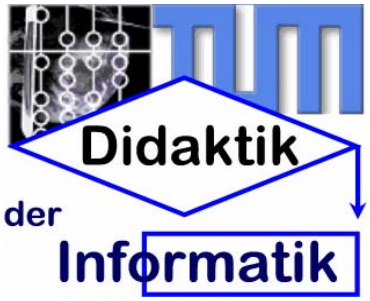


Prof. Dr. Peter Hubwieser
Fachgebiet Didaktik der Informatik
Fakultät für Informatik der TU München

Informatik in der Mittelstufe

10. Objektorientierte Modellierung und Programmierung

http://ddi.in.tum.de

Peter.Hubwieser@in.tum.de

Überblick

Ziel: Sie sollen einen Eindruck davon bekommen, was Sie ab 2008 unterrichten werden

Inhalt:

- Didaktische und fachliche Vorüberlegungen
- Werkzeuge
- Unterrichtskonzepte aus Schülersicht

ACHTUNG: kein fertiger Unterrichtsgang!

Didaktik der Informatik

1. Didaktische und fachliche Vorüberlegungen

Der Lehrplan

10 Informatik (NTG 2)

Nachdem sich die Schüler in der zweiten Hälfte des vorangegangenen Schuljahrs mit statischer Modellierung beschäftigt haben, wenden sie sich nun grundlegenden Konzepten der automatischen Verarbeitung von Information zu. Anknüpfend an ihre Erfahrungen mit einfachen Abläufen bei der Bedienung von Automaten aus ihrer Lebensumgebung, wie z. B. Fahrkartenautomaten, lernen sie, dass sich diese Abläufe durch eine Gliederung in Zustände und Zustandsübergänge der beteiligten Objekte beschreiben lassen. Folgen von Zustandsübergängen führen die Schüler zu den Grundstrukturen von Algorithmen. Die Jugendlichen erkennen dabei die wesentlichen Bausteine algorithmischer Ablaufbeschreibungen wieder, mit denen sie sich bereits in Jahrgangsstufe 7 beschäftigt haben. Sie verwenden nun Algorithmen zur Beschreibung der Funktionsweise von Methoden und vertiefen dabei ihre Kenntnisse im Erstellen von Ablaufvorschriften.
Den Schülern wird deutlich, dass in der objektorientierten Sichtweise alle bisher angewandten Techniken zielgerichtet und miteinander verzahnt zur Lösung umfangreicherer Aufgabenstellungen genutzt werden können. Gleichzeitig gewinnen sie ein deutlich tieferes Verständnis für die bereits in den Jahrgangsstufen 6 und 7 eingeführten Fachbegriffe der objektorientierten Sichtweise.
Die neuen Inhalte begegnen den Schülern im Rahmen von ausbaufähigen Aufgabenstellungen, wobei die praktische Arbeit einen großen Anteil des Unterrichts umfasst. Gleichzeitig erfahren die Jugendlichen auch schrittweise die grundlegenden Vorgehensweisen bei der Planung von Softwareprojekten. Sie erkennen, dass erst durch sorgfältig geplante Teamarbeit in Verbindung mit einem soliden Fundament an Wissen und einer klar strukturierten Vorgehensweise die Lösung von schwierigen, für den Einzelnen zu umfangreichen Aufgabenstellungen möglich wird.

In der Jahrgangsstufe 10 erwerben die Schüler folgendes Grundwissen:

- Sie können zeitliche Abläufe strukturieren, indem sie sie mit Hilfe von Zuständen und Übergängen

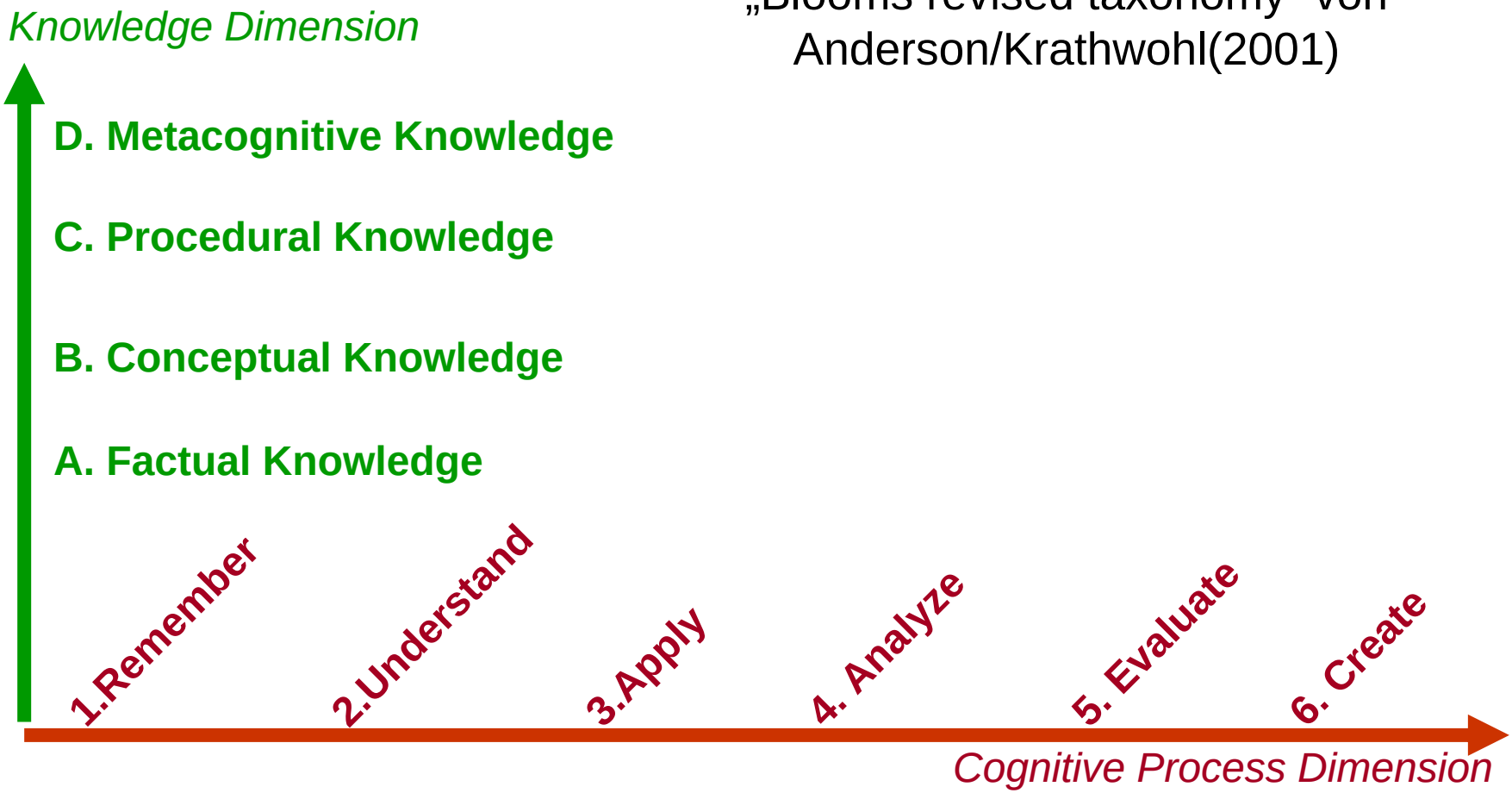
VORSICHT!

- In Gegensatz zu früheren Wahlkursen handelt es sich hier um **Pflichtunterricht.**
- Der Unterricht konkurriert mit Sprachen, Mathematik, Geschichte, Physik ....
  - Die bloße Fähigkeit „zu programmieren“ reicht als Lernziel keinesfalls aus!
  - Es geht um das gymnasialadäquate vertiefte Verständnis der Konzepte!
- Informatik ist ohne Ausnahme für **alle** Schülerinnen und Schüler der NTG-Richtung verpflichtend und **vorrückungsrelevant!**

=> **Jede(r)** muss eine Chance haben, die vorgestellten Konzepte verstehen und beherrschen zu lernen!

Einordnung in Lernzieltaxonomie

„Blooms revised taxonomy“ von Anderson/Krathwohl(2001)



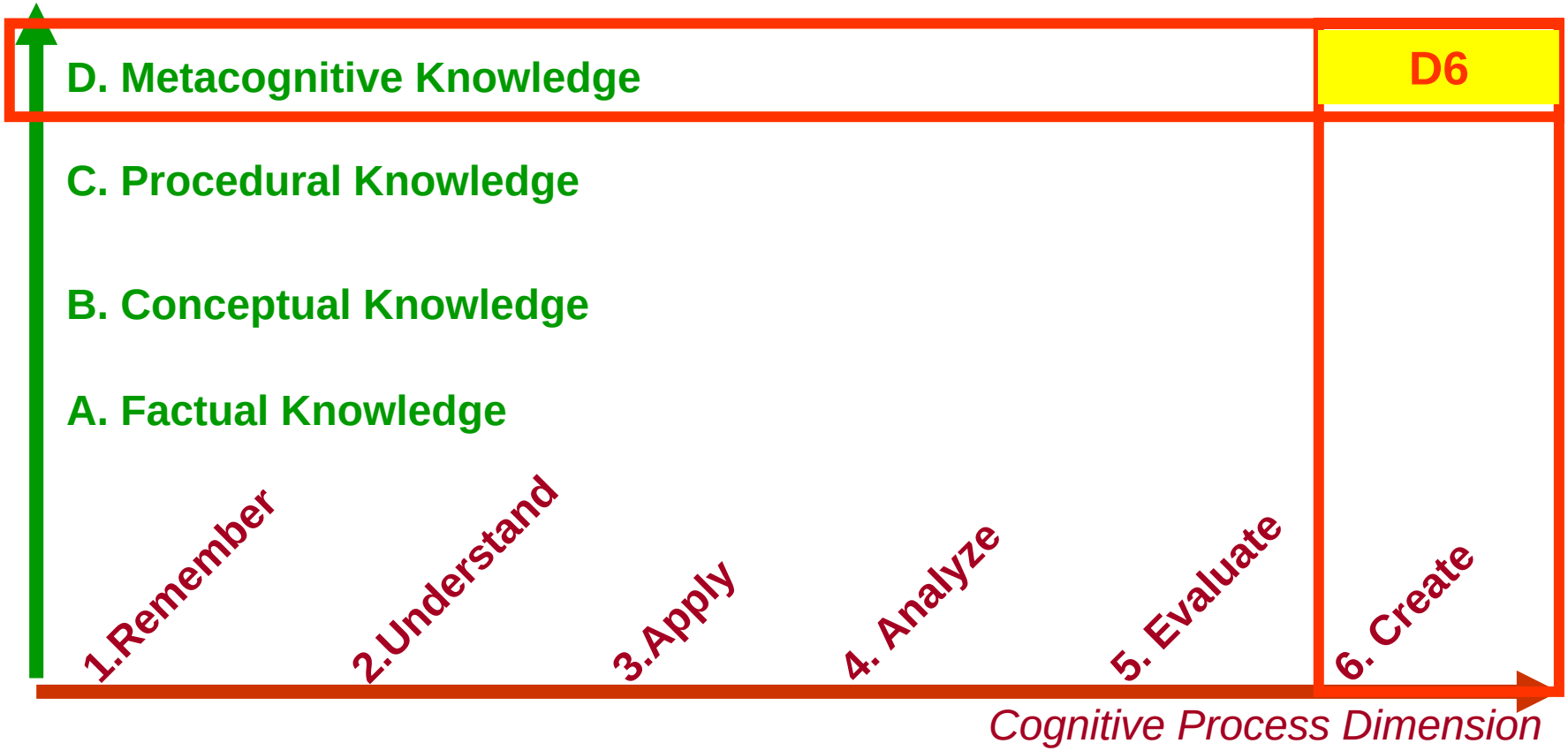
OOM/OOP ist sehr schwierig!

1. ***Cognitive process dimension:*** Schülerinnen und Schüler …
   - **analysieren** Systeme, um sie durch OO-Modelle darstellen zu können (Stufe 4),
   - **evaluieren** alternative Modelle, um das passende zu finden (Stufe 5),
   - **erzeugen** (create) Modelle und Programme zu ihren Modellen (Stufe 6)
2. ***Knowledge dimension*** Schülerinnen und Schüler …
   - kennen die Syntaxelemente einer Programmiersprache (**A. Factual knowledge**),
   - kennen die Strukturelemente von Algorithmen (**B. Conceptual knowledge**)
   - Simulieren Modelle mit Programmen (**C. Procedural knowledge**),
   - **D. Metacognitive knowledge**: beherrschen den Modellierungs- und Programmiervorgang im Ganzen.

Anderson and Krathwohl(2001)

Revidierte Bloomsche Taxonomie

Wir erreichen die schwierigste Stufe!



