekte Wertzuweisung durch eine andere Methode ausgelöst werden (z.B. Änderung des Attributes „Radius“ durch die Methode „Vergrößern“.

Zustände und Attribute

- Definition: Der Zustand eines Objektes wird durch die Gesamtheit der Werte aller seiner Attribute definiert, d.h.:
  - Ein Objekt ändert seinen Zustand, wenn sich der Wert mindestens eines seiner Attribute ändert.
  - Zwei Objekte der gleichen Klasse sind im gleichen Zustand, wenn sie in den Werten aller ihrer Attribute übereinstimmen.

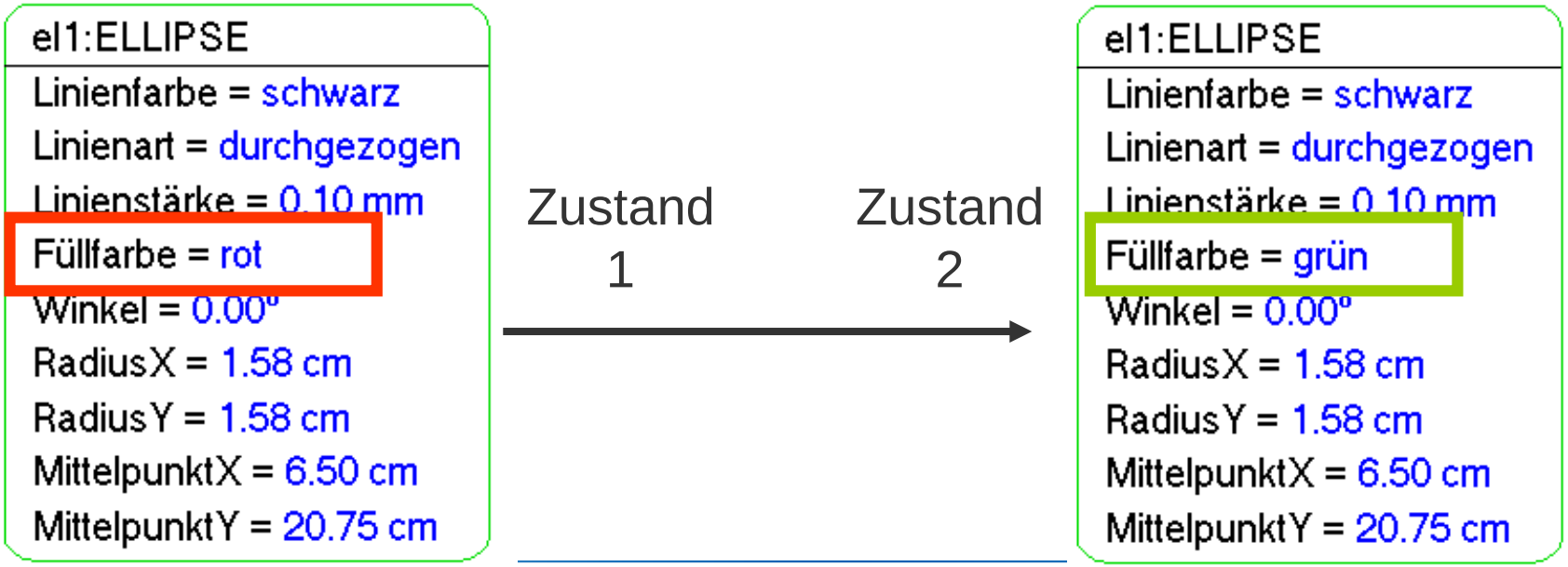
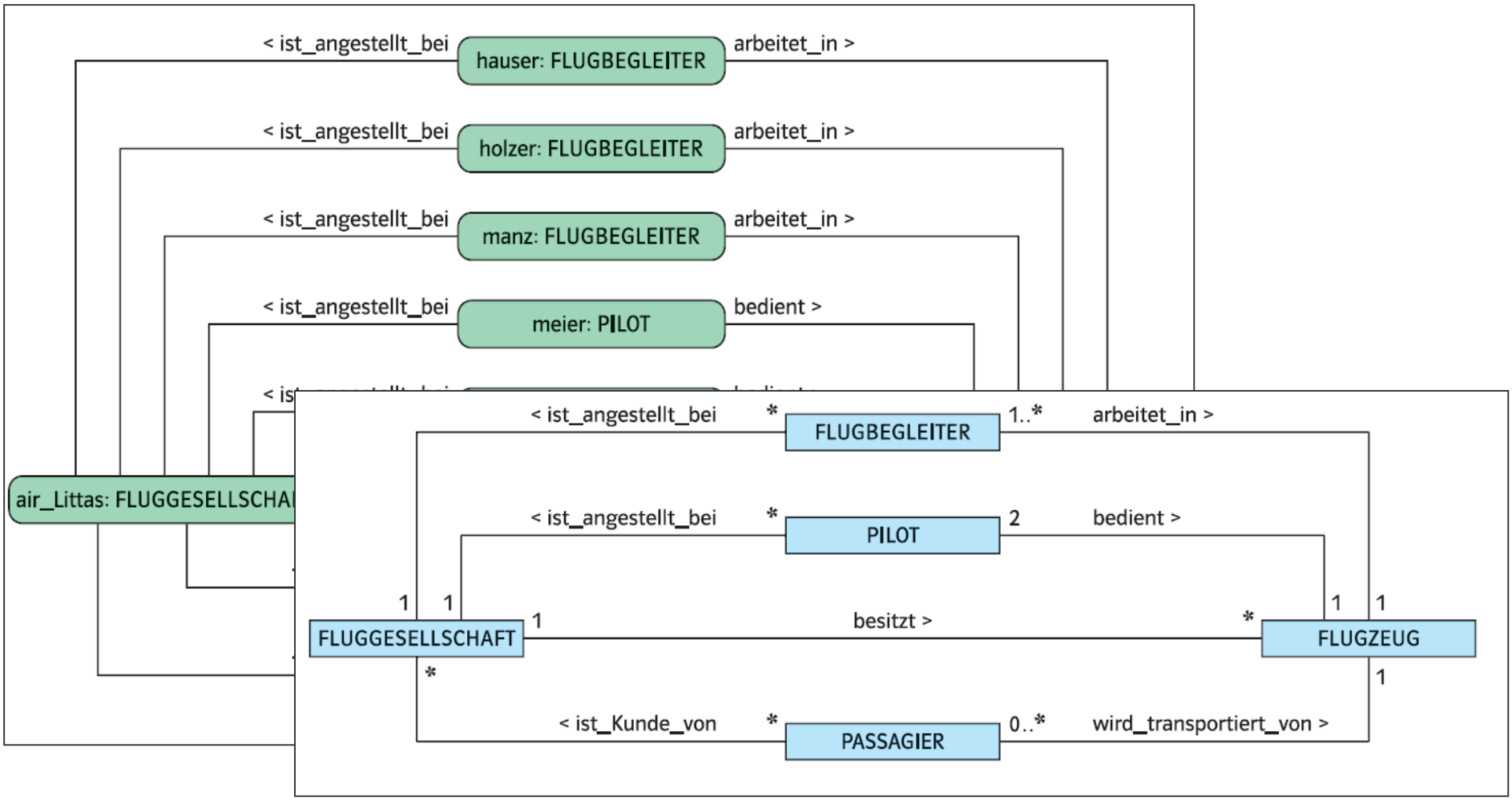


