Programmiermethodik

Objektorientierte Entwicklung

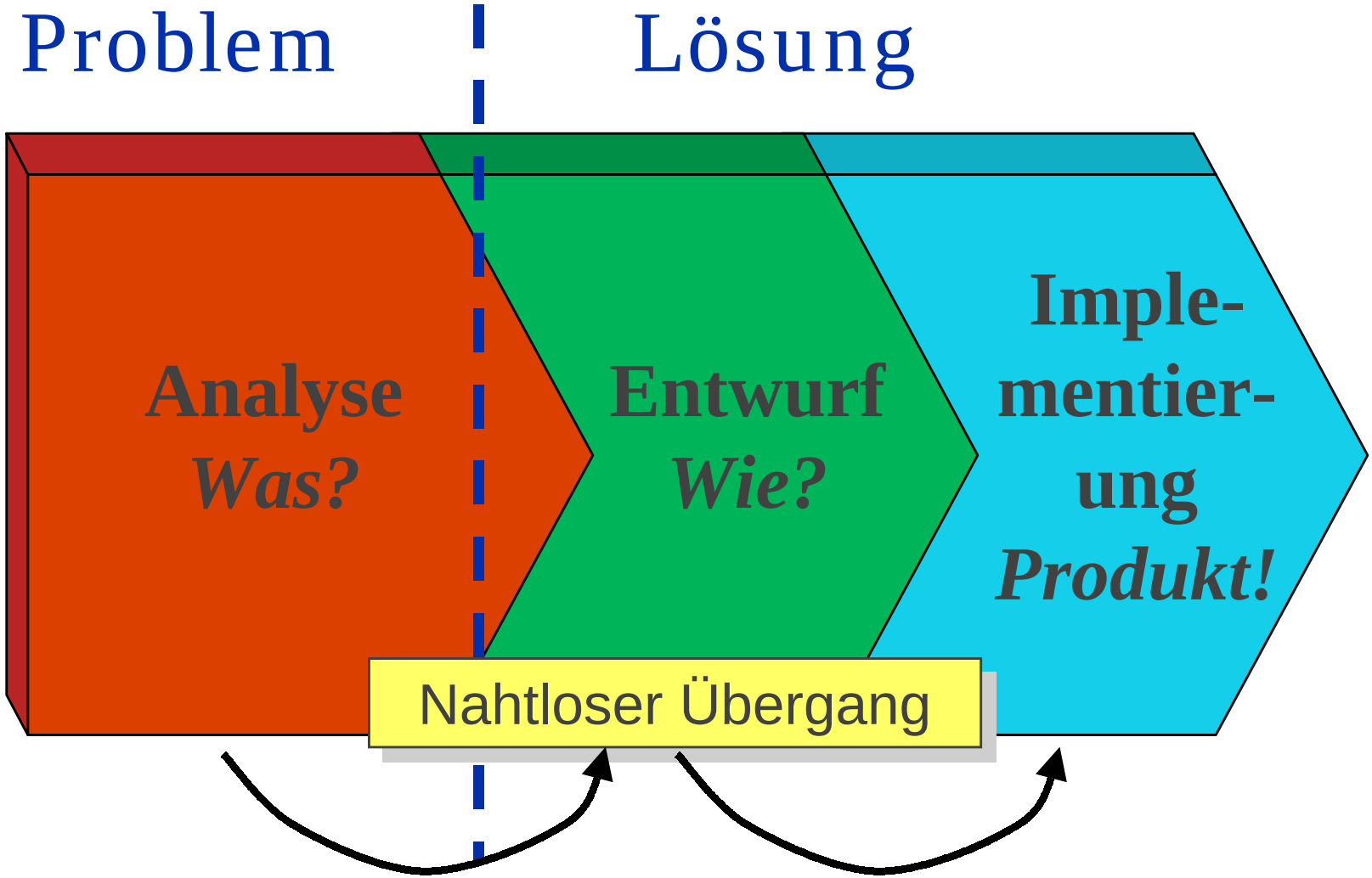
SS 2002

Thomas Kühne

kuehne@informatik.tu-darmstadt.de

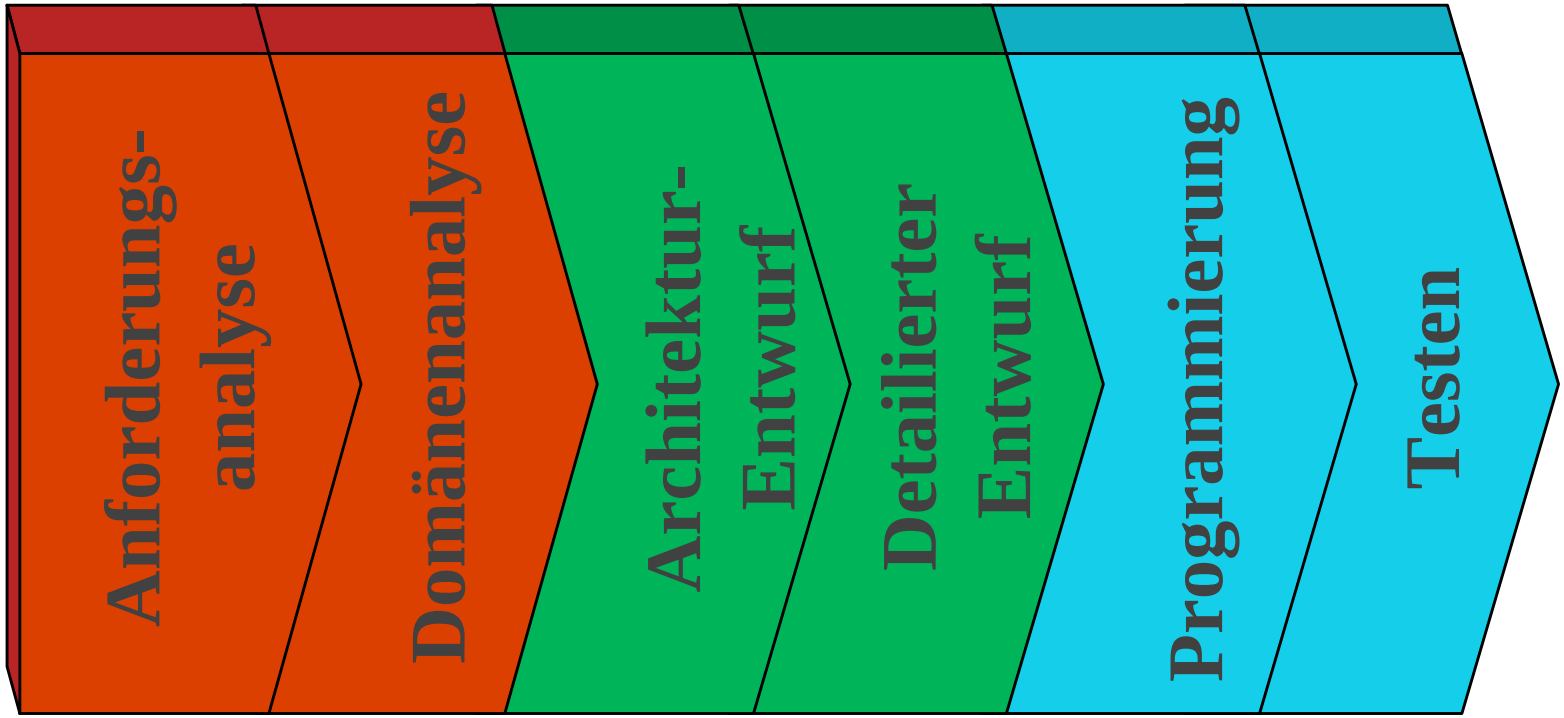
http://www.informatik.uni-mannheim.de/informatik/softwaretechnik

OO-Entwicklungsphasen

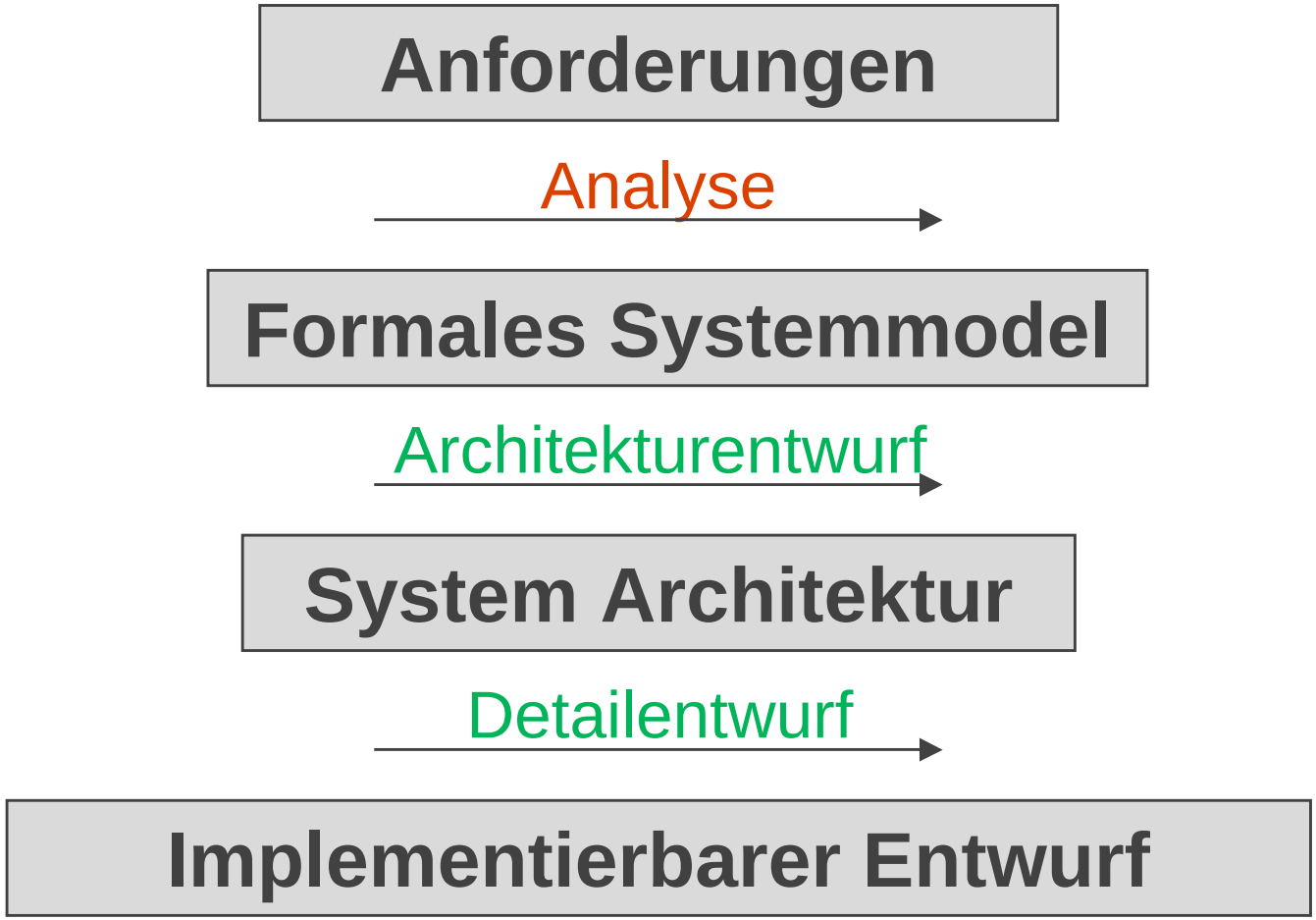


OO-Entwicklungsphasen

Genauer...



Entwicklungsprodukte



Entwicklungsprodukte

Anforderungsanalyse	➡	Use Cases
Domainenanalyse	➡	Konzeptmodel
Architekturentwurf	➡	Subsysteme
Detailentwurf	➡	Klassen Zustandsautomaten Interaktionen
Programmierung	➡	Code
Testen	➡	Fehlerbericht

Geschichte der UML

Das Ende des "Methodenkriegs"

- OOA/OOD (Coad/Yourdon, 1991)
- OMT (Rumbaugh, 1991)
- Booch (Booch, 1994)
- Objectory (Jacobson, 1994)
- Fusion (Hewlett-Packard)
- UML
  - `http://www.omg.org`

UML Diagramme

- Struktur
  - » Use Case Diagramm — *Systembenutzungen*
  - » Strukturdiagramm — *statische Klassen/Objekt Beziehungen*
- Verhalten
  - » Zustandsdiagramm — *reaktives Verhalten*
  - » Aktivitätsdiagramm — *Kontrollfluß*
  - » Sequenzdiagramm — *(zeitliche) Interaktionen*
  - » Kollaborationsdiagramm — *(strukturelle) Interaktionen*
- Implementierung
  - » Komponentendiagramm — *Ausführungseinheiten*
  - » Deploymentdiagramm — *Installationsplan*

Was ist Analyse?

- Anforderungsanalyse
  - » Benutzerwünsche feststellen
  - » Mögliche Leistungen herausarbeiten
- Domänenanalyse
  - » Domänenmodell entwickeln
  - » Systemverständnis validieren
  - » Anforderungen validieren

Warum Analyse?

Bestimmung der Anforderungen schwierig, da

- “Tech Talk”, domänenspezifischer “Slang”
- Expertenwissen ist nicht einfach zu erschliessen
- Implizite Annahmen sind oft falsch
- Zukünftige Benutzer haben oft nur unvollständige Vorstellungen/Ideen
- Nie endender Appetit nach mehr Funktionalität

Problembeschreibung

Eine Universitätsbücherei, nett am Fluß gelegen, hat Universitäts-Angestellte und Studenten als primäre Kunden.

Ein Angestellter kann bis zu 20 Bücher bis zu 4 Wochen ausleihen. Studenten können bis zu 10 Bücher bis für maximal 1 Woche ausleihen. Magazine sind höchstens für 3 Tage ausleihbar.

Ein Benutzer kann nach Bücher mit [illegible] Kataloges suchen und Reservierung[illegible] Bücher anmelden. Eine Erinnerung wird an Ausleiher ausgestellt sobald die Ausleihzeit überschritten wurde. Bibliothekare können ausleihen, reservieren, Bücher hinzufügen oder entfernen.