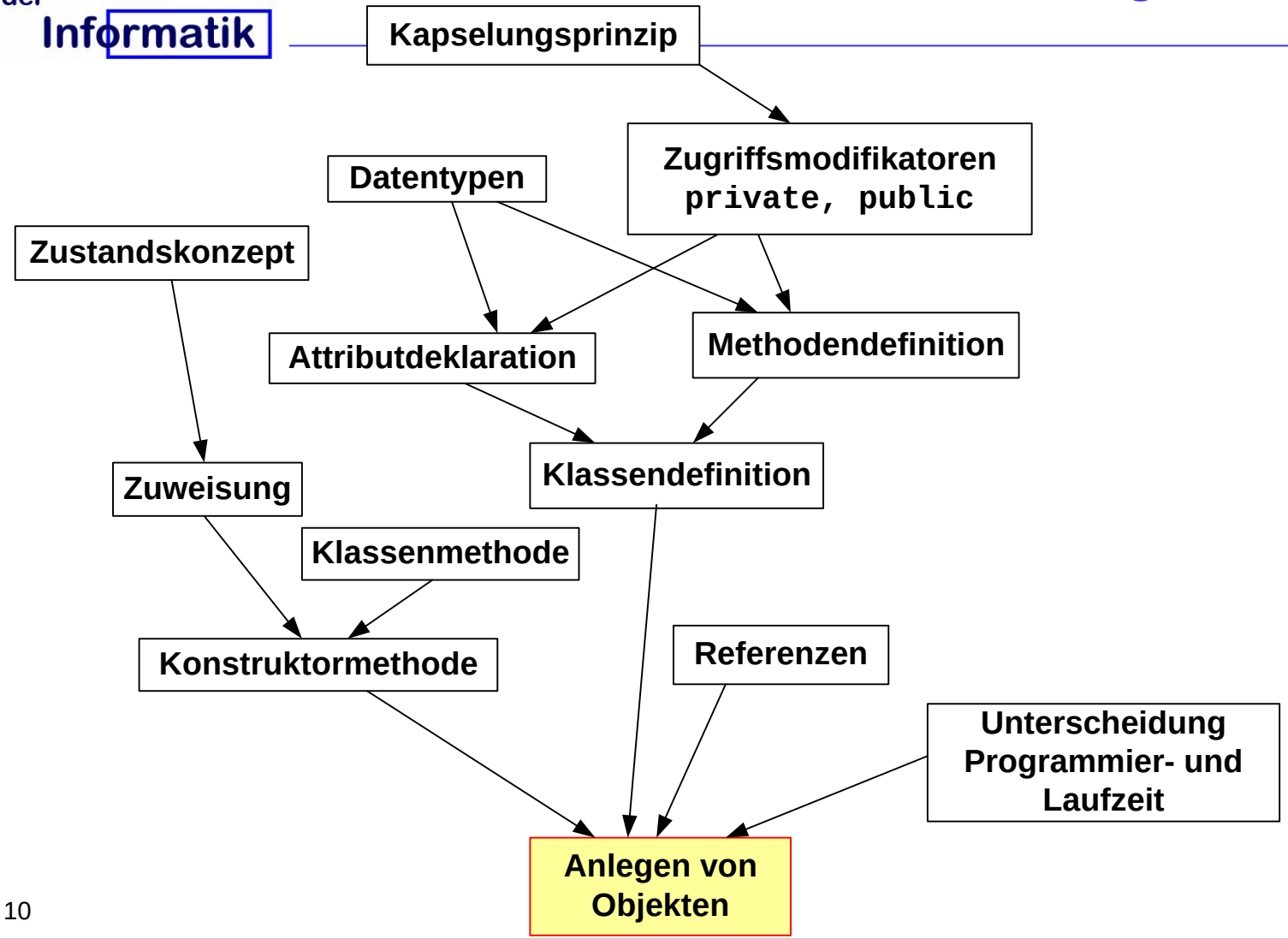
Didaktisches Dilemma

EINERSEITS verlangen moderne konstruktivistische pädagogische Ansätze den Einstieg:

- über Aufgabenstellungen aus authentischen Anwendungsbereichen mit realistischer Komplexität
- mit Bezug zur Erfahrungswelt der Schülerinnen und Schüler
- die zeigen, dass die zu lerndenen Konzepte tatsächlich Relevanz im Alltag haben
- also keine „Hello World“-Programme!

ANDERERSEITS verlangt ein solcher Einstieg, dass die Schülerinnen und Schüler **viele schwierige Konzepte auf einmal** lernen müssen!

z.B. für das erste Programm:



Didaktik der

Verzahnung von 4 Themensträngen

Zustandsmodellierung

- Zustände von Objekten, Attributen, lokalen Variablen
- Zustandsübergänge: auslösende und ausgelöste Aktion, Übergangsbedingung
- Implementierung von Zustandsmodellen

Algorithmen

- Algorithmusbegriff
- Informelle Algorithmen
- Strukturelemente
- Ein-/Ausgabe
- Implementierung
- Darstellung

OOM

- Objekte. Klassen, Attribute, Methoden
- Unterscheidung Klasse/Objekt
- Klassen- und Objektkarten
- Assoziationen, insbesondere Aggregation
- Vererbung
- Objekt- und Klassendiagramme

OOP

- Unterscheidung Programmier- und Laufzeit
- Implementierung von Klassenmodellen, insbesondere von Assoziationen
- Referenzen
- Lebenszyklus von Objekten
- Konstruktoren
- Kapselung: Zugriffsmodifikatoren

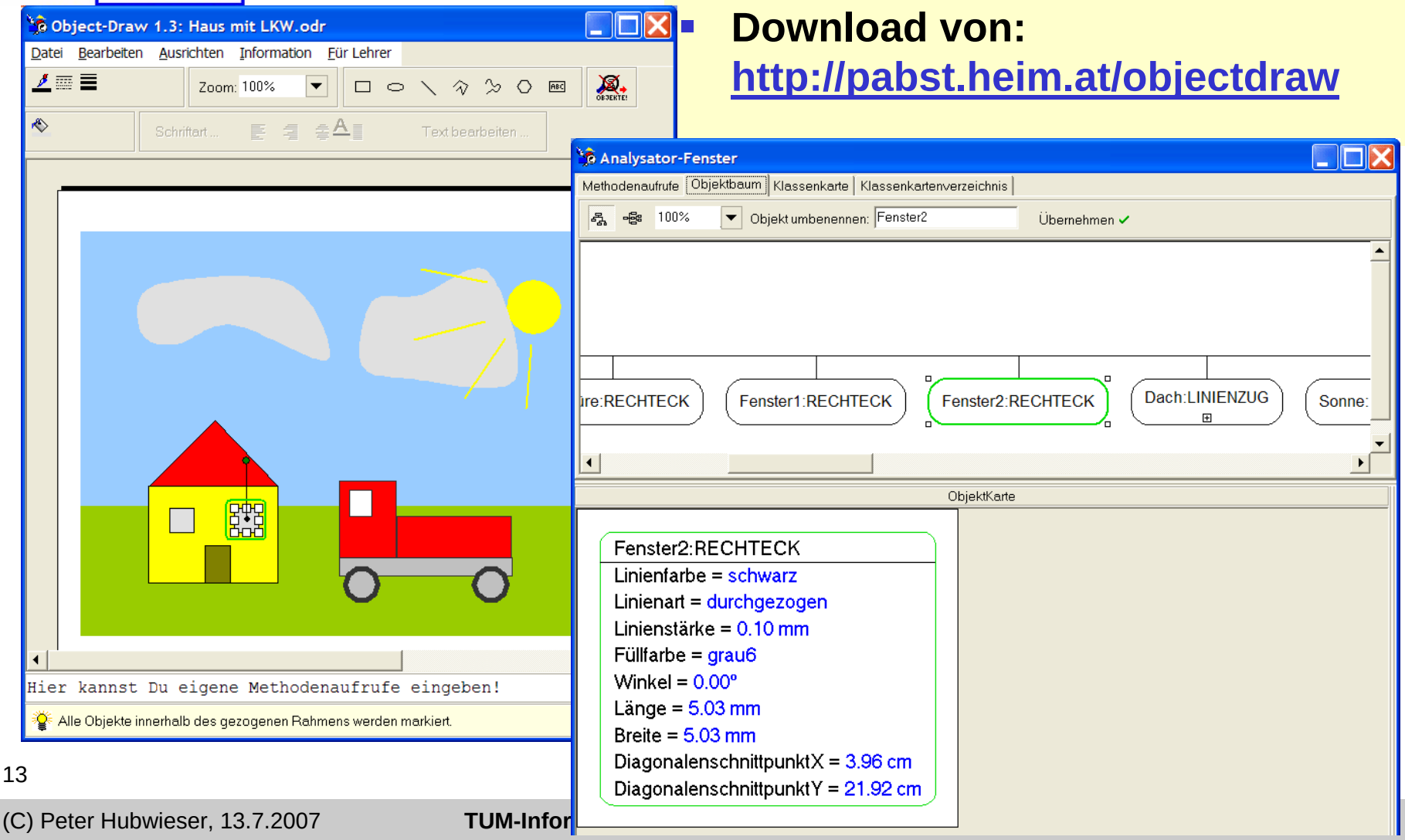
Didaktik der Informatik

2. Werkzeuge

Didaktik der Informatik

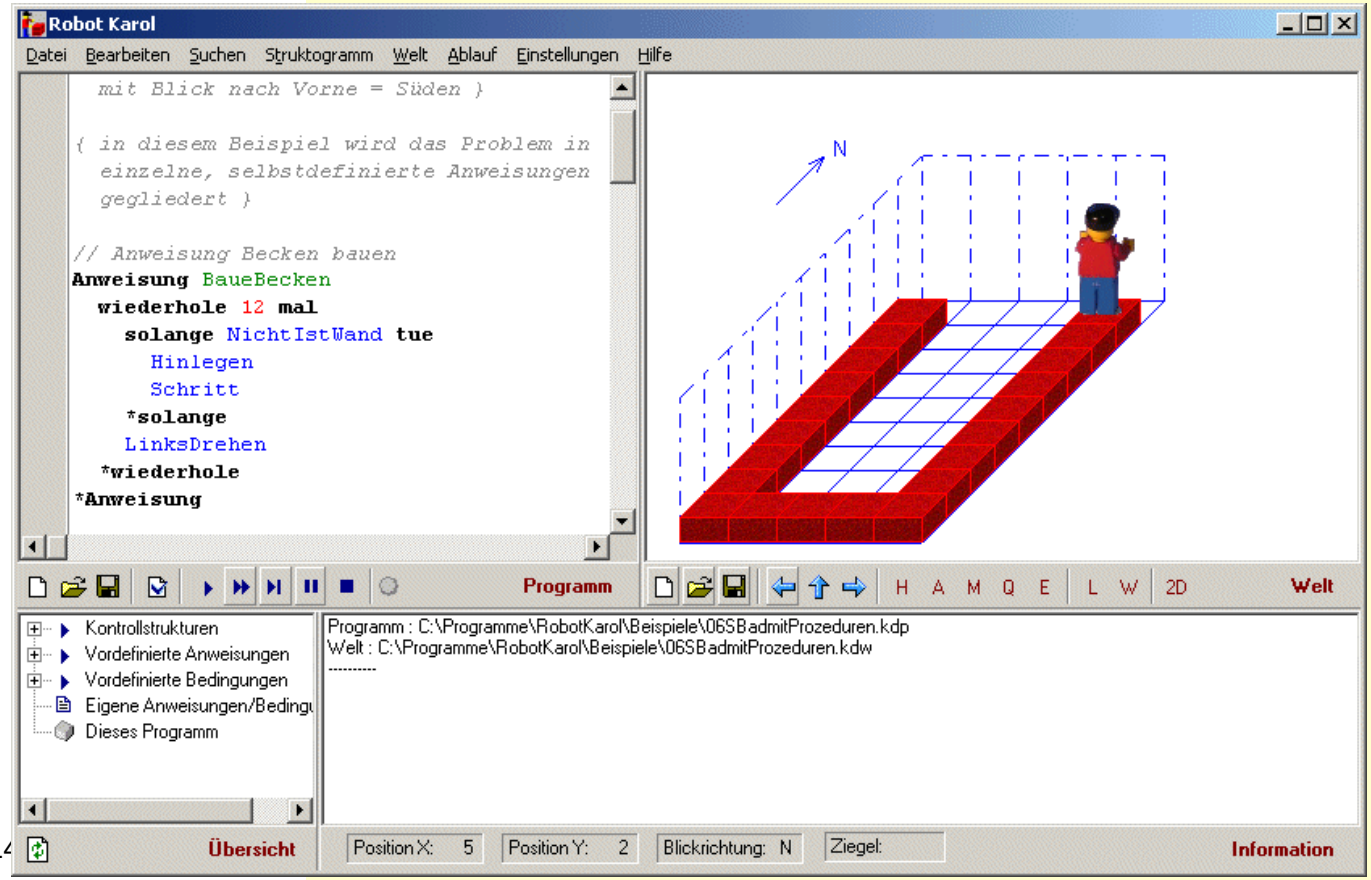
ObjectDraw [Martin Pabst]

- **Download von:**
  http://pabst.heim.at/objectdraw



Karol [Freiberger, Krsko]

- Download: http://www.schule.bayern.de/karol



Didaktik der Informatik

EOS [Martin Pabst]

