

# 11. Vererbung und Polymorphie

## Die Filter gegen Codeverschmutzung

## Die Basismittel zur Erweiterung von Software

Prof. Dr. rer. nat. Uwe Aßmann  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden

Version 20-0.5, 06.04.20

- 1) Vererbung zwischen Klassen
- 2) Vererbung im Speicher
- 3) Polymorphie

### Sprechstunde Prof. Aßmann:

- Montags während der Vorlesungszeit 11:10
- Donnerstags nach Vereinbarung 11:00-13:00

<https://matrix.tu-dresden.de/#/room/#Sprechstunde-INF-ST-Prof.Assmann:tu-dresden.de>



DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

# Begleitende Literatur

2

Softwaretechnologie (ST)

- ▶ Das **Vorlesungsbuch** von Pearson: **Softwaretechnologie für Einsteiger**. Vorlesungsunterlage für die Veranstaltungen an der TU Dresden. Pearson Studium, 2014. Enthält ausgewählte Kapitel aus:
  - UML: Harald Störrle. UML für Studenten. Pearson 2005. Kompakte Einführung in UML 2.0.
  - Softwaretechnologie allgemein: W. Zuser, T. Grechenig, M. Köhle. Software Engineering mit UML und dem Unified Process. Pearson.
  - Bernd Brügge, Alan H. Dutoit. Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java. Pearson Studium/Prentice Hall.
  - Erhältlich in SLUB
- Noch ein sehr gutes, umfassend mit Beispielen ausgestattetes Java-Buch:
  - C. Heinisch, F. Müller, J. Goll. Java als erste Programmiersprache. Vom Einsteiger zum Profi. Teubner.
- ▶ Für alle, die sich nicht durch Englisch abschrecken lassen:
- ▶ eBooks, von unserer Bibliothek SLUB gemietet:
  - [http://www.dbod.de/db/start.php?database=ebl\\_ebl](http://www.dbod.de/db/start.php?database=ebl_ebl) (DBoD)
- ▶ Free Books: <http://it-ebooks.info/>
  - Kathy Sierra, Bert Bates: Head-First Java <http://it-ebooks.info/book/255/>



# Obligatorische Literatur

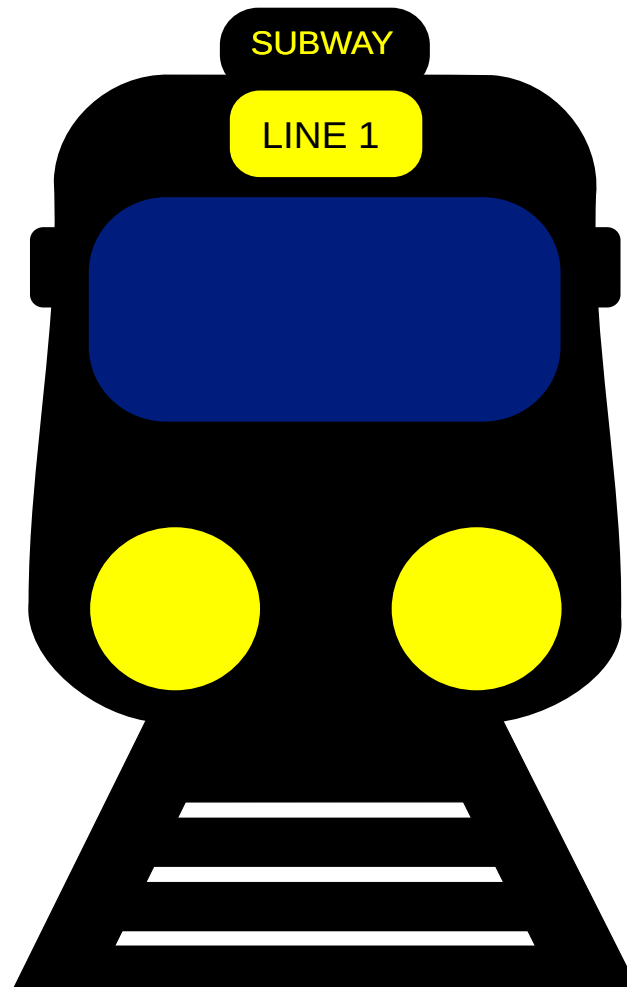
- ▶ ST für Einsteiger Kap.4+ 9, Teil II (Störrle, Kap. 5.2.6, 5.6)
  - Zuser Kap 7, Anhang A
- ▶ Java
  - Oracle Tutorial <https://docs.oracle.com/javase/tutorial/java/landl/index.html>
  - Balzert LE 9-10
  - Boles Kap. 7, 9, 11, 12

**Hörsaalübung "Objektorientierte Softwareentwicklung"  
findet bis auf Weiteres er nicht statt**

- ▶ Elementare Techniken der **Wiederverwendung** von objektorientierten Programmen kennen
  - **Generalisierung** und **Spezialisierung** mit einfacher Vererbung zwischen Klassen, konzeptuell und im Speicher
  - **Merkmalsuche** in einer Klasse und in der Vererbungshierarchie aufwärts nachvollziehen können
    - Überschreiben von Merkmalen verstehen
- ▶ **Dynamische Architektur** eines objektorientierten Programms verstehen
  - **Lebenszyklen** von Objekten verstehen
  - **Polymorphie** verstehen

# Java Herunterladen

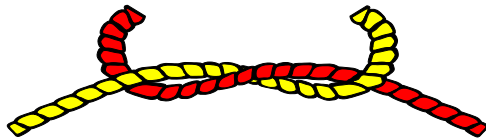
- ▶ Das Java Development Kit (JDK)
- ▶ <https://adoptopenjdk.net/>, brew tap adoptopenjdk/openjdk
- ▶ <http://openjdk.java.net/>, brew info openjdk



# Problem: Was tut man gegen Codeverschmutzung bzw. Copy-And-Paste-Programming (CAPP)?

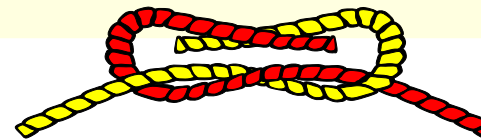
6

Softwaretechnologie (ST)

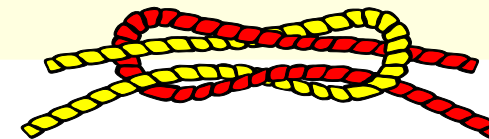


**Codeverschmutzung** durch CAPP: Nach einer Weile entdeckt man in einem gewachsenen System, dass jede Menge Code repliziert wurde  
**Große Software kann 10-20% an Replikaten (code clones) enthalten (Code-Explosion, code bloat)**

**Plagiat**  
**Ignoranz**  
**Aufwandsreduktion**  
**Mangelnde Anforderungsanalyse**  
**der Anwendungsdomäne**



**Aufwandsreduktion:**  
Wiederverwendung von Tests



- ▶ <http://c2.com/cgi/wiki?CopyAndPasteProgramming>
- ▶ [http://en.wikipedia.org/wiki/Copy\\_and\\_paste\\_programming](http://en.wikipedia.org/wiki/Copy_and_paste_programming)

# Hinweise zu weiterer Literatur: Linking Replicates

- ▶ Interessante Technik, Code-Replikate zu finden und dauerhaft zu verlinken:
  - Michael Toomim, Andrew Begel, and Susan L. Graham. Managing duplicated code with linked editing. In VL/HCC, pages 173-180. IEEE Computer Society, 2004.
  - <http://harmonia.cs.berkeley.edu/papers/toomim-linked-editing.pdf>
- ▶ Optional, mit vielen schönen Visualisierungen von Code Clones:
  - Matthias Rieger, Stéphane Ducasse, and Michele Lanza. Insights into system-wide code duplication. In WCRE, pages 100-109. IEEE Computer Society, 2004.
  - <http://rmod.lille.inria.fr/archives/papers/Rieg04b-WCRE2004-ClonesVisualizationSCG.pdf>

# 11.1 Vererbung zwischen Klassen beseitigt Codereplikate

Ähnlichkeit von Klassen sollten in Oberklassen ausfaktoriert  
werden

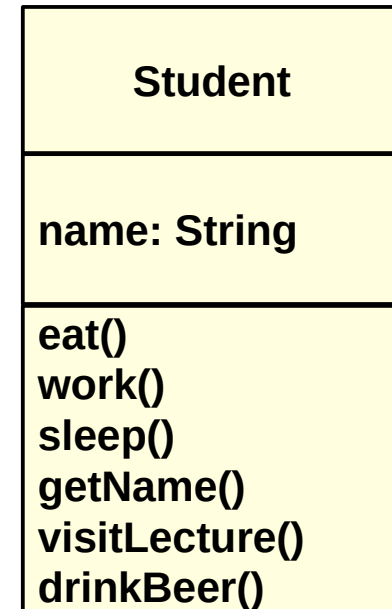
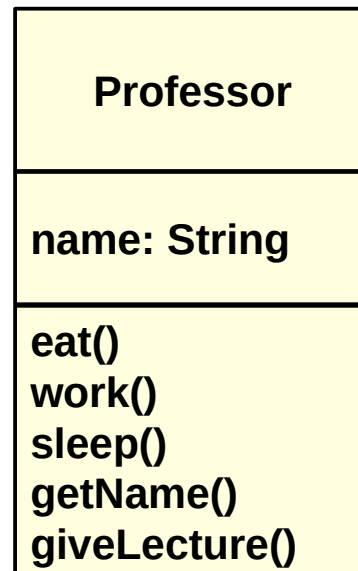
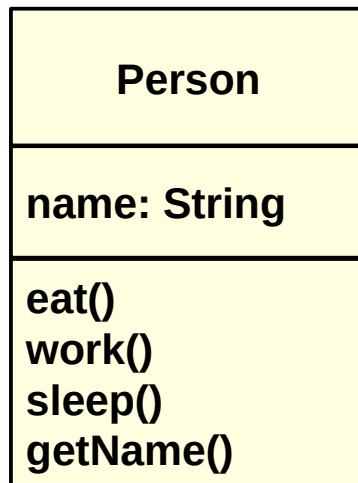


DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur



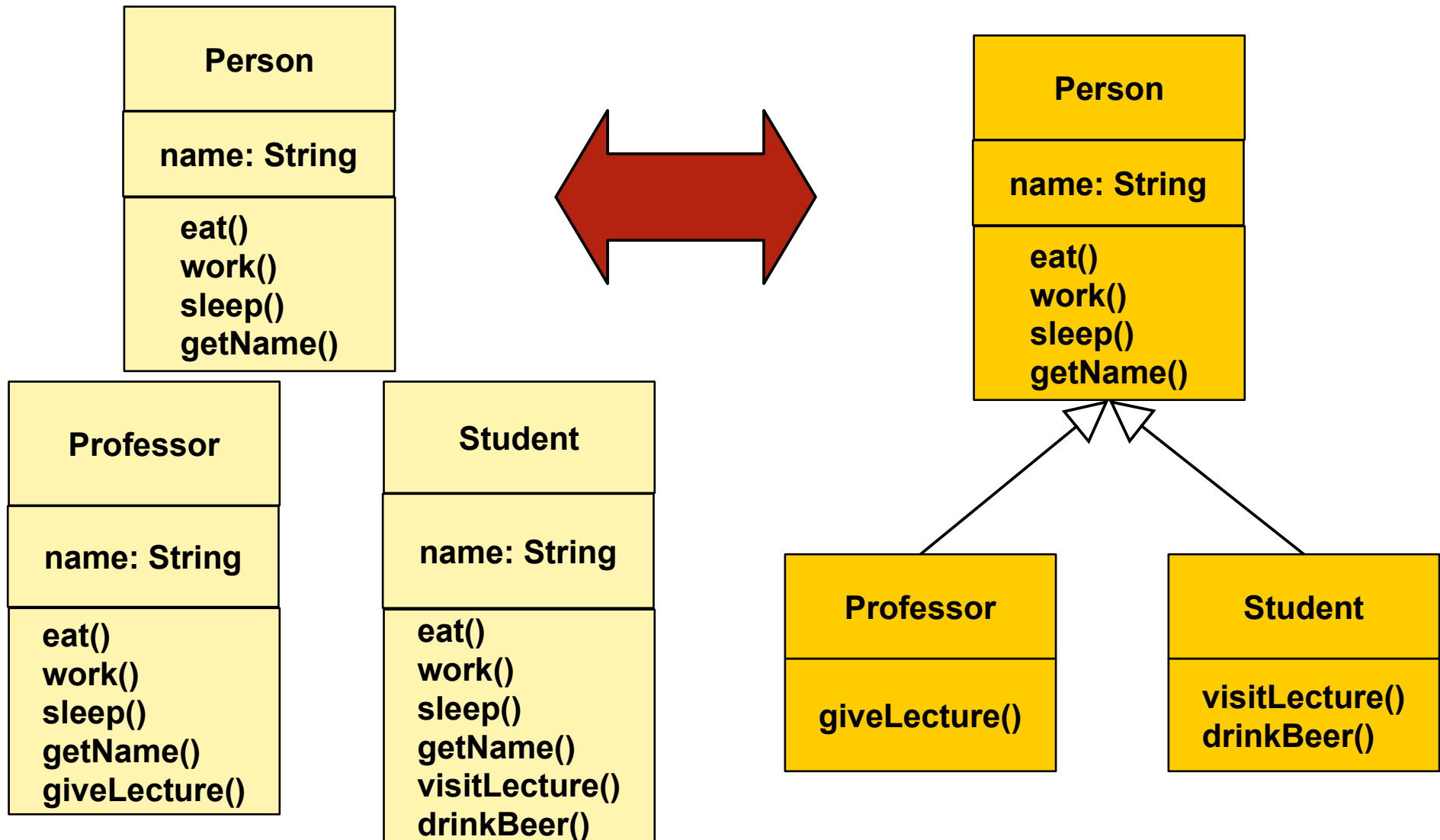
# Codeverschmutzung am Beispiel (unsoziales Programmieren)

- ▶ **Hier:** Person wurde zu Professor und Student kopiert und danach erweitert
- ▶ Warum ist diese Art des Programmierens “unsozial”?