Diagramme und Beziehungen



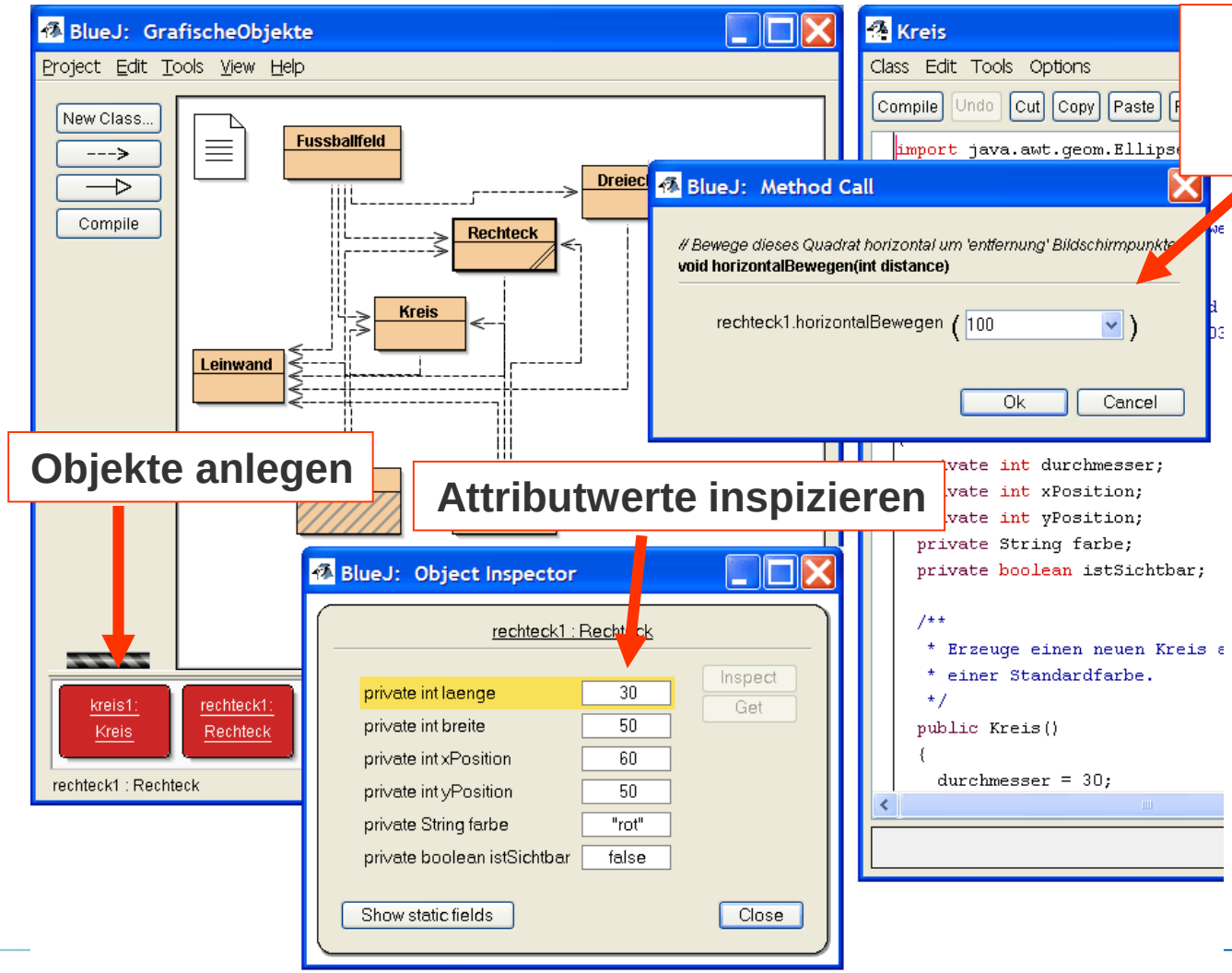
3. Stufe:

OO-Programmierung

9. Jahrgangsstufe

Werkzeug: BlueJ

www.bluej.org



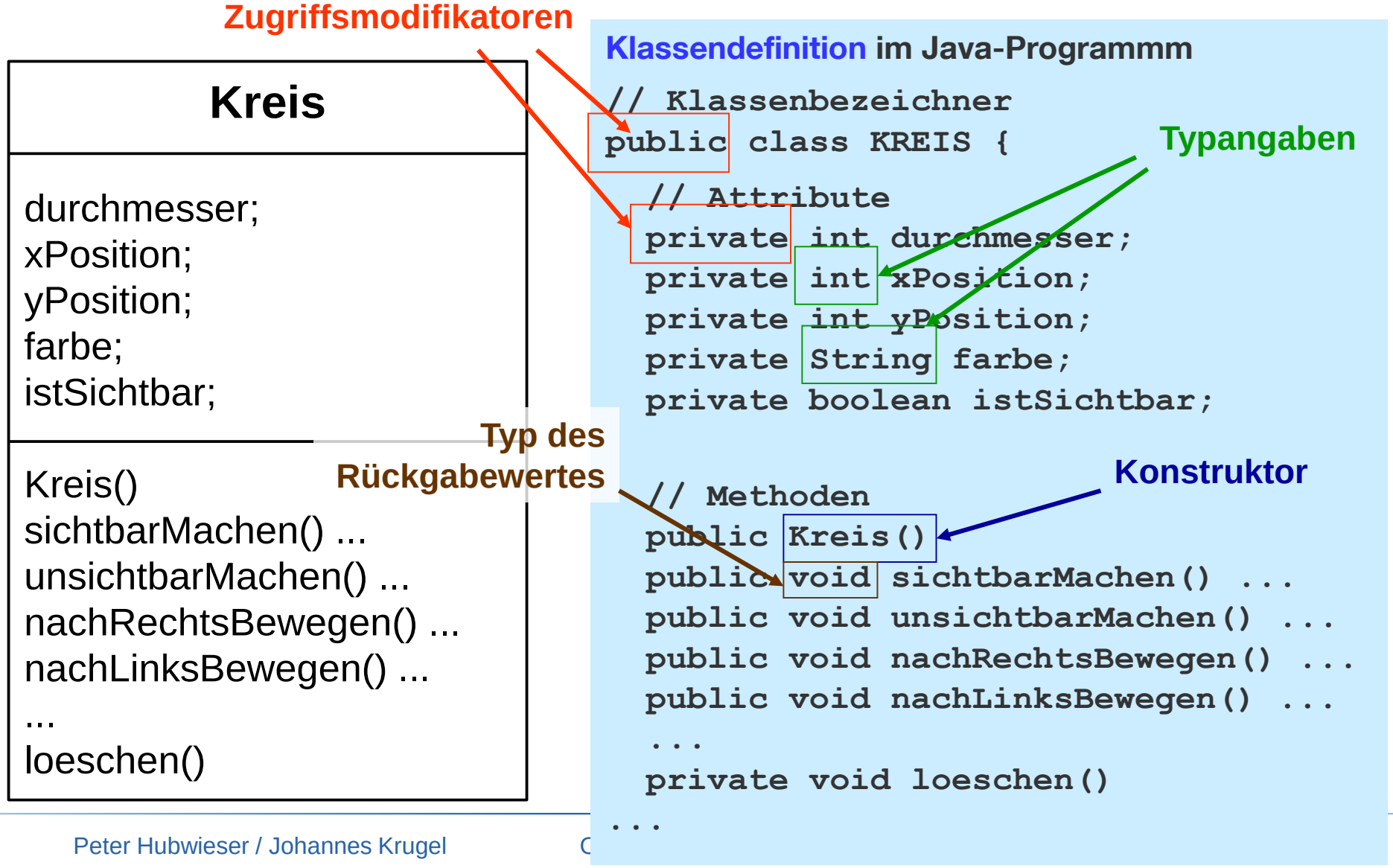
Methoden aufrufen und Parameterwerte übergeben