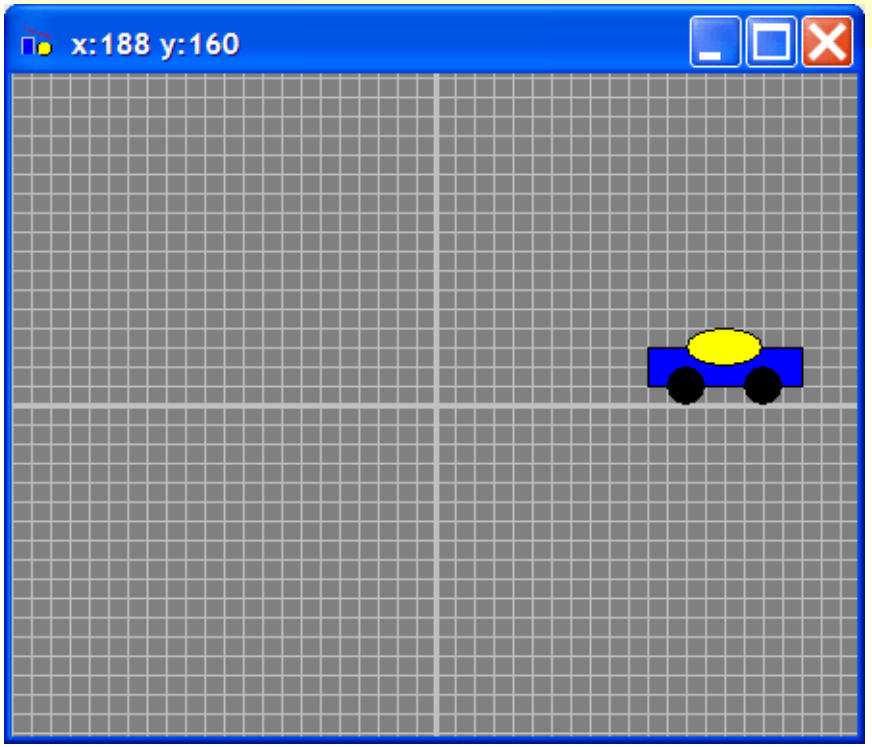
EOS 1.5.7 - Auto1.eos

Datei Bearbeiten Programm Information

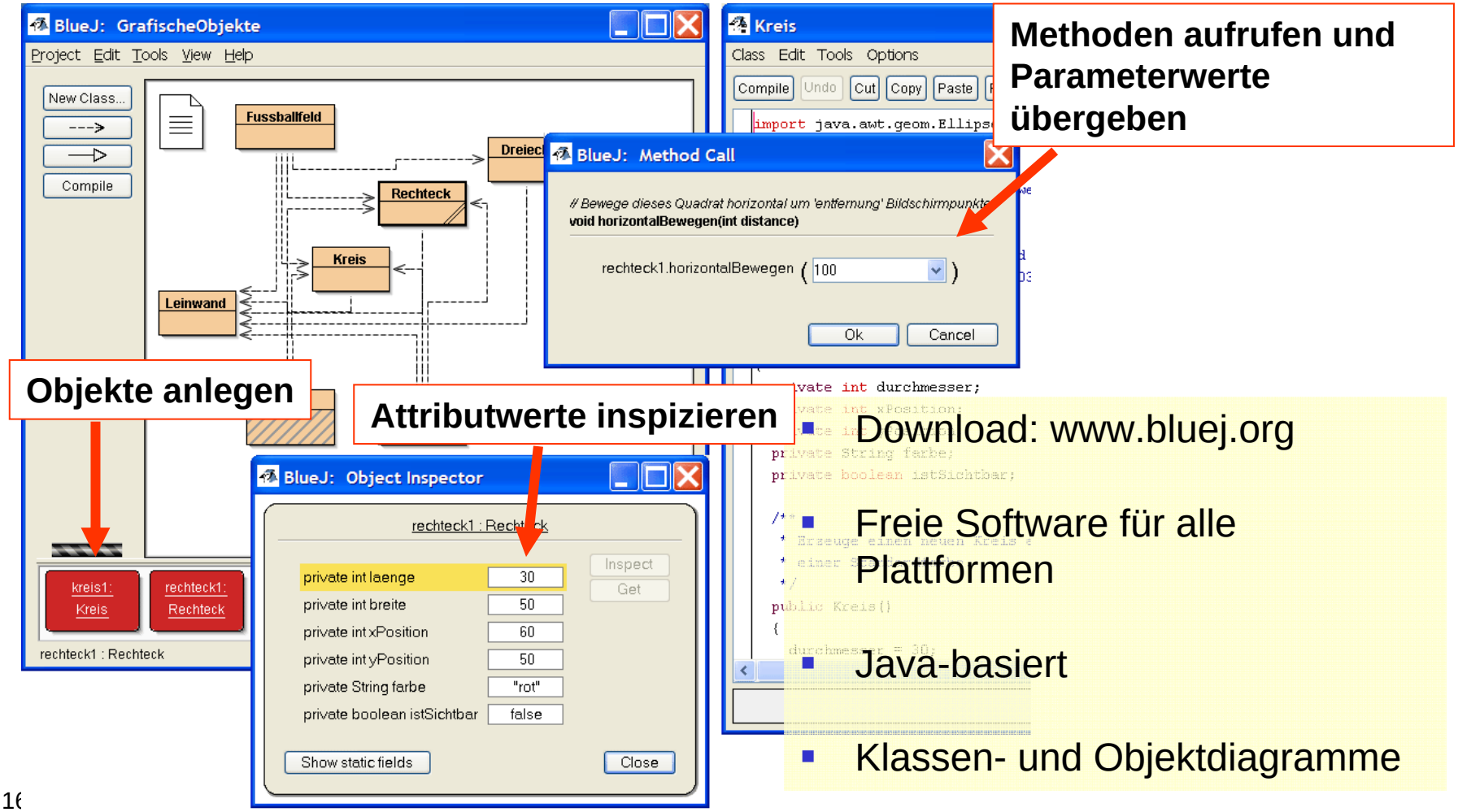
Programm bearbeiten | Geschwindigkeit: | Hilfe zeigen ...

```
// Zeichnet ein Auto und lässt es durch
// Verschieben der Einzelteile fahren
r:RECHTECK
k1,k2:KREIS
e:ELLIPSE
f:FENSTER

f.zeichne(r) f.zeichne(k1) f.zeichne(k2) f.zeichne

r.EckenSetzen(-40,30,40,10)
r.FüllfarbeSetzen(blau)

k1.RadiusSetzen(10)
k1.MittelpunktSetzen(-20,10)
k1.FüllfarbeSetzen(schwarz)

k2.RadiusSetzen(10)
k2.MittelpunktSetzen(20,10)
k2.FüllfarbeSetzen(schwarz)

e.MittelpunktSetzen(0,30)
e.RadiusxSetzen(20)
e.RadiusySetzen(10)
e.FüllfarbeSetzen(gelb)

wiederhole 150 mal
   r.verschieben(1,0)
   k1.verschieben(1,0)
   k2.verschieben(1,0)
   e.verschieben(1,0)
*wiederhole
```

EOS: „**Einfache objektorientierte Programmiersprache**“
http://pabst.heim.at/eos/index.htm



BlueJ



- Download: www.bluej.org
- Freie Software für alle Plattformen
- Java-basiert
- Klassen- und Objektdiagramme

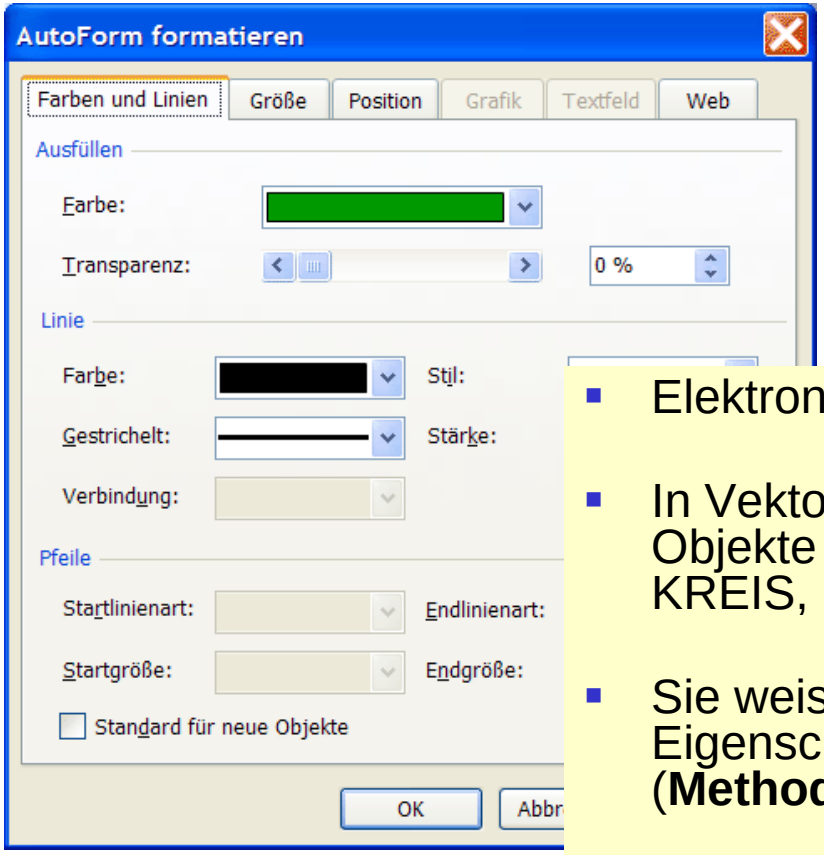
WIE SAG ICH'S MEINEN SCHÜLER(-INNEN)?

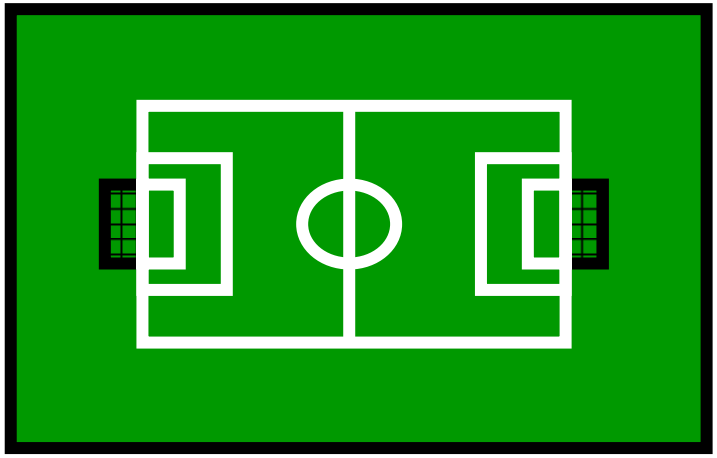
Didaktik der Informatik

3. Unterrichtskonzepte
.. in loser Folge!

3.1. Zustände von Objekten

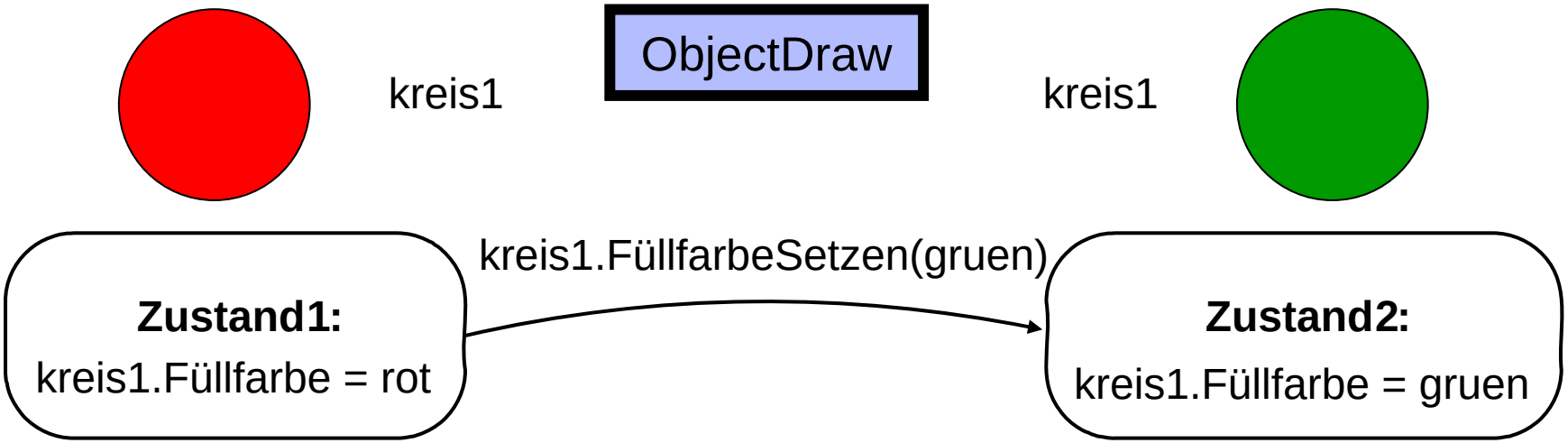
Wiederholung: Objekte in Vektorgrafiken





- Elektronische Dokumente bestehen aus **Objekten**.
- In Vektorgrafikdokumenten finden sich grafische Objekte vieler verschiedener Klassen: LINIE, KREIS, RECHTECK ...
- Sie weisen jeweils bestimmte charakteristische Eigenschaften (**Attribute**) und Fähigkeiten (**Methoden**) auf ...

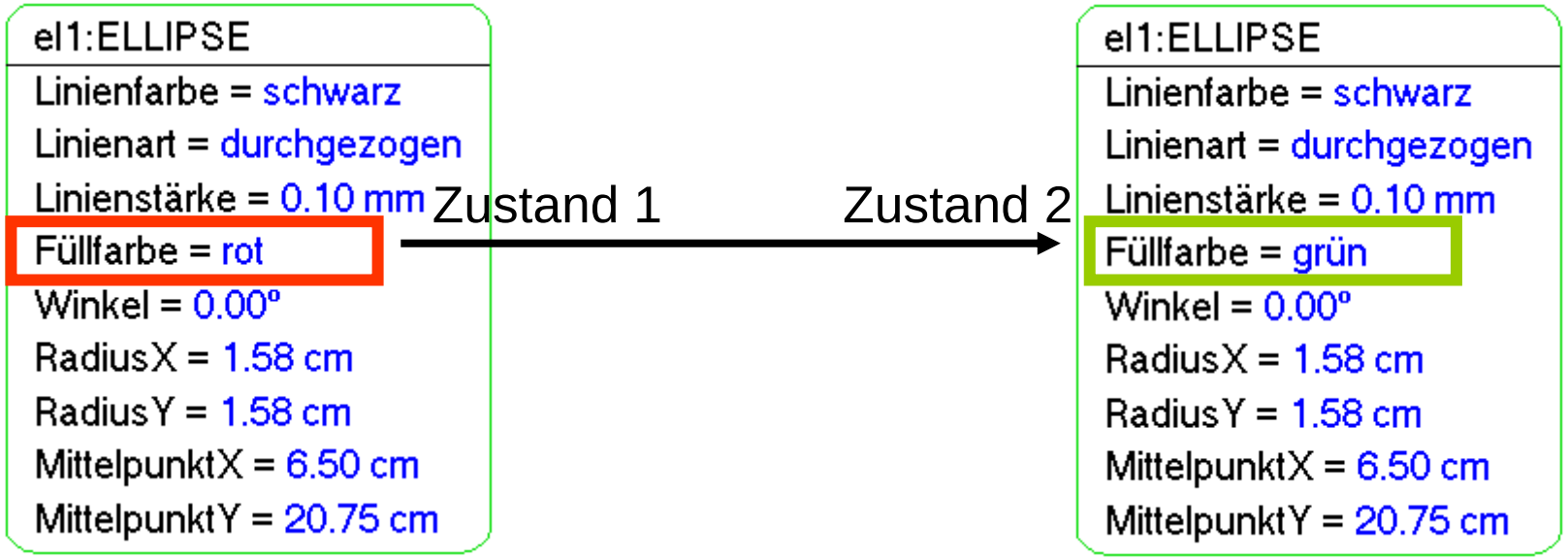
Zustände und Zustandsdiagramme



- 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“.

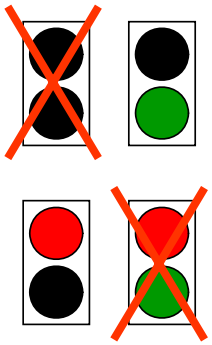
Zustände von Objekten

- 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.