# Einfache Vererbung

- ▶ **Vererbung:** Eine Klasse kann Merkmale von einer Oberklasse **erben**
- ▶ **Hier:** Oberklasse Person enthält alle gemeinsamen Merkmale der Unterklasse

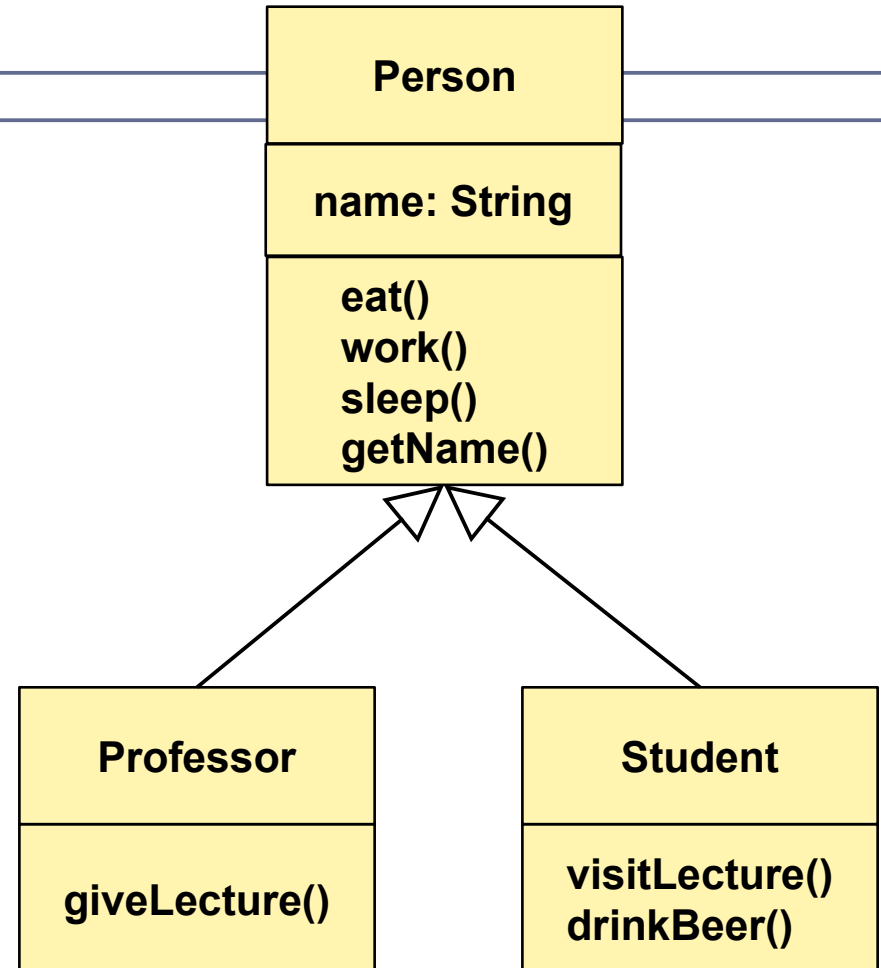


# Einfache Vererbung

11

Softwaretechnologie (ST)

- ▶ **Vorteil:** Vererbung drückt Gemeinsamkeiten aus
  - Die Unterklasse ist damit ähnlich zu dem Elter und den Geschwistern
  - Vererbung stellt *is-a*-Beziehung her
- ▶ **Hier:** Oberklasse Person enthält alle gemeinsamen Merkmale der Unterklasse
- ▶ Vererbung entspricht *Ausfaktorisierung*
- ▶ Bei **einfacher Vererbung** hat jede Klasse nur *eine* Oberklasse
  - Dann ist die Vererbungsrelation ein Baum



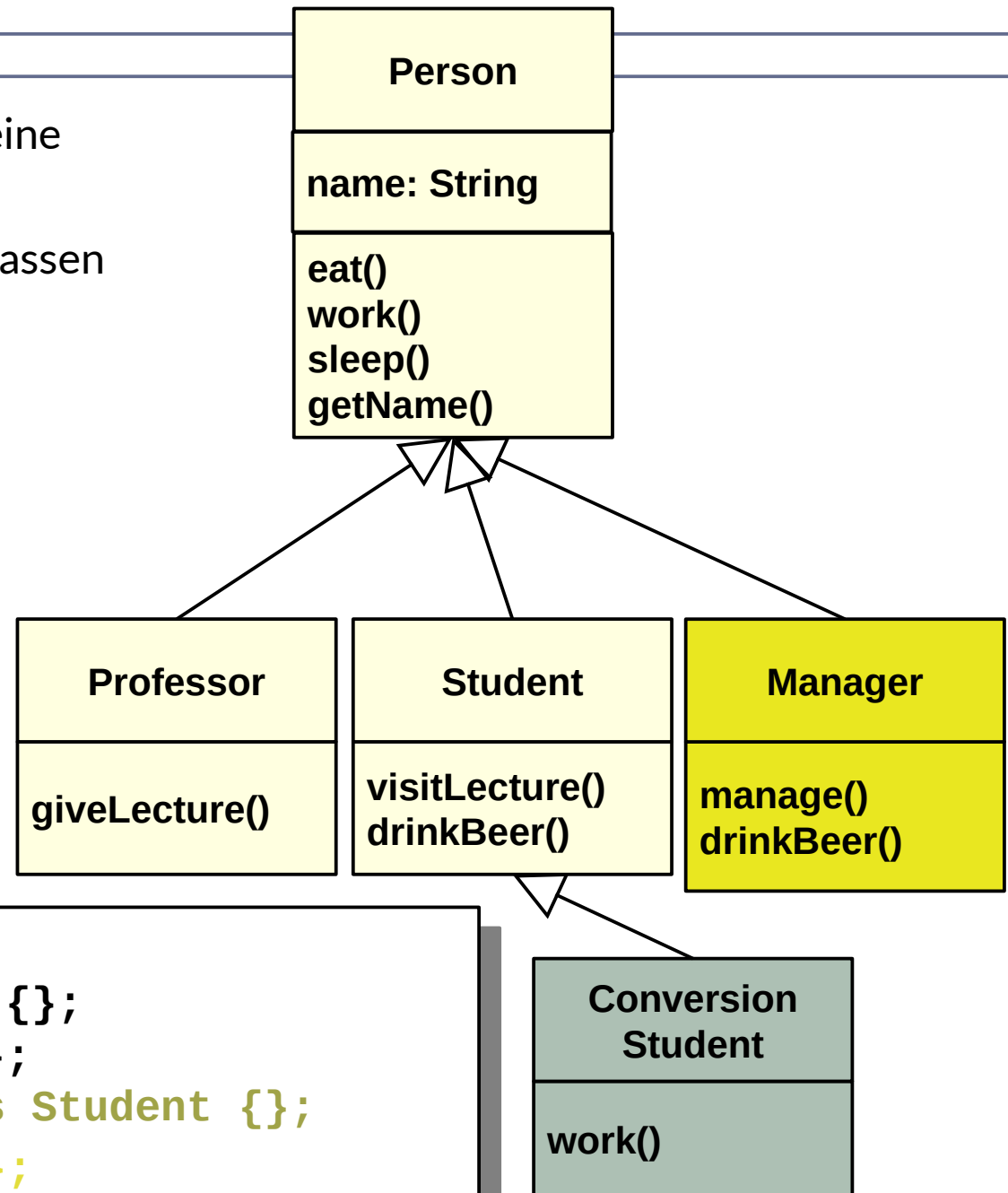
```
// Java
Professor extends Person {};
Student extends Person {};
```

# 11.1.1 Vererbungshierarchien erlauben nachträgliche Erweiterbarungen



# Horizontale und vertikale Erweiterbarkeit der Vererbung

- ▶ **Vorteil:** Mit Vererbung kann man eine Klassenhierarchie erweitern
- ▶ **Horizontal** durch neue Schwesterklassen
- ▶ **Vertikal** durch neue Unterklassen
- ▶ Und wie mittendrin?



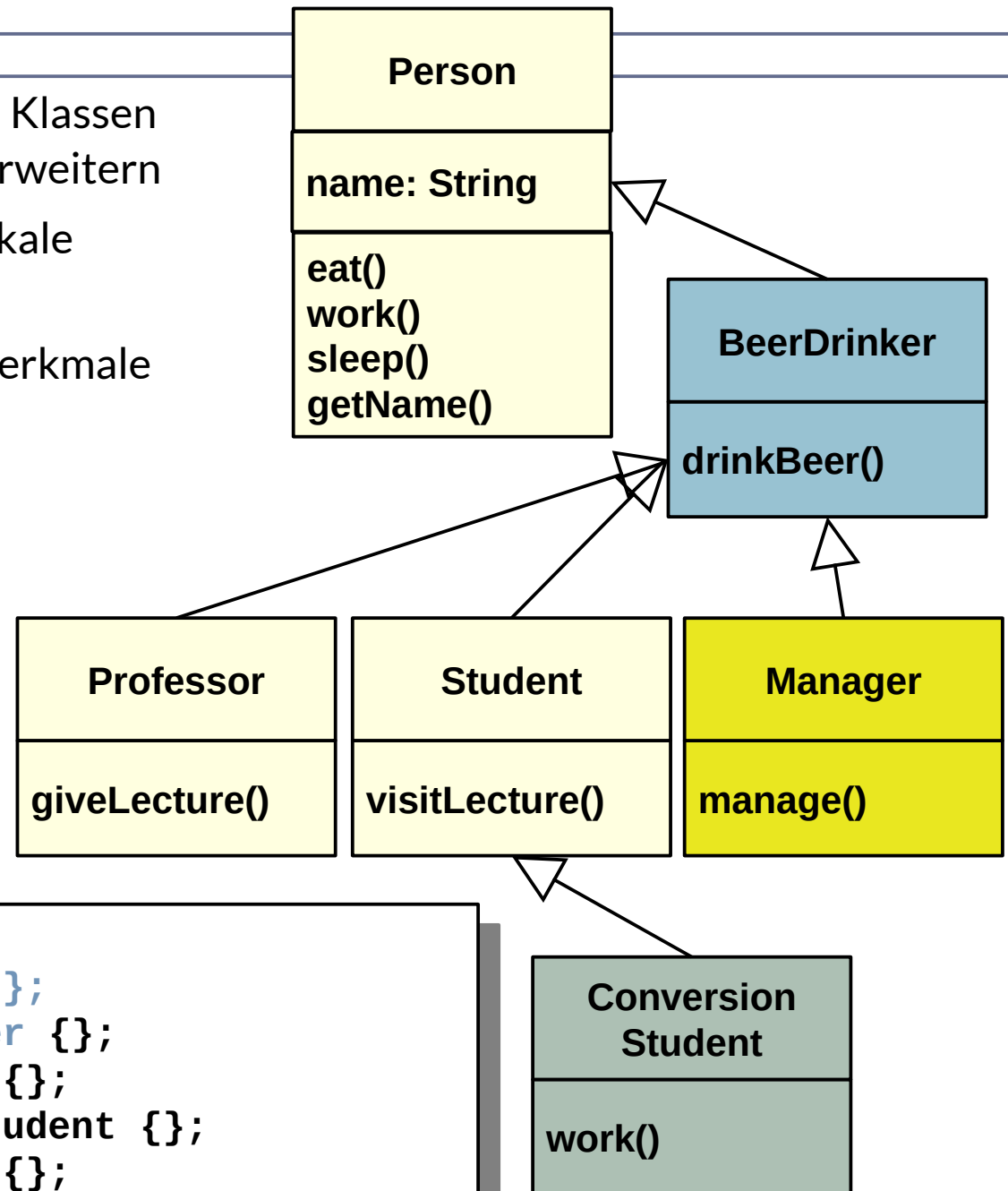
```
// Java
Professor extends Person {};
Student extends Person {};
ConversionStudent extends Student {};
Manager extends Person {};
```

# “Middle-Out” Erweiterbarkeit der Vererbung

14

Softwaretechnologie (ST)

- ▶ **Vorteil:** Mit zwischengeschobenen Klassen kann man eine Klassenhierarchie erweitern
- ▶ Und für neue horizontale und vertikale Erweiterungen vorbereiten
- ▶ “Zwischenschieben” faktorisiert Merkmale in einer Vererbungshierarchie um (*Refactoring*)



```
// Java
BeerDrinker extends Person {};
Professor extends BeerDrinker {};
Student extends BeerDrinker {};
ConversionStudent extends Student {};
Manager extends BeerDrinker {};
```

# Liskow'sches Ersetzungsprinzip (Liskow's Substitution Principle, LSP)

- ▶ Es gibt Programmiersprachen, in denen das Prinzip per Sprachdefinition gilt.
- ▶ In Java muss dieses Prinzip leider durch Testen abgesichert werden (Kapitel Test)

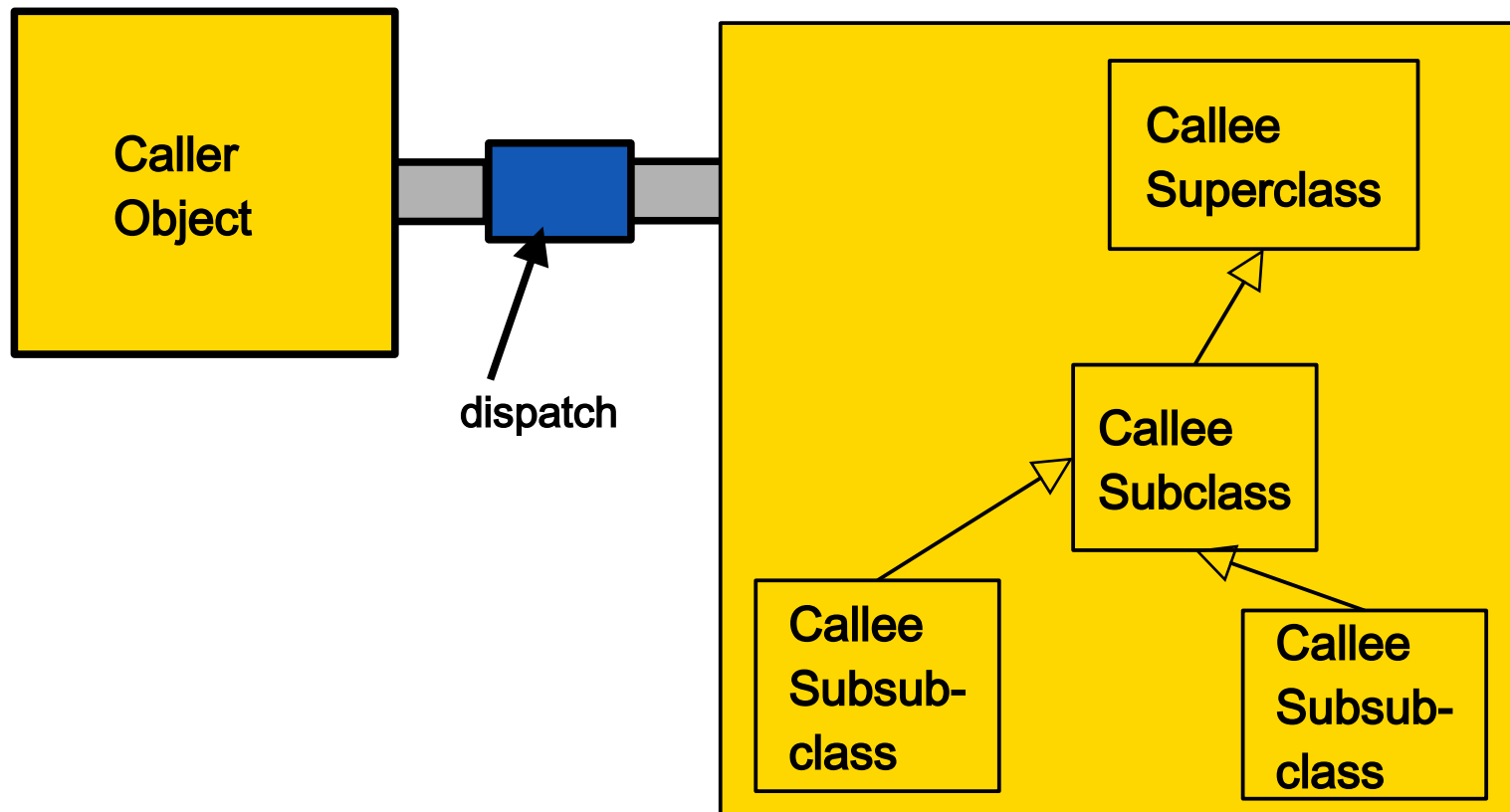
Ein Programm, das mit einem Objekt einer Klasse verwendet, kann fehlerfrei mit jedem Objekt einer ihrer Unterklassen arbeiten.

- ⊕ ▶ Horizontale, vertikale und zwischengeschobene Erweiterungen werden immer angewendet, um Code zu erweitern(!)
  - aber das LSP muss abgesichert werden.

- Biologisches Programmieren in Java bedeutet,
- eine Vererbungshierarchie nachträglich horizontal, vertikal zu erweitern,
    - zu refaktorisieren, um weitere Erweiterungen vorzubereiten und
  - das Liskow'sche Prinzip für die Erweiterungen mit Tests zuzusichern.