Identifizieren von Fachbegriffen und Aktivitäten

Problembeschreibung

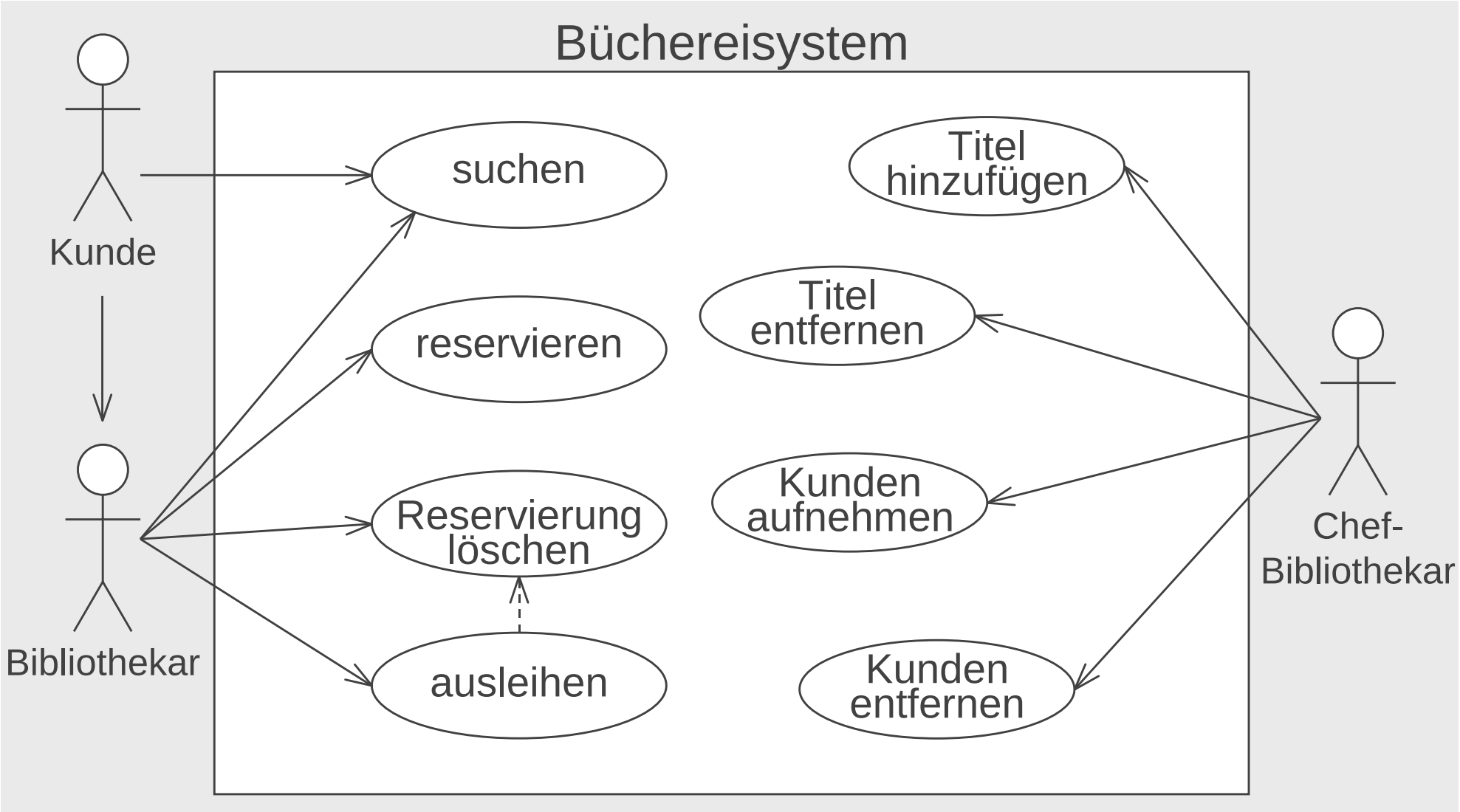
Eine **Universitätsbücherei**, nett am Fluß gelegen, hat Universitäts-**Angestellte** und **Studenten** als primäre **Kunden**.

Ein Angestellter kann bis zu 20 **Bücher** bis zu 4 Wochen *ausleihen*. Studenten können bis zu 10 Bücher bis für maximal 1 Woche ausleihen. **Magazine** sind höchstens für 3 Tage ausleihbar.

Ein **Benutzer** kann nach Bücher mit Hilfe eine Online-**Kataloges** *suchen* und **Reservierungen** für **ausgeliehene Bücher** *anmelden*. Eine **Erinnerung** wird an Ausleiher *ausgestellt* sobald die Ausleihzeit überschritten wurde. **Bibliothekare** können ausleihen, reservieren, Bücher hinzufügen oder entfernen.

Konzept oder Eigenschaft?

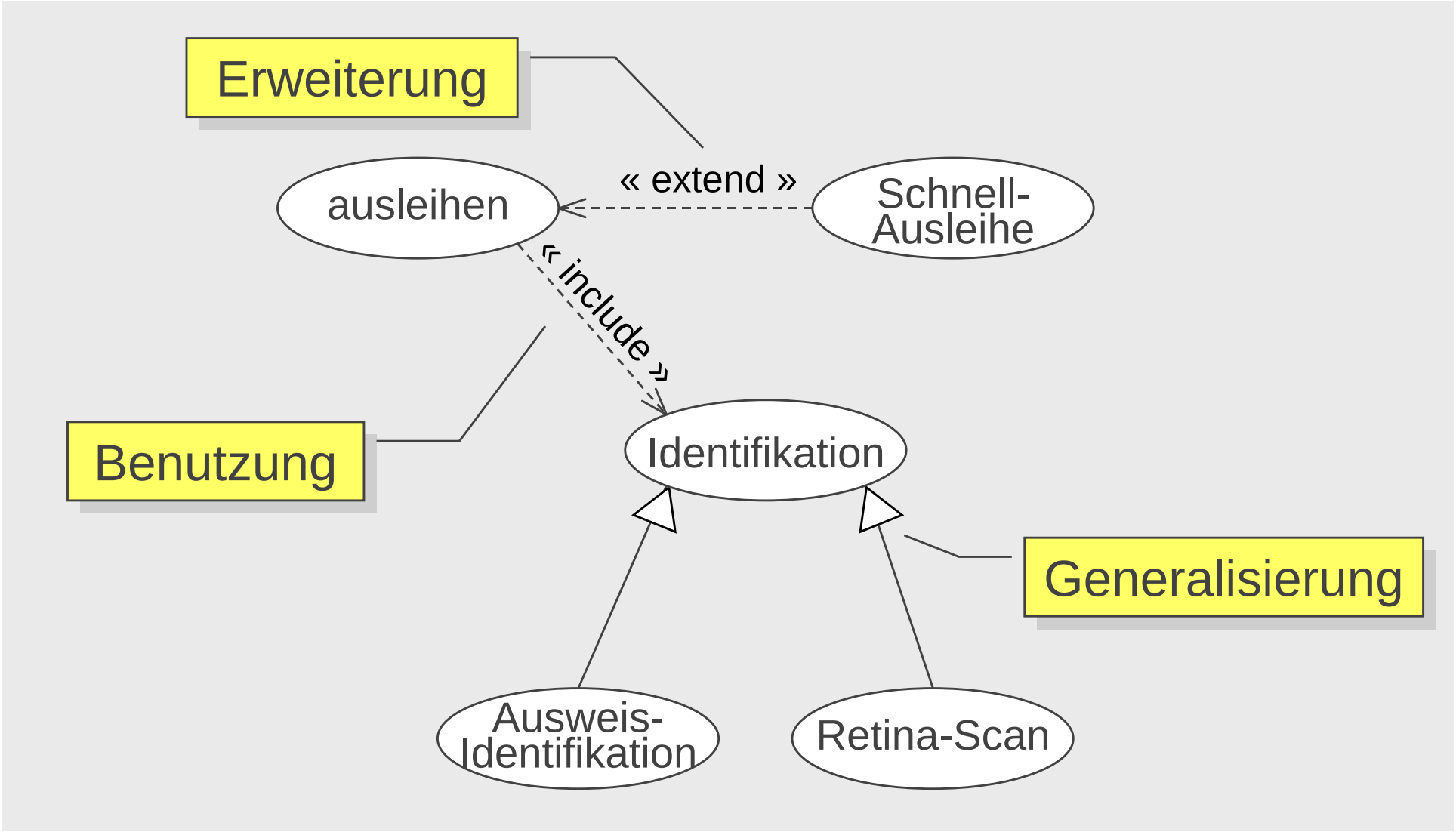
Bücherei Use Cases



Use Case Beziehungen

- Erweiterung (extend)
  - » Eine Variante oder Ausnahmesituation erweitert den Normallfall
- Benutzung (include)
  - » zur Vermeidung doppelter Beschreibungen können sich Use Cases untereinander "enthalten"
- Generalisierung
  - » Beziehung zwischen einer allgemeinen Formulierung und spezialisierten Formen

Use Case Beziehungen

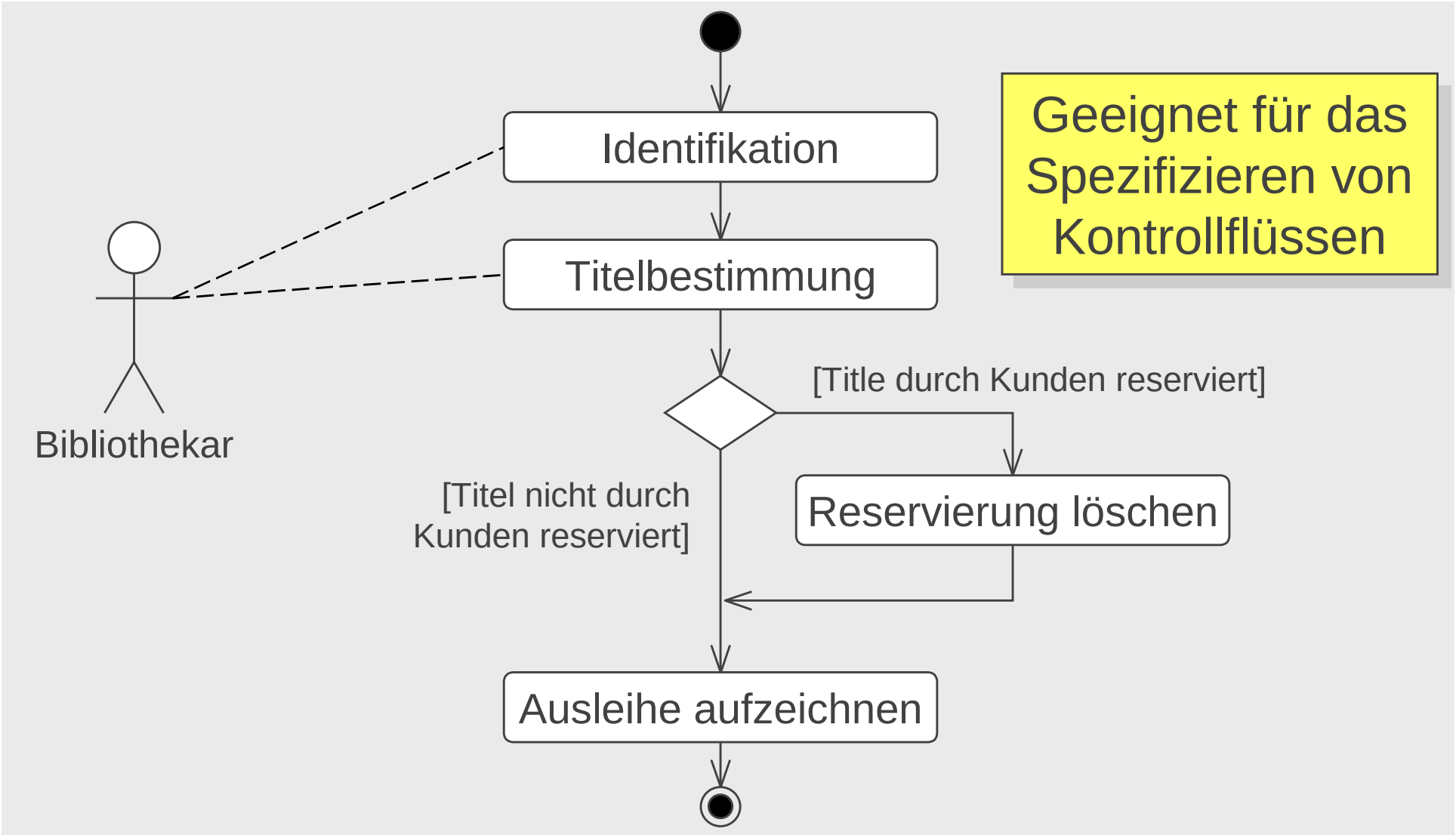


Use Case Beschreibung

Titel ausleihen

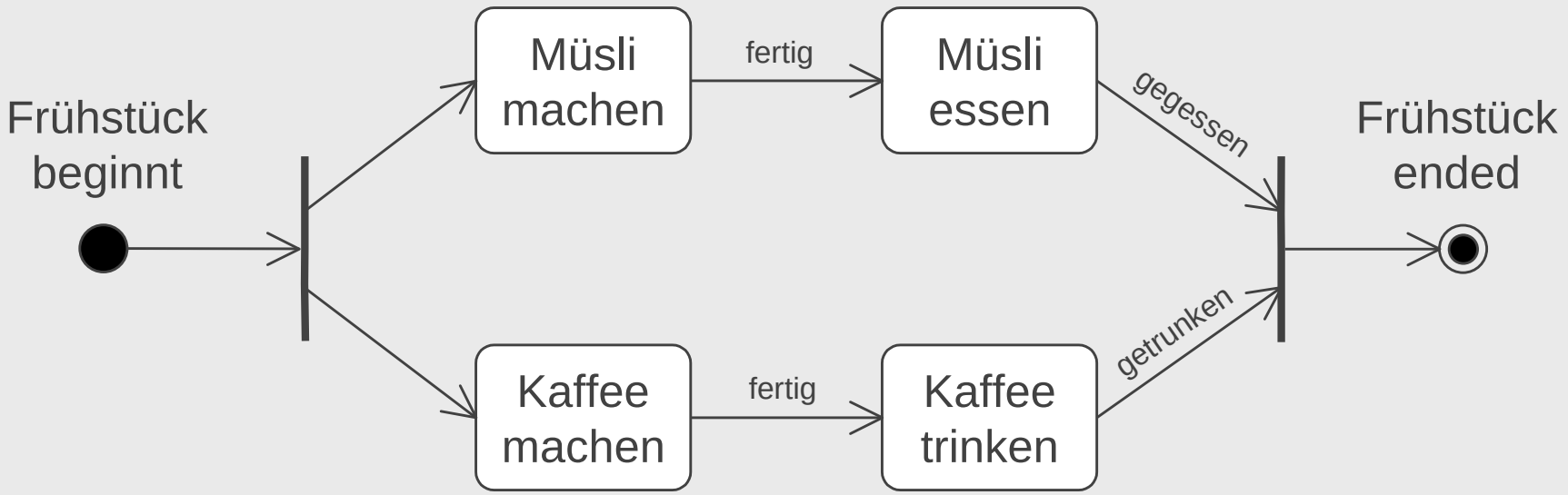
- Kunden identifizieren
- Titel bestimmen
- Ist der Titel bereits (vom Kunden) reserviert?
  - » ja ➔ Reservierung löschen
- Ausleihaufzeichnung erzeugen
- Titel an Kunden aushändigen

Aktivitätsdiagramm



Aktivitätsdiagramm

Paralleles Frühstück

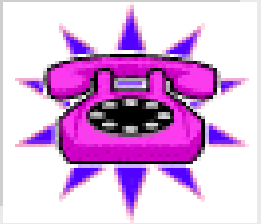


Objektorientierte Analyse

Weltsicht: Kooperierende Objekte

Chef-Schere

Bildtelefon am Eingang



Telefon auf Schreibtisch

The basic principle of recursive design is to make the parts have the same power as the whole rather than dividing the computer into lesser stuffs, like data structures and procedures, we should divide it into lots of little computers that communicate together.