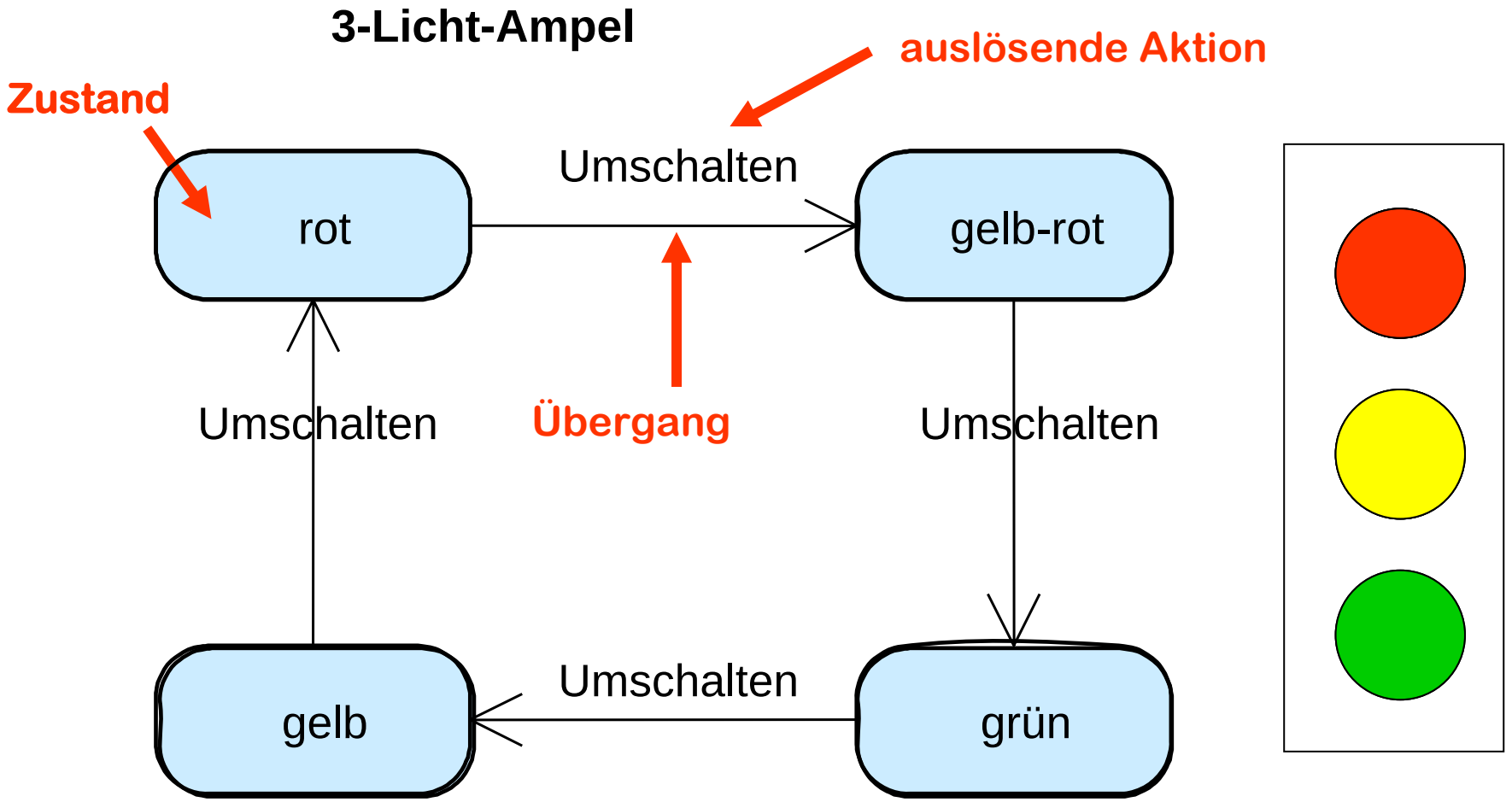


Anzahl der Zustände

- Wenn das Objekt O die Attribute a_1, .. a_n besitzt
  und diese jeweils k_1, .. k_n Zustände annehmen können,
  dann existieren für O genau $k_1 \cdot k_2 \cdot .. \cdot k_n$ Zustände.
- Beispiel: Eine Fußängerampel mit zwei Lichtern hat zwei Attribute:
  - FarbeOben mit den 2 Werten schwarz und rot
  - FarbeUnten mit den 2 Werten schwarz und gruen.
- Damit ergeben sich für dieses Objekt $2 \cdot 2 = 4$ mögliche Zustände:
  - schwarz-schwarz,
  - schwarz-gruen,
  - rot-schwarz,
  - rot-gruen.
- Technisch genutzt werden (im Betrieb) davon jedoch nur zwei:
  - schwarz-gruen (gehen!),
  - rot-schwarz (stop!)



Zustände, Übergänge und Aktionen



Vollständige Zustandsdiagramme

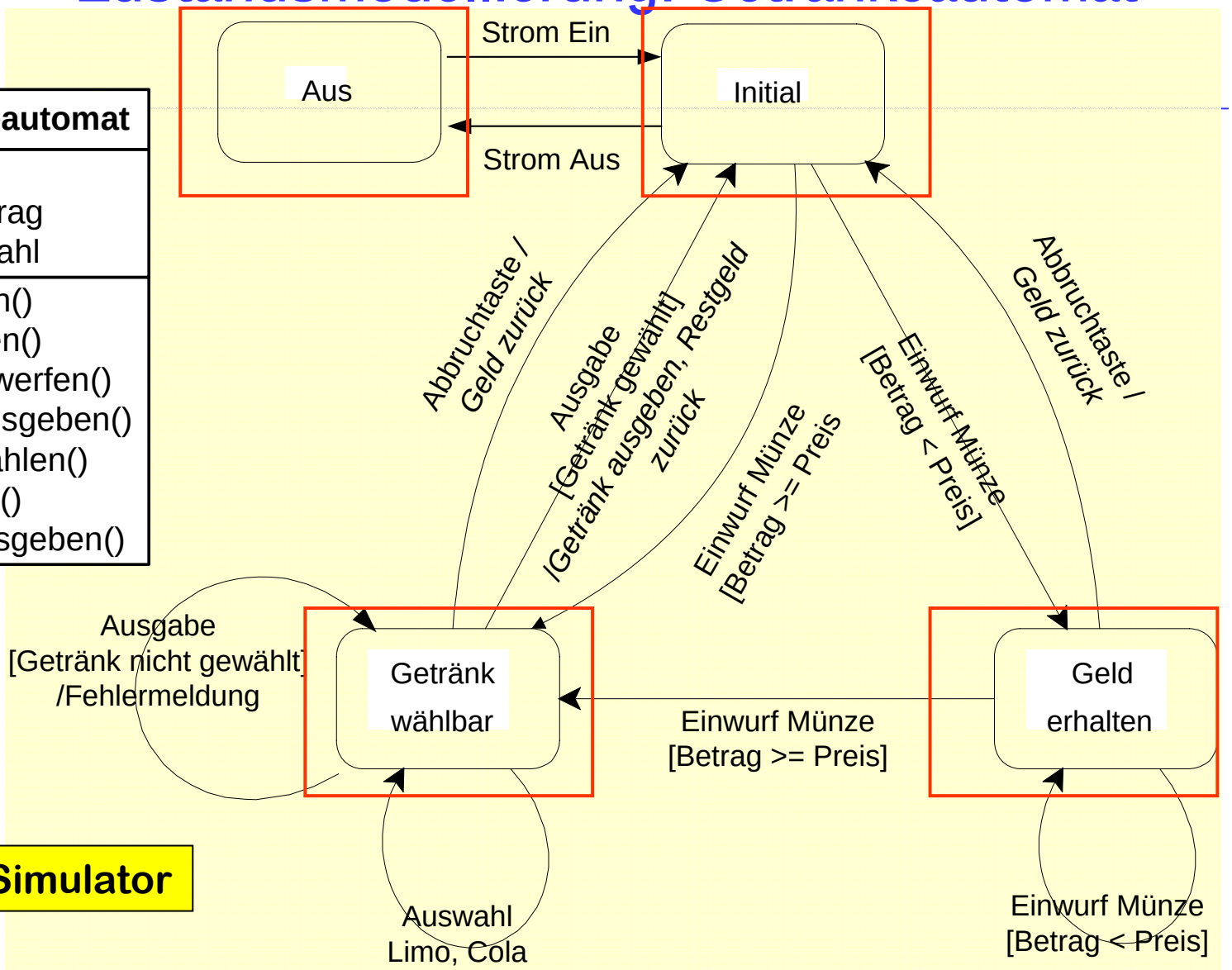
- Der Übergang von einem **Zustand z1** zu einem **Zustand z2** findet genau dann statt, wenn
  - sich das System im **Zustand z1** befindet und
  - die **auslösende Aktion a1** stattfindet und
  - unmittelbar vor dem Übergang die **Bedingung b** erfüllt ist.
- Dann wird mit dem Übergang auch die **Aktion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

Zustandsmodellierung: Getränkeautomat

Getränkeautomat

zustand
einwurfbetrag
getränkewahl

einschalten()
ausschalten()
münzeEinwerfen()
restgeldAusgeben()
getränkWählen()
abbrechen()
getränkAusgeben()



Java-Script-Simulator

3.3. Algorithmen

Algorithmen

- Algorithmen sind eine der ältesten (abstrakten) Beschreibungstechniken für Abläufe:
  - Benannt nach dem Mathematiker Al-Khwarizmi (ca. 780–850), der am Hof der Kalifen von Bagdad wirkte.
- Bereits in der Antike kannte man informelle algorithmische Beschreibungen für Rechenverfahren, z.B. den berühmten Euklidischen Algorithmus:
  - „Nimmt man beim Vorliegen zweier ungleicher Zahlen abwechselnd immer die kleinere von der größeren weg, so müssen, wenn niemals ein Rest die vorangehende Zahl genau misst, bis die Einheit übrig bleibt, die ursprünglichen Zahlen gegeneinander prim (teilerfremd) sein“

    aus Euklids Buch VII, §1, zitiert nach Gericke H.: Mathematik in Antike und Orient – Mathematik im Abendland. Fourier, Wiesbaden, 3. Aufl. 1994.

Beispiele:
Eingabe: 12, 8
12 – 8 = 4
8 – 4 = 4
4 – 4 = 0
⇒ 12, 8 nicht prim

Eingabe; 13, 9
13 – 9 = 4
9 – 4 = 5
5 – 4 = 1
⇒ 13, 9 prim

Definition

- „Ein **Algorithmus** ist ein Verfahren
  - mit einer präzisen (d.h. in einer genau festgelegten Sprache abgefassten)
  - endlichen Beschreibung
  - unter Verwendung
    - effektiver (d.h. tatsächlich ausführbarer)
    - elementarer (Verarbeitungs-) Schritte.“

  (nach Broy, M.: Informatik: Eine grundlegende Einführung, Band 1. Springer-Verlag, Berlin, 2. Auflage, 1998)

Beispiel

Wechselgeld-Algorithmus:

- **Eingabe**: Preis P, gezahlter Betrag G
- **Ausgabe**: Folge von Münzen oder Scheinen, deren Werte aufsummiert genau gleich Werte G-P sind.
- **Verfahren:**

(1) Berechne Z = G-P;
(2) Suche größte Münze M mit dem Wert M<=Z;
(3) Berechne Z-M;
(4) Falls Ergebnis = 0:
  Berechnung beenden
  sonst
  wiederhole ab (2) mit Z-M anstatt Z.

Beispielhafter Ablauf:

Eingabe: P = 0,80€; G = 2€

(1) Z = G-P = 1,20€

(2) M = 1€

(3) Z-M = 0,20€ > 0

(2) M = 20ct

(3) Z-M = 0

(4) fertig!

Ausgabe: 1€, 20ct.

Programme und Algorithmen

- Ein **Programm** ist **eine** (von vielen möglichen) Darstellungen eines bestimmten Algorithmus als Text in einer speziellen Sprache, die vom Rechner automatisch interpretiert werden kann: Eine solche Sprache heißt **Programmiersprache**.

- Falls ein Programm P einen Algorithmus A beschreibt, dann ist:
  - P eine Konkretisierung von A
  - A eine Abstraktion von P.

Struktur von Algorithmen: Bubblesort

Didaktik der

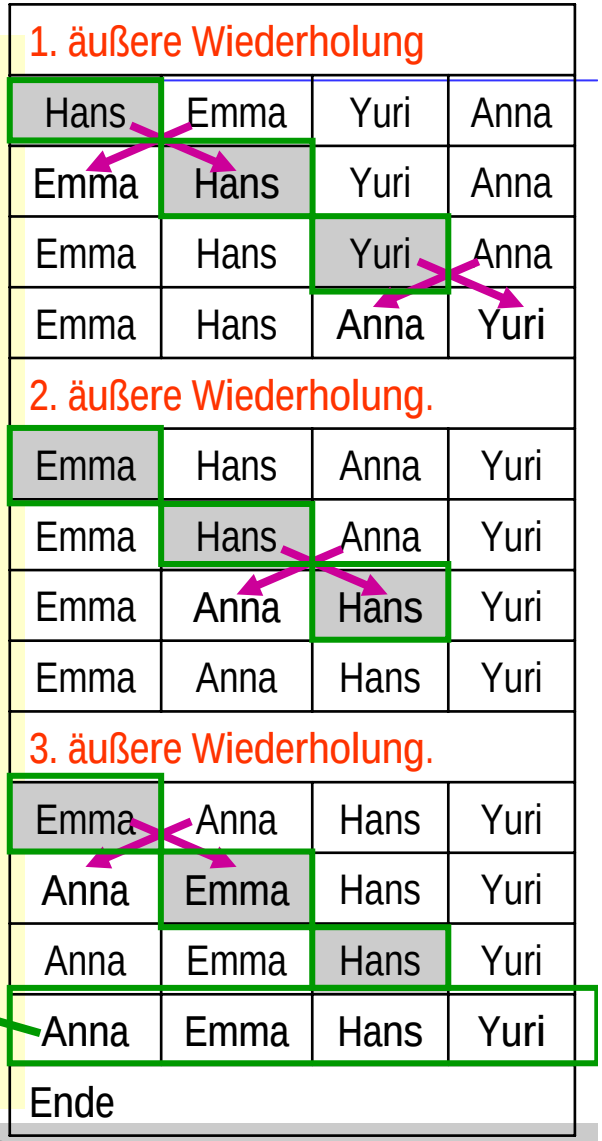
Algorithmus „Bubblesort“:

Wiederhole (Anzahl der Elemente der Liste – 1) mal
Wiederhole für alle Namen vom ersten bis zum vorletzten
Wenn der betrachtet Name alphabetisch hinter den folgenden gehört
Vertausche die beiden Namen
Ende Wenn
Ende Wiederhole
Ende Wiederhole
Ende Algorithmus

2. innere Wiederholung

Anwendung aus funktionaler Sicht:

Bubblesort(“Hans”, “Emma“, “Yuri”, “Anna”) =
(„Anna“, “Emma”, “Hans”, “Yuri”)



Struktur von Algorithmen: Unteilbare Schritte und Sequenzen

- Alle Algorithmusnotationen zeigen beim Ablauf gewisse strukturelle Gemeinsamkeiten: Es gibt ..