# Liskow'sches Ersetzungsprinzip

- ▶ Egal, welches Objekt einer Klasse aus einer Klassenhierarchie für die Abarbeitung eines Aufrufs genommen wird, - der Aufruf muss immer funktionieren und darf nicht zum Absturz des Aufrufers führen



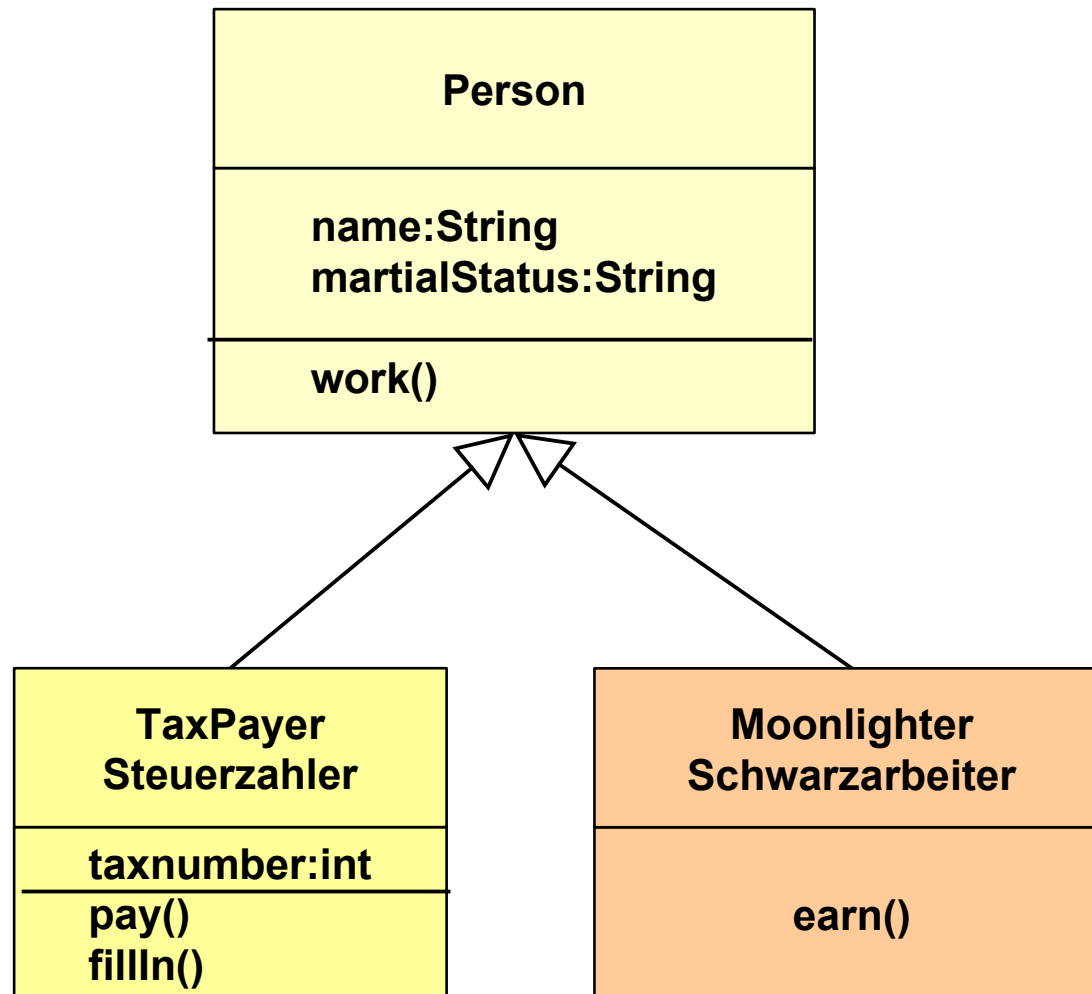


## 11.2 Wie stellt sich Vererbung im Speicher der JVM dar?



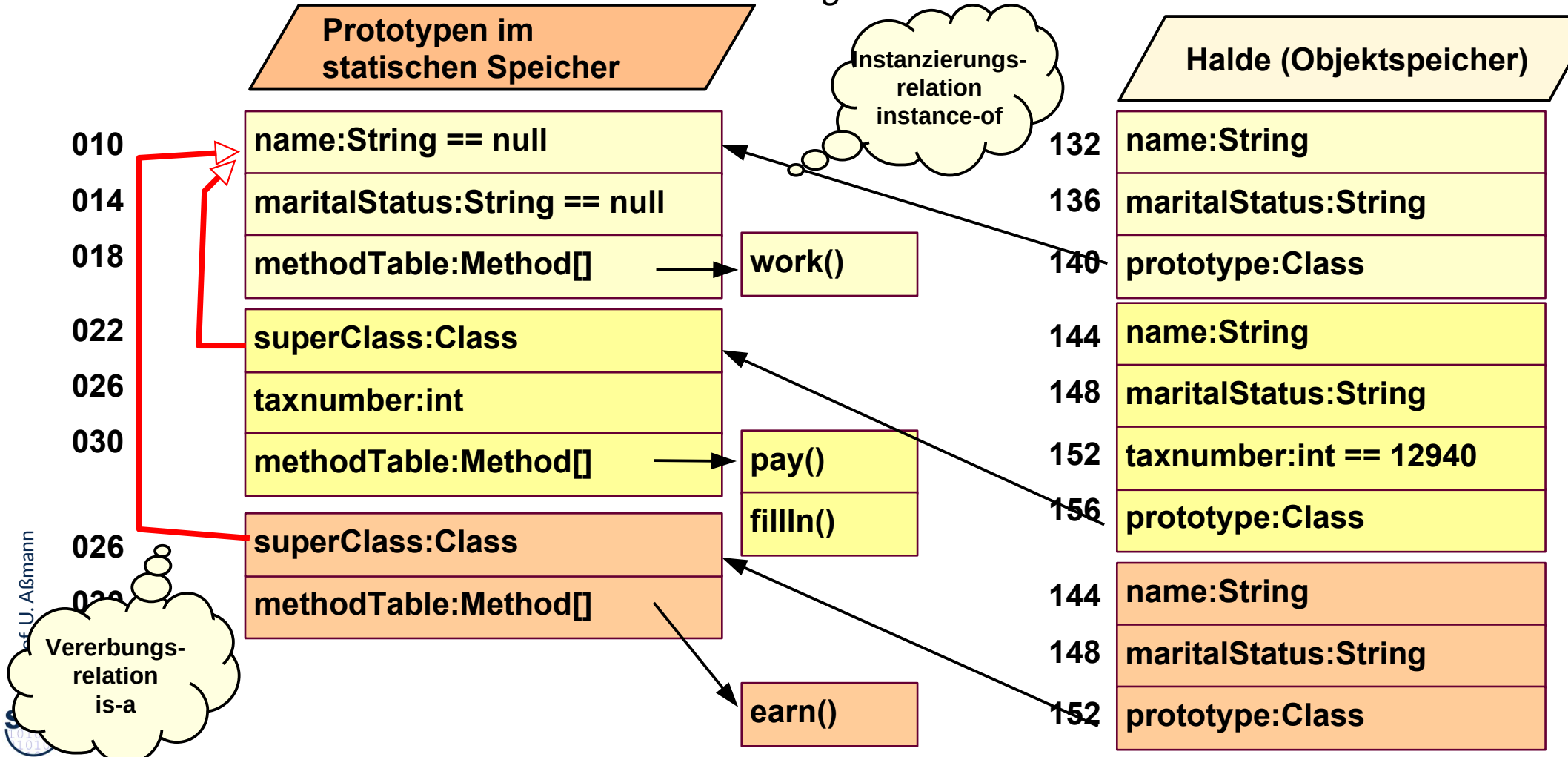
# 11.1.2 Vererbung im Speicher

- ▶ ... am Beispiel Steuerzahler



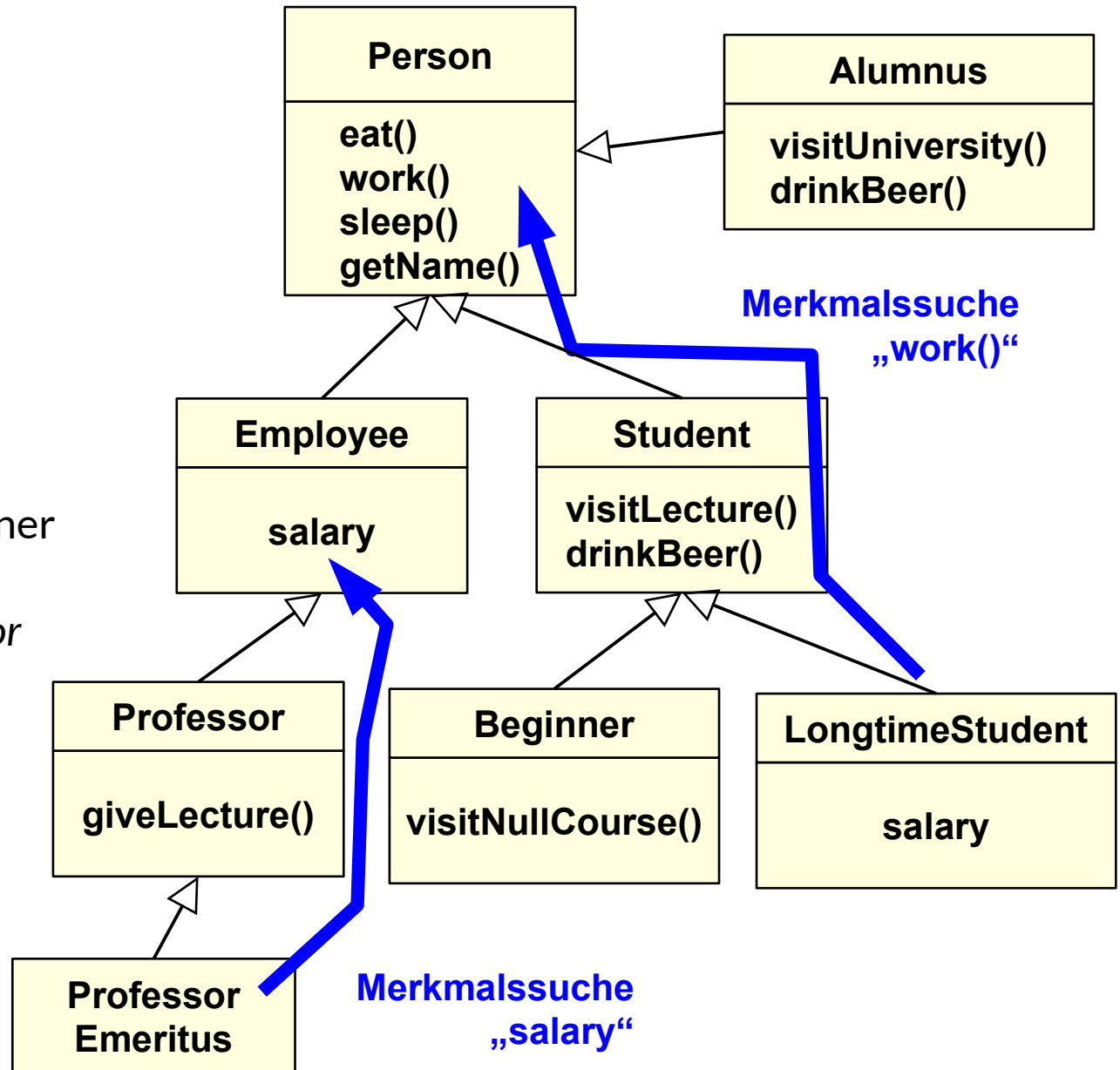
# Vererbung im Speicher

- Die Vererbungsrelation wird im Speicher als *Baum* zwischen den Prototypen der Ober- und Unterklassen dargestellt (Verzeigerung von unten nach oben)
  - Unterscheide davon die Objekt-Prototyp-Relation instance-of!
- Methoden werden zwischen den Klassen *geteilt*



# Merkmalsuche im Vererbungsbaum

- ▶ Oberklassen sind *allgemeiner* als Unterklassen (Prinzip der **Generalisierung**)
- ▶ Unterklassen sind *spezieller* als Oberklassen (**Spezialisierung**)
  - Unterklassen *erben* alle Merkmale der Oberklassen
- ▶ *Methoden- bzw. Merkmalsuche:*
  - Wird ein Merkmal nicht in einer Klasse definiert, wird in der Oberklasse *gesucht* (*method or feature resolution*)