Objekte anlegen

Attributwerte inspizieren

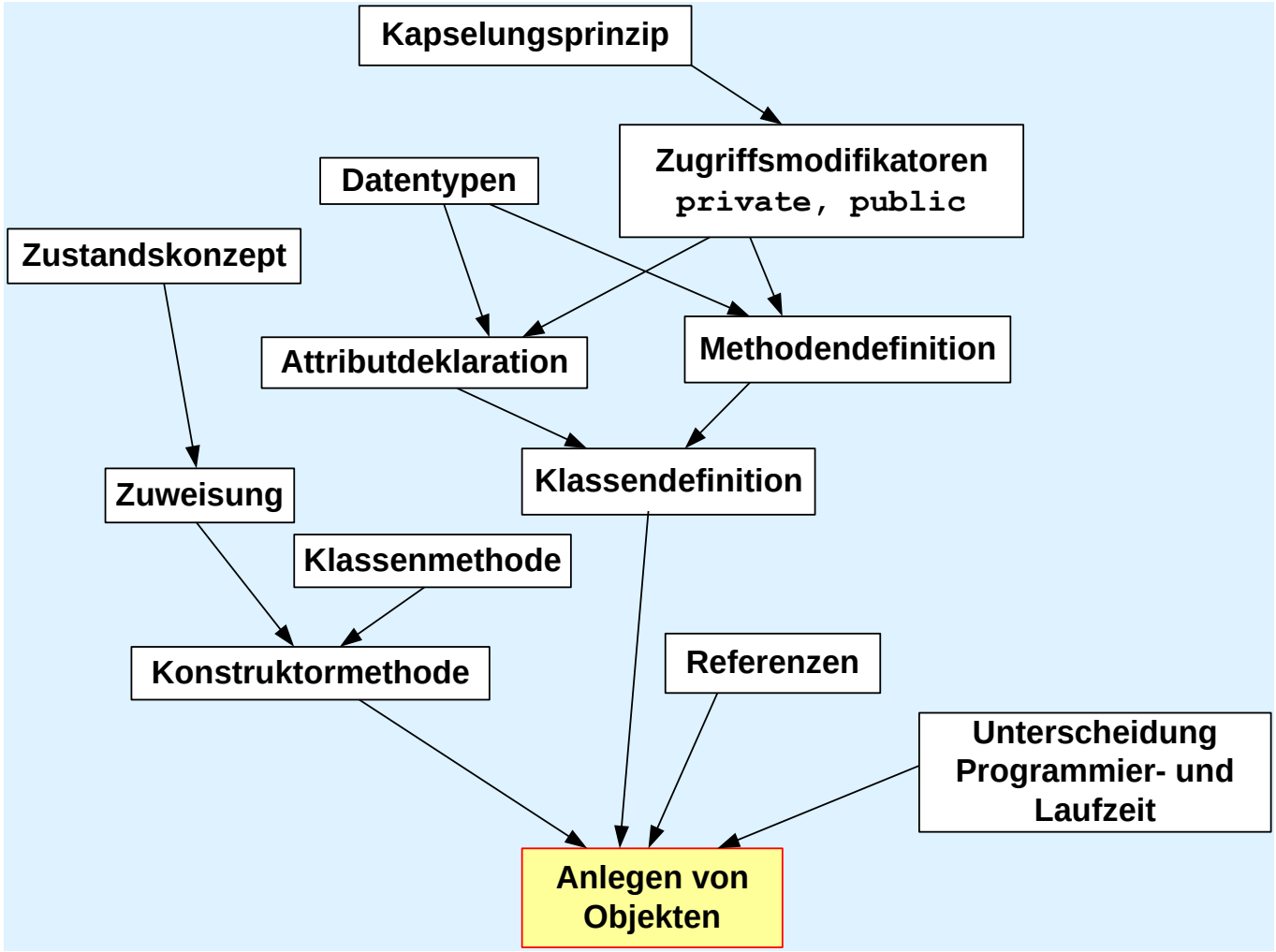
- Freie Software für alle Plattformen
- Java-basiert
- Klassen- und Objektdiagramme

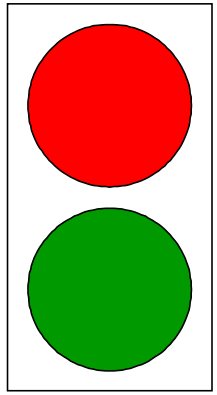
1. Definition von Klassen



Konstruktoraufruf: Vorkenntnisse

```
lichtOben = new Kreis();
```





Wirkung einer Zuweisung

- Bei einer Zuweisung geht der vorherige Wert des jeweiligen Attributes verloren.
- Nach der letzten Zuweisung haben beide Kreise denselben Durchmesser. Der Wert 10 ist also verloren gegangen.

Zeit

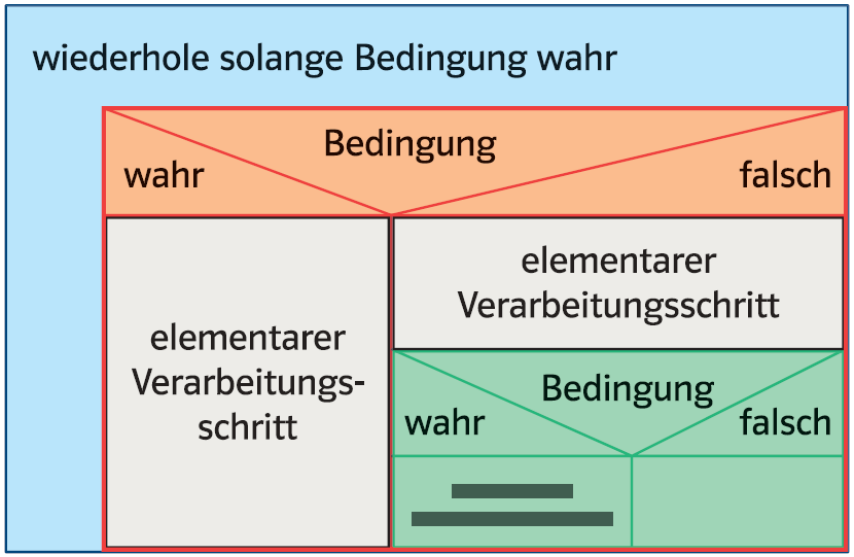
Anweisung	Wert von `kreis1.durchmesser`	Wert von `kreis2.durchmesser`
`kreis1.durchmesser = 10`		
	10	unbekannt
`kreis2.durchmesser = 20`		
	10	20
`kreis1.durchmesser = kreis2.durchmesser`		
	20	20