1. **elementare (unteilbare) Verarbeitungsschritte**

   z.B.: schalte rotes Licht aus

2. **Sequenzen** (Folgen) von Verarbeitungsschritten:

   z.B:

   schalte rotes Licht ein;

   warte eine Minute;

   schalte gelbes Licht ein;

KAROL-Programm:

```
Schritt
LinksDrehen
Schritt
Hinlegen
RechtsDrehen
Schritt
Hinlegen
```

Struktur von Algorithmen: Bedingte Verarbeitungsschritte

```
KAROL-Programm:

wenn IstZiegel dann
        LinksDrehen
        Schritt
        RechtsDrehen
        Schritt
        Schritt
        RechtsDrehen
        Schritt
        LinksDrehen
sonst
        Schritt
        Schritt
*wenn
```

- **Bedingte Verarbeitungsschritte:** Manche Verarbeitungsschritte sollen nur unter bestimmten Bedingungen ausgeführt werden:

  Wenn der betrachtet Name alphabetisch hinter den folgenden gehört dann

  Vertausche die beiden Namen

  Ende Wenn

- **Alternative:** Oft soll im Fall, dass die o.g. Bedingung **nicht** erfüllt ist, ein anderer Verarbeitungsschritt ausgeführt werden:

  Wenn $b \neq 0$ dann
  gib a/b aus
  sonst
  melde Fehler
  Ende Wenn

Struktur von Algorithmen: Wiederholungen

```
KAROL:

wiederhole 3 mal
  wenn IstZiegel dann
    LinksDrehen
    Schritt
    RechtsDrehen
    Schritt
    Schritt
    RechtsDrehen
    Schritt
    LinksDrehen
  sonst
    Schritt
    Schritt
  *wenn
*wiederhole
```

- Wiederholung mit (vor dem ersten Durchlauf) **festgelegter Anzahl von Durchläufen**:

  Wiederhole (Anzahl der Elemente der Liste – 1) mal

  Vertausche jedes Element mit dem folgenden

  Ende Wiederhole

- mit **Wiederholungs- oder Abbruchbedingung**

  Wiederhole bis Eingabe = „Ende“

  reagiere auf Eingabe

  Ende Wiederhole.

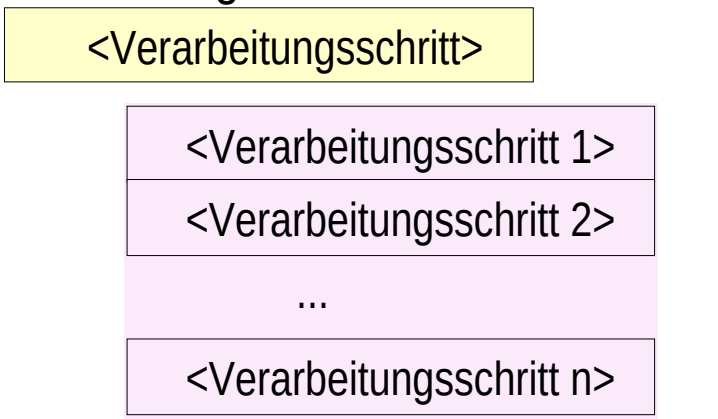
  ```
  wiederhole solange NichtIstWand
  ```

- Mit der zweiten Variante kann man mehr Funktionen berechnen als mit der ersten. So kann man hier z.B. die Wiederholung auf Wunsch des Benutzers beenden.

Pseudocode und Struktogramme

Strukturelement	Pseudocode
elementarer Verarbeitungsschritt	<Verarbeitungsschritt>
Sequenz	<Verarbeitungsschritt 1>; <Verarbeitungsschritt 2>; ... <Verarbeitungsschritt n>;
Bedingter Verarbeitungsschritt	Wenn <Bedingung> Dann<Verarbeitungsschritt1> Sonst<Verarbeitungsschritt2> Ende Wenn
Wiederholung	Wiederhole solange <Bedingung> < Verarbeitungsschritt > Ende Wiederhole

Struktogramm-Element



Mehrfache Auswahl

- Anstatt einer geschachtelten Folge bedingter Anweisungen gibt es in vielen Algorithmusnotationen die mehrfache Auswahl:

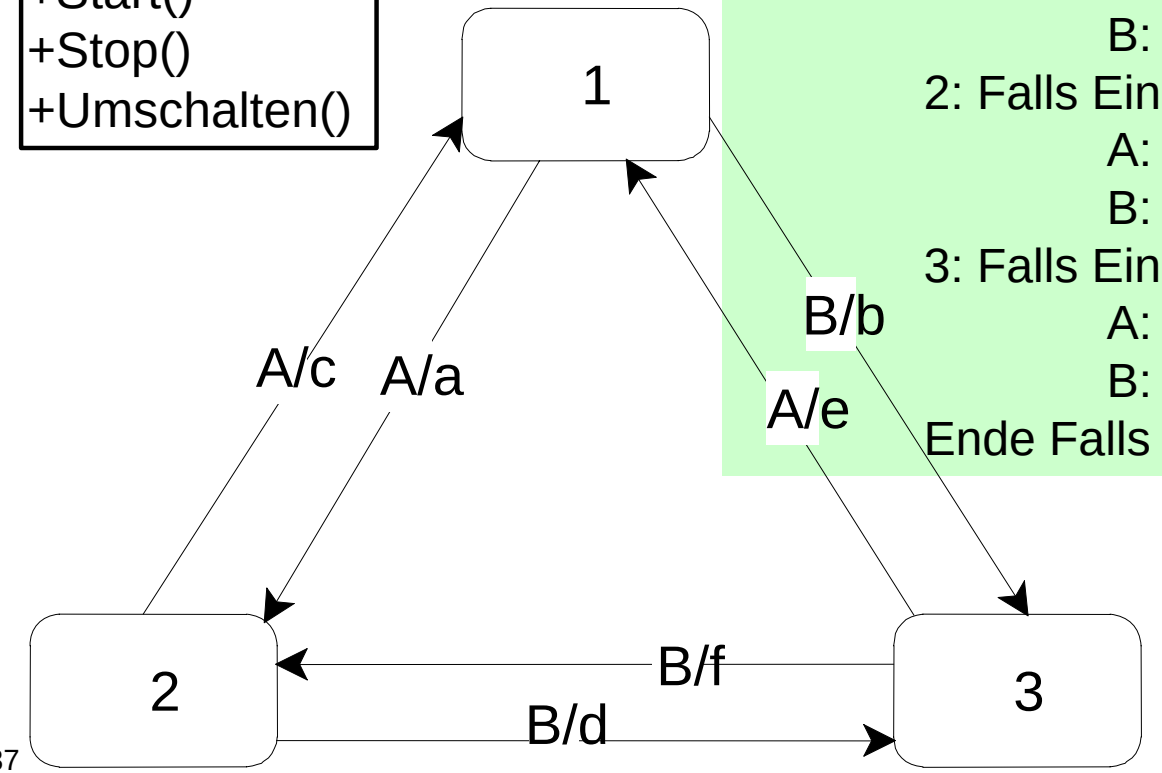
- Falls Zahl =

  1: Anweisung A
  2: Anweisung B
  3: Anweisung C
  [sonst: Anweisung D]
  Ende Falls

- Wenn Zahl = 1 dann Anweisung A
  sonst wenn Zahl = 2 dann Anweisung B
  sonst wenn Zahl = 3 dann Anweisung C
  [sonst Anweisung D]
  Ende wenn
  Ende wenn
  Ende wenn.

Algorithmen und Automaten

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




```
Wiederhole für immer
Falls Zustand =
        1: Falls Eingabe =
                A: 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
```

Didaktik der Informatik

3.4. Programmierung

Automatisierung durch Programmierung

- Maschinen (z.B: Roboter) werden **programmiert**, um Abläufe zu automatisieren
- Dies geschieht aus unterschiedlichen Motiven:
  - Einsparung von **Ressourcen** (Zeit, Geld): Maschinen arbeiten meist schneller und oft billiger als Menschen
  - **Sicherheit**: Manche Arbeiten sind für Menschen zu riskant (Brennstabwechsel in Kernkraftwerken)
  - **Bequemlichkeit**: So kann man es Menschen ersparen, z.B. sehr monotone oder unangenehme Arbeiten übernehmen zu müssen.
- Nachteil: Damit fallen immer auch Arbeitsplätze weg.z.B. Automobilindustrie:
  - Vor 50 Jahren arbeiteten Hunderte von Menschen in einer Fertigungshalle, heute weniger als 10.

Programmiersprachen

- Diese Programme liegen ebenso wie die Daten, mit denen sie arbeiten, im **Arbeitsspeicher** des Rechners („**von-Neumann-Prinzip**“).
- Dort sind Programme und Daten als Bitmuster durch digitale Spannungszustände dargestellt.
- Da die direkte Eingabe dieser Bitmuster sehr mühsam wäre, hat man so genannte **höhere Programmiersprachen** entwickelt, die eher die Struktur der Aufgabenstellung als die der Rechenanlage widerspiegeln.

Höhere Programmiersprachen

```
public void umschaltenRot()
    {
        lichtOben.farbeAendern("rot");
        lichtUnten.farbeAendern("schwarz");
        zustand = "ROT";
    }
```

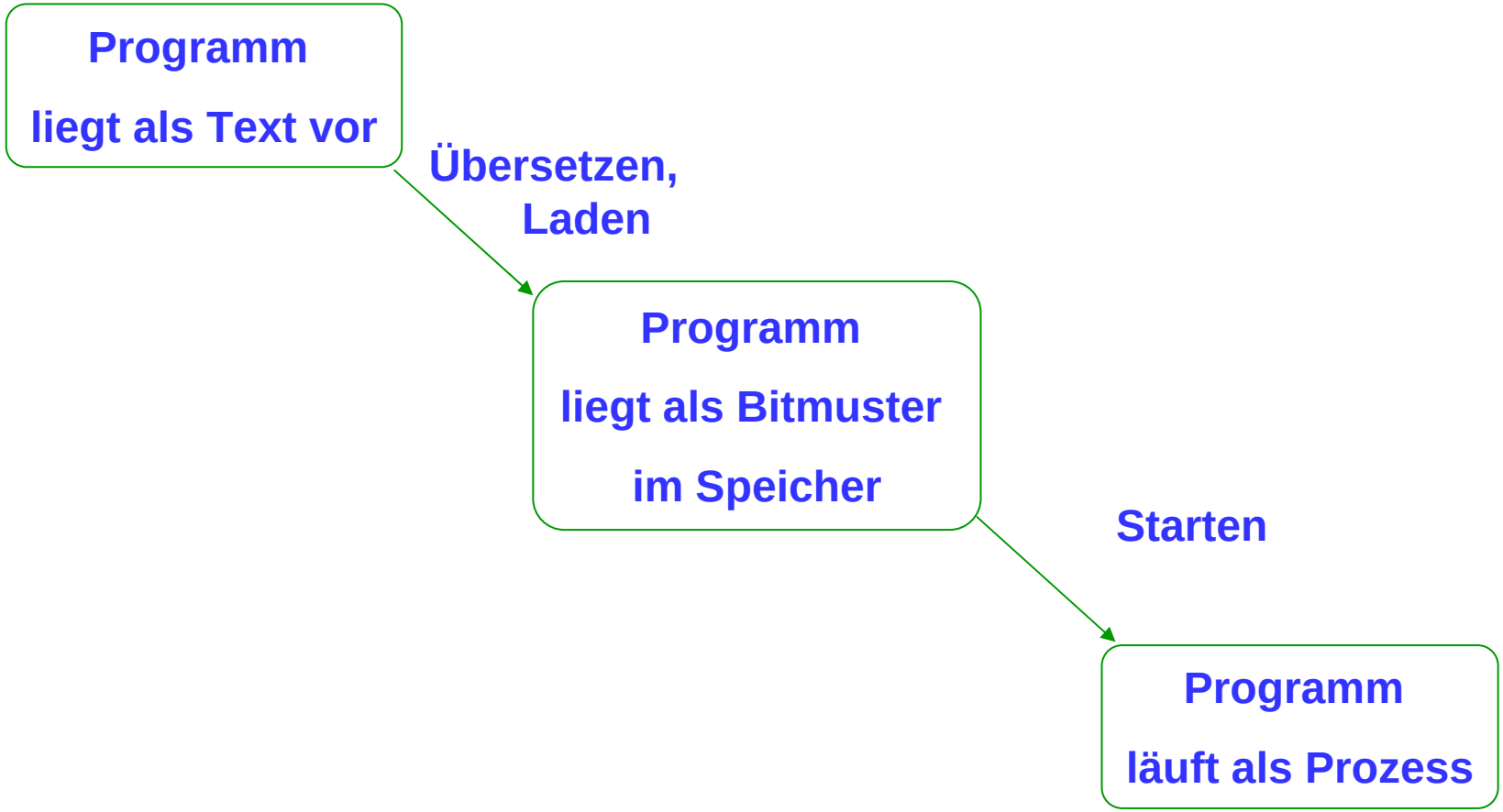
- Ein Programm einer höheren Programmiersprache (z.B. Java, C. C++. Basic, Pascal): besteht aus einer Folge von Anweisungen in einer **für den Menschen lesbaren** Darstellung.
- Wenn dieses Programm von der Maschine ausgeführt werden sollen, muss es in eine maschinenlesbare Form transformiert werden, d.h. in eine Bitfolge, z.B. 01010011 00101111 11001010 00101010 ...
- Dieser „Maschinencode“ hängt aber (im Gegensatz zum Text des Java-Programms) sehr stark von der jeweils verwendeten Maschine ab.