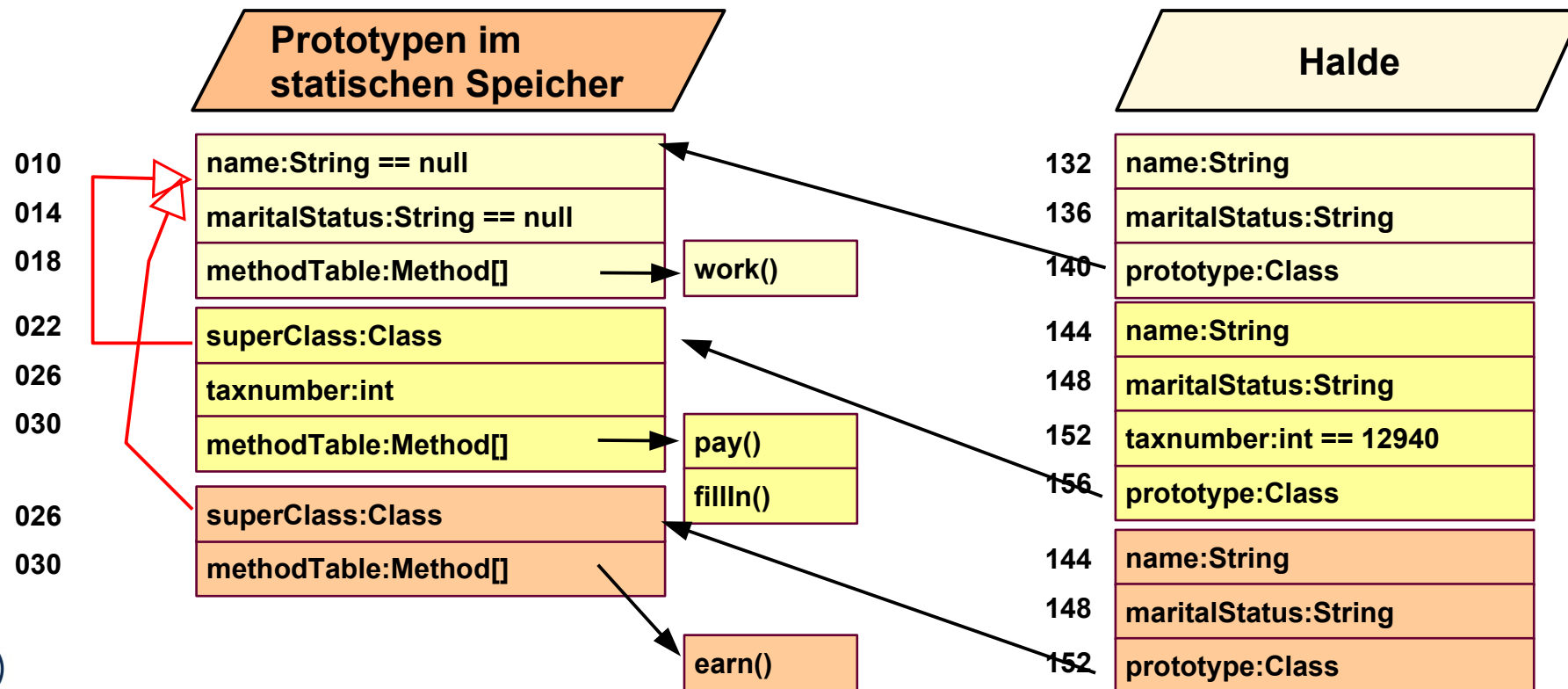


PersonInheritanceDrinkBeer.java

PersonInheritanceDemo.java

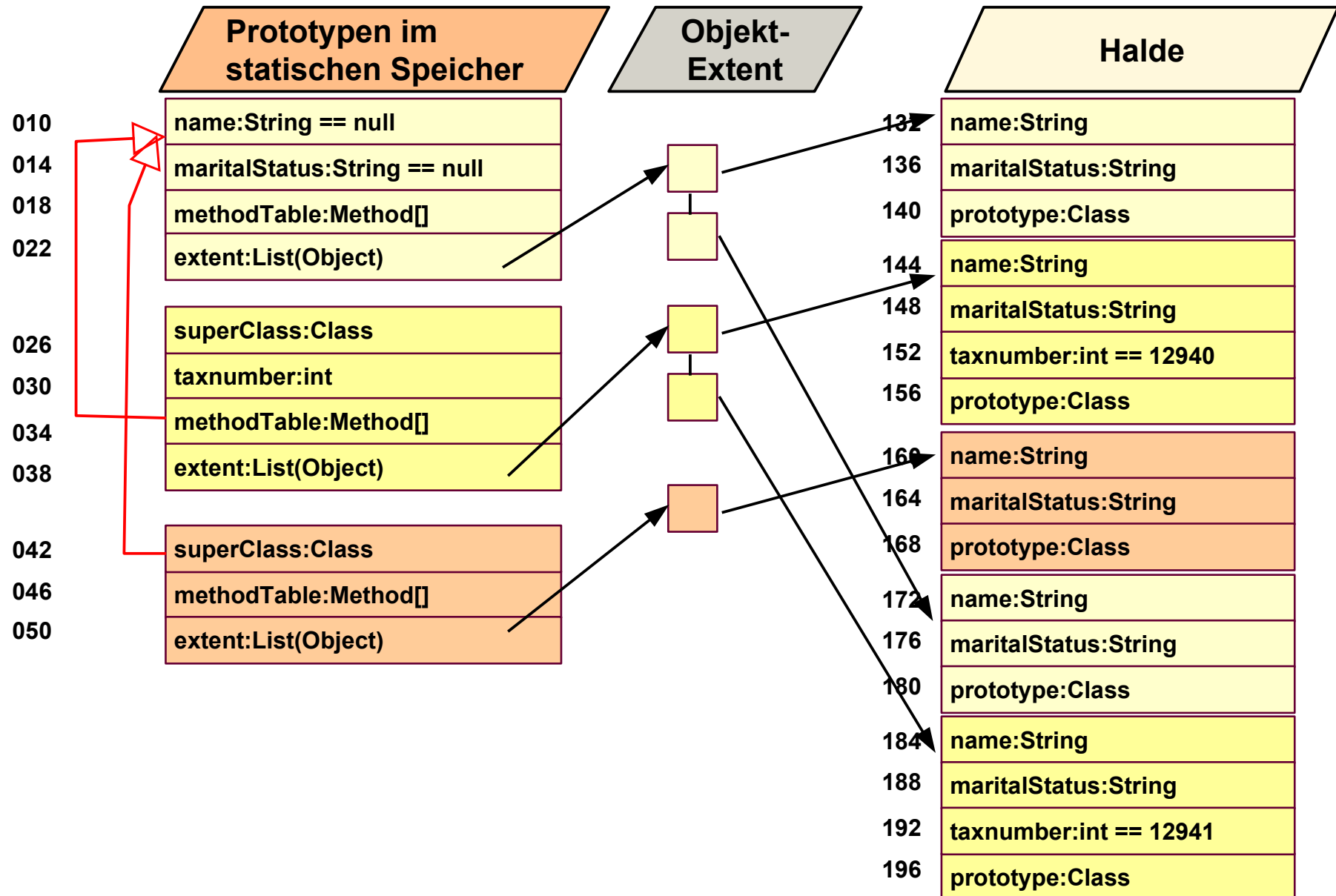
# Merkmalsuche im Speicher - Beispiele

- 1) Suche Attribut *name* in Steuerzahler: direkt vorhanden
- 2) Suche Methode *pay()* in Steuerzahler: Schlage Prototyp nach, finde in Methodentabelle des Prototyps
- 3) Suche Methode *work()* in Steuerzahler: Schlage Prototyp nach, Schlage Oberklasse nach (Person), finde in Methodentabelle von Person
- 4) Suche Methode *payback()* in Steuerzahler: Schlage Prototyp nach, Schlage Oberklasse nach (Person); existiert nicht in Methodentabelle von Person. Da keine weitere Oberklasse existiert, wird ein Fehler ausgelöst "method not found" "message not understood"



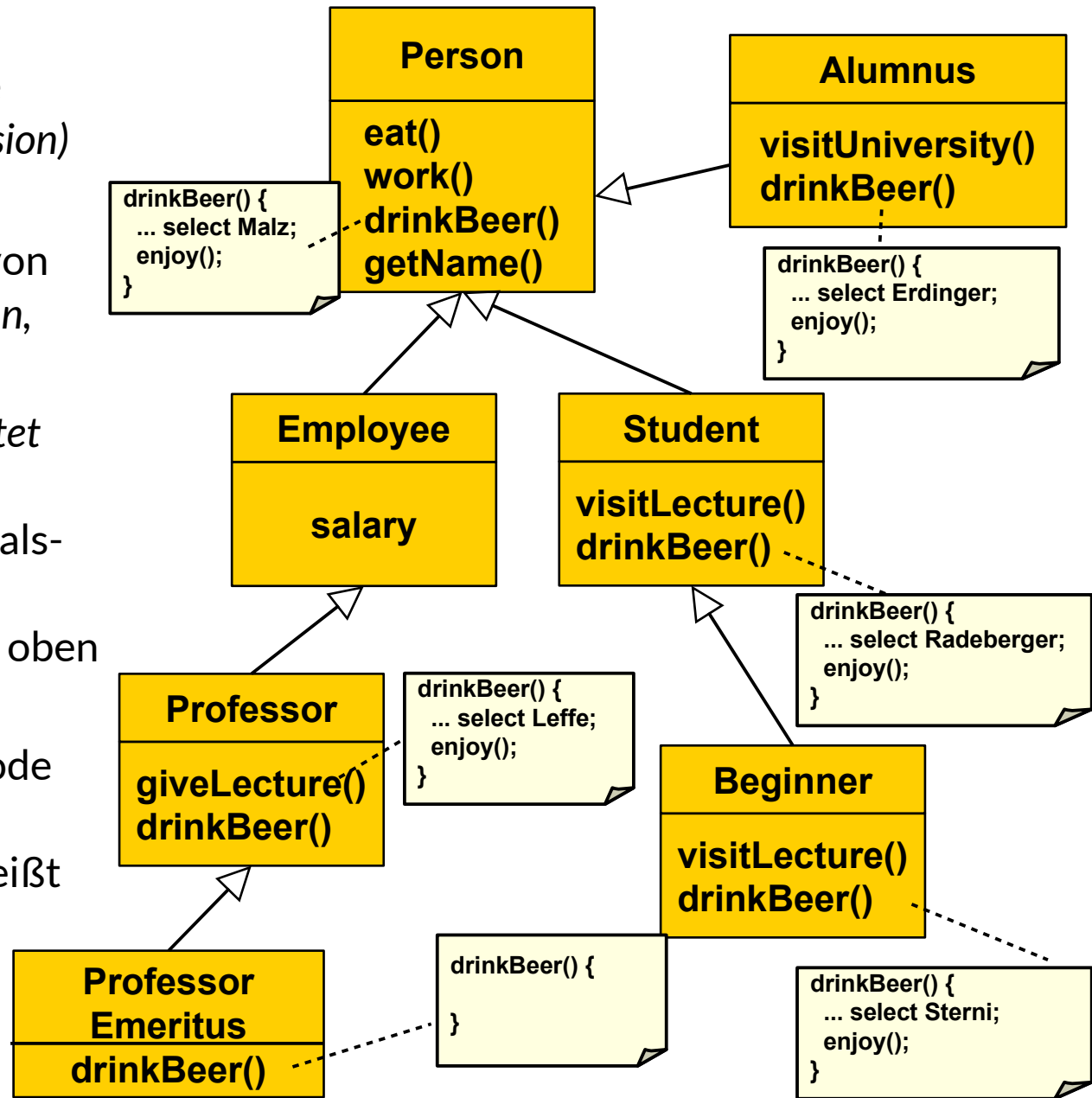
# Objekt-Extent im Speicher, mit Vererbung

- ▶ Zu einer Klasse vereinige man alle Extents aller Unterklassen



# Erweitern und Überschreiben von Merkmalen

- ▶ Eine Unterklasse kann neue Merkmale zu einer Oberklasse hinzufügen (*Erweiterung, extension*)
- ▶ Definiert eine Unterklasse ein Merkmal erneut, spricht man von einer *Redefinition* (*Überschreiben, overriding*)
  - Dieses Merkmal *überschattet* (*verbirgt*) das Merkmal der Oberklasse, da der Merkmals-suchalgorithmus in der Hierarchie von unten nach oben sucht.
  - Die überschriebene Methode hat mehrere Implementierungen und heißt *polymorph* oder *virtual*

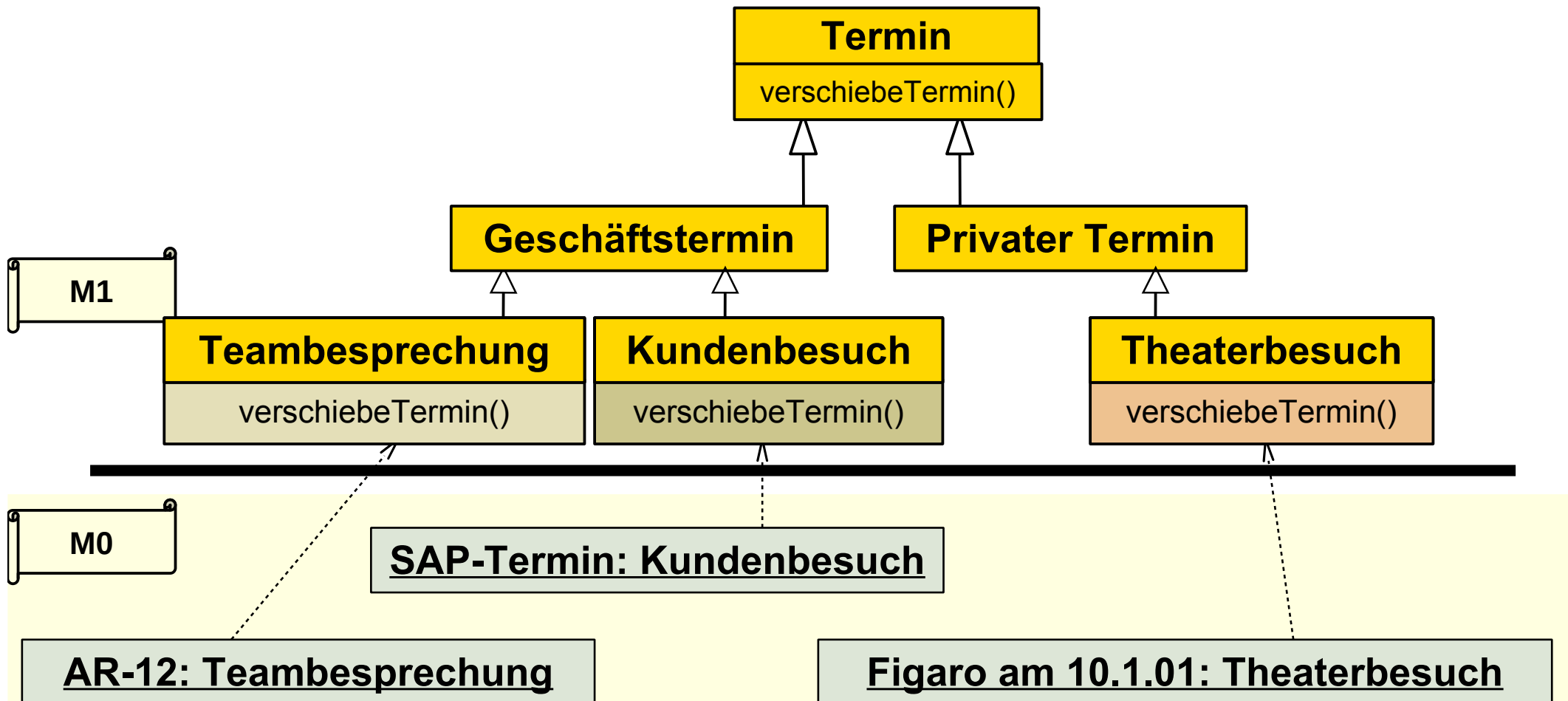


# Beispiel: Termin-Klasse und Termin-Objekte im fachlichen Modell "Terminverwaltung"

24

Softwaretechnologie (ST)

- ▶ Allgemeines Merkmal: Jeder Termin kann verschoben werden.
  - Daher schreibt die Klasse **Termin** vor, daß auf die Nachricht „verschiebeTermin“ reagiert werden muß.
- ▶ Unterklassen *spezialisieren* Oberklassen; Oberklassen *generalisieren* Unterklassen



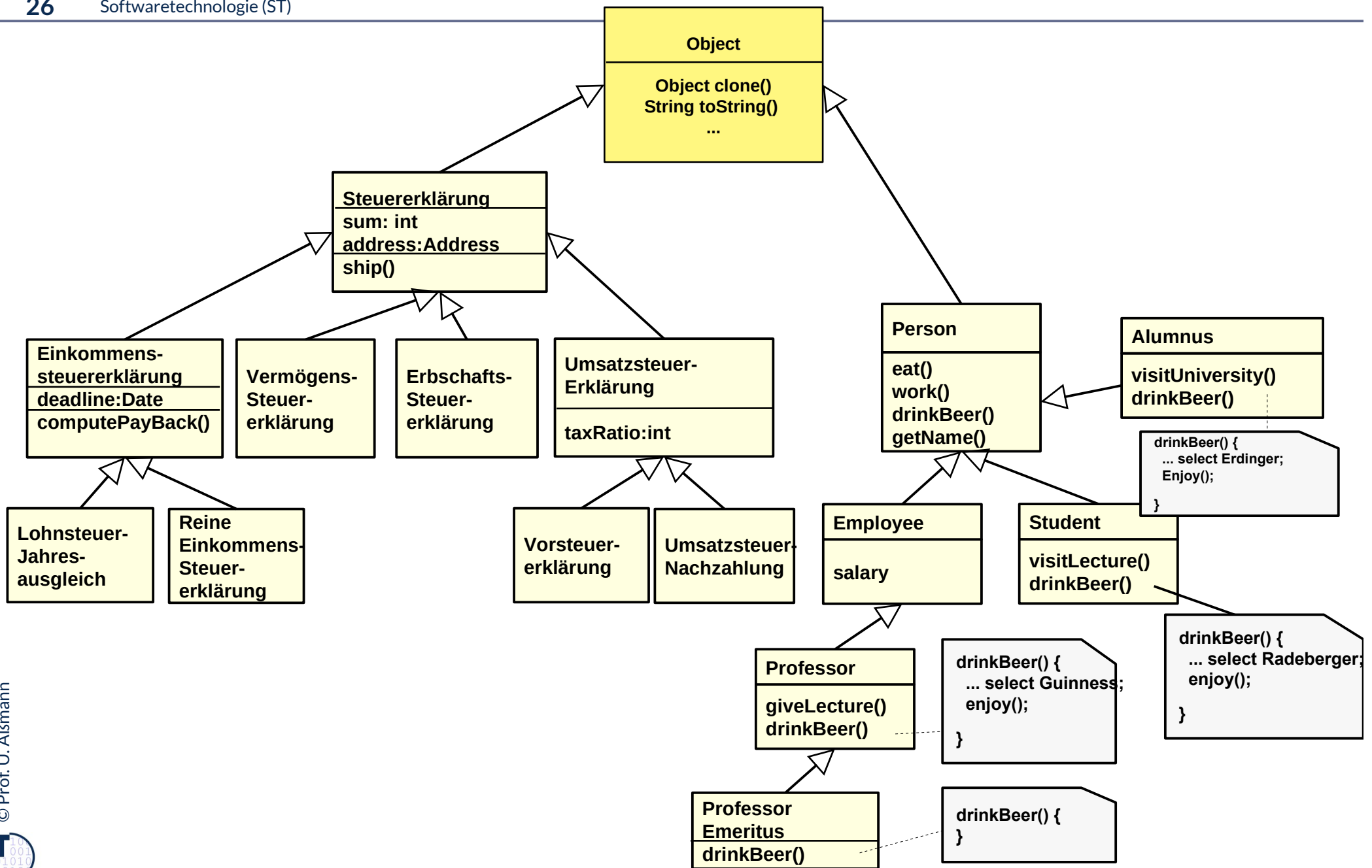


## 11.1.3. Die oberste Klasse von Java: "Object"

- ▶ **java.lang.Object:** allgemeine Eigenschaften aller Objekte und Klassen
  - Jede Klasse ist Unterklasse von Object ("extends Object").
  - Diese Vererbung ist *implizit* (d.h. man kann "extends Object" weglassen).
  - Wiederverwendung in der gesamten JDK-Bibliothek!
- ▶ Jede Klasse kann die Standard-Operationen überdefinieren:
  - equals: Objektgleichheit (Standard: Referenzgleichheit)
  - hashCode: Zahlcodierung
  - toString: Textdarstellung, z.B. für println()

```
class Object {
    protected Object clone (); // kopiert das Objekt
    public boolean equals (Object obj);
        // prüft auf Gleichheit zweier Objekte
    public int hashCode();      // produce a unique identifier
    public String toString();  // produce string representation
    protected void finalize(); // lets GC run
    Class getClass();         // gets prototype object
}
```

# Vererbung von Object auf Anwendungsklassen



# 11.E1 Exkurs: Lernen mit Begriffshierarchien

Begriffshierarchien können zum Lernen eingesetzt werden

“Der einzige Weg, auf welchem wahre Kenntnis erreicht werden kann, ist durch liebevolles Studium.”