Alan Kay

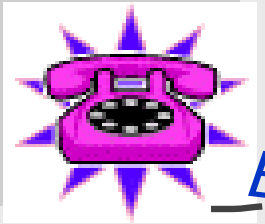
Objektorientierte Analyse

Weltsicht: Kooperierende Objekte

Klassen sind in einer Generalisierungshierachie organisiert

Chef-Schere

Bildtelefon am Eingang



Telefon auf Schreibtisch

Exemplar

Typ

Büromaterial

is-a

is-a

mechanisches Bürogerät

elektro-mechanisches Bürogerät

is-a

is-a

Schere

Telefon

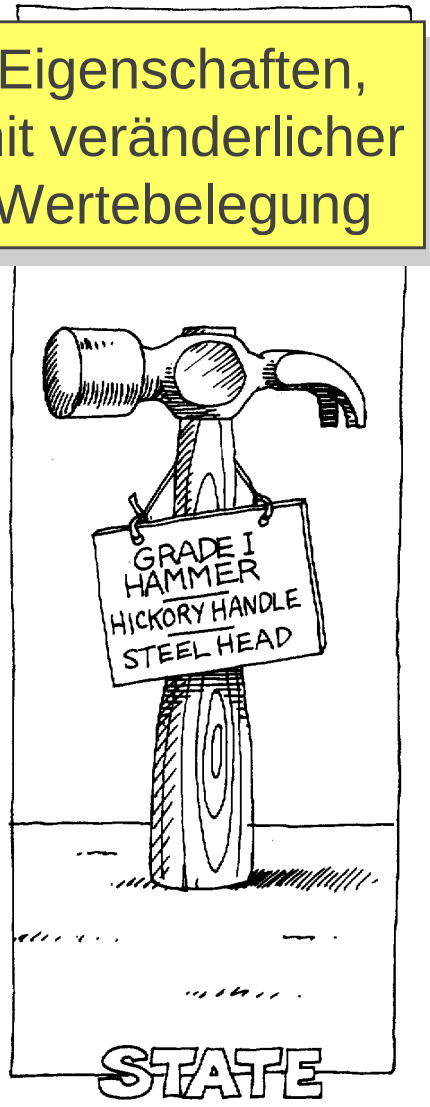
is-a

Bildtelefon

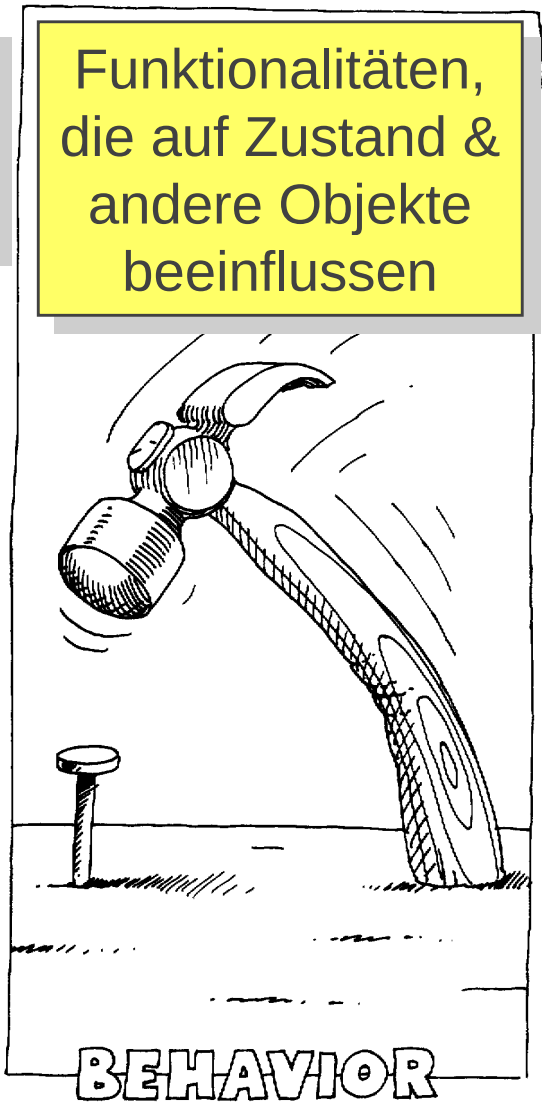
Jedes Objekt ist Instanz einer Klasse

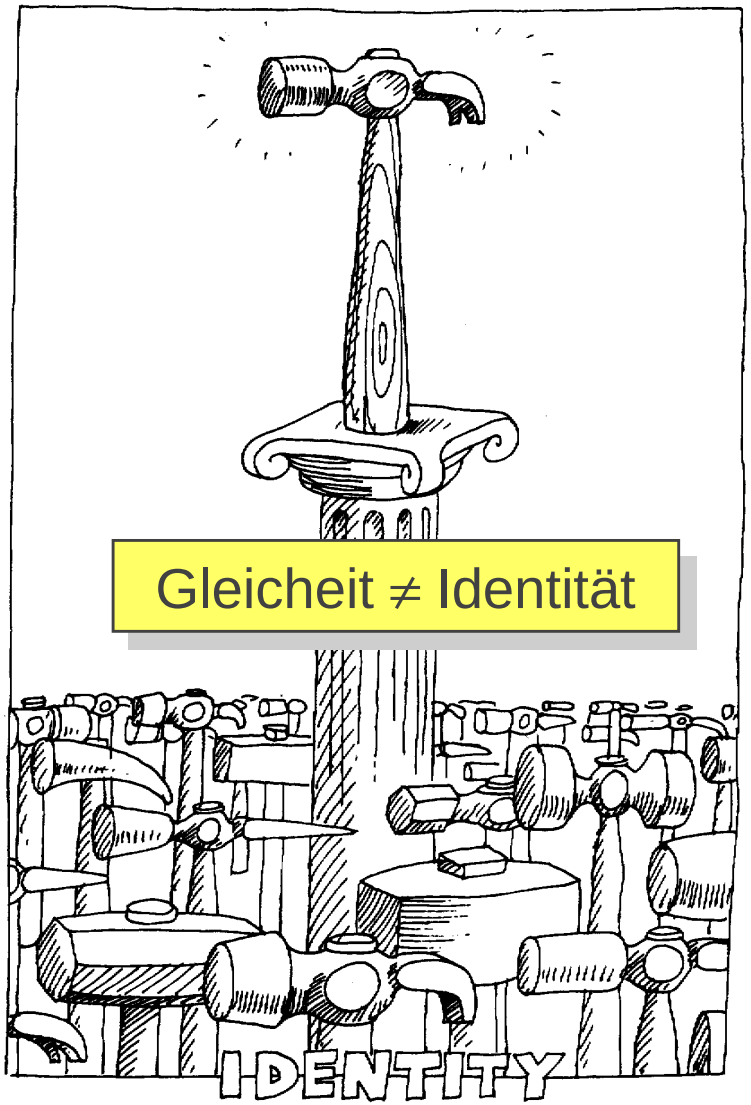
Was ist ein Objekt?

Eigenschaften, mit veränderlicher Wertebelegung



Funktionalitäten, die auf Zustand & andere Objekte beeinflussen



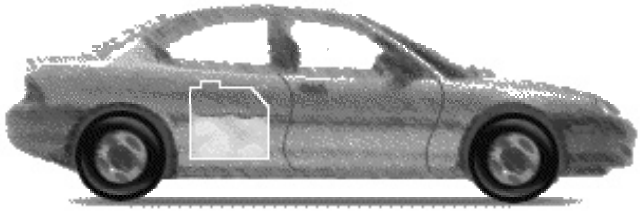


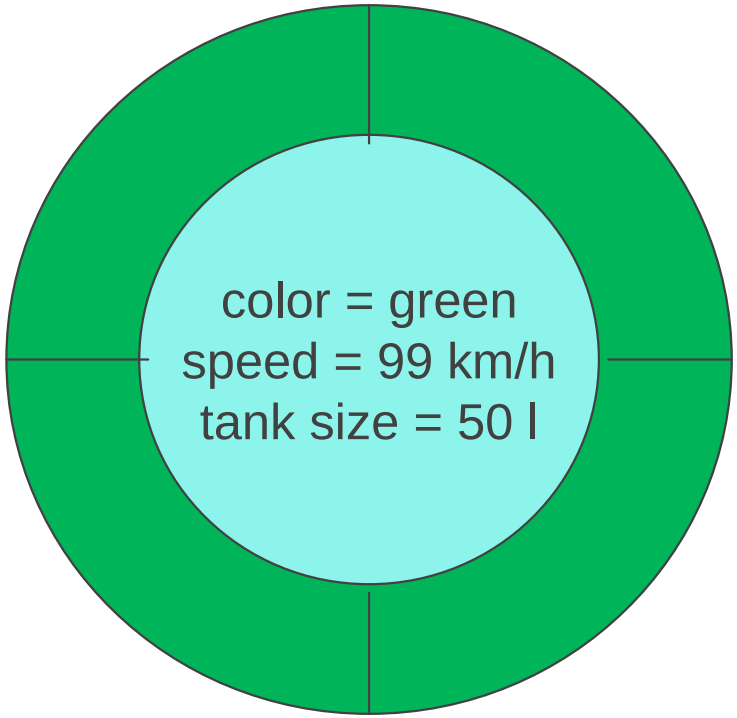
Gleicheit ≠ Identität

Zeichnungen aus Booch

Eigenschaften

Zustand wird durch Wertebelegungen definiert

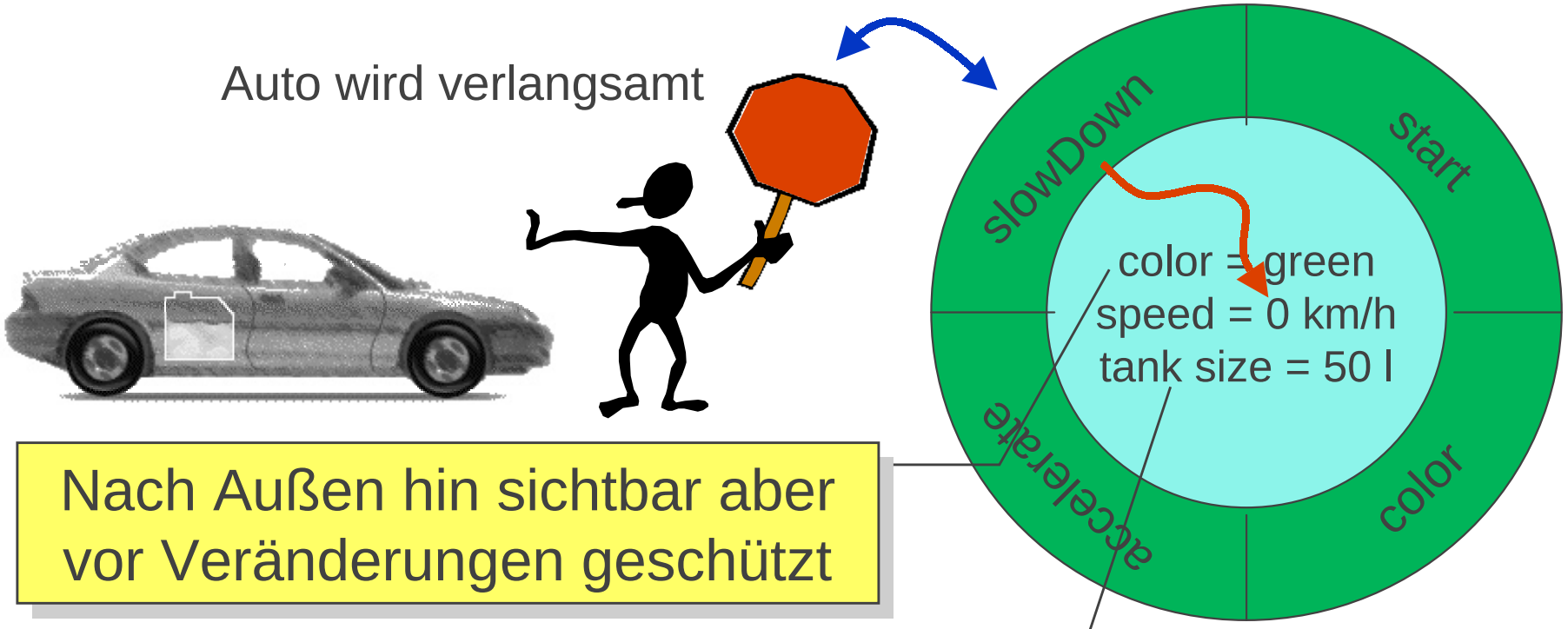




Auto Objekt

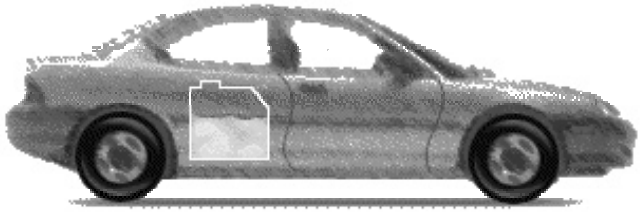
Funktionalitäten

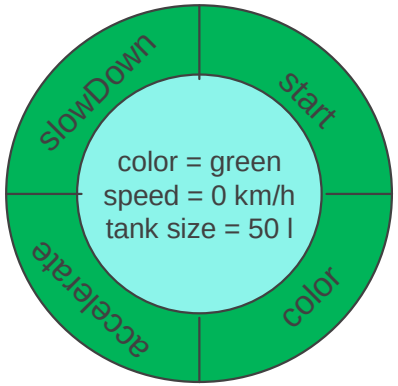
Methoden erlauben disziplinierte Änderungen des Zustands