Übersetzerprogramme

- Für die Umwandlung von Programmen einer höheren Programmiersprache in solche Bitfolgen gibt es spezielle Programme:
- **Compiler**: **Vor der Ausführung des Programms** wird der **gesamte Programmtext** aus der höheren Programmiersprache in Maschinencode übersetzt, der im Hauptspeicher des Rechners aufbewahrt (oder auch für spätere Verwendung auf die Festplatte ausgelagert) wird, bis seine tatsächliche Abarbeitung beginnt (z.B. Java):
  1. **Schritt: Übersetzen**
  2. **Schritt: Starten**
- **Interpreter** übersetzen die Anweisungen des Programmtextes **während der Ausführung des Programms einzeln** und sofort führen sie sofort aus (z.B. Karol):

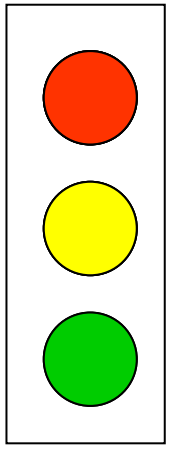
  Programm Starten, Übersetzen während des Ablaufs

Zustandmodell: Vom Programm zum Prozess



Simulation durch Programme

- Mit Hilfe von **Computerprogrammen** kann man das Verhalten realer Objekte (oder Systeme) bis zu einem gewissen Grad nachbilden (Simulation).
- Solche **Simulationsverfahren** werden in der Forschung und in der Industrie sehr häufig genutzt, z.B. zur Nachbildung des Verhaltens von Brücken bei Belastungen, für die Wettervorhersage oder für die Flugbahn von Satelliten.



Simulation der 3-Licht-Ampel mit EOS

Programme und Abläufe

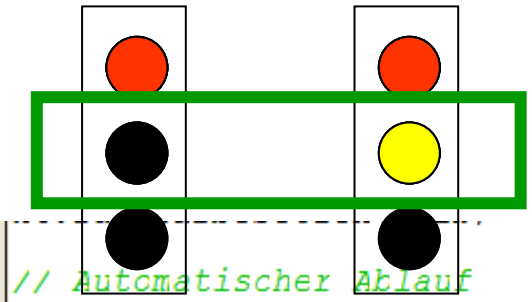
Programm starten

```
r:RECHTECK
f:FENSTER
f.zeichne(r)
r.eckenSetzen(-20,-10,20,10)
r.füllfarbeSetzen(weiß)

z:Integer

wiederhole 100 mal
   wenn zufall(-100,100)<0 dann
      r.füllfarbeSetzen(gruen)
      wiederhole 10 mal
         r.Verschieben(-1,0)
      *wiederhole
      sonst
      r.füllfarbeSetzen(rot)
      wiederhole 10 mal
         r.Verschieben(1,0)
      *wiederhole
   *wenn
*wiederhole
```

EOS-Programm „Zufall“

- Ein Programm (hier „Zufall“ in EOS) beschreibt einen oder mehrere **mögliche Abläufe** eines Systems (z.B. eines Roboters oder einer animierten Grafik).
- Ein Ablauf ist eine Folge von Zuständen der beteiligten Objekte
- Nach jedem Start eines Programms wird genau einer dieser Abläufe realisiert.
- Wenn es mehrere mögliche Abläufe gibt, entscheiden Rahmenbedingung zur Laufzeit (z.B. Eingaben oder zufällige Zwischenergebnisse) welcher dieser Abläufe eingeschlagen wird.

Objekte und Programme

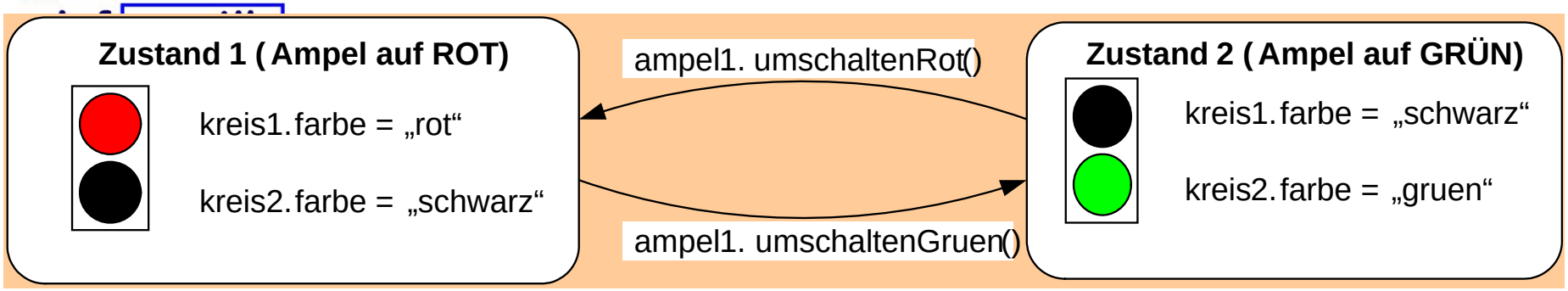


```
// Automatischer Ablauf
wiederhole immer
   //Zustand rot
   k1.füllfarbeSetzen(rot)
   k2.füllfarbeSetzen(schwarz)
   //warten
   k4.mittelpunktSetzen(-100,100)
   wiederhole 20 mal
      k4.verschieben(0,-10)
   *wiederhole
   //Zustand rot-gelb
   k2.füllfarbeSetzen(gelb)
   //warten
   k4.mittelpunktSetzen(-100,100)
   wiederhole 20 mal
      k4.verschieben(0,-10)
   *wiederhole
   //Zustand grün
   k1.füllfarbeSetzen(schwarz)
   k2.füllfarbeSetzen(schwarz)
   k3.füllfarbeSetzen(grün)
```

- Der Ablauf eines **objektorientierten Programms** besteht aus einer Folge von Zuständen der beteiligten (Programm-) Objekte.
- Die Zustandsübergänge werden durch die Änderung von Attributwerten dargestellt und durch den Aufruf von Methoden ausgelöst.
- Beispiel (3-Licht-Ampel): Übergang vom Zustand rot zum Zustand rot-gelb
  - Realisierung durch Wechsel der Füllfarbe von k2 von schwarz auf gelb
  - Auslösung durch Aufruf von k2.füllfarbeSetzen(gelb)

Didaktik der

Reale Objekte und Programmobjekte



- Das **reale Objekt** „Fußgängerampel“ hat 2 Zustände: ROT, GRÜN
- Zur Simulation benötigt man ein **Programmobjekt** mit 2 Attributen:
  kreis1.Füllfarbe,
  kreis2.Füllfarbe
  mit jeweils z.B. 16 möglichen Werten (Farben), also mit insgesamt z.B. **256** Zuständen.

kreis1.farbe → ↓ kreis2.farbe	rot	schwarz	blau	gelb
rot				
schwarz	Zustand 1: Ampel ROT			
blau				
gelb				
gruen		Zustand 2: Ampel GRÜN		
lila				
weiß				

3.5. OO-Programmierung

Klassenkarte und Klassendefinition

Zugriffsmodifikatoren

Kreis
durchmesser; xPosition; yPosition; farbe; istSichtbar;
Kreis() sichtbarMachen() ... unsichtbarMachen() ... nachRechtsBewegen() ... nachLinksBewegen() ... ... loeschen()

Klassendefinition im Java-Programmm

Typangaben

```
// Klassenbezeichner
public class Kreis

// Attribute
private int durchmesser;
private int xPosition;
private int yPosition;
private String farbe;
private boolean istSichtbar;


// Methoden
public Kreis()
public void sichtbarMachen() ...
public void unsichtbarMachen() ...
public void nachRechtsBewegen() ...
public void nachLinksBewegen() ...
...
private void loeschen()
```

Zugriffsmodifikatoren

- Vor den Attribut- und Methodenbezeichnern steht einer der beiden **Zugriffsmodifikatoren private** oder **public**
- **private**: Nur Objekte der gleichen Klasse können
  - auf das Attribut lesend oder schreibend zugreifen
  - die Methode aufrufen.
- **public**: freier Zugriff für alle anderen Objekte.
- **Ziel ist Kapselung:** Abschottung der inneren Strukturen von Objekten nach außen (Sicherheit, Arbeitsteilung).
  - **arbeitsteilige Entwicklung** von großen Programmen
  - **Sicherheitsaspekte**
- In der Regel definiert man
  - Attribute als **private**
  - Methoden als **public**.

```
private int centerY
public void moveUp()
```

Typangaben

```
private int centerY
public void moveUp()
```

- Nach dem Zugriffsspezifikator folgt eine **Typangabe** für die Werte, z.B.:
  - **int** für ganze Zahlen wie 12, -34225, 65544 usw.
  - **float** für Fließkommazahlen (d.h. Dezimalbrüche) wie 3.17 oder - 0.1124
  - **String** für Texte wie „Hallo“, guten Tag“ oder „Hans“
  - **boolean** für die Wahrheitswerte True bzw. False
- Bei Methoden wird damit der Typ des Rückgabewertes der Methode festgelegt, z.B.:
  - RÖMISCH hat Rückgabewerte vom Typ String
  - RUNDEN hat Rückgabewerte vom Typ float
- **void** gibt an, dass die Methode **keinen** Rückgabewert liefert.
- Die Anweisung `return x` in einer Methode bewirkt, dass diese den Wert von x als Rückgabewert liefert.

NEUER TEXT! >>

Konstruktoren

```
public Kreis()
```

- Für die Erzeugung von Objekten benötigt man in vielen Programmiersprachen eine spezielle Methode, die man **Konstruktur** nennt.
- Oft erledigt man damit auch gleich die Belegung der Attribute mit Anfangswerten (Initialisierung).
- Da das anzulegende Objekt zum Zeitpunkt des Aufrufs dieser Methode ja noch nicht existiert, muss eine Konstruktormethode der jeweiligen **Klasse** anstatt einem Objekt zugeordnet werden (Klassenmethode).
- Daher wird sie oft mit dem gleichen Bezeichner wie die Klasse (z.B. Kreis) versehen und (im Gegensatz zu den anderen Methoden) groß geschrieben.

Wertzuweisung

- **Wertzuweisung:** Anweisung zur Änderung des Wertes eines Attributes
- **Leider** wird sie in vielen Sprachen (z.B. C, C++ oder Java) durch das Symbol „=“ dargestellt
- **Beispiel**:
  `kreis1.farbe = „rot“`
  weist dem Attribut *farbe* des Objektes *kreis1* den Wert „rot“ zu.
- Auf der linken Seite einer Zuweisung steht immer ein Attributbezeichner, auf der rechten Seite ein Term, dessen Wert berechnet und dem Attribut auf der linken Seite als Wert zugewiesen wird, z.B.
- `linie1.laenge = 3+5*7`

Wirkung einer Zuweisung

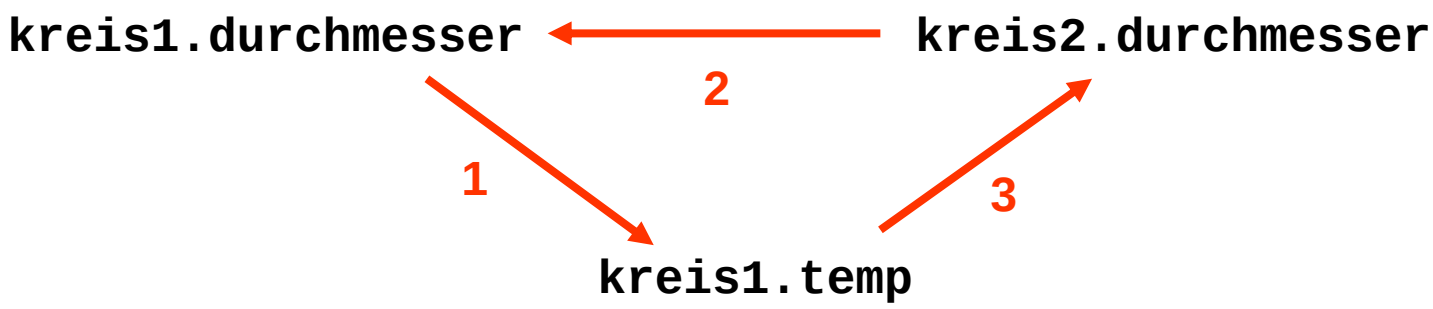
- Bei einer Zuweisung geht der vorherige Wert des jeweiligen Attributes verloren.
- Nach der letzten Zuweisung haben beide Kreise denselben Durchmesser. Der Wert 10 ist also verloren gegangen.

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

Ringtausch

- Will man die Werte zweier Attribute austauschen, so benötigt man ein **drittes Attribut** zum *Zwischenspeichern* desjenigen Wertes, der ansonsten überschrieben werden würde:



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

Ändern von Attributwerten

```
public class Kreis {
private int durchmesser;
...
public void groesseAendern(int neuerDurchmesser) {
                loeschen();
                durchmesser = neuerDurchmesser;
                zeichnen();
        }
```

- Attribute sollten der Regel von außen (d.h. von Objekten anderer Klassen) nicht über direkte Wertzuweisungen verändert werden können.
- Daher werden sie meist durchwegs als **private** gekennzeichnet.
- Die eigentliche Wertzuweisung wird in **spezielle Setze-Methoden** eingebaut (z.B. groesseAendern).
- In diese Setze-Methoden baut man daneben oft diverse Sicherheitsmaßnahmen ein, z.B. Passwortabfragen, Typüberprüfungen, Abfangen problematischer Werte wie 0 bei Divisionen.

Gleichheitssymbol

- Die Verwendung des mathematischen Gleichheitssymbols „=“ für die Zuweisung ist sicher eine der unglücklichsten Entscheidungen in der Geschichte der Programmiersprachen.
- Eine der negativen Konsequenzen ist, dass man für die Gleichheitsaussage ein anderes Symbol verwenden muss. In Java ist das das doppelte Gleichheitszeichen:
- **`kreis1.durchmesser == 5`** steht für die folgende **Aussage**:
- „Das Attribut **`kreis1.durchmesser`** hat den Wert 5“
- Zu einem bestimmten Zeitpunkt ist diese Aussage entweder wahr oder falsch.