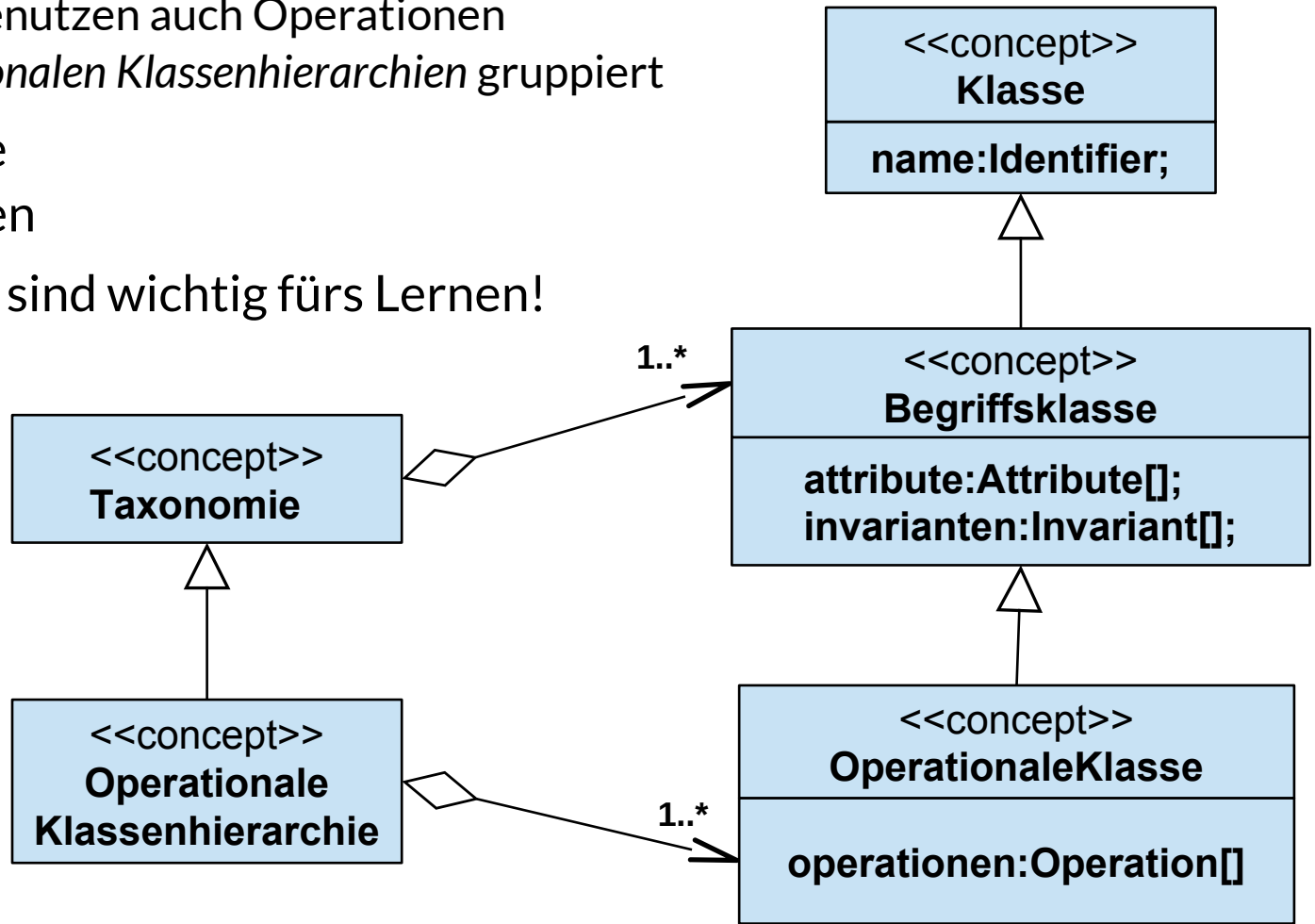
Carl Hilty (1831 - 1909), Schweizer Staatsrechtler und Laientheologe

<http://www.aphorismen.de/>



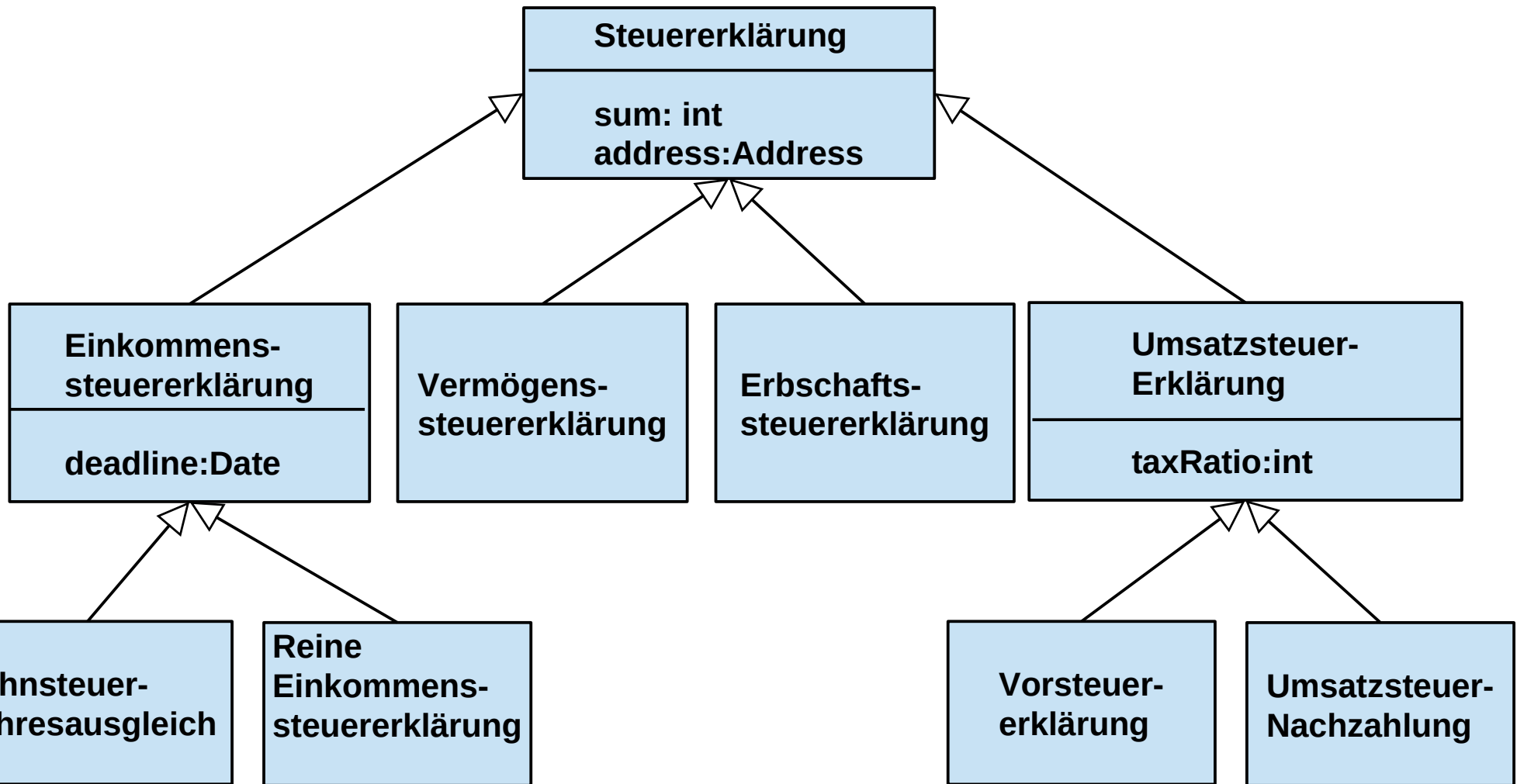
# Q1: Begriffshierarchien (Taxonomien) nutzen einfache Vererbung

- ▶ Domänenmodelle werden durch *Klassifikation* der Domänenobjekte und Domänenkonzepte ermittelt
- ▶ Klassifikationen führen zu **Begriffshierarchien (Taxonomien)**
  - *Begriffsklassen* besitzen nur Attribute und Invarianten (leicht blau)
- *Operationale Klassen* benutzen auch Operationen und werden zu *operationalen Klassenhierarchien* gruppiert
- **Beispiel:** die Begriffe der Arten von Klassen
- **Merke:** Taxonomien sind wichtig fürs Lernen!



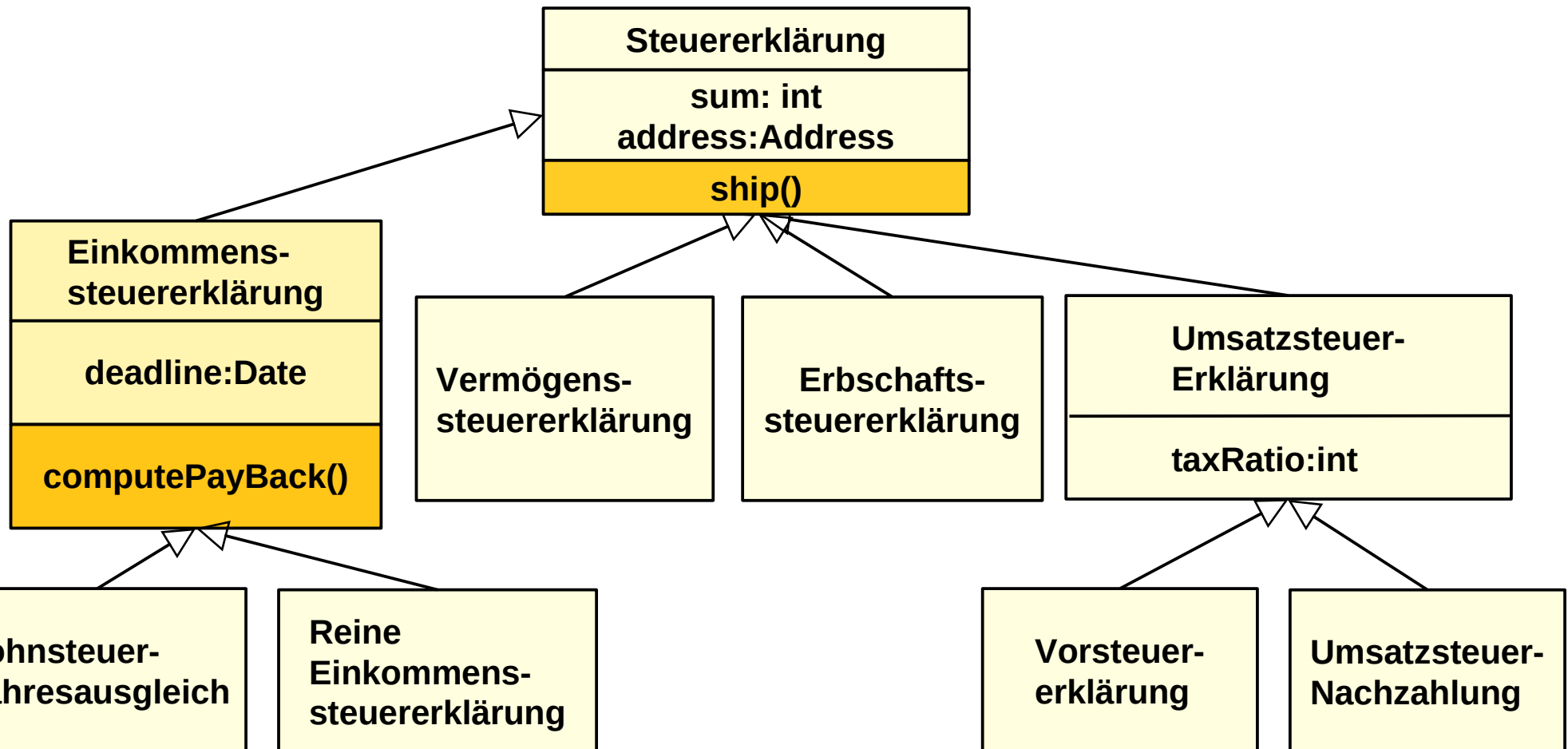
# Bsp. Taxonomie der Steuererklärungen im fachlichen Modell "Steuererklärung"

- ▶ Domäne: Finanzbuchhaltung: Das deutsche Steuerrecht kennt viele Arten von Steuererklärungen
- ▶ Eine Klassifikation führt zu einer Begriffshierarchie
- ▶ Warum haben Informatiker durch ihr Verständnis von Begriffshierarchien große Vorteile im Leben?



# Bsp. Erweiterung einer Begriffshierarchie hin zu operationalen Klassenhierarchie

- ▶ Programmiert man eine Steuerberater-Software, muss man die Begriffshierarchie der Steuererklärungen als Klassen einsetzen.
- ▶ Daneben sind aber die Klassen um eine neue **Abteilung (compartment)** mit **Operationen** zu erweitern, denn innerhalb der Software müssen sie ja etwas tun.



# Blooms Taxonomie des Lernens

- ▶ [Wikipedia, Lernziele] Die 6 Stufen im kognitiven Bereich lauten:

- ▶ **Lehrlingsschaft**

- **Stufe 1) Kenntnisse / Wissen:** Kenntnisse konkreter Einzelheiten wie Begriffe, Definitionen, Fakten, Daten, Regeln, Gesetzmäßigkeiten, Theorien, Merkmalen, Kriterien, Abläufen; Lernende können Wissen abrufen und wiedergeben.
- **Stufe 2) Verstehen:** Lernende können Sachverhalt mit eigenen Worten erklären oder zusammenfassen; können Beispiele anführen, Zusammenhänge verstehen; können Aufgabenstellungen interpretieren.

- ▶ **Gesellschaft**

- **Stufe 3) Anwenden:** Transfer des Wissens, problemlösend; Lernende können das Gelernte in neuen Situationen anwenden und unaufgefordert Abstraktionen verwenden oder abstrahieren.
- **Stufe 4) Analyse:** Lernende können ein Problem in einzelne Teile zerlegen und so die Struktur des Problems verstehen; sie können Widersprüche aufdecken, Zusammenhänge erkennen und Folgerungen ableiten, und zwischen Fakten und Interpretationen unterscheiden.
- **Stufe 5) Synthese:** Lernende können aus mehreren Elementen eine neue Struktur aufbauen oder eine neue Bedeutung erschaffen, können neue Lösungswege vorschlagen, neue Schemata entwerfen oder begründete Hypothesen entwerfen.

- ▶ **Meisterschaft**

- **Stufe 6) Beurteilung:** Lernende können den Wert von Ideen und Materialien beurteilen und können damit Alternativen gegeneinander abwägen, auswählen, Entschlüsse fassen und begründen, und *bewusst Wissen zu anderen transferieren*, z. B. durch Arbeitspläne.