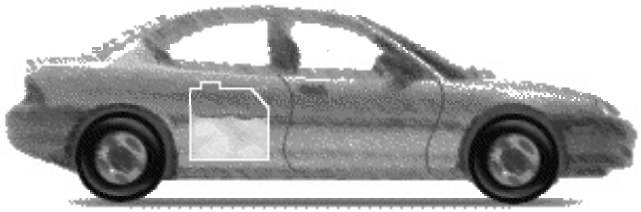
Identität

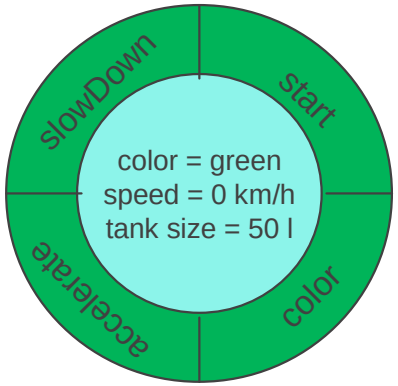
Identität ermöglicht die Unterscheidung von Objekten mit gleichem Zustand





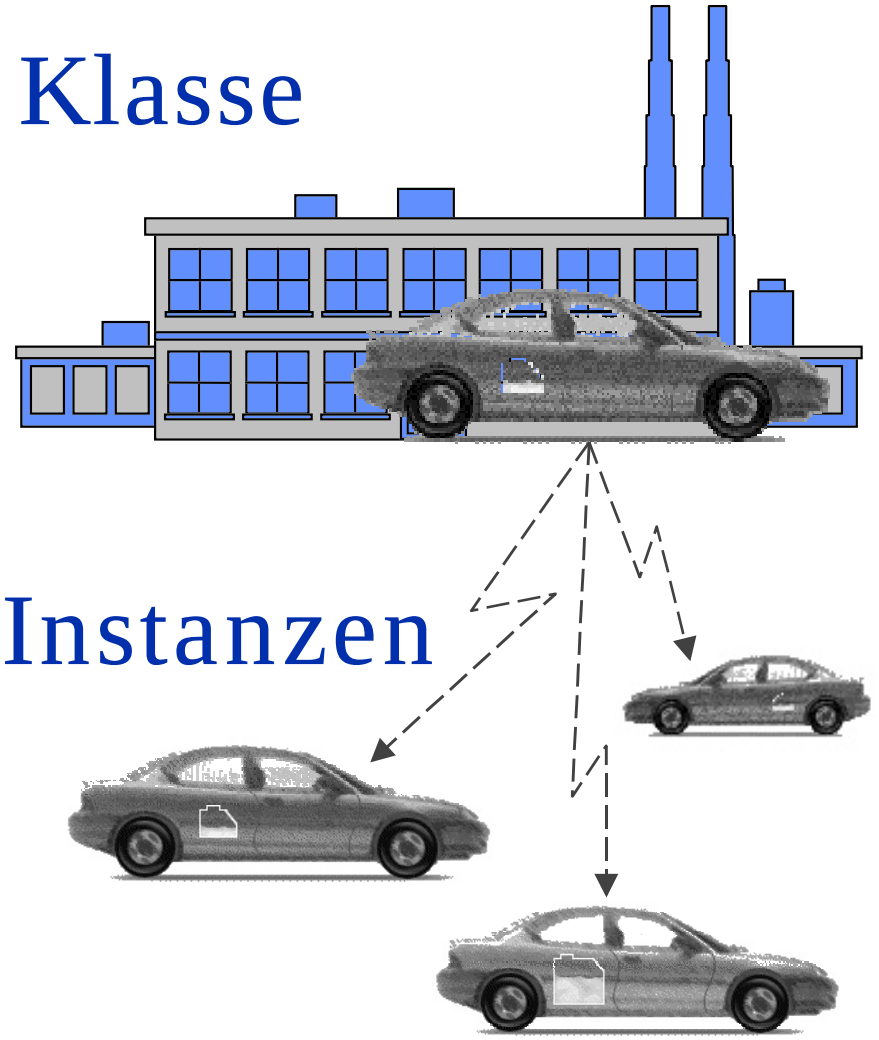
Gleich aber nicht identisch



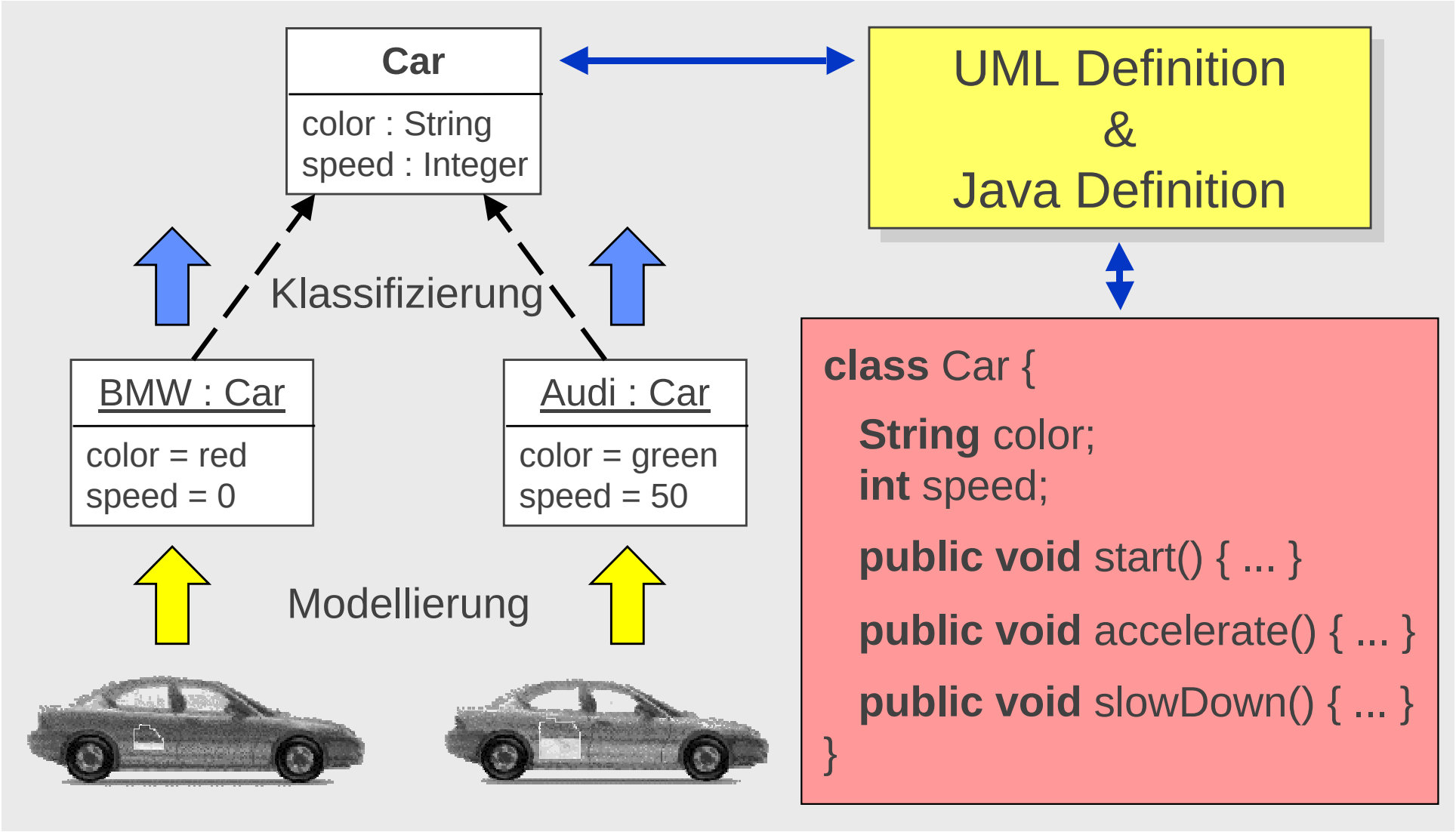


Objekte und Klassen

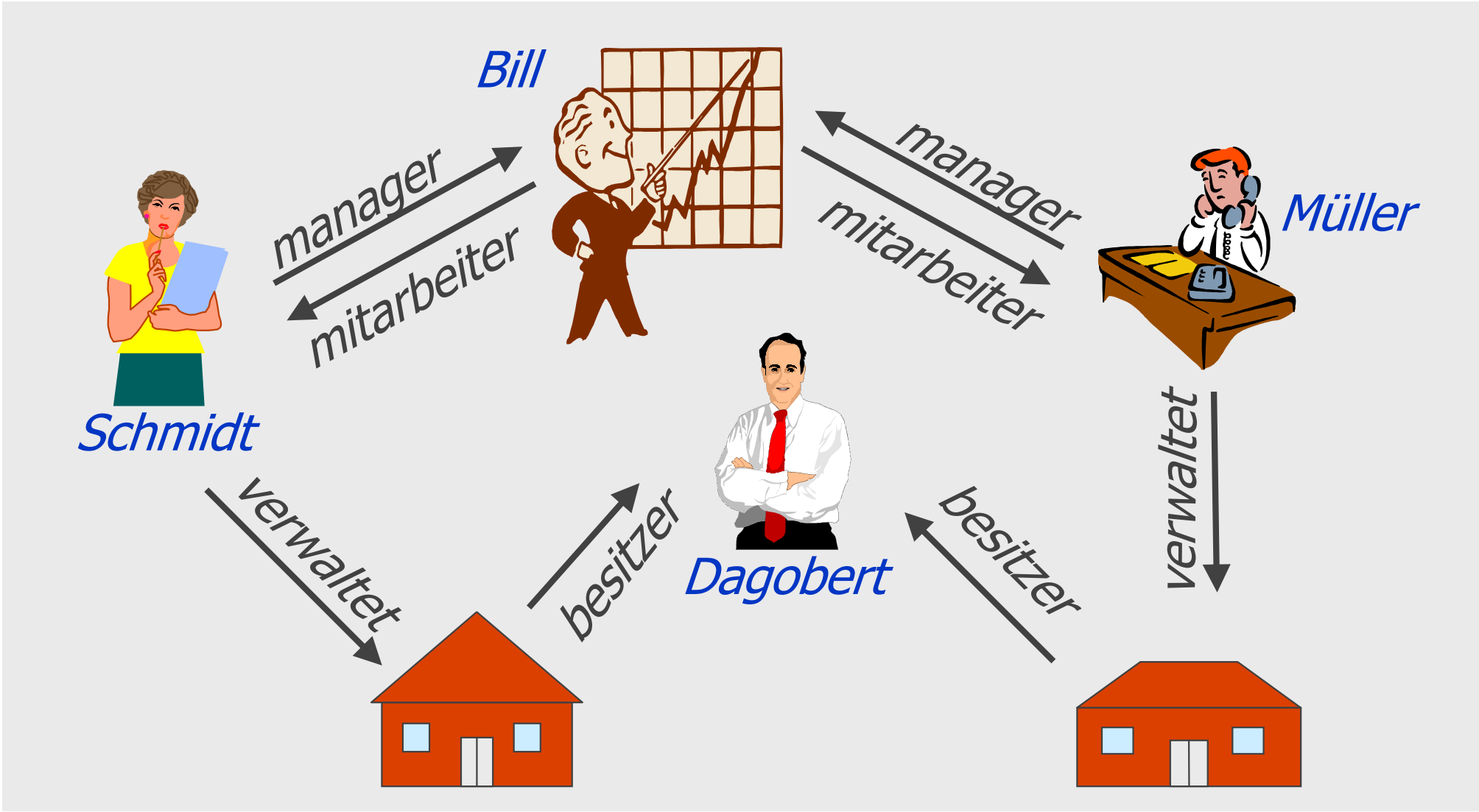
- Eine Klasse ist eine "Fabrik" für Objekte eines bestimmten Typs
- Sie definiert die Eigenschaften und das Verhalten der Objekte
- Jedes Objekt hat seinen individuellen Zustand