Didaktik der Informatik

Sequenzen

```
public void umschaltenRot()
    {
        lichtOben.farbeAendern("rot");
        lichtUnten.farbeAendern("schwarz");
        zustand = "ROT";
    }
```

- In C, C++ und Java werden Sequenzen durch geschweifte Klammern begrenzt.
- Die einzelnen Anweisungen einer Sequenz werden durch Strichpunkte voneinander getrennt.

Bedingte Anweisung

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

public void umschaltenRot()
{
    lichtOben.farbeAendern("rot");
    lichtUnten.farbeAendern("schwarz");
    zustand = "ROT";
}
public void umschaltenGruen()
{
    lichtOben.farbeAendern("schwarz");
    lichtUnten.farbeAendern("gruen");
    zustand = "GRUEN";
}
```

- **Aufgabe**: Vereinfachen Sie die Bedienung Ihrer 2-Lichtampel durch den Einbau einer Methode umschalten, die Objekte der Klasse Ampel je nach Ausgangszustand jeweils den passenden Folgezustand versetzt. Verwenden Sie dabei die Alternative in Java:

```
if (<Bedingung>)
{Anweisungsfolge}
else
{Anweisungsfolge}
```

Wiederholung

```
public void setzeI(int neuerWert)
    {
        i=neuerWert;
    }
    public void warte (int msec){
        Leinwand leinwand = Leinwand.gibLeinwand();
        leinwand.warte(msec);
    }
     public void ablauf(int anzahl, int wartezeit)
    {
        umschaltenRot();
        setzeI(1);
        while (i<anzahl)
        {
            umschalten();
            setzeI(i+1);
            warte(wartezeit);
        }
    }
```

- **Aufgabe**: Automatisieren Sie den Ablauf der Ampel vollständig durch eine Methode ***ablauf***.
- Verwenden Sie dazu die Wiederholung in Java:

```
while <Bedingung>
   {Anweisungsfolge}
```

Mehrfache Auswahl

```
public void umschalten()
    {
        switch (zustand) {
            case 1:umschaltenRotGelb();
            break;
            case 2:umschaltenGruen();
            break;
            case 3:umschaltenGelb();
            break;
            case 4:umschaltenRot();
            break;
        }
    }
```

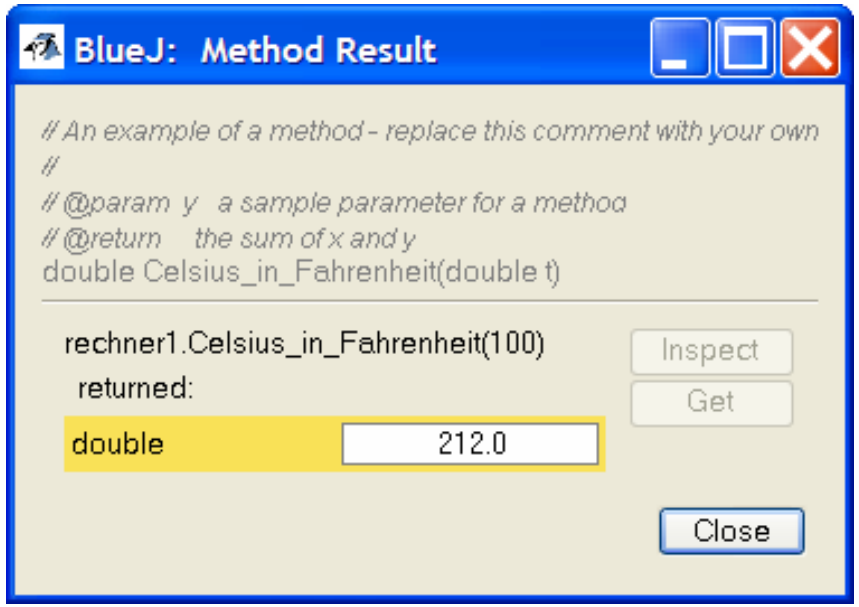
Beispiel: 3-Licht-Ampel

- Bisher: zwei Modellzustände (rot, grün)
- Bei mehreren Zuständen benötigt man die Kontrollstruktur „mehrfache Auswahl“:
- Leider muss der steuernde Ausdruck in der switch-Anweisung (hier das Attribut Zustand) zwingend vom Typ int sein
- Zustände müssen also durch Zahlen dargestellt werden

Ausgabe von Werten

```
public double Celsius_in_Fahrenheit(double t)
    {
        return t*1.8+32;
    }
```

Datentyp `double`: Standard-Fließkommatyp von Java (15 Stellen); `float` hat nur 7 Stellen.



- Für die Ausgabe von Werten kann oft das BlueJ-Ausgabefenster genutzt werden

Ausgabemethode

```
// Projekt Rechnen; Klasse Rechner
public void Celsius_in_Fahrenheit(int t)
{
   System.out.println(t+"°C entsprechen");
   System.out.println(t*1.8+32+" Grad Fahrenheit");
}
```

- Für die Ausgabe von Texten oder Zahlen steht die Methode **`System.out.println()`** zur Verfügung.
- Wenn sie mehrere Ausgaben hintereinander ausgeben soll (z.B. Texte oder Zahlen), dann werden diese mit Hilfe des Operatorsymbols + verkettet.

Lokale Variablen („Hilfsattribute“)

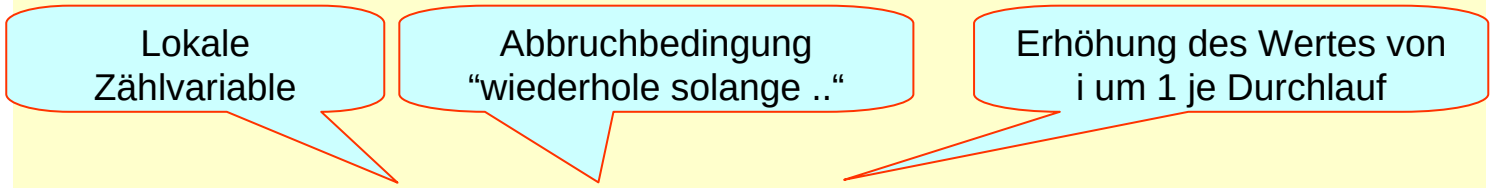
```
public void wechselgeldAuszahlen(int preis, int einwurf){
// Vereinfachung: es gibt nur Münzen zu 10, 50, 100, 200ct.
// und Preise in Vielfachen von 10ct.
int m = 0;
int z = einwurf-preis;
while (z>0) {
    if (z>=200) {
        m = 200;
        }  else {
            if (z>=100) {
            m = 100;
            }  else {
                if (z>=50){
                    m = 50;
                    }  else
                        m = 1
                        }
                }
            }
            System.out.print
            z = z-m;
    }
}
```

- Oft benötigt man für eine Berechnung innerhalb einer Methode eine Hilfsvariable (z.B. für ein Zwischenergebnis), deren Wert aber für den Zustand des Objektes keine Rolle spielt.
- Dann deklariert man innerhalb der jeweiligen Methode eine Variable, die mit dem Aufruf der Methode angelegt und mit deren Ende wieder aufgelöst wird.
- Solche Variablen heißen dann **lokal** (im Gegensatz zu globalen Variablen, die im ganzen Programm gültig sind.

Wiederholung mit fester Wiederholungszahl

```
public void Zinsen (double kapital, double zinssatz, int jahre) {
    double kontostand = kapital;
    for (int i=1; i<=jahre; i++){
        kontostand = kontostand*(1+zinssatz/100);
    }
    System.out.println("Kontostand nach "+jahre+" Jahren:");
    System.out.println(kontostand+" €");
}
```

- Wiederholungen mit fester Anzahl von Durchläufen:




```
for (int i=1; i<=jahre; i++){ .. }
```

- Äquivalent zu „Wiederhole für i=1 bis i=jahre mit Schrittweite 1“

Felder (arrays)

```
public void bubbleSort(){
    String temp;
    String[] name = new String[5];
    //Eingabe der Namen
    name[1] = "Hans";
    name[2] = "Emma";
    name[3] = "Yuri";
    name[4] = "Anna";
    //Sortieren
    for (int i=1;i<4;i++){
        for (int k=1; k<4; k++) {
            if
(name[k].compareTo(name[k+1])>0){
                temp = name[k];
                name[k] = name[k+1];
                name[k+1] = temp;
            }
        }
    }
    //Ausgabe
    for (int i=1;i<=4;i++){
        System.out.println(name[i]);
    }
}
```

- Oft will man mehrere Variablen mit einem Befehl ansprechen, z.B. eine Reihe von Namen in einer Wiederholung.
- Dann gibt man diesen Variablen denselben Bezeichner (z.B. `name`) und greift über einen Index auf die einzelnen Werte zu, z.B. `name[3] =„Yuri“`)
- `compareTo(String s2)` ist eine Methode der Klasse `String`, die den Wert des aufgerufenen Objektes (z.B. s1) mit dem Wert von s2 vergleicht. Das Ergebnis ist negativ, wenn s1 alphabetisch vor s2 steht, Null bei Gleichheit und positiv, wenn s1 nach s2 steht.

Didaktik der Informatik

Rekursive Methoden

```
public int ggT(int zahl1, int zahl2) {
    if (zahl1 == zahl2) {
       return zahl1;
       } else {
          if (zahl1 > zahl2) {
             return ggT(zahl1-zahl2, zahl2);
             } else {
                return ggT(zahl1, zahl2-zahl1);
         }
    }
}
```

Methodenaufrufe

ggT(12, 8)	ggT(13, 9)
↓	↓
ggT(12- 8, 8)	ggT(13- 9, 9)
↓	↓
ggT(4, 8- 4)	ggT(4, 9- 4)
↓	↓
4	ggT(4, 5- 4)
	↓
	ggT(4- 1, 1)
	↓
	ggT(3- 1, 1)
	↓
	ggT(2- 1, 1)
	↓
	1

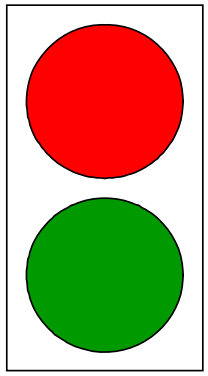
Eingabe: 12, 8
12 – 8 = 4
8 – 4 = 4
4 – 4 = 0
⇒ 12, 8 nicht prim

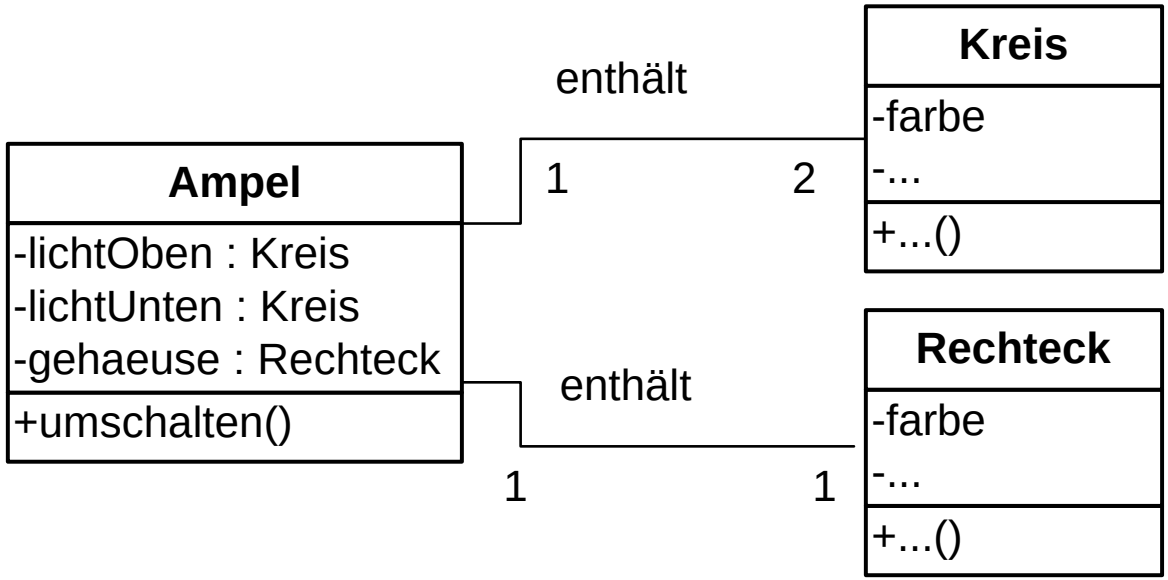
Eingabe; 13, 9
13 – 9 = 4
9 – 4 = 5
5 – 4 = 1
⇒ 13, 9 prim

- Eine Methode heißt **rekursiv**, wenn ihre Ausführung (mindestens) einen weiteren Aufruf derselben Methode (eventuell mit anderen Parameterwerten) bewirkt.
- Bei jedem Aufruf einer solchen Funktion wird ein **neuer Prozess** erzeugt.

3.6. Beziehungen

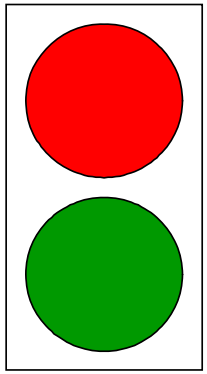
Zusammengesetzte Objekte





- „**Enthält“-Beziehung**: Ein Objekt der Klasse `Ampel` enthält
  - zwei Objekte der Klasse `Kreis: lichtOben, lichtUnten` und
  - ein Objekt der Klasse `Rechteck: gehaeuse`.

Zusammengesetzte Objekte



```
public class Ampel
{
   private Kreis lichtOben;
   private Kreis lichtUnten;
   private Rechteck gehaeuse;
   ..
}
```

- In Programmobjekten wird die enthält-Beziehung durch **Attribute** des enthaltenden Objektes (hier der Klasse `Ampel`) realisiert.
- Der Wert dieses Attributes ist das enthaltene Objekt (hier z.B. der Klasse `Kreis`).
- Der Typ dieses Attribute entspricht der Klasse der enthaltenen Objekte (hier z.B. `Kreis`).

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). Einweiteres 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).

Referenzen in Java