Ablauf des Ringtauschs

Anweisung	Wert von `kreis1.durchmesser`	Wert von `kreis2.durchmesser`	Wert von `kreis1.temp`
`kreis1.durchmesser = 10`			
	10	unbekannt	unbekannt
`kreis2.durchmesser = 20`			
	10	20	unbekannt
`kreis1.temp = kreis1.durchmesser`			
	10	20	10
`kreis1.durchmesser = kreis2.durchmesser`			
	20	20	10
`kreis2.durchmesser = kreis1.temp`			
	20	10	10

Implementieren von Algorithmen

- Umsetzung der algorithmischen Strukturelemente in eine Programmiersprache, z.B. Java




```
lichtOben.farbeAendern("rot");
lichtUnten.farbeAendern("schwarz");
zustand = "ROT";
```

```
if (zustand == "ROT") {
        umschaltenGruen();
}
else {
        umschaltenRot();
}
```

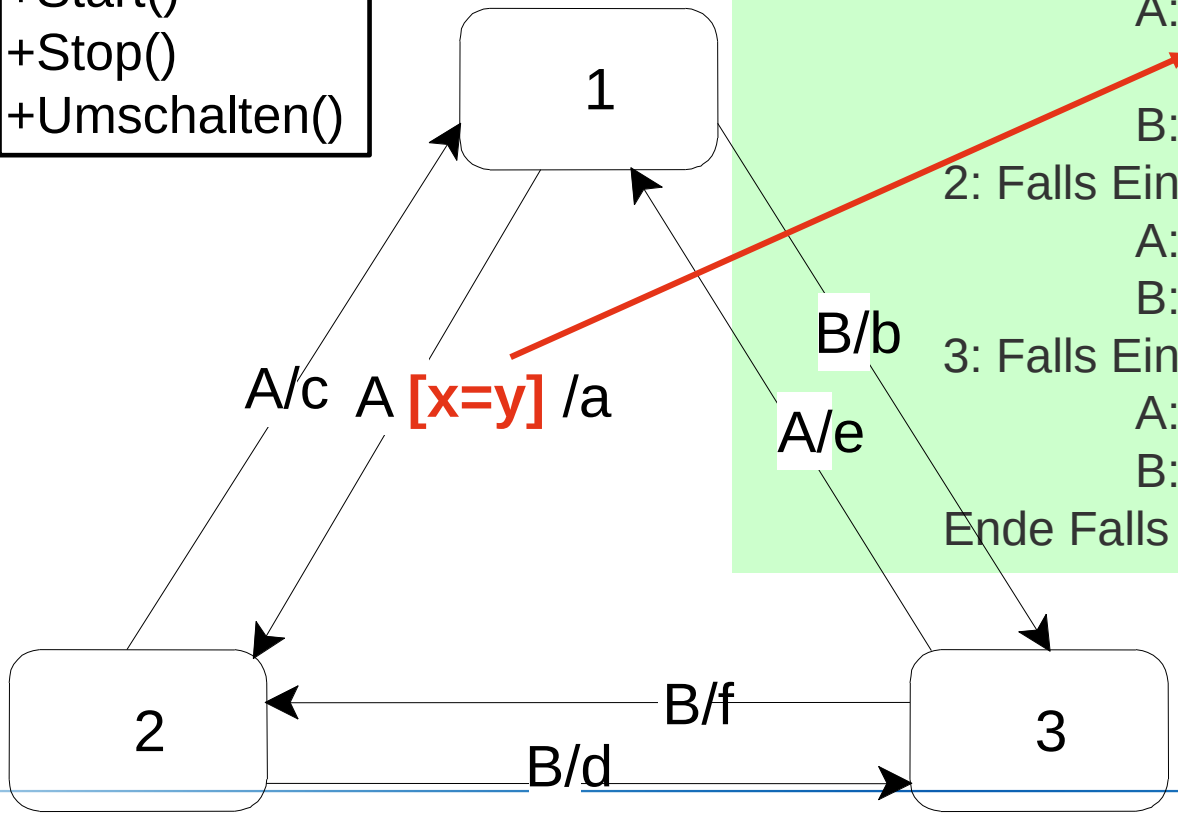
```
while (i<anzahl) {
  umschalten();
  setzeI(i+1);
  warte(wartezeit);
}
```

Vollständige Zustandsmodellierung

- Der Übergang von einem **Zustand z1** zu einem **Zustand z2** findet genau dann statt, wenn
  - sich das System im **Zustand z1** befindet und
  - die **auslösende Aktion a1** stattfindet und
  - unmittelbar vor dem Übergang die **Bedingung b** erfüllt ist.
- Dann wird mit dem Übergang auch die **Aktion a2** ausgelöst.
- Abstraktionen: Übergänge finden (im Vergleich zur Dauer der Zustände) in **unendlich kurzer Zeit** statt.

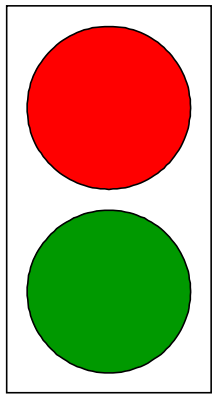
Implementierung von Zustandsmodellen

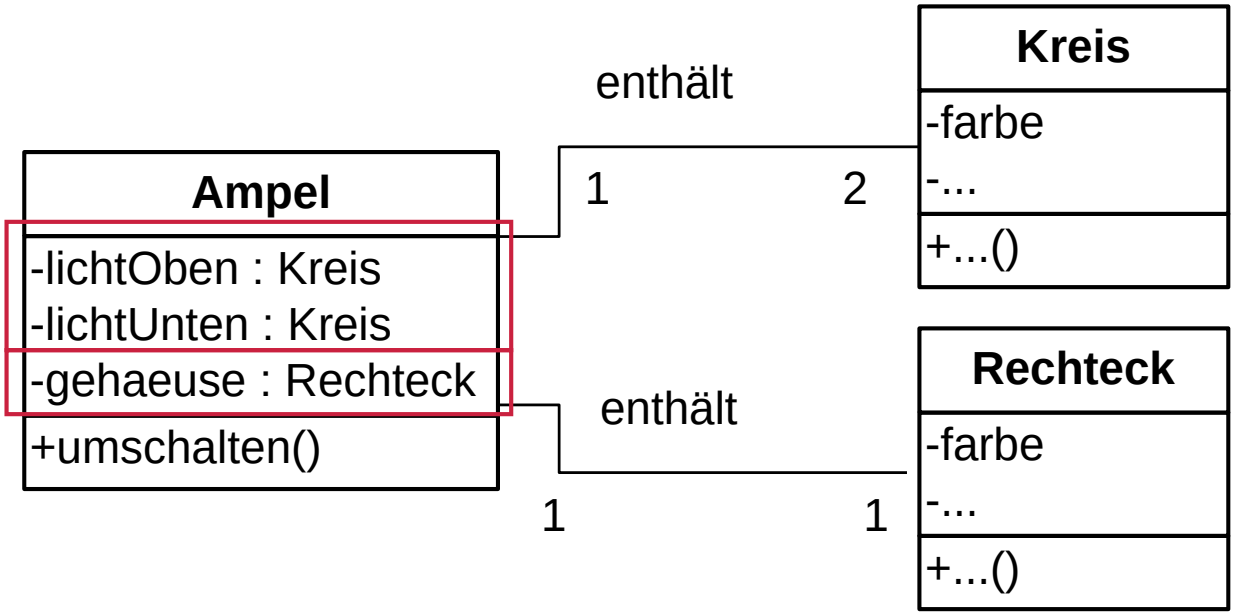
AUTOMAT
-zustand
+Start() +Stop() +Umschalten()




```
Wiederhole für immer
Falls Zustand =
        1: Falls Eingabe =
                A: Wenn (x=y) dann
                        Aktion a; Zustand = 2
                B: Aktion b; Zustand = 3
        2: Falls Eingabe =
                A: Aktion c; Zustand = 1
                B: Aktion d; Zustand = 3
        3: Falls Eingabe =
                A: Aktion e; Zustand = 1
                B: Aktion f; Zustand = 2
        Ende Falls
```