# Lernlandkarten und Lernmatrizen als Hilfsmittel

- ▶ Erstellen Sie eine **Strukturkarte (concept map)** der Vorlesung zur Vorbereitung für die Klausur
- ▶ Die Strukturkarte enthält alle **Begriffshierarchien**, die in der VL diskutiert wurden
- ▶ **Vorlesungslandkarte**: Quasi-hierarchische Darstellung der Inhalte der Vorlesung
  - gegliedert wie die Vorlesung
  - gefüllt mit Begriffen, die Sie erklären können (Bloom-Stufe 1+2)
  - gefüllt mit Fragen
- ▶ **Vorlesungsmatrix**: Matrixartige Darstellung der Inhalte
  - auf die Vorlesungslandkarte aufbauend
  - Kreuzen mit zweiter Dimension: Querschneidende Aspekte wie Analyse, Design, Entwurfsmuster in die zweite Dimension eintragen
  - Damit die Vorlesungslandkarte in einen zweiten Zusammenhang bringen (Bloom-Stufe 3+4)
- ▶ **Übung**: Erstellen Sie eine Vorlesungslandkarte von Vorlesung 10, "Objekte und Klassen"
  - Erstellen Sie eine Vorlesungslandkarte von Vorlesung 11, "Vererbung und Polymorphie"
  - Ermitteln sie querscheidende Aspekte wie Objektallokation, Speicherrepräsentation
  - Entwickeln Sie eine Vorlesungsmatrix





# Exkursion: ProQuest eBooks Online

- ▶ **Achtung, neuer e-Book-Service unserer Bibliothek SLUB:**
- ▶ [http://www.dbod.de/db/start.php?database=ebf\\_ebl](http://www.dbod.de/db/start.php?database=ebf_ebl) (DBoD)
- ▶ <https://www.slub-dresden.de/recherche/datenbanken/erweitertes-angebot-an-e-medien-waehrend-covid-19/> (Alle)

Sehr empfohlen für die Technik des Lernens und wiss. Arbeitens:

- ▶ Stickel-Wolf, Wolf. Wissenschaftliches Arbeiten und Lerntechniken. Gabler. Blau. Sehr gutes Überblicksbuch für Anfänger.
- ▶ **Kurs “Academic Skills in Software Engineering” (2/2/0)**
  - Sommersemester
  - <https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/23071653920?20>
  - [https://tu-dresden.de/ing/informatik/smt/st/studium/lehrveranstaltungen?leaf=1&lang=en&subject=418&embedding\\_id=47eddfa7c5a54ed5be49042aff35a31b](https://tu-dresden.de/ing/informatik/smt/st/studium/lehrveranstaltungen?leaf=1&lang=en&subject=418&embedding_id=47eddfa7c5a54ed5be49042aff35a31b)

## 11.3. Polymorphie (Wechsel der Gestalt)

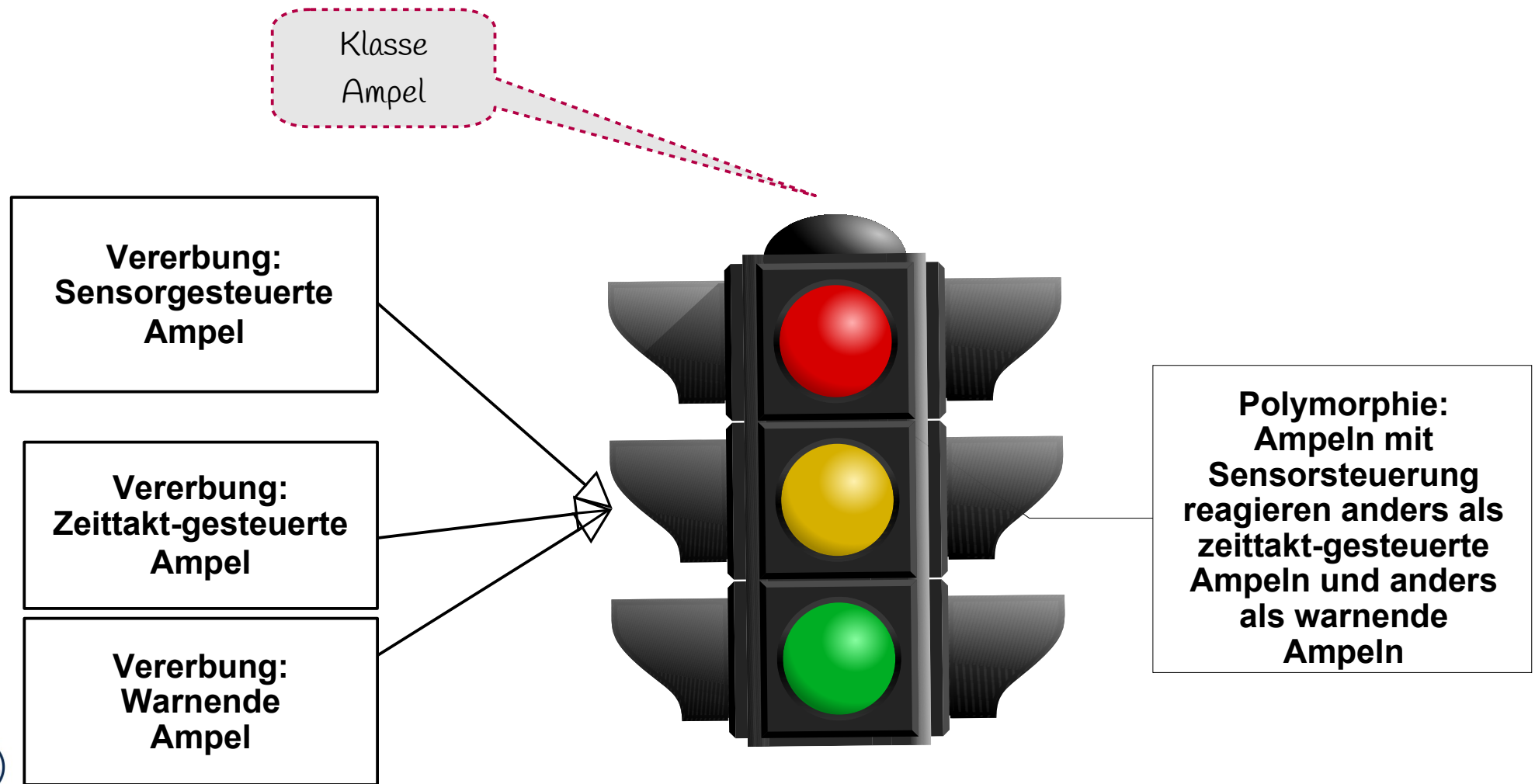
.. verändert das Verhalten einer Anwendung, ohne den Code zu verändern

- ▶ Polymorphie erlaubt *dynamische Architekturen*
  - Dynamisch wechselnd
  - Unbegrenzt viele Objekte
- ▶ Polymorphie erlaubt die Spezifikation von *Lebenszyklen von Objekten*
- ▶ Zentraler Fortschritt gegenüber einfachem imperativen Programmieren



# Vererbung und Polymorphie

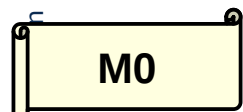
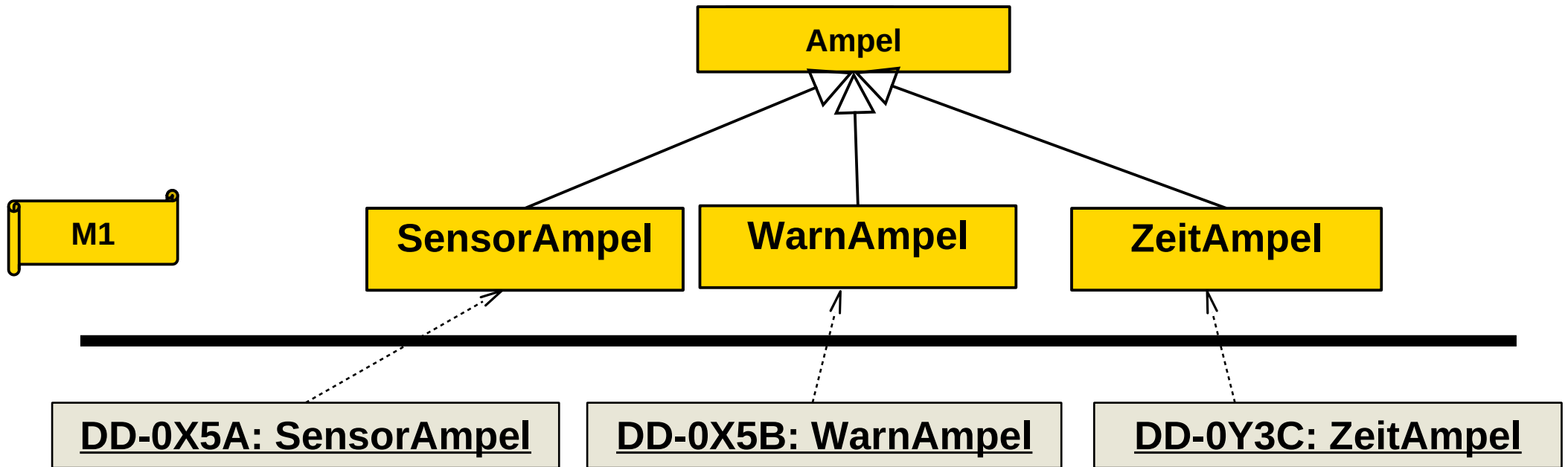
- ▶ Welcher Begriff einer Begriffshierarchie wird verwendet (Oberklassen/ Unterklassen)?
- ▶ Wie hängt das Verhalten des Objektes von der Hierarchie ab (spezieller vs allgemeiner)?



# Beispiel: Der Lebenszyklus von Ampeln

Jede Ampel schaltet auf eine spezifische Weise

- ▶ Die Klasse **ZeitAmpel** schreibt vor, daß auf die Nachricht „Zeittakt“ mit Schalten reagiert werden muss
- ▶ Die Klasse **SensorAmpel** schreibt vor, auf das Sensorereignis “Auto kommt an” geschaltet werden muss
- ▶ Die Klasse **WarnAmpel** schreibt nur vor, dass geblinkt wird



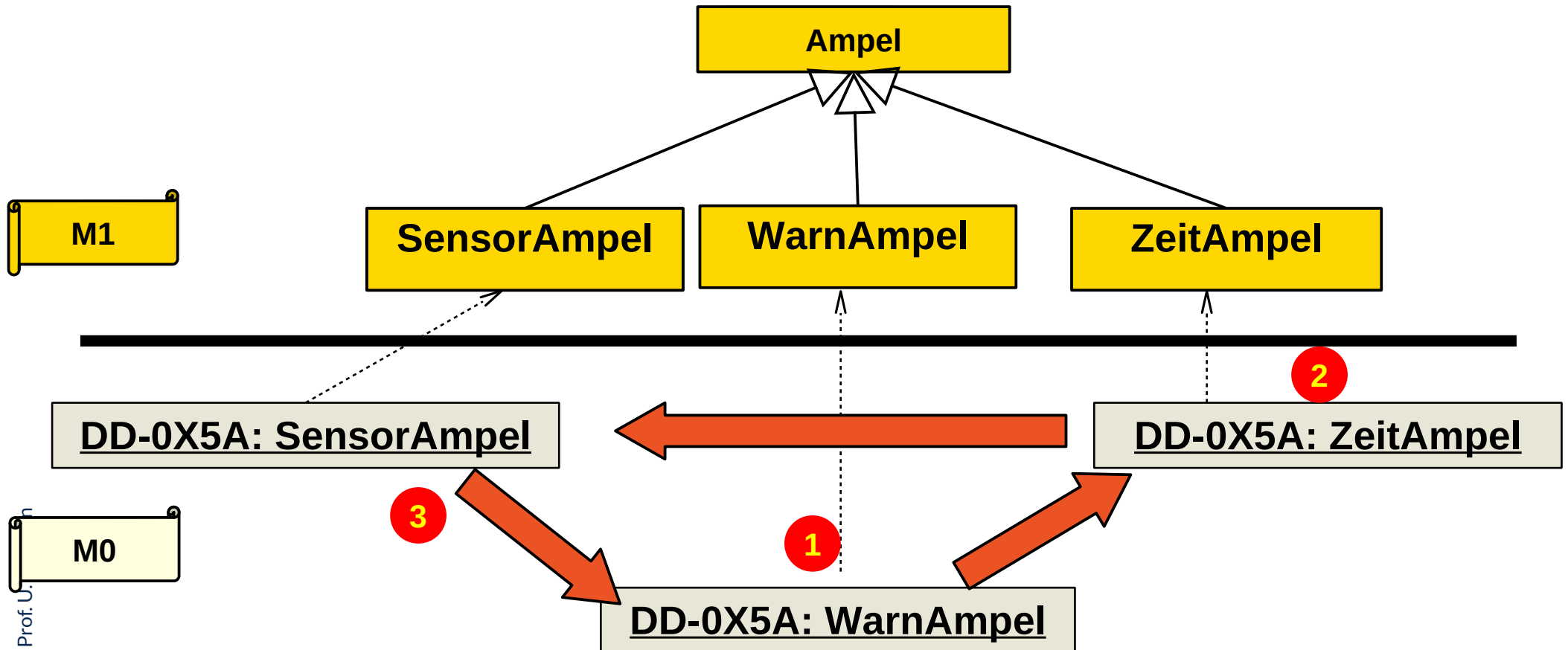
- > "instance-of": Objekt ist Instanz einer Klasse
- > "is-a" Generalisierung (Vererbung)

# Beispiel: Lebenszyklus von Ampeln (Polymorphie)

37

Softwaretechnologie (ST)

- ▶ Ampeln folgen **Lebenszyklen**: nachts blinken sie, zur Rushhour sind sie Zeit-getaktet, und ansonsten sensor-getrieben

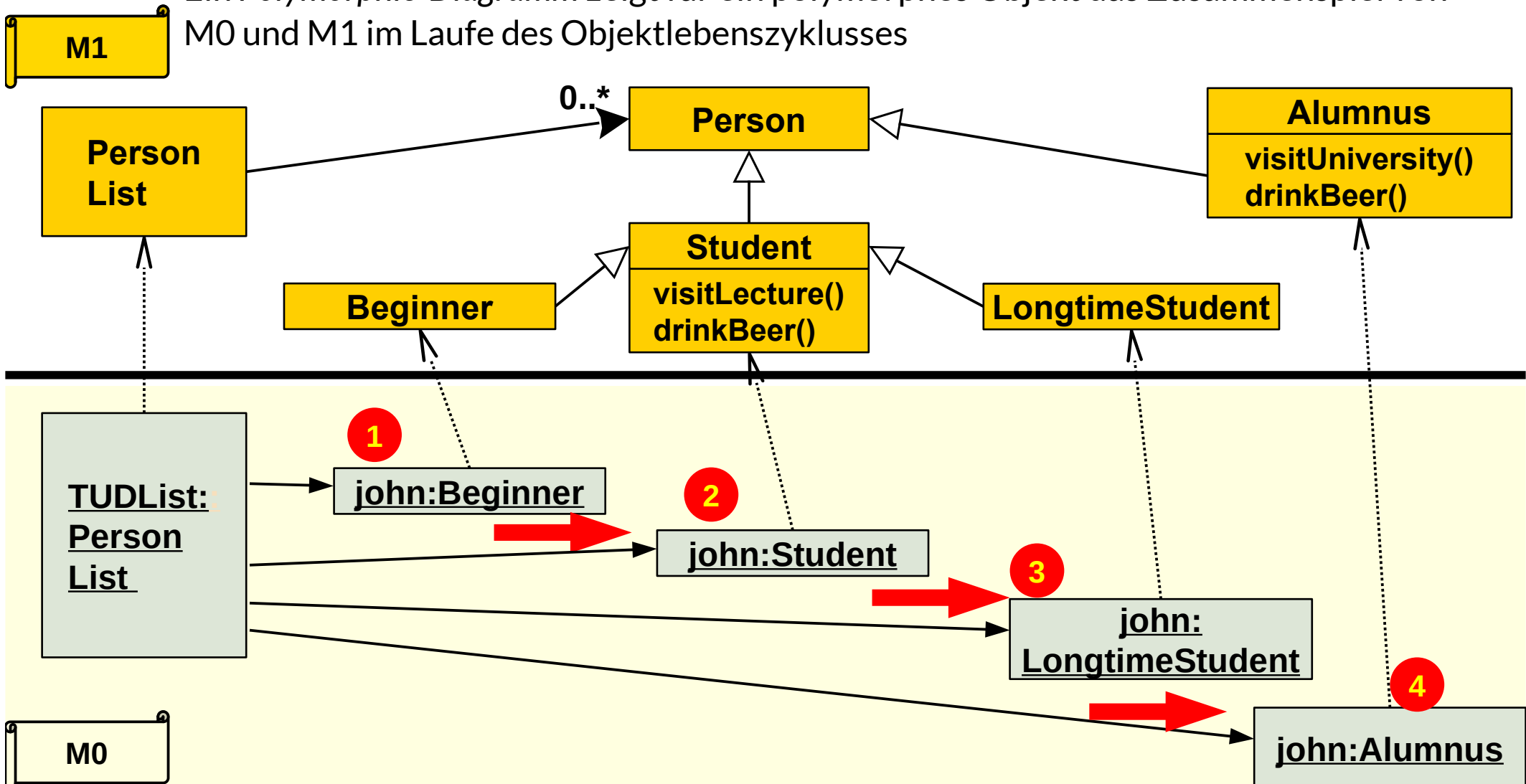


# Polymorphie (polymorphism) als Wechsel von Phasen des Lebens eines Objekts (im Polymorphie-Diagramm)

38

Softwaretechnologie (ST)