```
public class Ampel
{
   private Kreis lichtOben;
   private Kreis lichtUnten;
   private Rechteck gehaeuse;
    /**
..}
```

Objektbezeichner

Attributbezeichner

Attributwert

- In der OOP verwendet man (technisch, für den Benutzer unsichtbar) als Wert der **Referenz** die Nummer der Speicherzelle, ab der das Objekt im Hauptspeicher abgelegt ist.
- So werden z.B. Objekte, die in anderen Objekten enthalten sind, durch **Referenzen** in diese eingebunden.
- In unserem Beispiel wird also das z.B. Objekt `lichtOben` der Klasse Kreis nicht in das Objekt ampel1 kopiert, sondern **nur seine Adresse** im Speicher des Rechners als Wert des Attributes eingetragen.

Arbeitsspeicher

Objekt ampel1

Objekt rotesLicht

Nr. der Zelle	Inhalt
10000	ampel1
...	
10010	lichtOben
10020	10200
...	
10200	lichtOben
10210	farbe
10220	„rot“
...	
10300	lichtUnten
10310	farbe
10320	„gruen“

Mehrfache Referenzierung

Nr. der Zelle	Inhalt
10000	ampel1
10001	kreis1
10010	10200
...	
10100	ampel2
10110	kreis1
10120	10200
...	
10200	kreis1
10210	farbe
10220	„blau“

- Wenn es in einem Kino zwei Eingänge gibt, an denen je ein Platzanweiser je einem Kunden eine Referenz auf denselben Sitzplatz übergeben, dann bekommen die beiden Kunden Probleme.
- Ähnlich problematisch wird es, wenn ein Objekt von zwei anderen Objekten referenziert wird, z.B.:

  `ampel1` enthält **`kreis1`**
  `ampel2` enthält **`kreis1`**

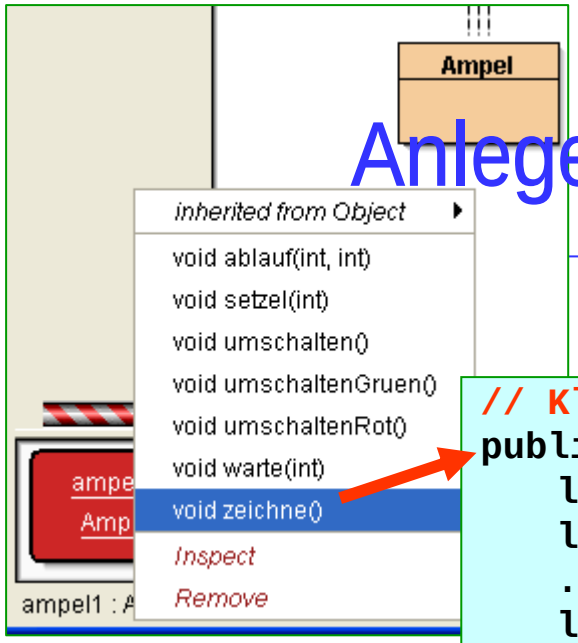
- Nach folgender Zuweisungsfolge:

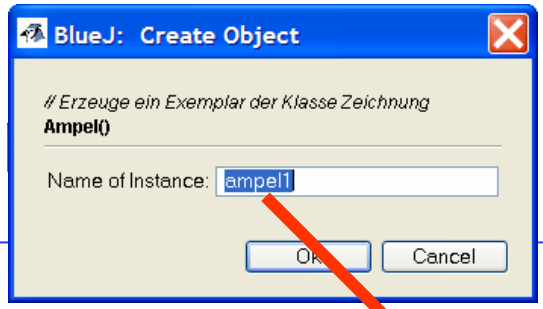
```
ampel1.kreis1.farbe = „rot“
ampel2.kreis1.farbe = „blau“
```

- gilt:

```
ampel1.kreis1.farbe == „blau“
```

Anlegen von Objekte






```
// Klasse Ampel
public void zeichne(){
    lichtOben= new Kreis();
    lichtOben.farbeAendern("rot");..
    ..
    lichtUnten= new Kreis();
    lichtUnten.farbeAendern("gruen");
    ..
    }
```

```
//Klasse Kreis
public Kreis()
{
    durchmesser = 30;
    xPosition = 20;
    yPosition = 60;
    farbe = "blau";
    istSichtbar = false;
  }
```

Nr. der Zelle	Inhalt
10000	**ampel1**
10010	lichtOben
10020	**10200**
10030	lichtUnten
10040	**10300**
...	

10200	**lichtOben**
10210	farbe
10220	„rot“

10300	**lichtUnten**
10310	farbe
10320	„gruen“

Zugriff auf Objekte anderer Klassen?

```
// In Klasse KREIS
private int xPosition;
```

```
// In Klasse BILLARD
kugel1 = new Kreis()
while (kugel1.Position <500) {
  kugel1.horizontalBewegen(1)
}
```

Fehler!: „xPosition has private Access in KREIS“

```
// In Klasse KREIS
public int xPosition;
```

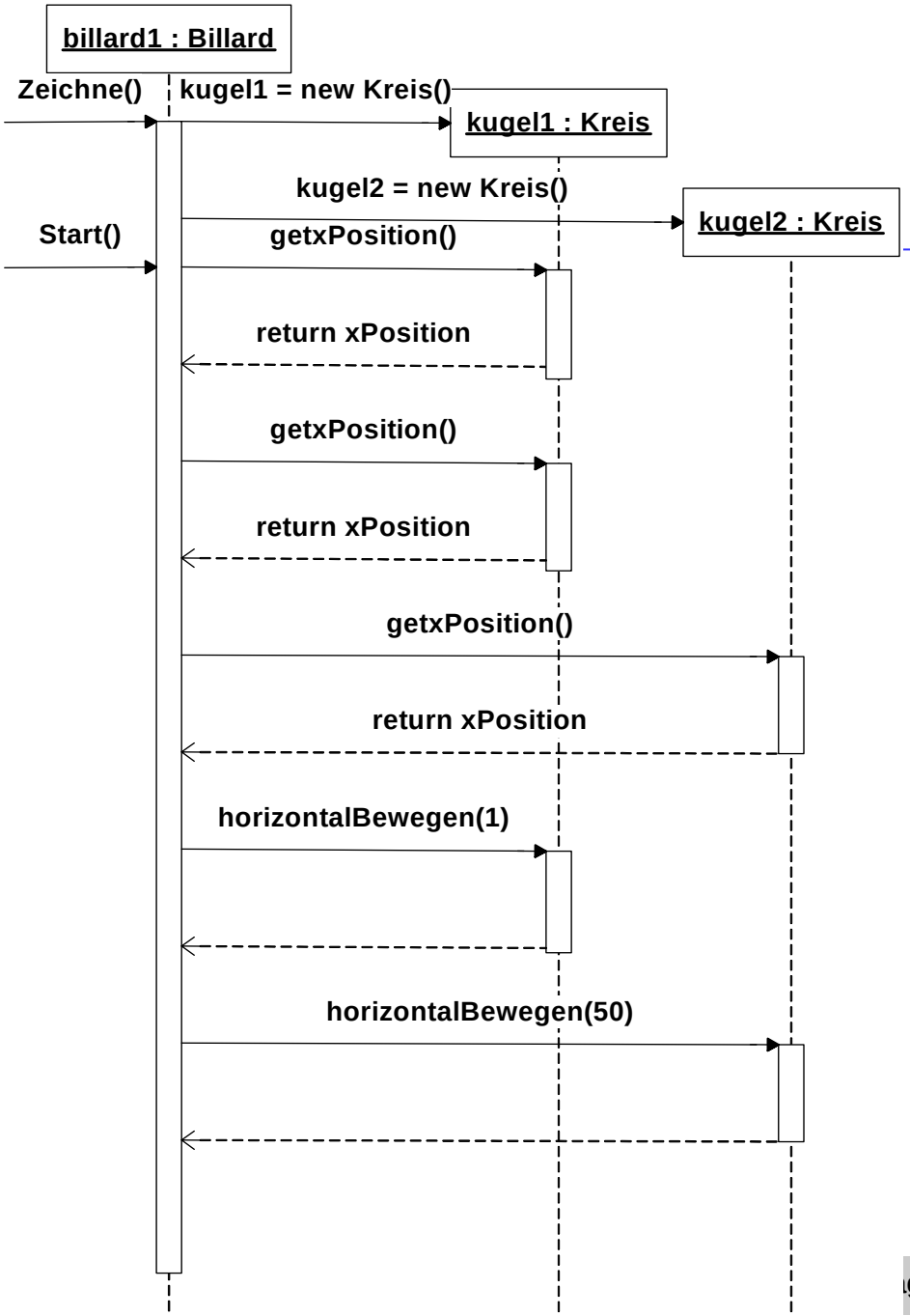
- In der Regel werden Attribute als **private** deklariert, z.B. `xPosition` in Klasse `Kreis`
- Damit haben Objekte anderer Klassen keinen Zugriff auf die Werte dieser Attribute.
- Wenn ein Objekt ein anderes enthält dann benötigt es oft (zumindest lesenden) Zugriff auf dessen Attribut
- Beispiel: Objekte der Klasse `Billard` enthalten Objekte der Klasse `Kreis`, auf deren Position `xPosition` sie reagieren sollen.
- 1. (schlechte) Lösung: Die Attribute werden als **public** deklariert. Nachteil: **Alle** anderen Objekte (aller Klassen!) können dann die Werte nicht nur lesen, sondern auch verändern.

Lesemethoden

```
// In Klasse KREIS
public int getxPosition(){
        return xPosition;
}
```

- 2. Gute Lösung : Zu jedem Attribut wird eine Lesemethode definiert (z.B. `getxPosition()` zu am Attribut `xPosition`), die als **public** deklariert wird.
- Die Anweisung `return` sorgt dafür, dass die Lesemethode den Wert des Attributes als Rückgabewert transportiert.

```
// Aufruf der Lesemethode von Klasse Billard aus:
Kugel1 = new Kreis()
while (kugel1.getxPosition()<500) {
...
// anstatt
// while (kugel1.Position <500)
```



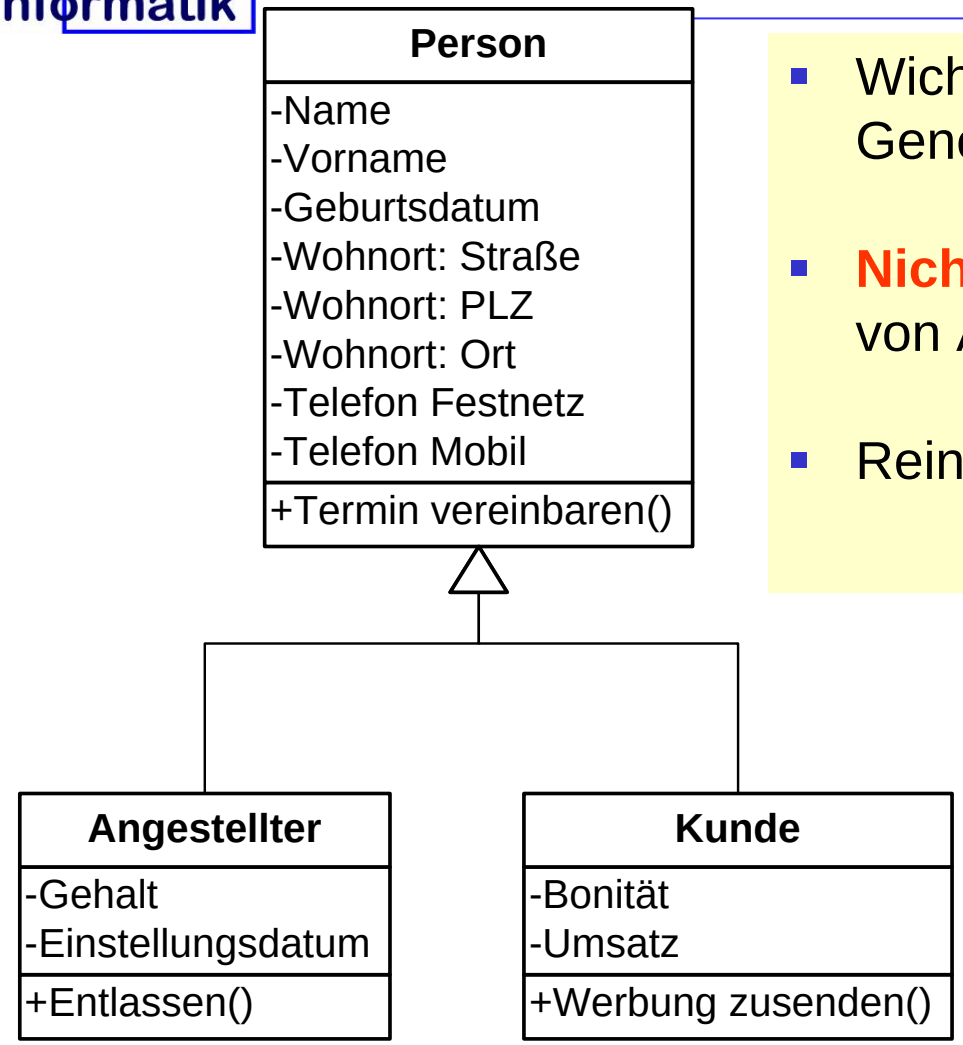
Sequenzdiagramme

```
public class Billard
{
public void zeichne()
   {
   kugel1 = new Kreis();
   kugel2 = new Kreis();

public void start()
   {
   while (kugel1.getxPosition()<500)
   {
      while (kugel1.getxPosition()<
          kugel2.getxPosition()-15) {
          kugel1.horizontalBewegen(1);
           }
        kugel2.horizontalBewegen(50);
     }
   }
}
```

Didaktik der Informatik

3.7. Generalisierung und Spezialisierung

Ober- und Unterklassen



- Wichtig: Betonung auf allgemeiner Generalisierung/Spezialisierung
- **Nicht** auf technischer Vererbung von Attributen!
- Reine Klassenbeziehung!

Literatur

- David J. Barnes / Michael Kölling: Java lernen mit BlueJ. Eine Einführung in die objektorientierte Programmierung. Pearson Studium, München 2006. ISBN: 978-3-8273-7152-2, 34,95 €

- Anderson L.W., Krathwohl D.R. (2001): (eds.): A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. New York: Longman (2001).