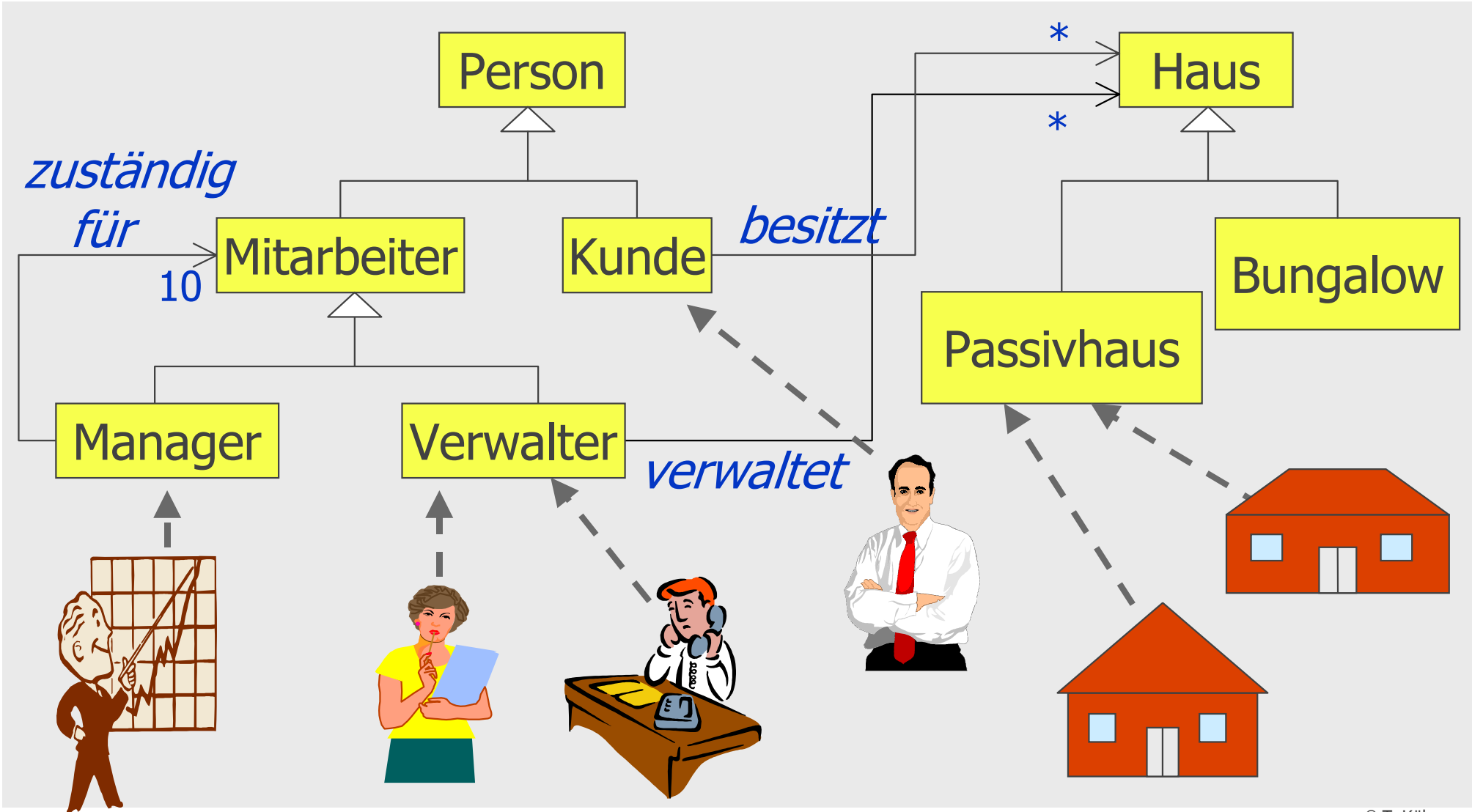
Objekte und Klassen



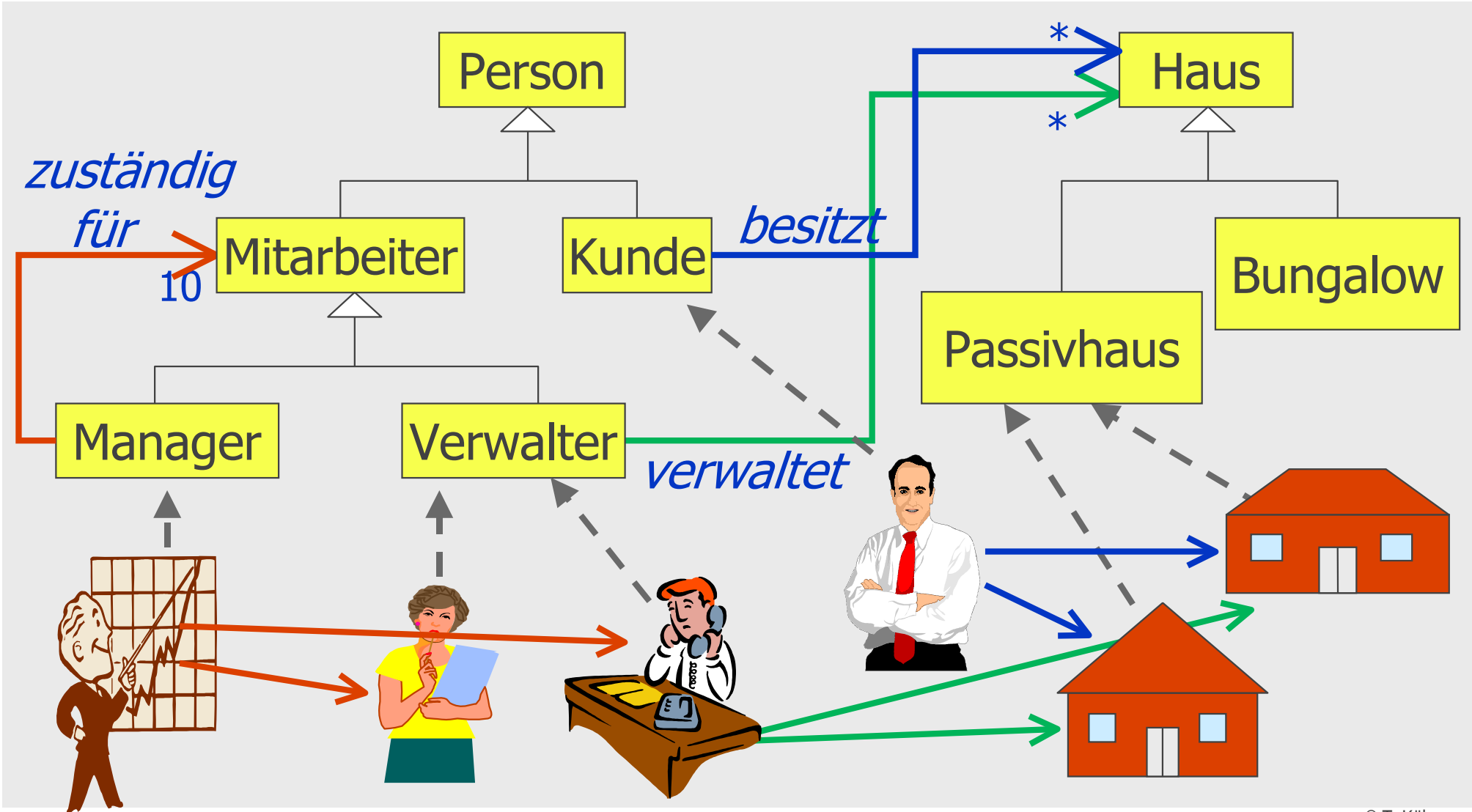
Kooperierende Objekte



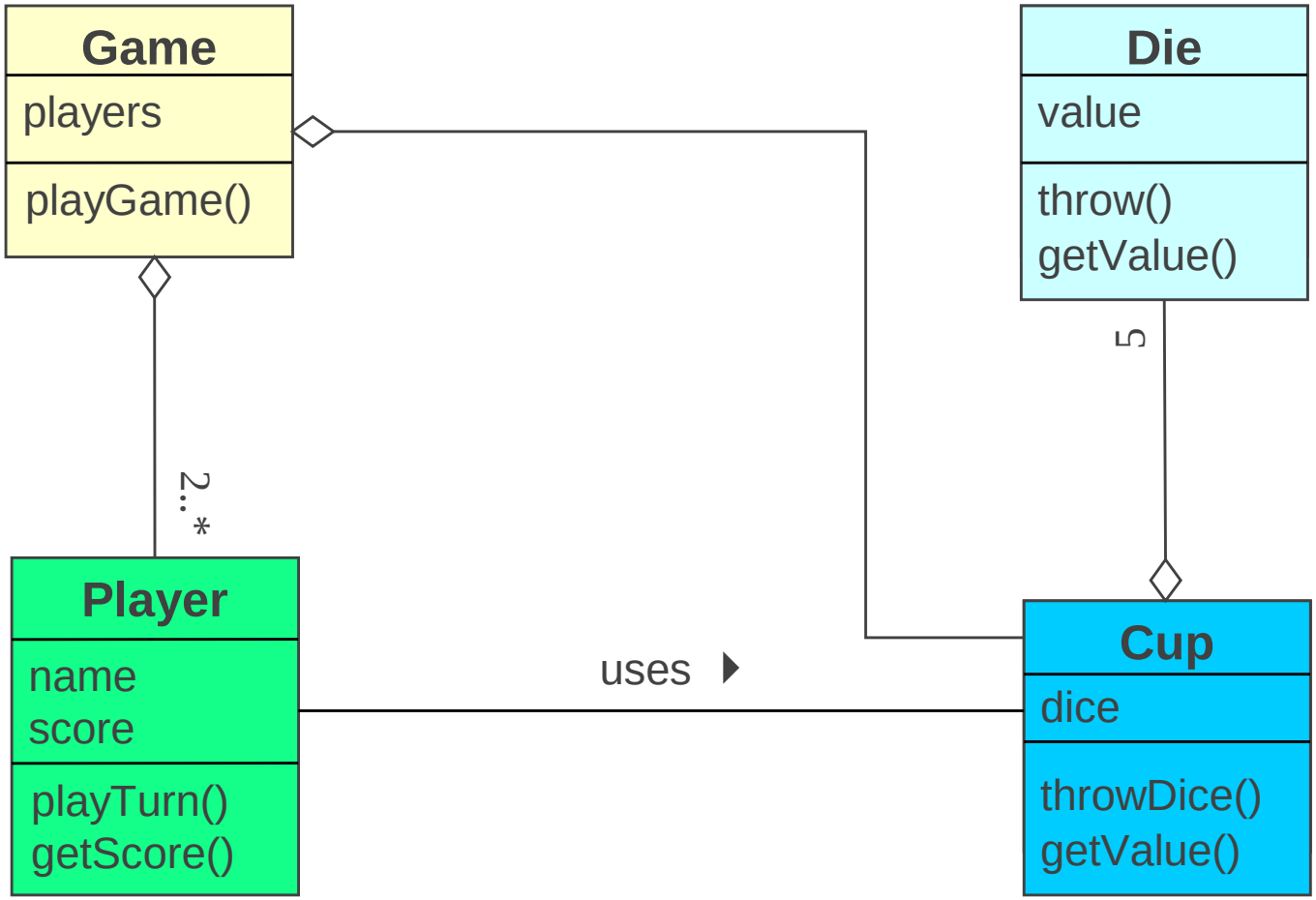
Klassifikation



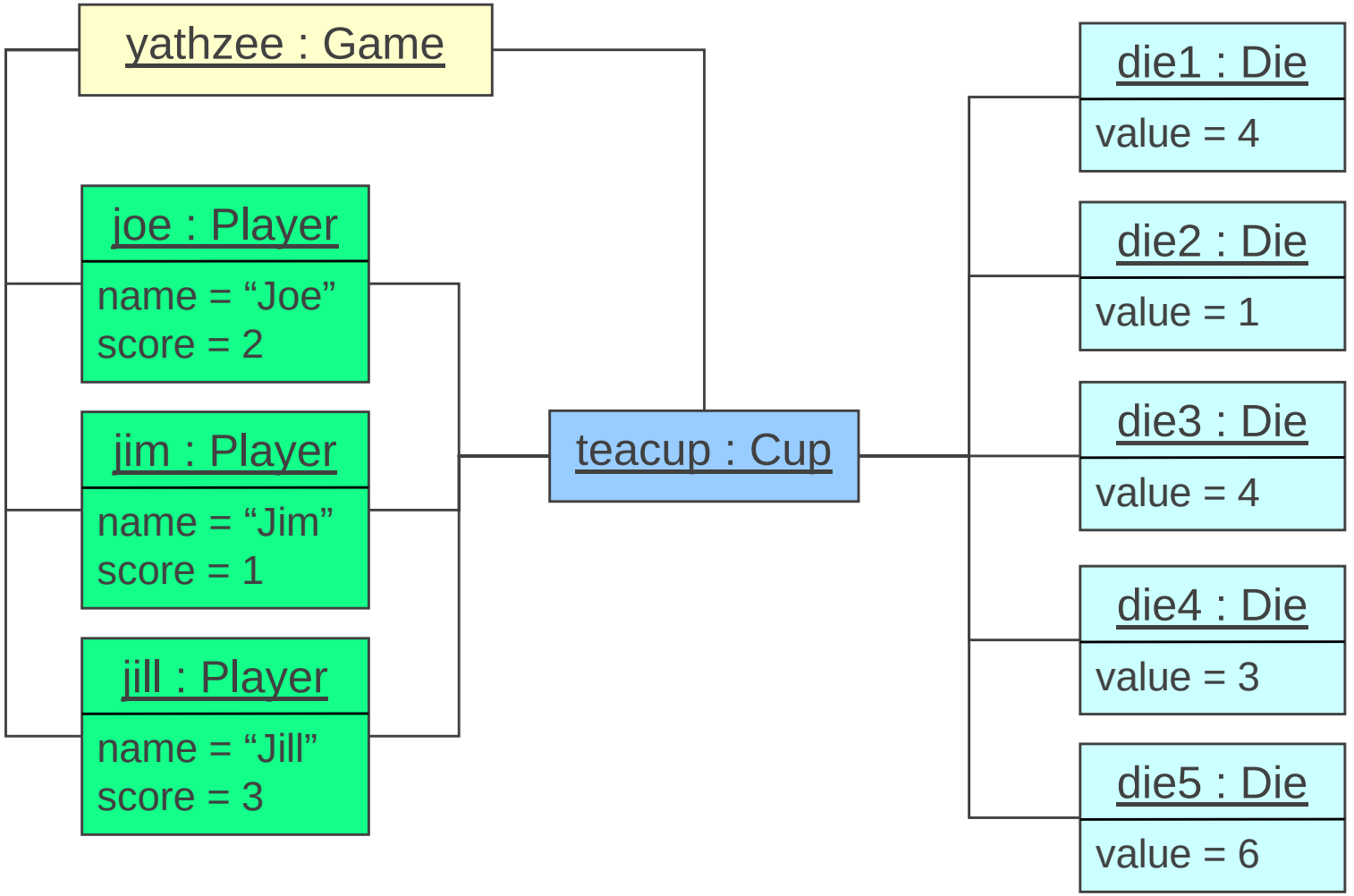
Links & Assoziationen



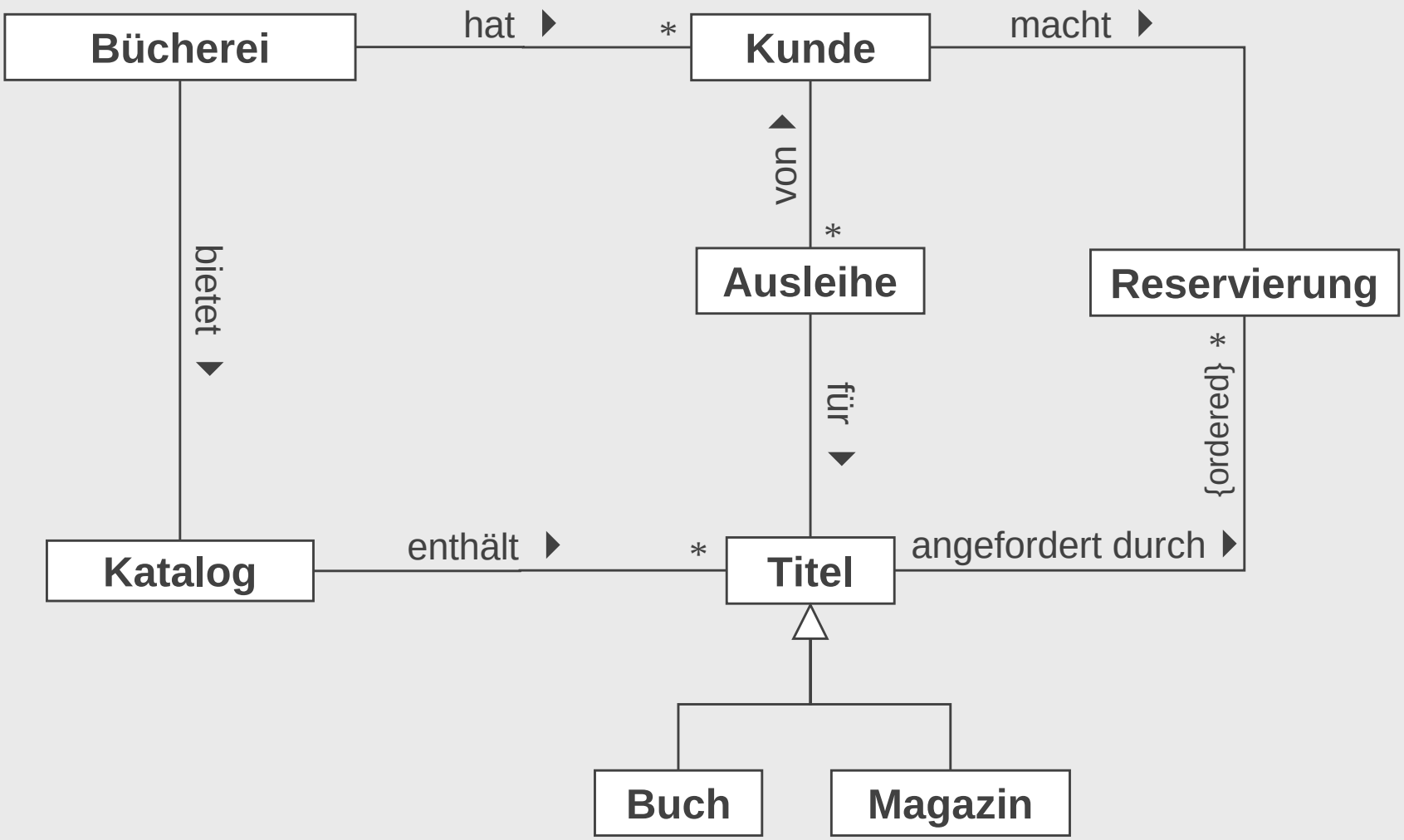
Würfelspiel: Klassen



Würfelspiel: Objekte



Analyse: Büchereisystem

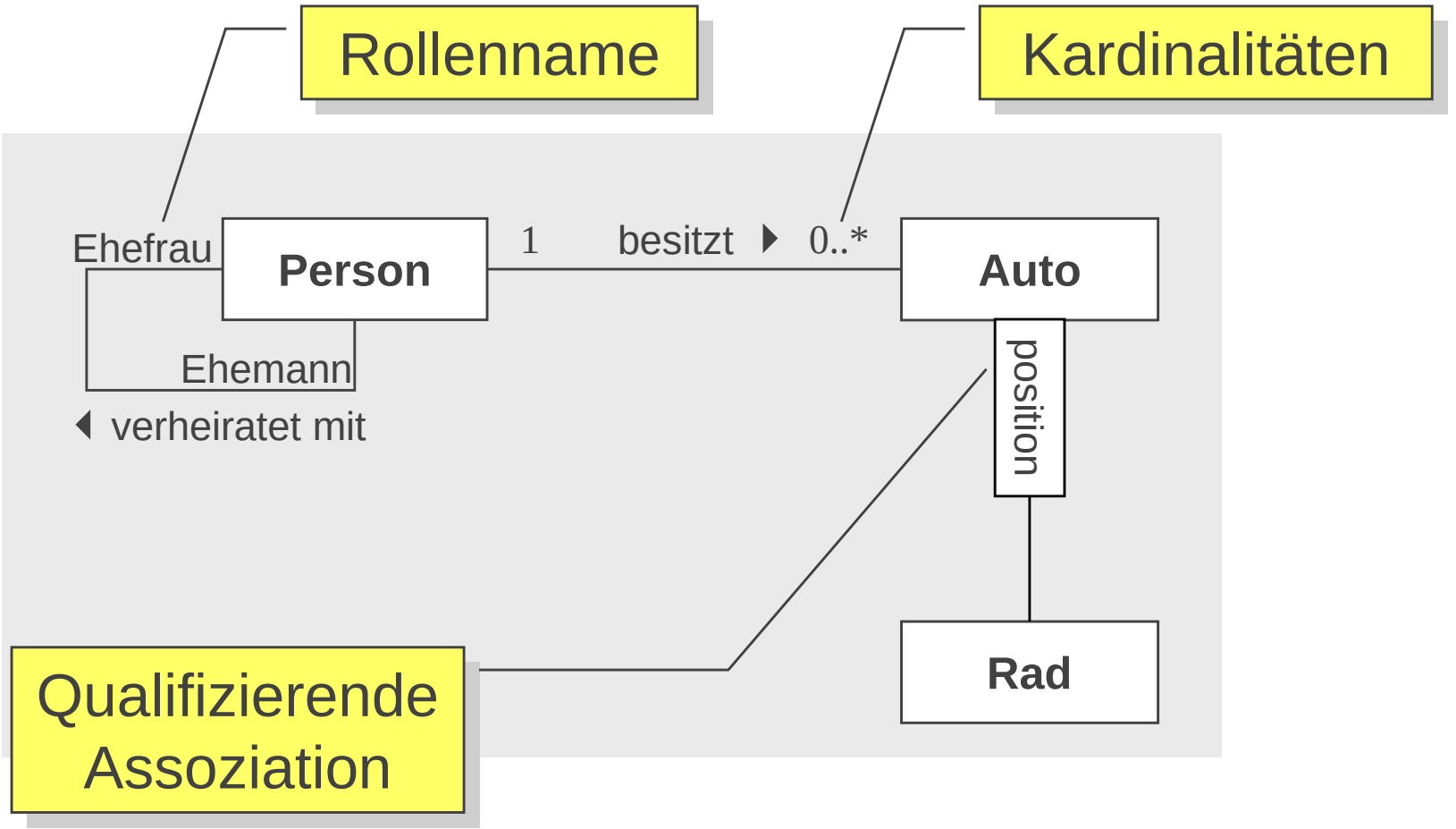


Konzepte finden

CRC Cards

Titel	
bibliographische Angaben speichern	