Beziehungen und Referenzen





- „**Enthält“-Beziehung**: Ein Objekt der Klasse `Ampel` enthält
  - zwei Objekte der Klasse `Kreis: lichtOben, lichtUnten` und
  - ein Objekt der Klasse `Rechteck: gehaeuse`.
- Diese Beziehung wird durch Attribute der enthaltenden Klasse AMPEL implementiert, die als Typ jeweils die Klasse der enthaltenen Objekte haben.

Verweise und Referenzen

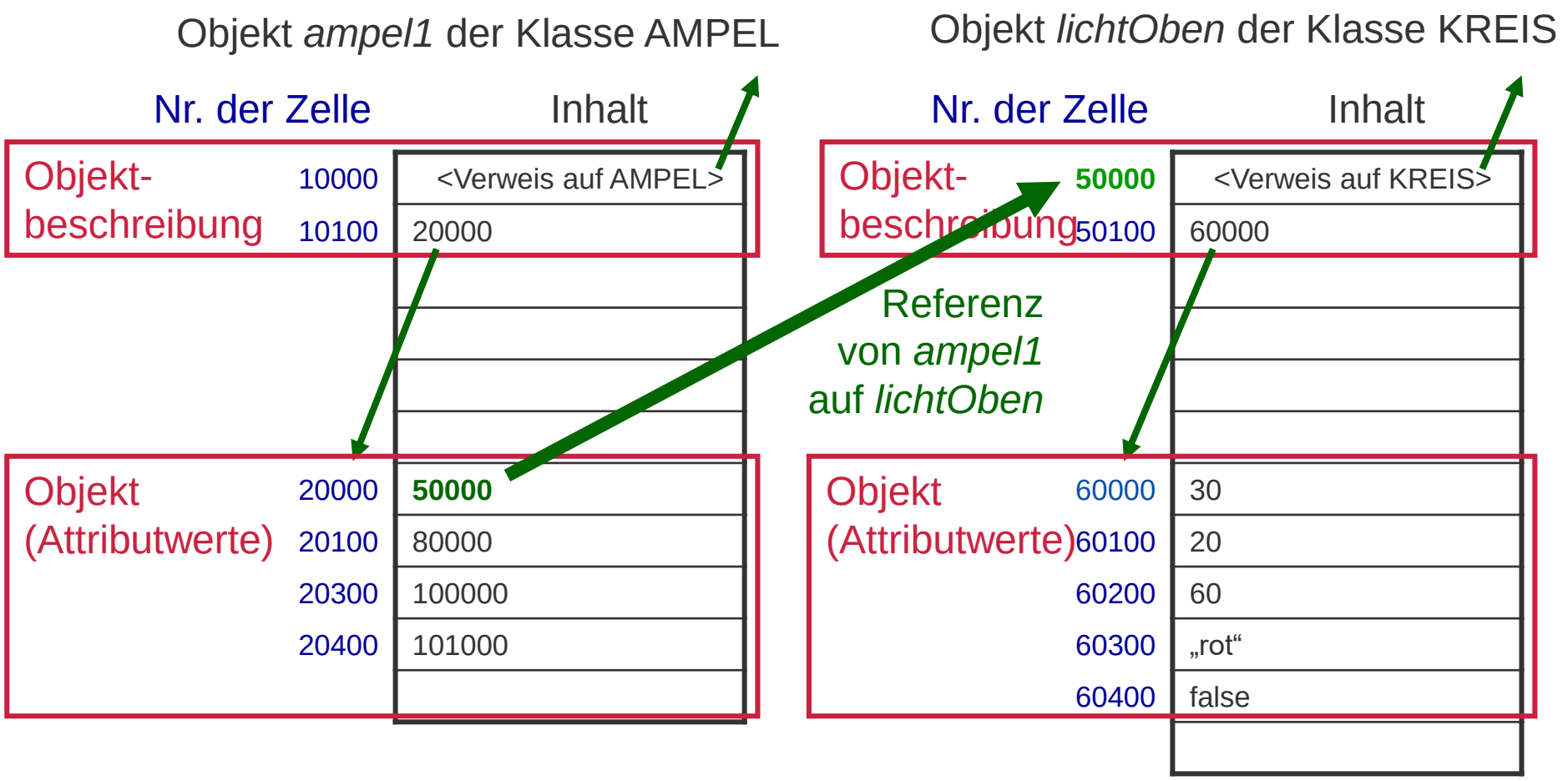
In vielen Situationen ist es praktisch, nicht mit den Objekten selbst zu hantieren, sondern **Verweise auf diese** zu verwenden:

- Ein Platzanweiser im Kino hätte z.B. große Mühe, wenn er am Eingang alle Sitzplätze stapeln und den Kunden mitgeben würde. Stattdessen arbeitet er mit einem Verweis auf den jeweiligen Platz in Form eines eindeutigen Bezeichners (z.B. Reihe 12 Platz 3).
- Ebenso werden im Kraftfahrtbundesamt in Flensburg nicht alle deutschen Kraftfahrzeuge selbst gestapelt, sondern Verweise darauf in Form der Kennzeichen (wie z.B. RO-K 223).
- Ein weiteres Beispiel sind die Verweise (Links) im WWW in Form der URLs (wie http://www.in.tum.de).

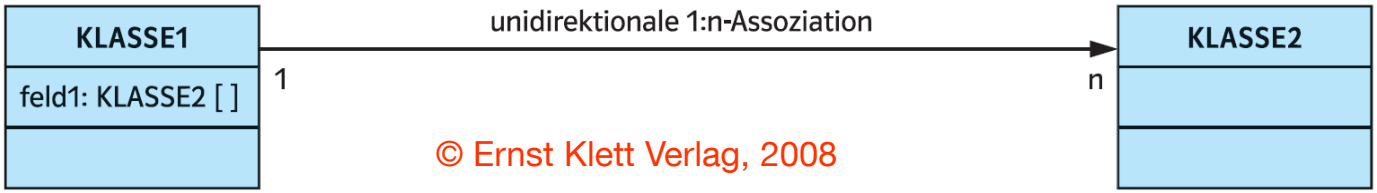
- Man spricht von **Referenzierung**, wenn man nicht mit dem Objekt selbst hantiert, sondern mit einem Verweis auf das Objekt in Form eines eindeutigen Bezeichners (z.B. einer Speicherstelle oder eines Dateinamens im Internet).