- ▶ Zur Laufzeit kann jedes Objekt einer Unterklasse ein Objekt einer Oberklasse vertreten. Das Objekt der Oberklasse ist damit *vielgestaltig* (*polymorphic*).
- ▶ Ein *Polymorphie-Diagramm* zeigt für ein polymorphes Objekt das Zusammenspiel von M0 und M1 im Laufe des Objektlebenszyklusses



# Wechsel der Gestalt (Objektevolution und Polymorphie)

- ▶ Die genaue Unterklasse einer Variablen wird festgelegt
  - Beim **Erzeugen** (der Allokation) des Objekts (Allokationszeit, oft in der Aufbauphase des Objektnetzes), oft in einem alternativen Zweig des Programms alternativ festgelegt
  - Bei einer **neuen Zuweisung** (oft in einer Umbauphase des Objektnetzes)

```
Data data; Person john;
```

```
if (data.hasLeftUniversity())  
    john = new Alumnus();  
else if (data.getYear()==1)  
    john = new Beginner();  
else if (data.getYear()>5)  
    john = new LongtimeStudent();  
else  
    john = new Student();
```

```
if (data.hasHabilitated())  
    john = new Professor();
```

```
// which type has Person john here?
```

```
...
```

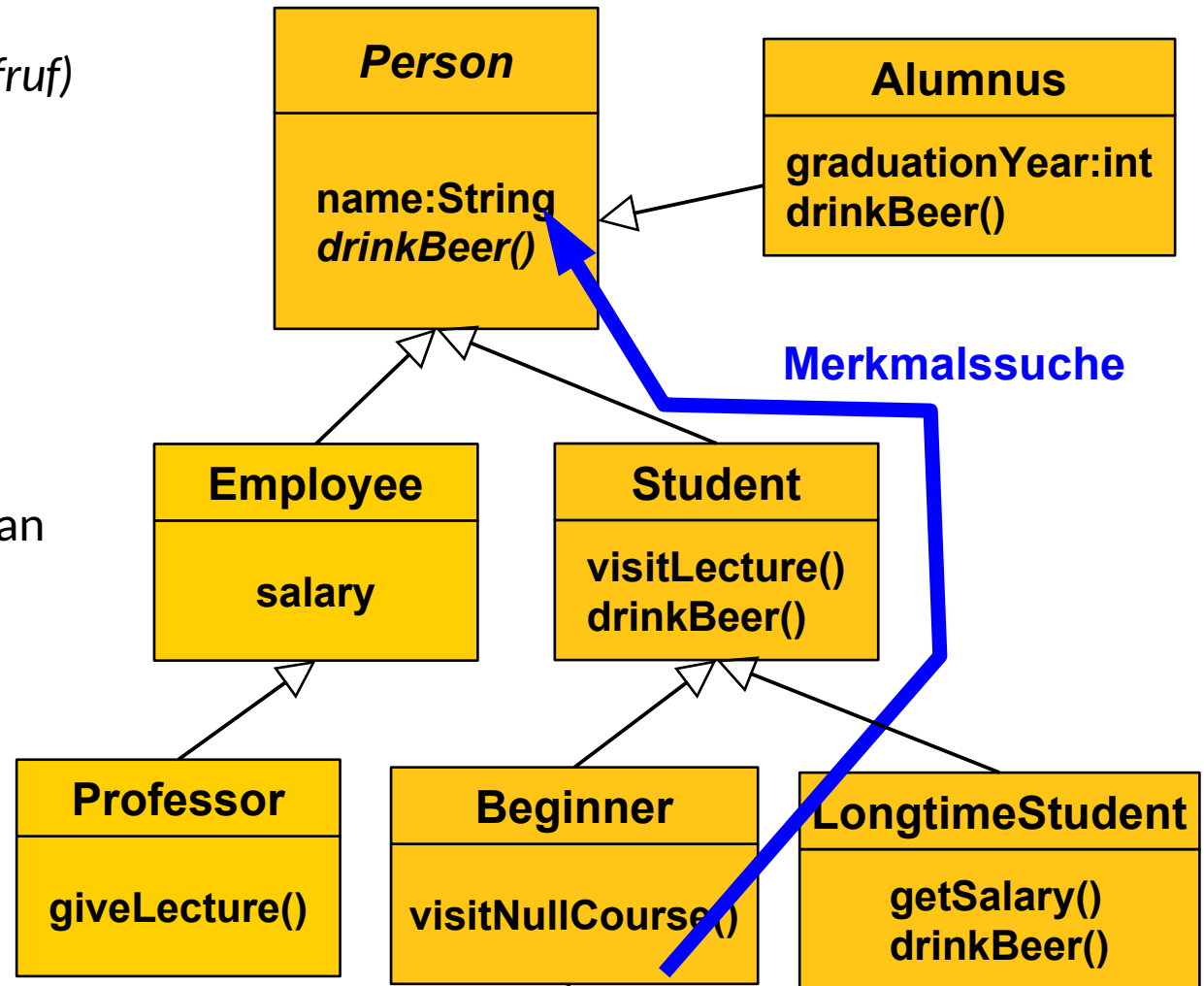
```
// which type has person here?
```

```
// how will the person act?
```

```
john.visitLecture();  
john.drinkBeer();
```

# Dynamischer Aufruf (Polymorpher Aufruf, dynamic dispatch)

- ▶ *Dynamischer Aufruf (polymorpher Aufruf)* realisiert Polymorphie zur Laufzeit
  - Aufruf an Objekte aus Vererbungshierarchien unter Einsatz von Merkmals- (Methoden-)suche (method resolution)
- ▶ Dynamischer Aufruf ist also Aufruf an Objekt + Methodensuche
  - Suche wird oft mit Tabellen realisiert (*dispatch*)



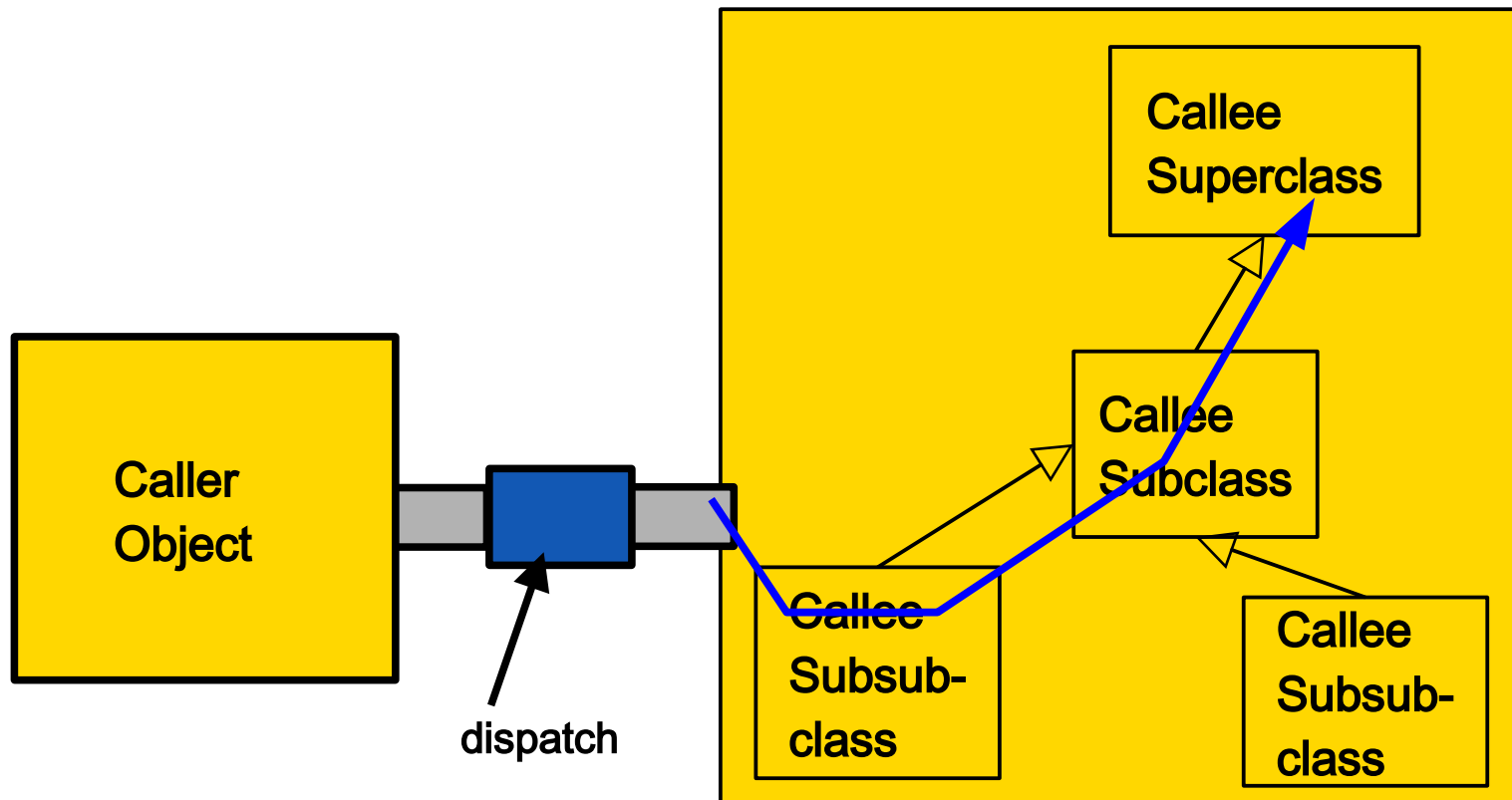
**john:Beginner**

**drinkBeer()**



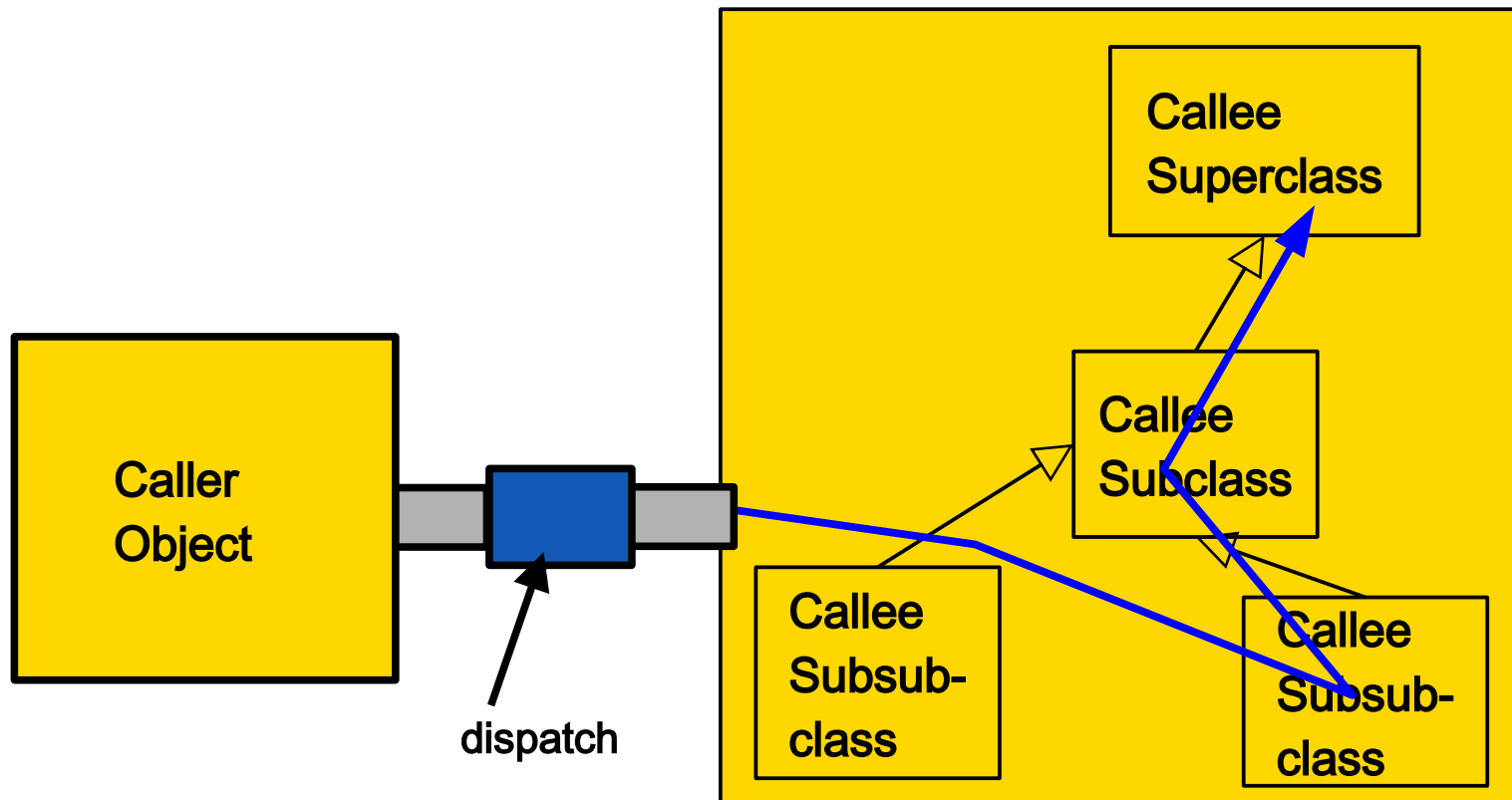
# Dynamischer Aufruf (Polymorpher Aufruf, dynamic dispatch)

- ▶ Vom Aufrufer aus wird ein Suchalgorithmus gestartet, der die Vererbungshierarchie aufwärts läuft, um die passende Methode zu finden
  - Die Suche läuft tatsächlich über die Klassenprototypen
  - Diese Suche kann teuer sein und muß vom Übersetzer optimiert werden (dispatch optimization)