maximale Ausleihperiode kennen	Buch, Magazin
Reservierungen merken	Reservierung

Fortgeschrittene Beziehungen



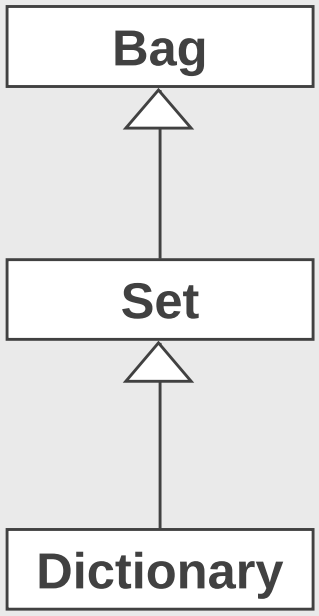
Generalisierung

Benutzt für...

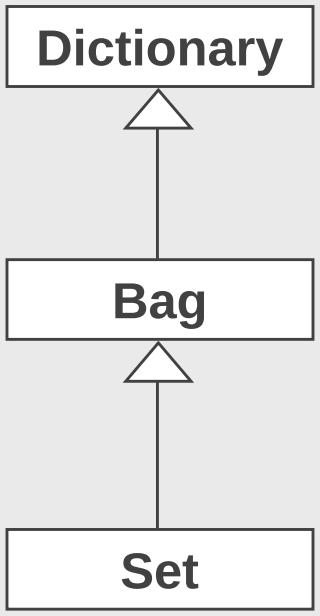
- Kategorisierung (*is-a*)
  - » Systemverständnis
- Code Reuse (*subclassing*)
  - » allgemeiner und spezialisierter Kode
  - » leichte *Erstellung* von Bibliotheken
- Substitutionsprinzip (*subtyping*)
  - » verhaltensäquivalente Subtypen
  - » leichte *Benutzung* von Bibliotheken