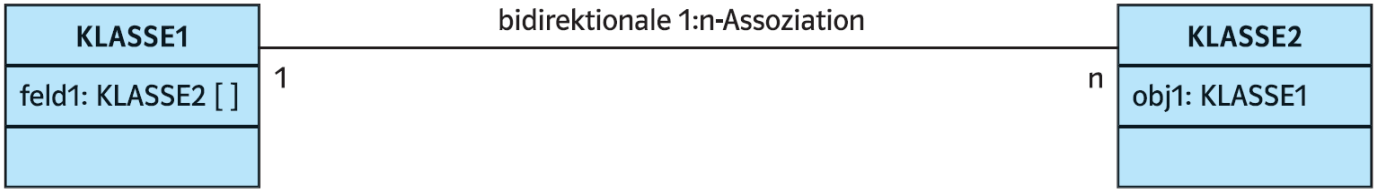
Objekte im Arbeitsspeicher

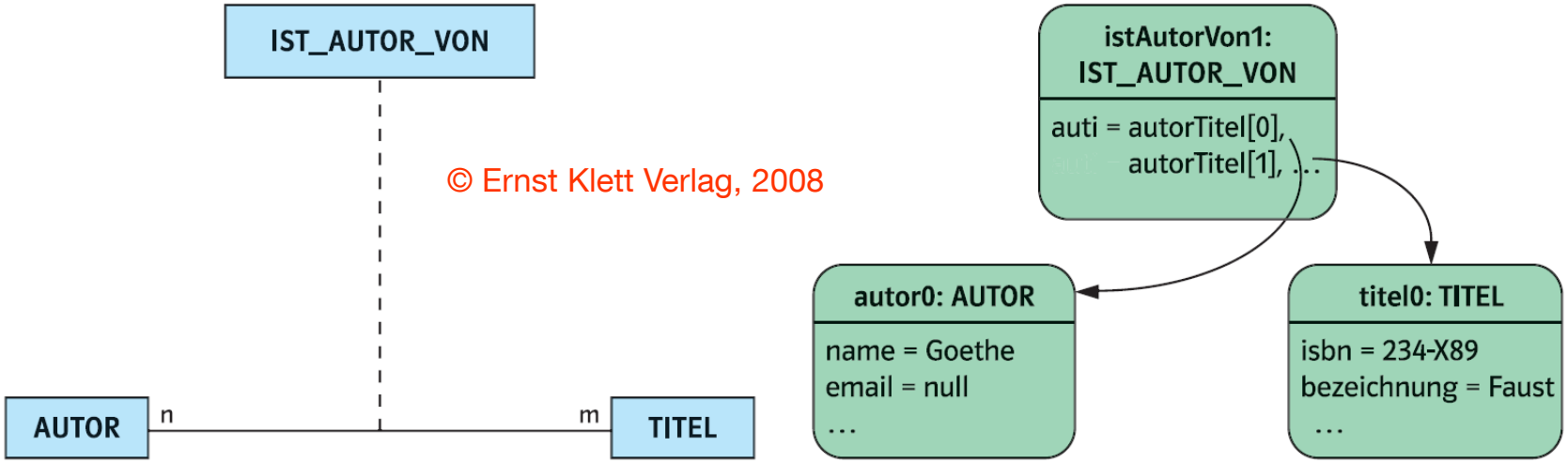
```
public class Ampel {
  private Kreis lichtOben;
  private Kreis lichtUnten;
  private Rechteck gehaeuse;
  private String zustand; …
}
```



Implementierung von Referenzen





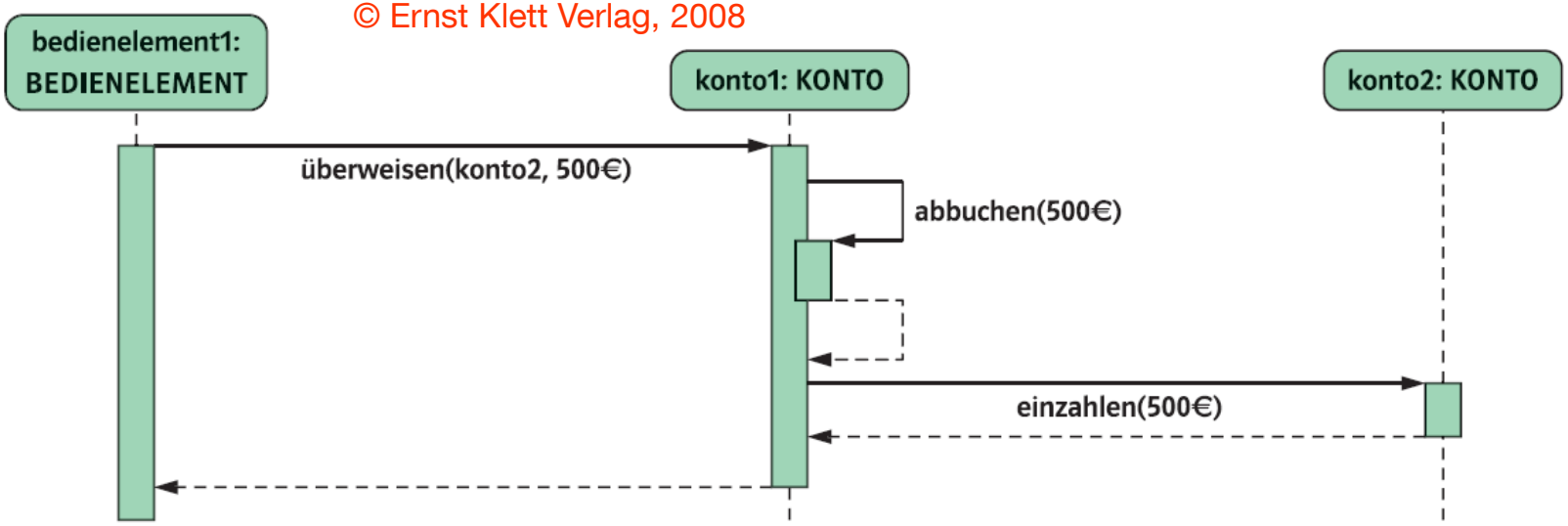


Sequenzdiagramme

Im neuen Lehrplan G9 weggelassen

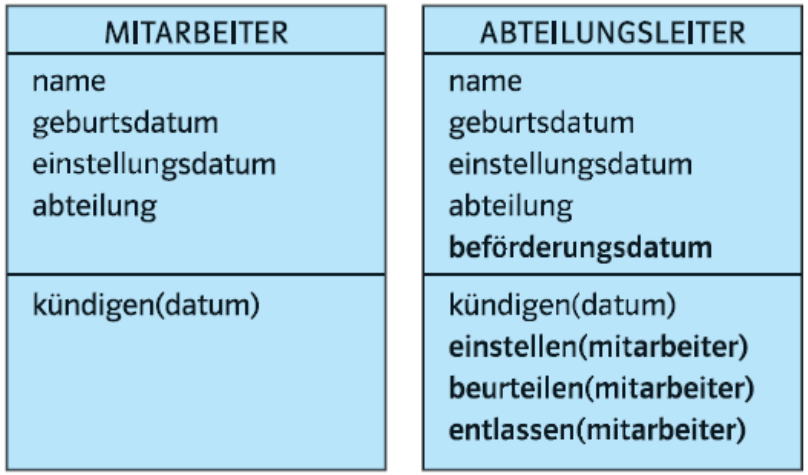
```
public void einzahlen(double b) {
    ...}
public void abbuchen(double b) {
    ...}

public void ueberweisen(Konto kto,
    double b) {
   abbuchen(b);
   kto.einzahlen(b);
}
```