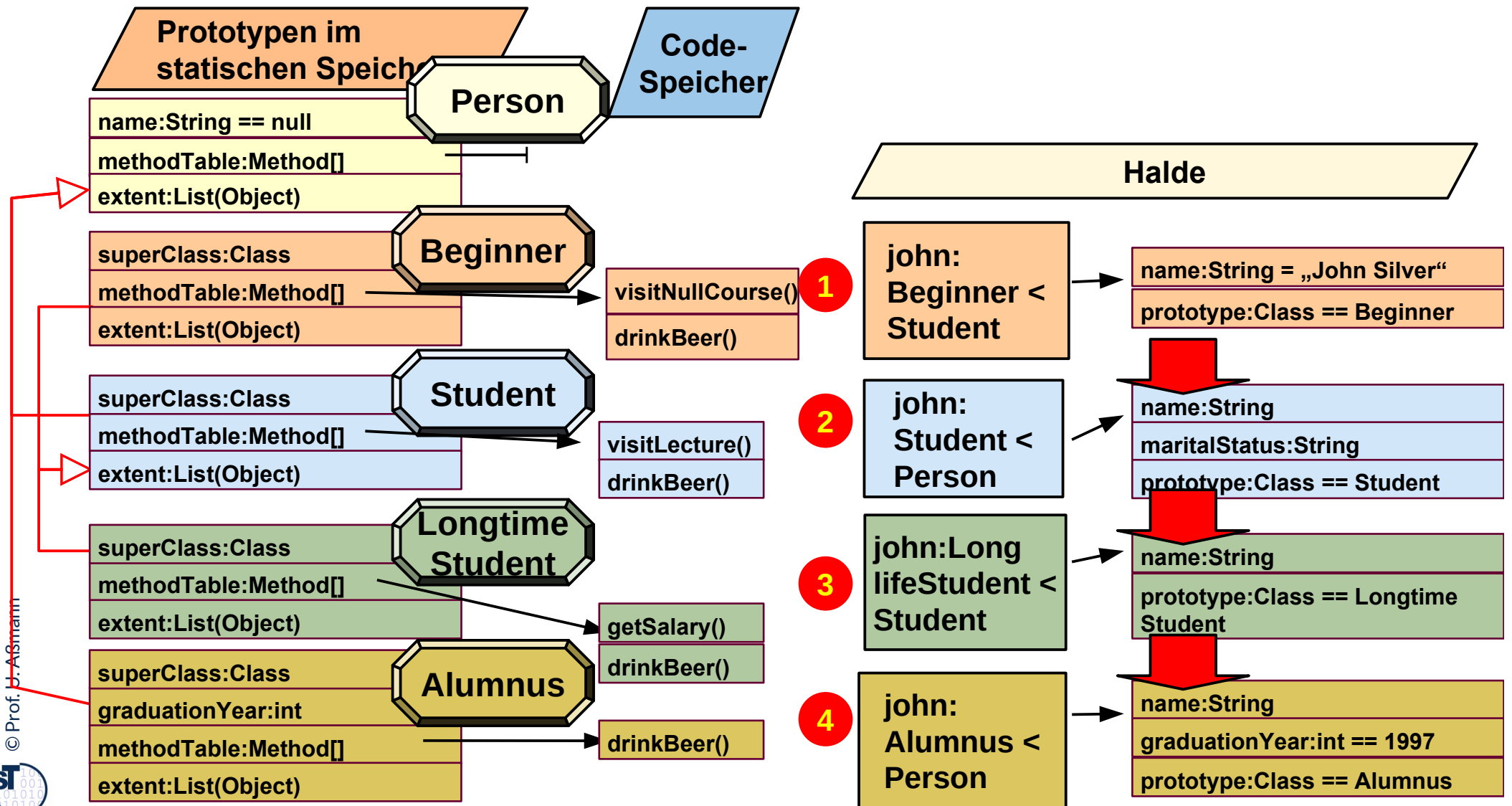
# Dynamischer Aufruf (Polymorpher Aufruf, dynamic dispatch)

- ▶ Dynamischer Wechsel des Typs des Objekts möglich (z.B. auf Schwesterklasse)



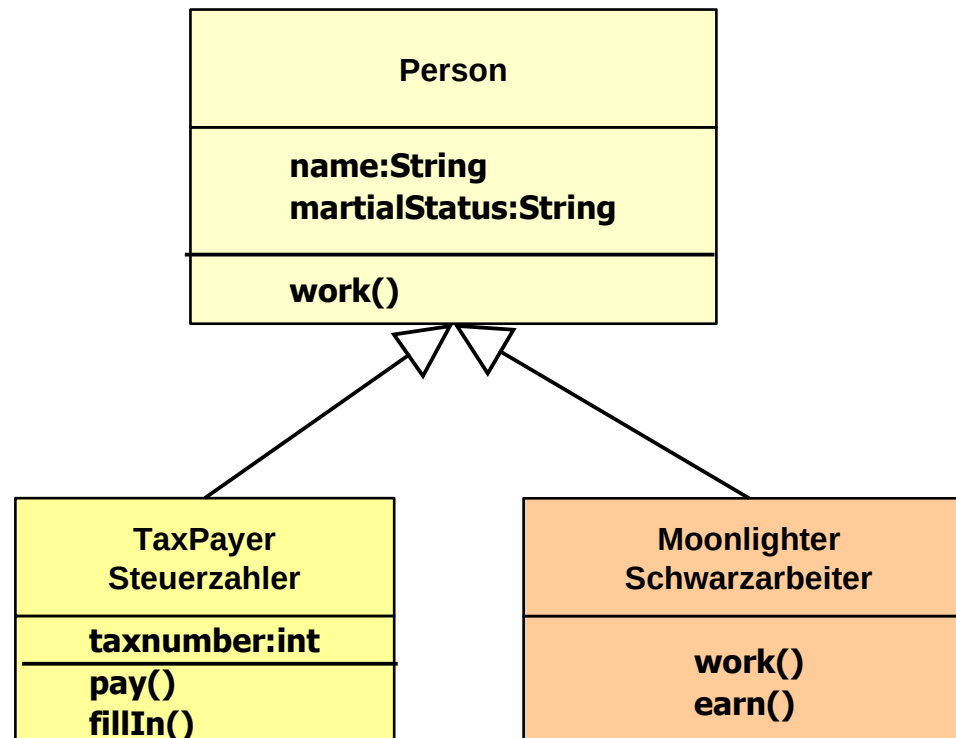
# Was passiert beim polymorphen Aufruf im Speicher?

- Frage: Welche Inkarnation der Methode *drinkBeer()* wird zu den verschiedenen Zeitpunkten im Leben Johns aufgerufen?



# Polymorphe und monomorphe Methoden

- ▶ Methoden, die nicht mit einer Oberklasse geteilt werden, können nicht polymorph sein
- ▶ Die Adresse einer solchen *monomorphen* Methode im Speicher kann statisch, d.h., vom Übersetzer ermittelt werden (*statischer Aufruf*). Eine Merkmalsuche ist dann zur Laufzeit nicht nötig
- ▶ **Frage:** Welche der folgenden Methoden sind poly-, welche monomorph?



- ▶ Codeverschmutzung wird vermieden durch Vererbung
  - Vererbung erlaubt die *Wiederverwendung* von Merkmalen aus Oberklassen,
  - Einfache Vererbung führt zu Vererbungshierarchien
- ▶ Polymorphie erlaubt dynamische Architekturen
  - Merkmalsuche (dynamic dispatch) löst die Bedeutung von Merkmalsnamen auf, in dem von den gegebenen Unterklassen aus aufwärts gesucht wird
  - Polymorphie benutzt Merkmalsuche, um die Mehrdeutigkeit von Namen in einer Vererbungshierarchie aufzulösen
  - Monomorphe Aufrufe sind schneller, weil Merkmalsuche eingespart werden kann
- ▶ Die Klasse `Object` enthält als implizite Oberklasse der Java-Bibliothek gemeinsam nutzbare Funktionalität für alle Java-Klassen
- ▶ Generische Klassen ermöglichen typsichere Wiederverwendung von Code über Typ-Parameter → der Compiler meldet mehr Fehler

# Warum ist das wichtig?

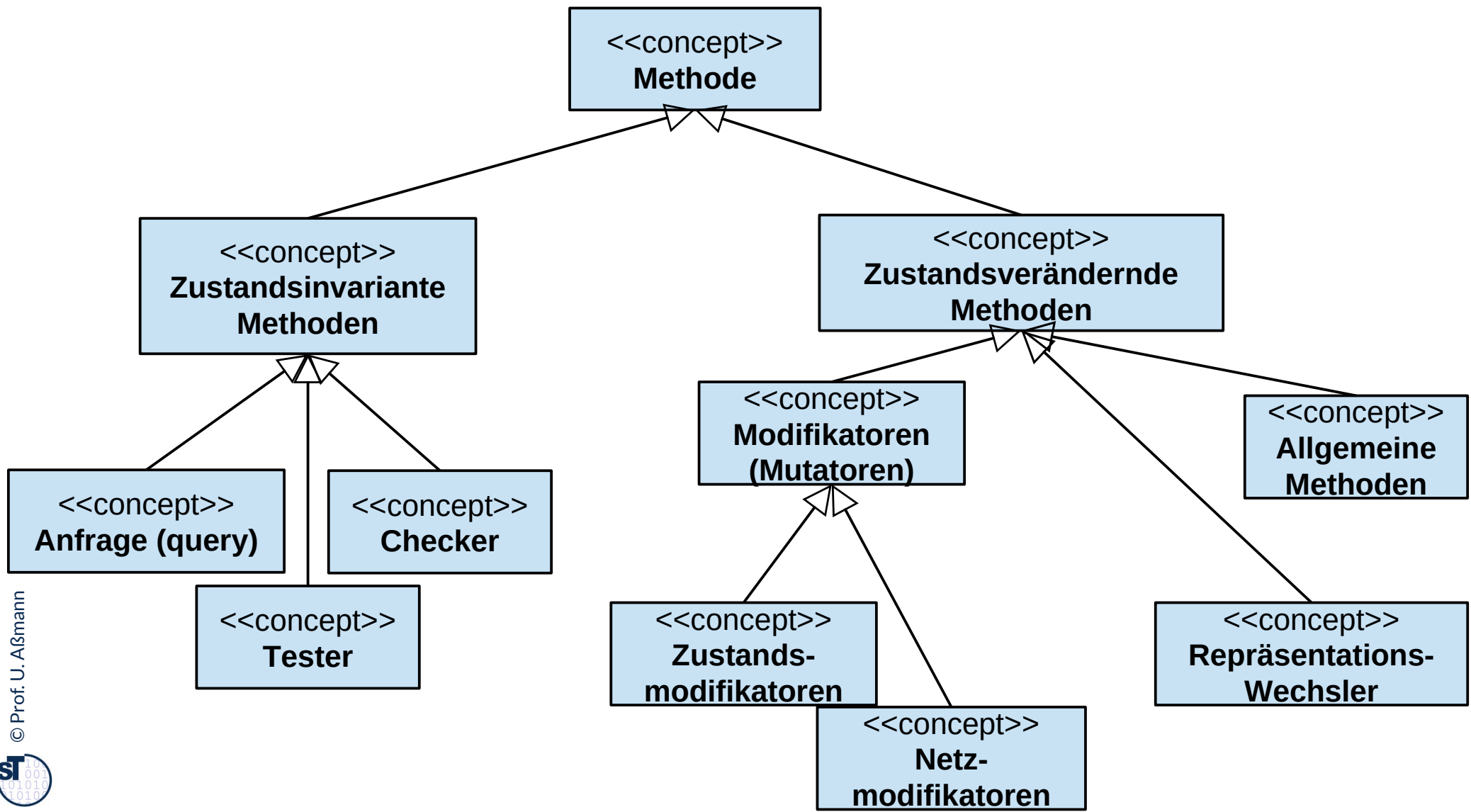
- ▶ **Wiederverwendung** ist eines der Hauptprobleme des Software Engineering
  - In einem Programm
  - Von Projekt zu Projekt
  - Von Produkt zu Produkt (Produktfamilien, Produktlinien)
- ▶ Wiederverwendung ist das Hauptmittel der Softwarefirmen, um **profitabel** arbeiten zu können:
  - Schreibe und teste einmal und wiederverwende oft
  - Alle erfolgreichen Geschäftsmodelle von Softwarefirmen basieren auf Wiederverwendung (→ Produktlinien, → Produktmatrix)
  - Ohne Wiederverwendung kein Verdienst und Überleben als Softwarefirma
- ▶ Firmen, die Wiederverwendung beherrschen, können neue Produkte sehr schnell erzeugen (reduction of time-to-market)
  - und sich an wechselnde Märkte gut anpassen
- ▶ Firmen mit guter Wiederverwendungstechnologie leben länger

# Verständnisfragen

- ▶ Geben Sie eine Begriffshierarchie der Methodenarten an. Welche könnten Sie sich noch denken?
- ▶ Geben Sie eine Begriffshierarchie des Klassenbegriffs an. Welche Klassenarten kennen Sie? Wie spezialisieren sie sich?
- ▶ Erweitern Sie die Vererbungshierarchie der Universitätsangehörigen um den Rektor und den Pedell (s. Wikipedia). Wo müssen sie eingeordnet werden?
- ▶ Was bedeutet der Begriff "Refactoring"? Welche Vorteile bietet er?
- ▶ Stellen Sie ein Polymorphie-Diagramm über die Phasen Ihres Lebens auf.
- ▶ Welchen Polymorphie-Zyklus durchläuft der Steuerzahler unseres Beispiels?
- ▶ Kann eine Steuererklärung polymorph sein?
- ▶ Wie würden Sie ein Testprogramm für ein polymorphes Objekt aus einem Polymorphie-Diagramm heraus entwickeln?
- ▶ Welche wesentliche Vorteile hat ein Informatiker in seinem Leben, der das Vererbungskonzept verstanden hat?

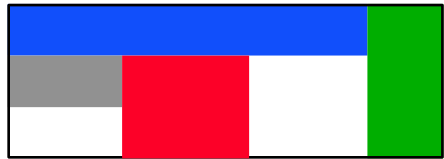
# Bsp: Begriffshierarchie (Taxonomie) der Methodenarten

- ▶ **Wiederholung:** Welche Arten von Methoden gibt es in einer Klasse?
- ▶ **Antwort:** Begriffshierarchie:

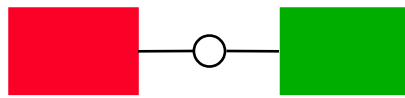




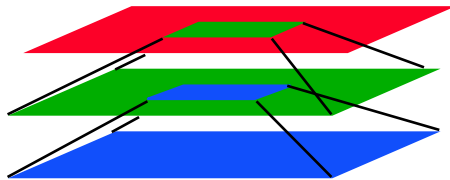
# Prinzipielle Vorteile von Objektorientierung



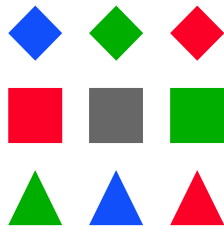
Zuständigkeitsbereiche



Klare Schnittstellen



Hierarchie



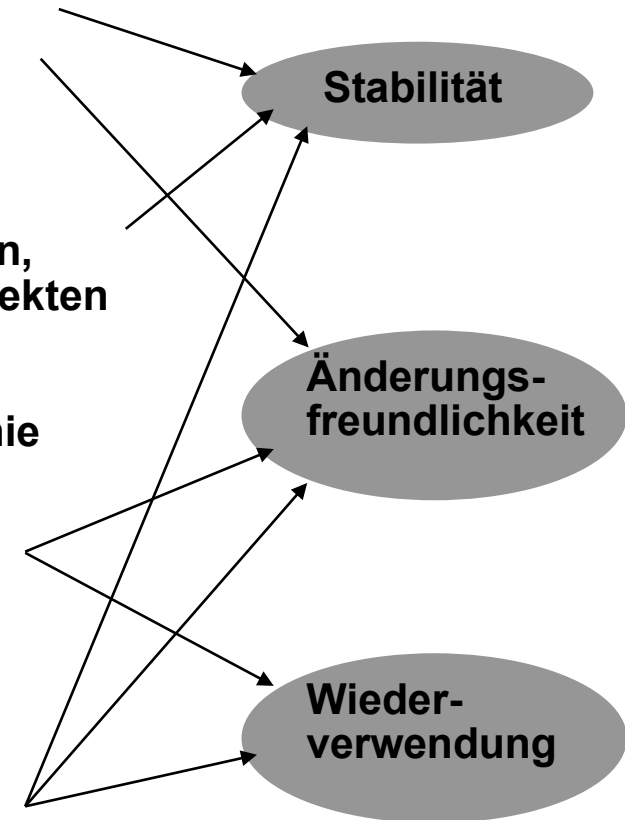
Baukastenprinzip

**Lokalität: Lokale Kapselung von Daten und Operationen, gekapselter Zustand**

**Typen und Typsicherheit**  
Definiertes Objektverhalten,  
Nachrichten zwischen Objekten

**Vererbung und Polymorphie**  
(Spezialisierung),  
Wiederverwendung  
Klassenschachtelung

**Benutzung vorgefertigter Klassenbibliotheken**  
(Frameworks),  
Anpassung durch Spezialisierung  
(Vererbung)



# Q10: Relationen