Konkurrierende Sichten

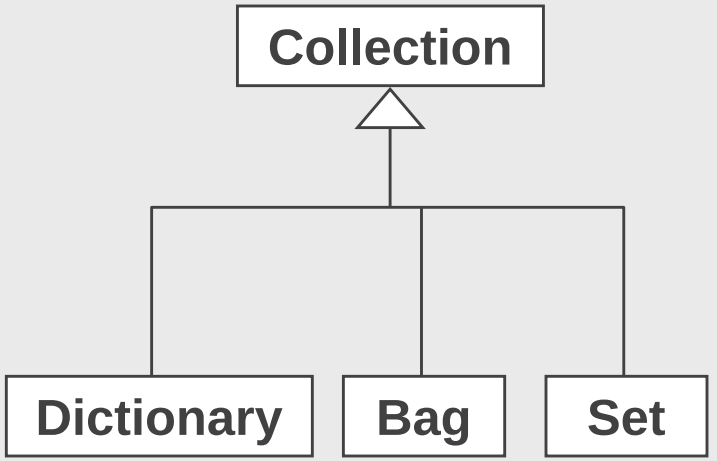
Is-a



Reuse

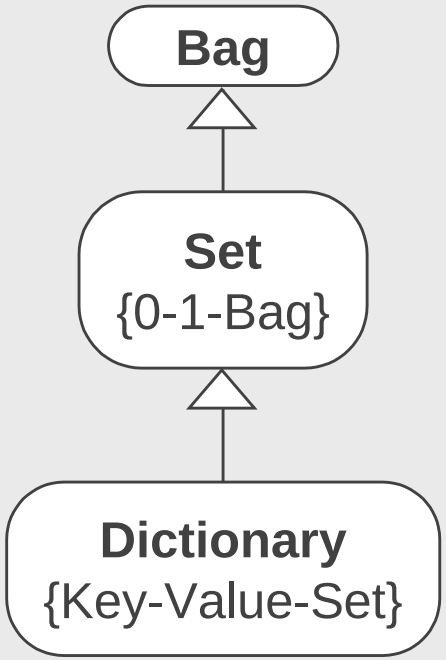


Subtyping

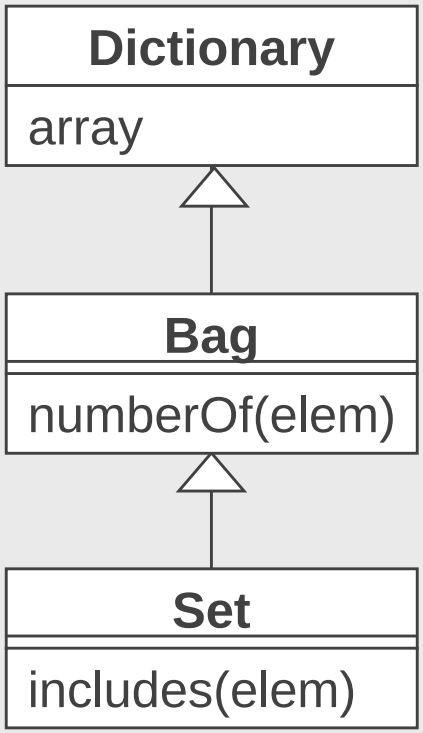


Konkurrenz verstehen

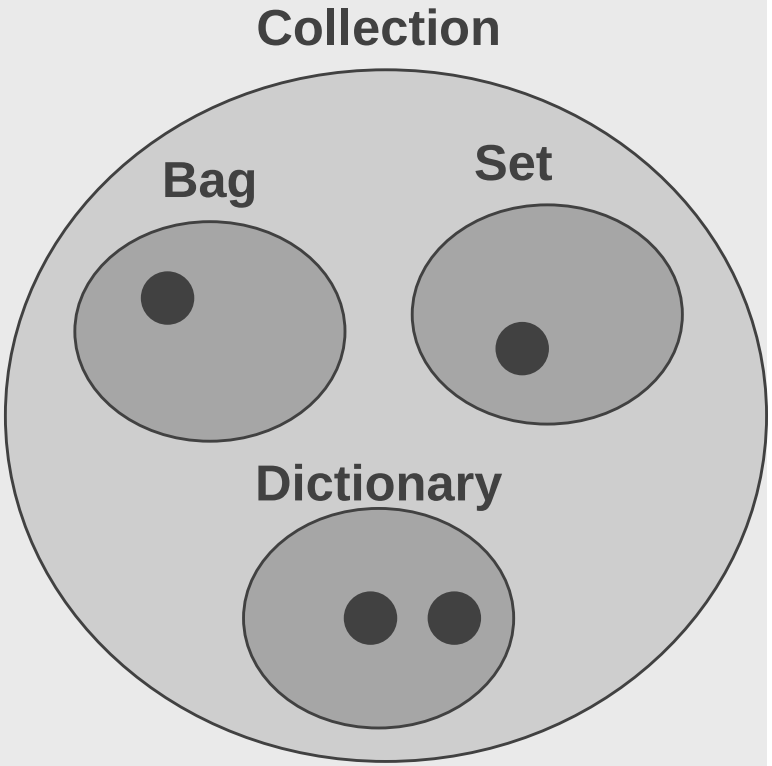
Is-A
bzgl. Begriffen



Reuse
bzgl. Definitionen

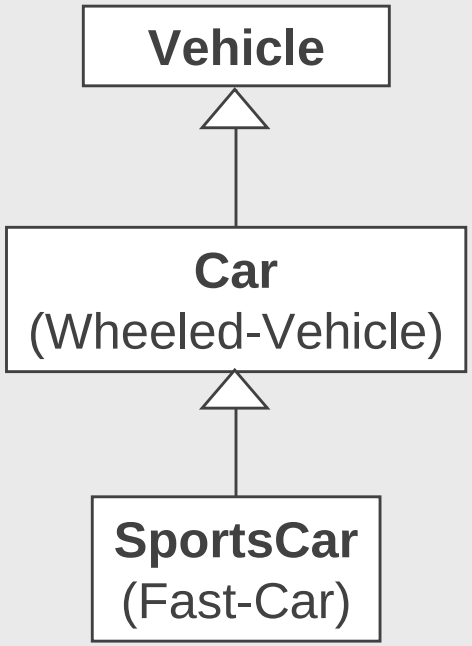


Subtyping
bzgl. Objekten

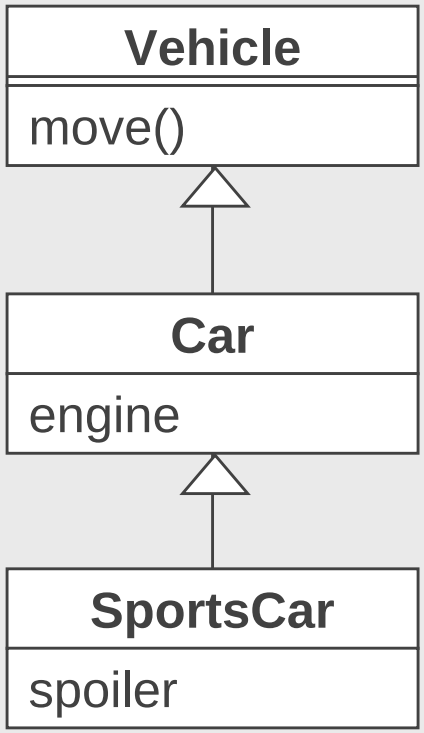


Keine Konkurrenz

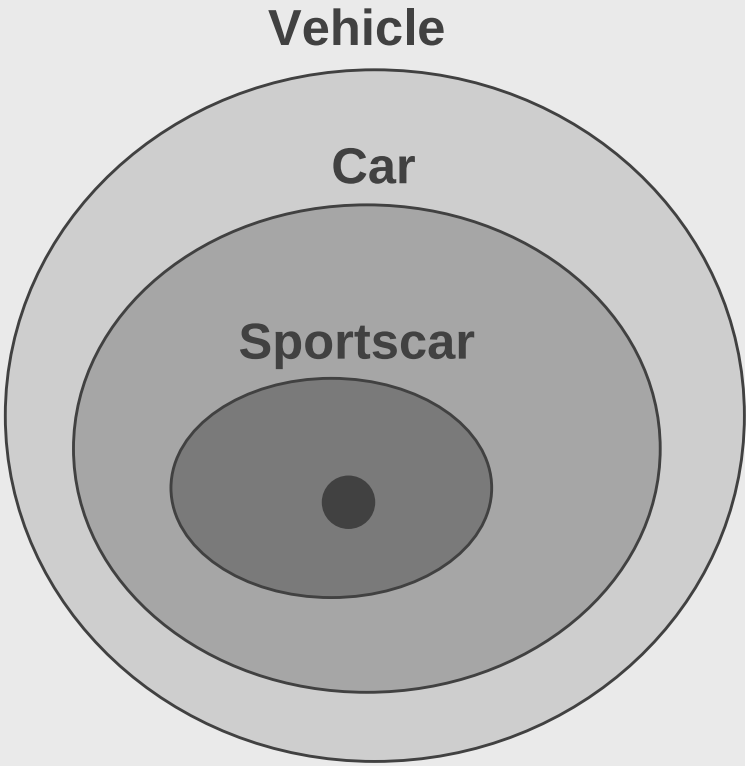
Is-A



Reuse



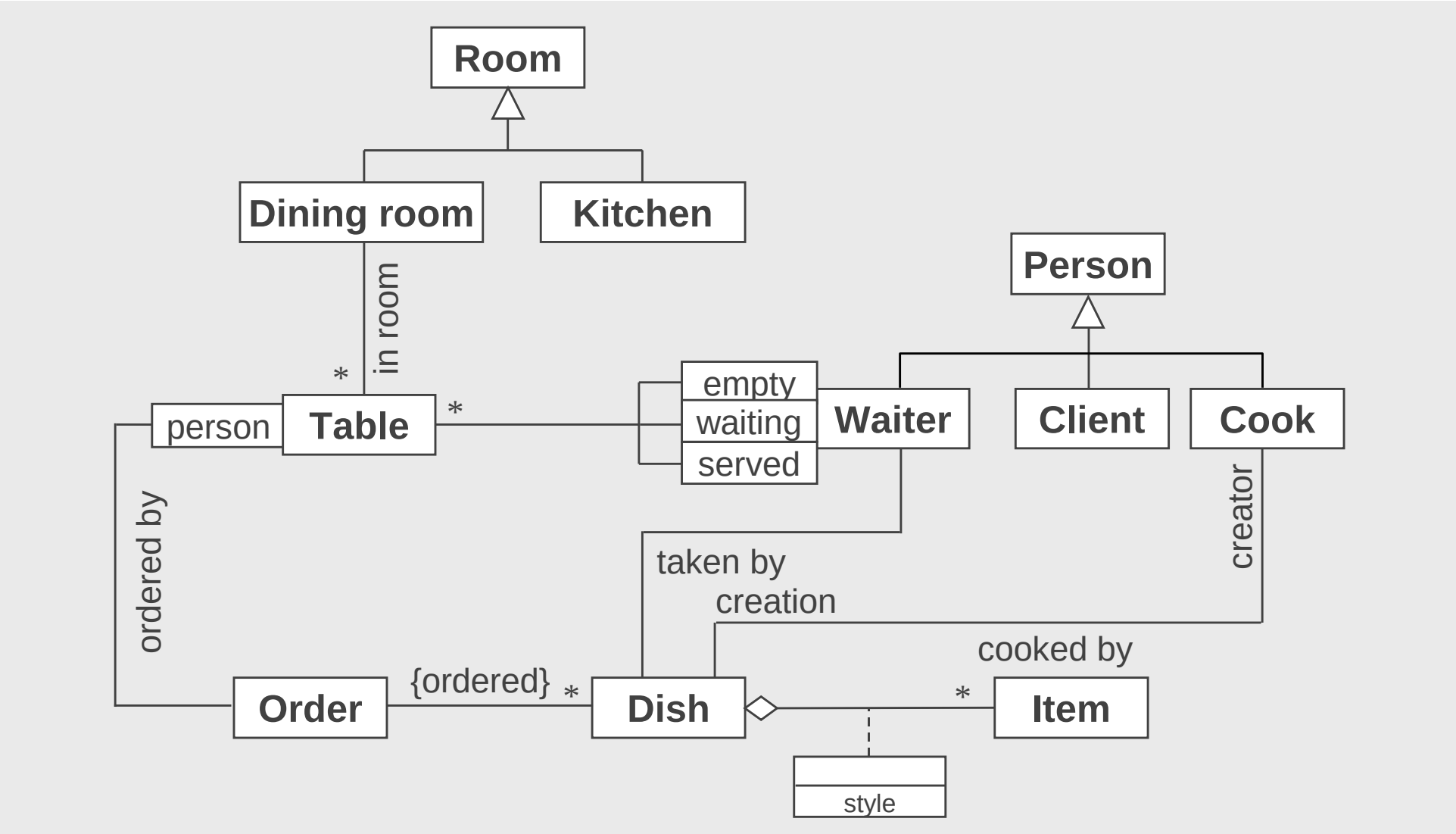
Subtyping



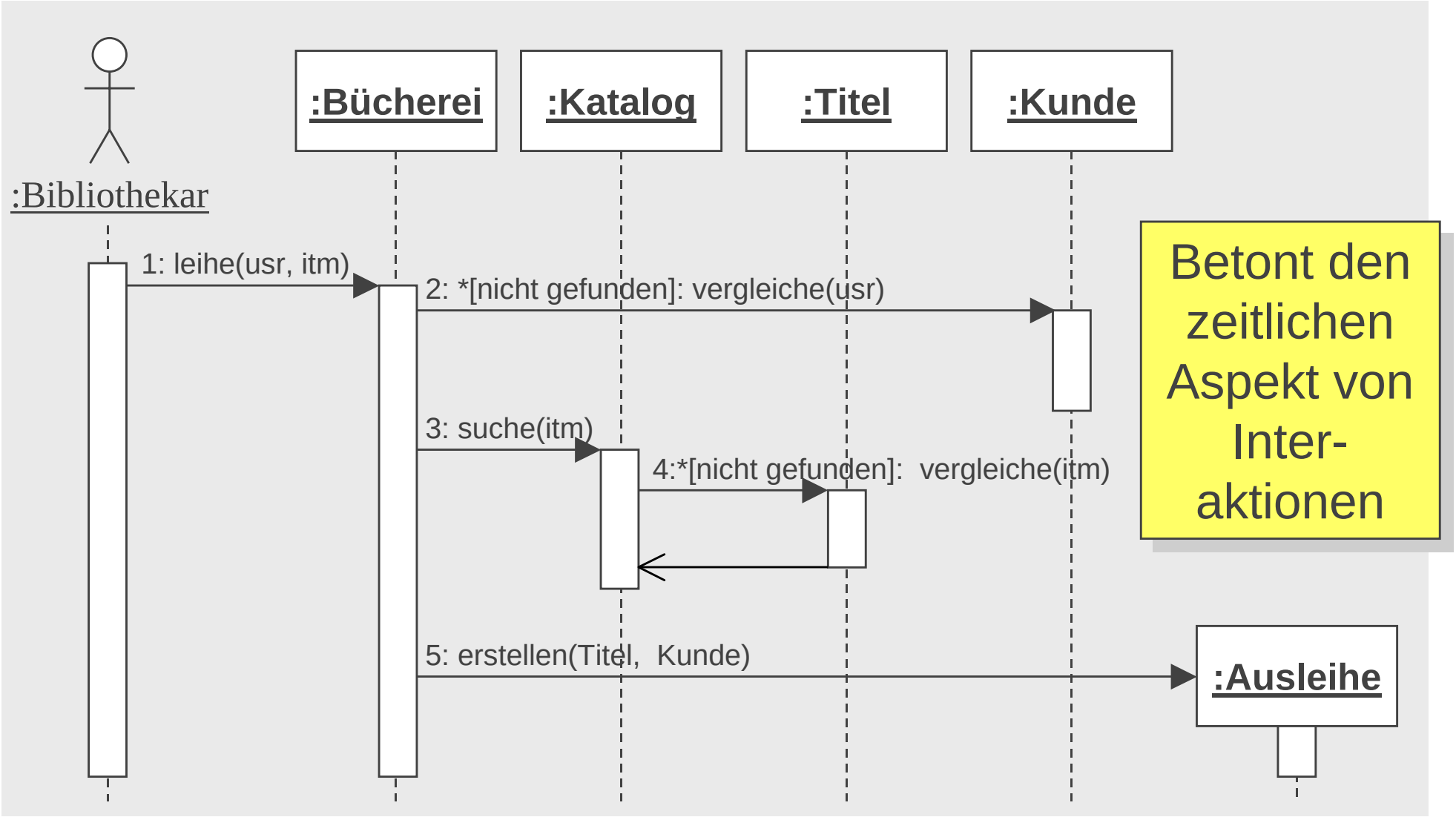
Assoziationen

- Assoziation
  - » Beziehung zwischen zwei Klassen
  - » oft strukturell motiviert
- Aggregation
  - » *teil-von* Beziehung
  - » strukturelle Abhängigkeit
- Assoziationsklasse
  - » erlaubt Beziehungen mit Informationen
  - » vermeidet voreilige Entwurfsentscheidungen