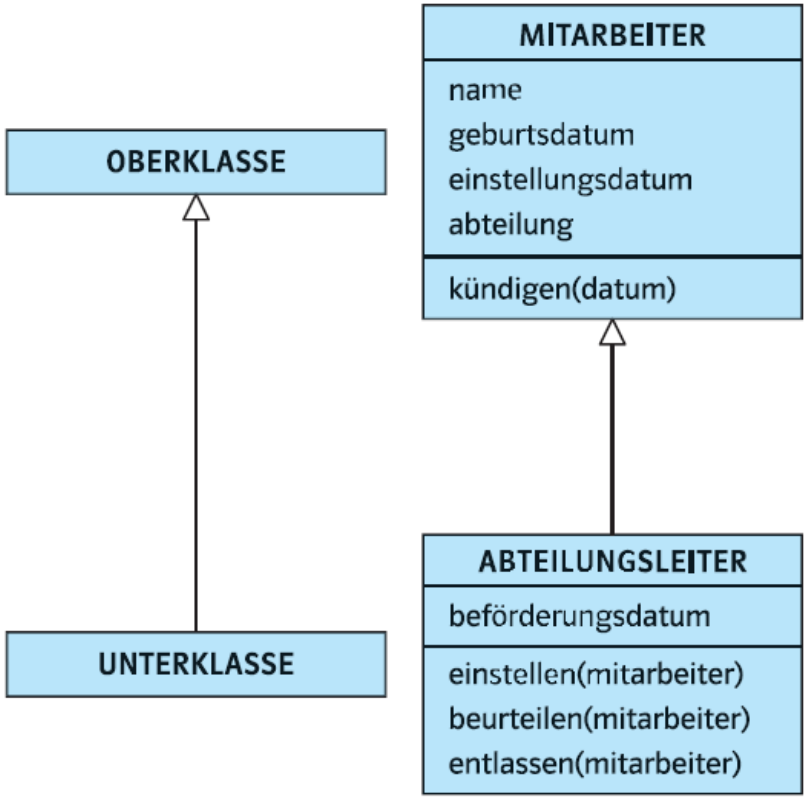
KONTO
kontonummer kontoinhaber kontostand
einzahlen(betrag) abbuchen(betrag) überweisen(konto, betrag)



Spezialisierung

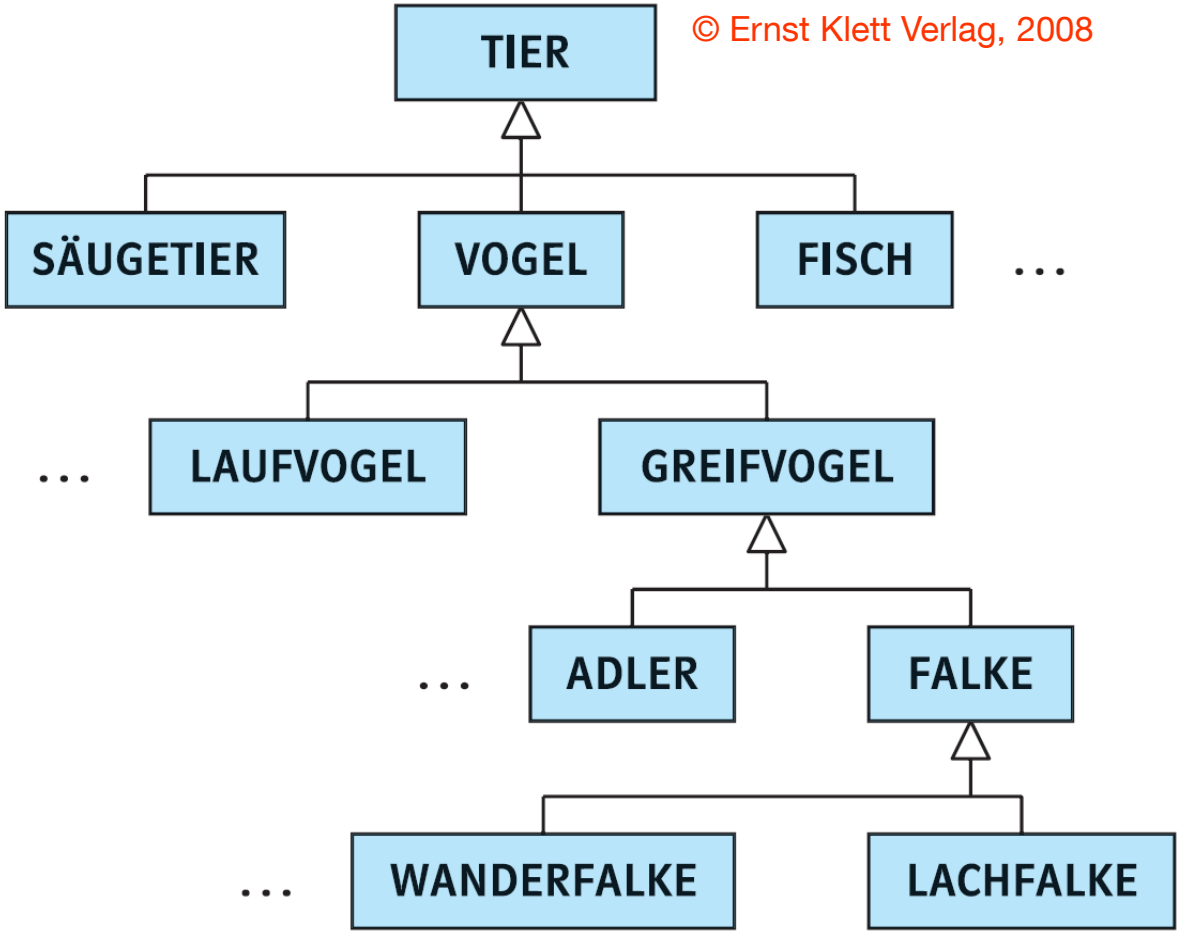
- Spezialisierung durch Bildung von Unterklassen
- Vererbung: Unterklassen erben alle Attribute und Methoden ihrer Oberklasse
- Betonung auf konzeptueller Spezialisierung
- **.. nicht** auf technischer Vererbung von Attributen!
- Die Spezialisierung ist eine reine Klassenbeziehung!





Inf9 Lernbereich 3

Generalisierung - Klassenhierarchien

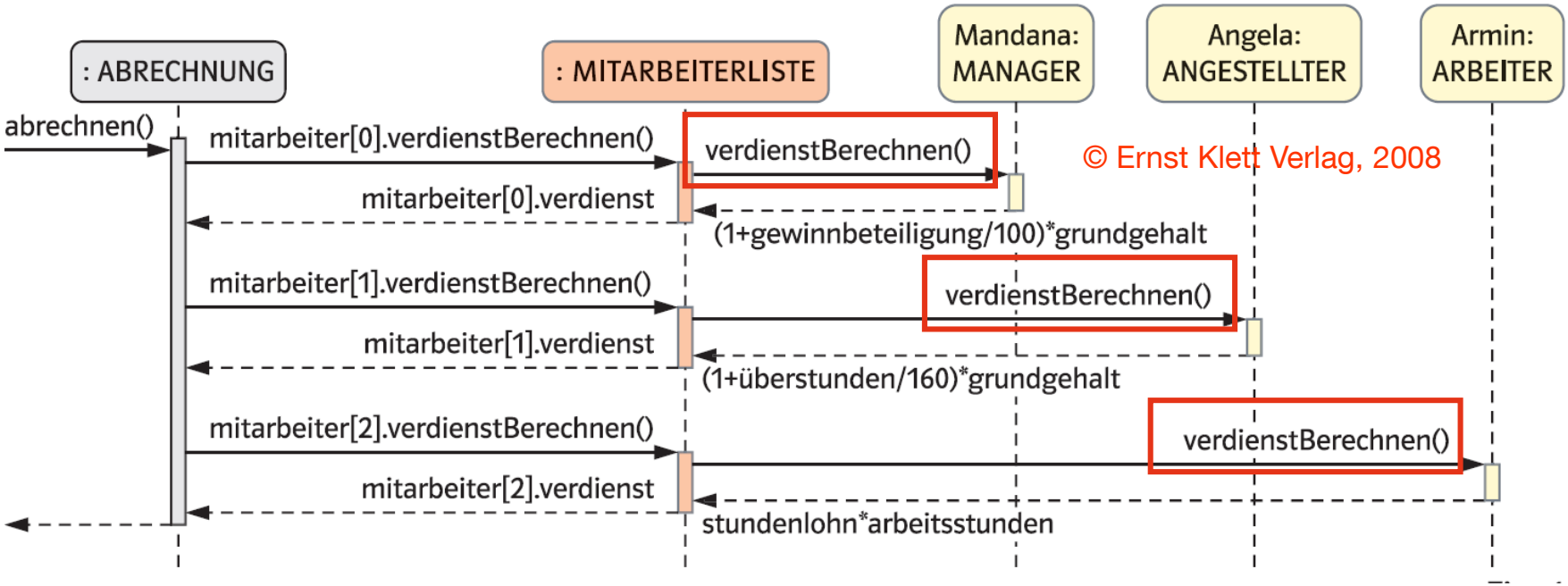


Inf9 Lernbereich 3

Polymorphismus

Inf10 Lernbereich 2

- Man spricht von Polymorphismus, wenn gleichnamige Methoden unterschiedliche Definitionen bzw. Implementierungen haben.
- Polymorphie wird durch Überschreiben einer ererbten Methoden realisiert.



Softwareprojekte im Unterricht

Management
- Projektdefinition
- Projektplanung
- Projektkontrolle
- Projektabschluss

Projektideen
- Zeichentrickfilm
- Supermarkt
- Börsenspiel
- DVD-Verleih
- Tic-Tac-Toe
- Kniffel
- Sudoku
- Pizzaservice

Vorschläge für Softwareprojekte

A Zeichentrickfilm
Mithilfe eines Programmpaketes für Grafiken soll ein Zeichentrickfilm entw
Eventuell können alternative Handlungsabläufe über Eingaben durch den B
steuert werden.

B Supermarkt
Es soll ein Programm entwickelt werden, das die Vorgänge in einem Supermarkt simuliert: An jeder Kasse kann eine begrenzte Anzahl von Kunden in einer Warteschlange stehen. Jeder neue Kunde, der bezahlen möchte, reiht sich in eine möglichst kurze Warteschlange ein. Übersteigt die Anzahl der wartenden Kunden ein bestimmtes Maß, wird eine weitere Kasse geöffnet. Wird eine Kasse gesperrt, können sich keine weiteren Kunden dort anstellen.

Literaturverzeichnis

ANDERSON, L.W., AND KRATHWOHL, D.R. 2001. *A taxonomy for learning, teaching, and assessing. A revision of Bloom's taxonomy of educational objectives*. Longman, New York.

ARMSTRONG, D.J. 2006. The quarks of object-oriented development. *Commun. ACM 49*, 123-128. http://doi.acm.org/10.1145/1113034.1113040.

BAYERISCHES STAATSMINISTERIUM FÜR UNTERRICHT UND KULTUS. 2010. *Schüler- und Absolventenprognose*. Reihe A: Bildungsstatistik 52.

BENNEDSEN, J., AND SCHULTE, C. What does Objects-First. In *Seventh Baltic Sea Conference on*.

BERGES, M., AND HUBWIESER, P. 2011. Minimally invasive programming courses – Learning OOP with(out) instruction. In *Proceedings of SIGCSE 2011. Dallas, TX*, To Appear, ACM, Ed.

DIETHELM, I. 2007. Strictly models and objects first" - ein Unterrichtskonzept für OOM. In *Didaktik der Informatik in Theorie und Praxis. INFOS 2007: 12. GI-Fachtagung Informatik und Schule, 19.-21. September 2007 in Siegen*, S. E. SCHUBERT, Ed. GI, 45–56.

EHLERT, A., AND SCHULTE, C. 2009. Empirical comparison of objects-first and objects-later. In *Proceedings of the fifth international workshop on Computing education research workshop*, ACM, Ed. ACM, New York, NY, USA, 15-26.

ENGEL, G., AND ROBERTS, E. 2002. *Computing curricula 2001. ACM/IEEE-CS Joint Curriculum Task Force, final report, December, 2001*. IEEE.

HENDERSON-SELLERS, B. 1992. *A book of object-oriented knowledge. Object-oriented analysis, design and implementation: a new approach to software engineering*. Prentice-Hall object-oriented series. Prentice-Hall, Englewood Cliffs NJ.

HUBWIESER, P. 2001. Informatik - Allgemeinbildung für die Informationsgesellschaft. In *Das ist Informatik*, J. DESEL, Ed. Springer, Berlin, 117–134.

HUBWIESER, P. 2006. Functions, Objects and States: Teaching Informatics in Secondary Schools: Invited talk. In *Informatics Education - The Bridge between Using and Understanding Computers*, R. MITTERMEIR, Ed. Springer.

HUBWIESER, P. 2007. A smooth way towards object oriented programming in secondary schools. In *Informatics, Mathematics and ICT: A golden triangle: Proceedings of the Working Joint IFIP Conference: WG3.1 Secondary Education, WG3.5 Primary Education; College of Computer and Information Science, Northeastern University Boston, Massachusetts, USA 27th - 29th June 2007*, D. BENZIE AND M. IDING, Eds., Boston,MA.

KÖLLING, M., AND ROSENBERG, J. 1996. Blue - a language for teaching object-oriented programming. In *ACM SIGCSE Bulletin*, 190–194.

LEWIS, J. 2000. Myths about object-orientation and its pedagogy. *SIGCSE Bull 32*, 245-249. http://doi.acm.org.eaccess.ub.tum.de/10.1145/331795.331863.

ROSSON, M.A.A.S. 1990. The cognitive consequences of object-oriented design. *Human Computer Interaction 5*, 4, 345–379.

SCHWILL, A. 1995. Programmierstile im Anfangsunterricht. In *Innovative Konzepte für die Ausbildung, 7. GI-Fachtagung Informatik und Schule, INFOS'95, Chemnitz, 25.-28. September 1995*, S. E. SCHUBERT, Ed. Springer, 178–187.

Vielen Dank für Ihre Aufmerksamkeit!

Noch Fragen? Gerne!