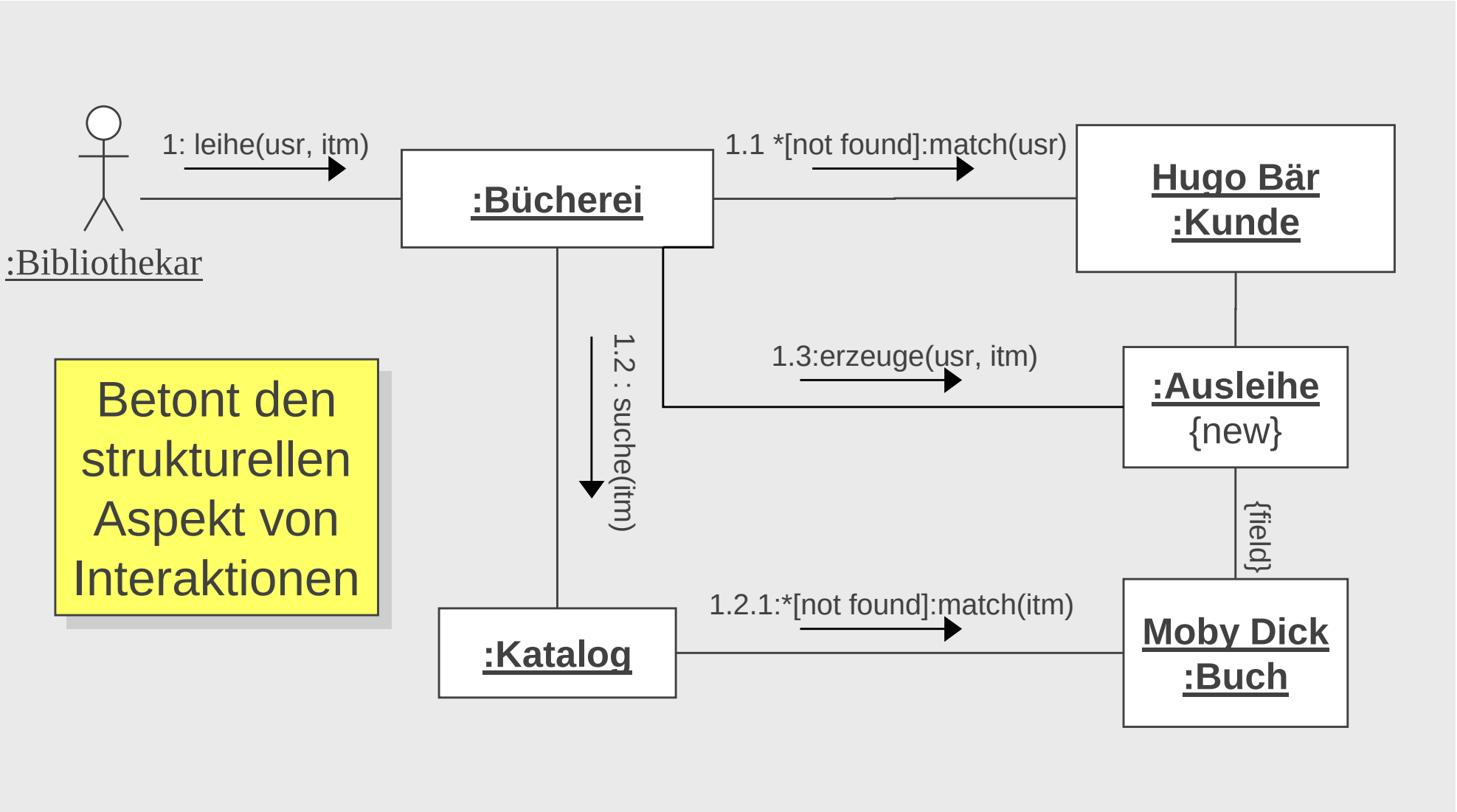
Ein Beispiel



Sequenzdiagramm

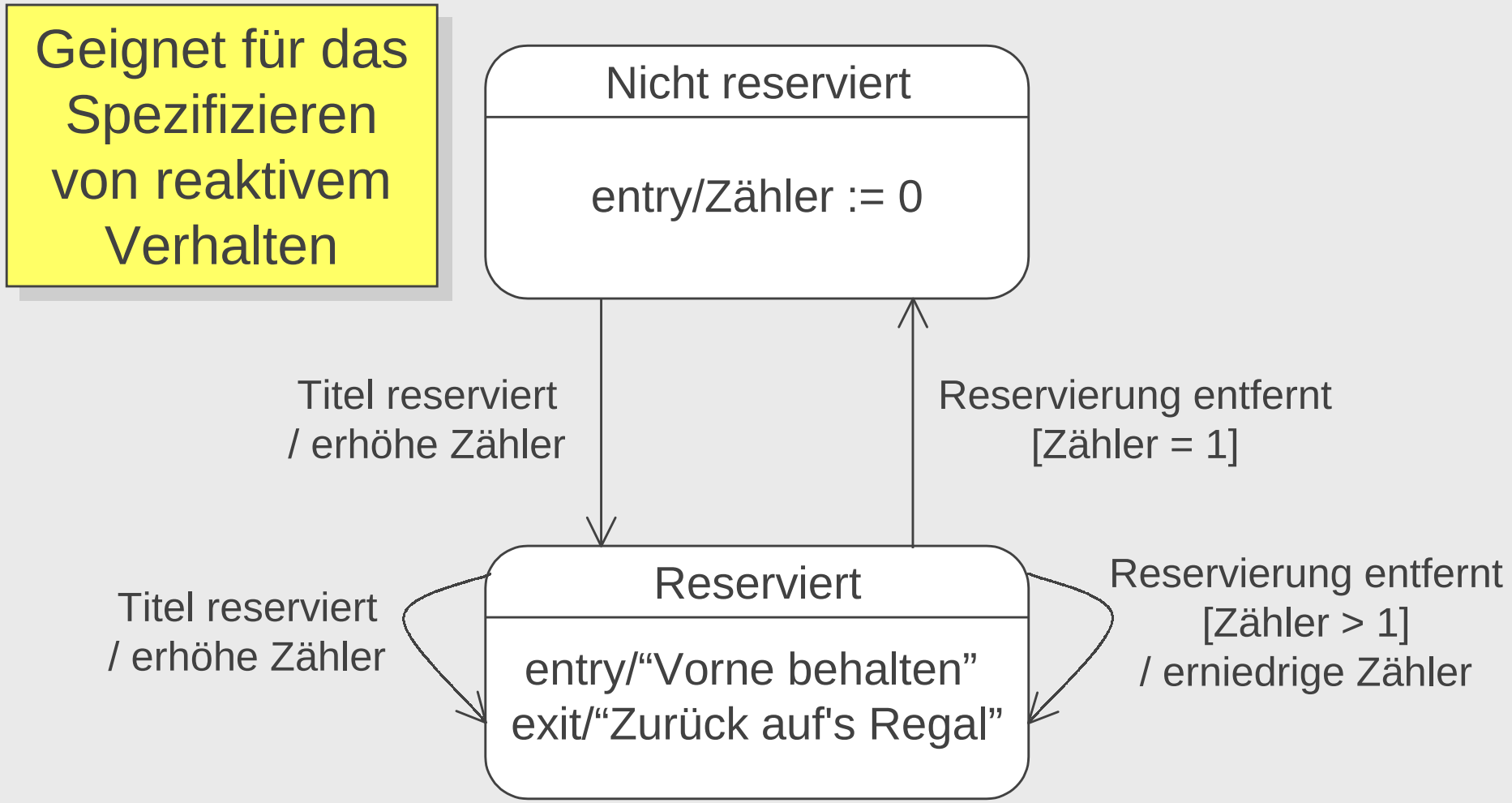


Kollaborationsdiagramm

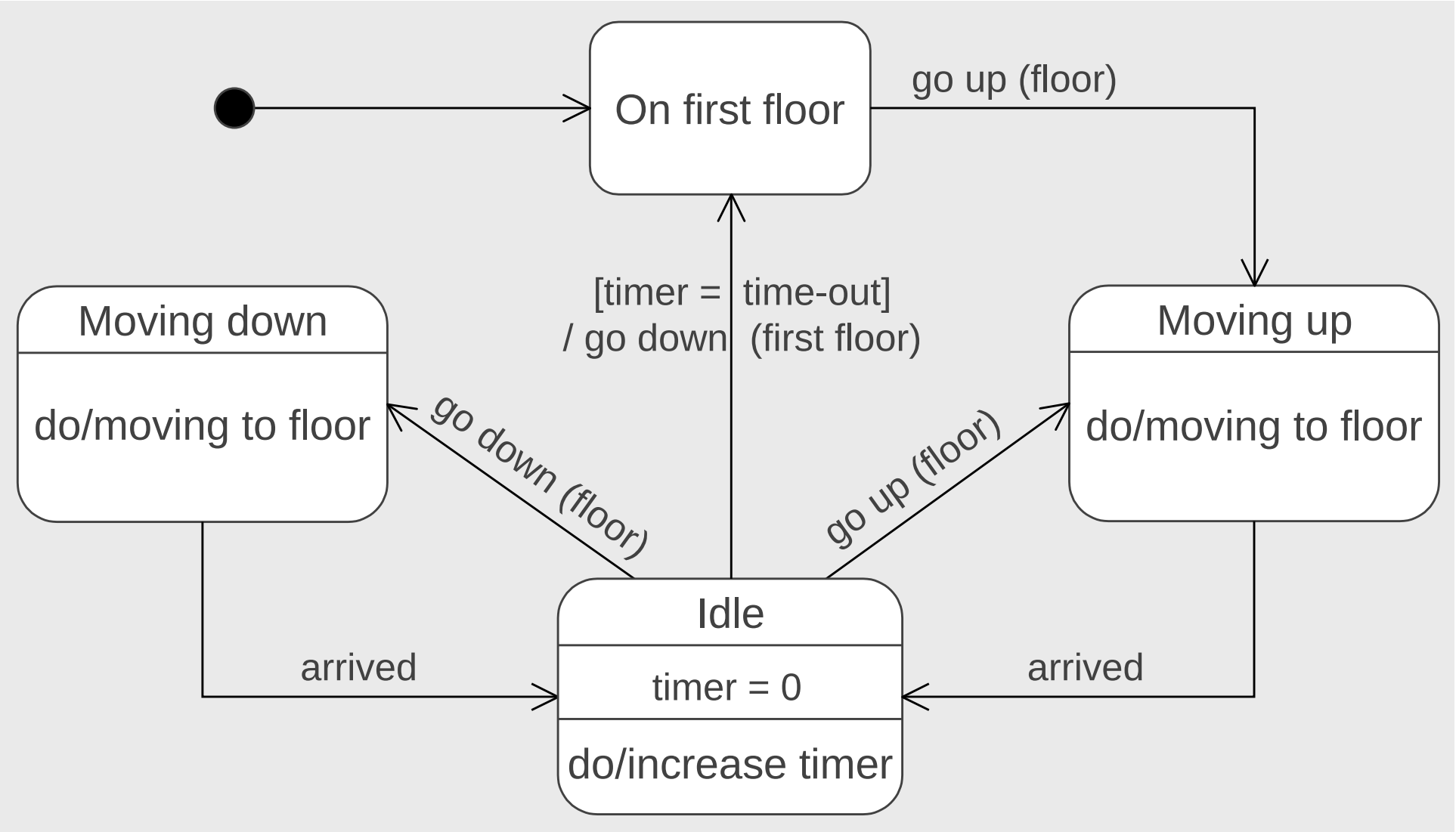


Zustandsdiagramm

Geignet für das Spezifizieren von reaktivem Verhalten



Zustandsdiagramm



Zustandsdiagramm

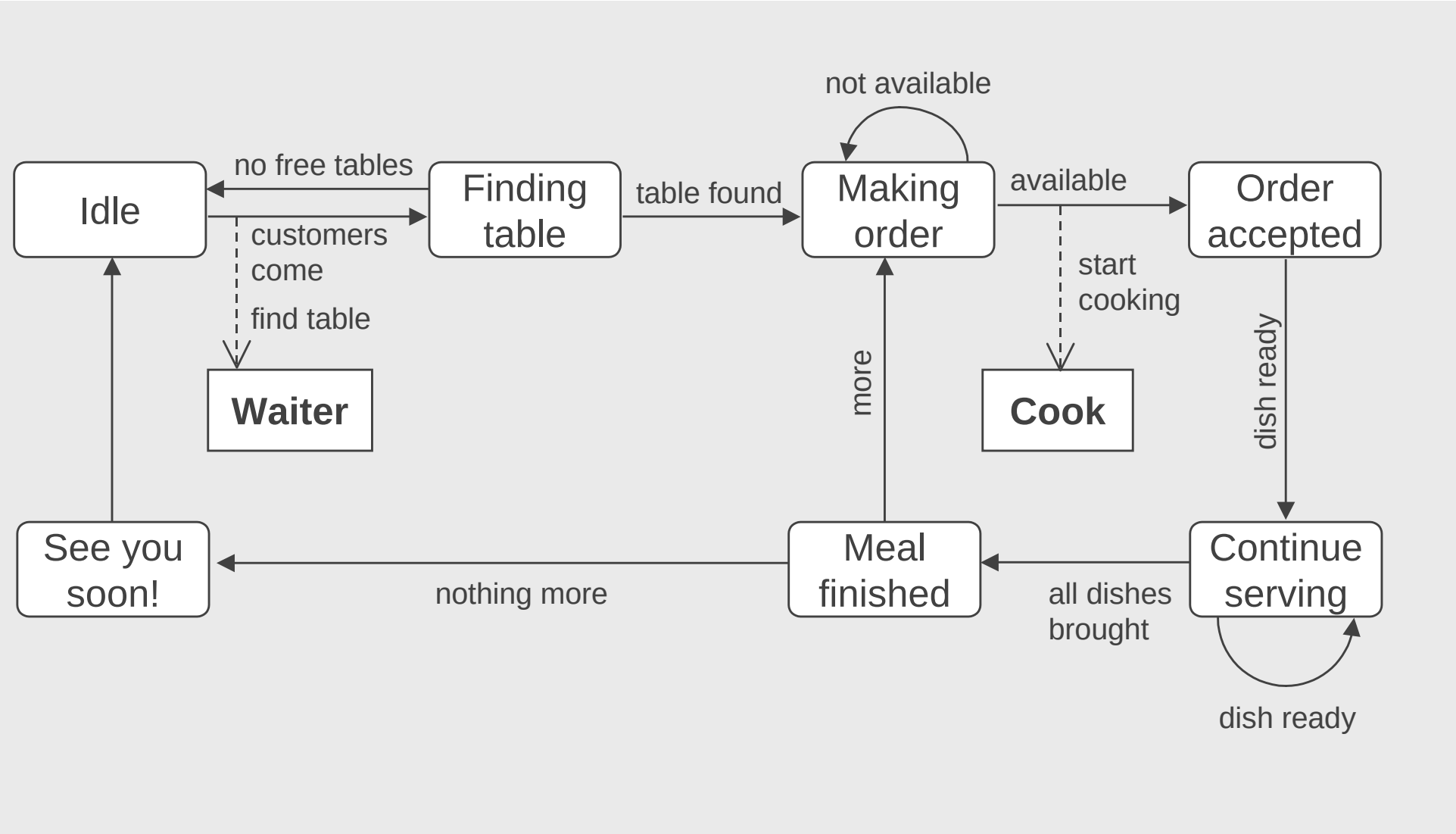
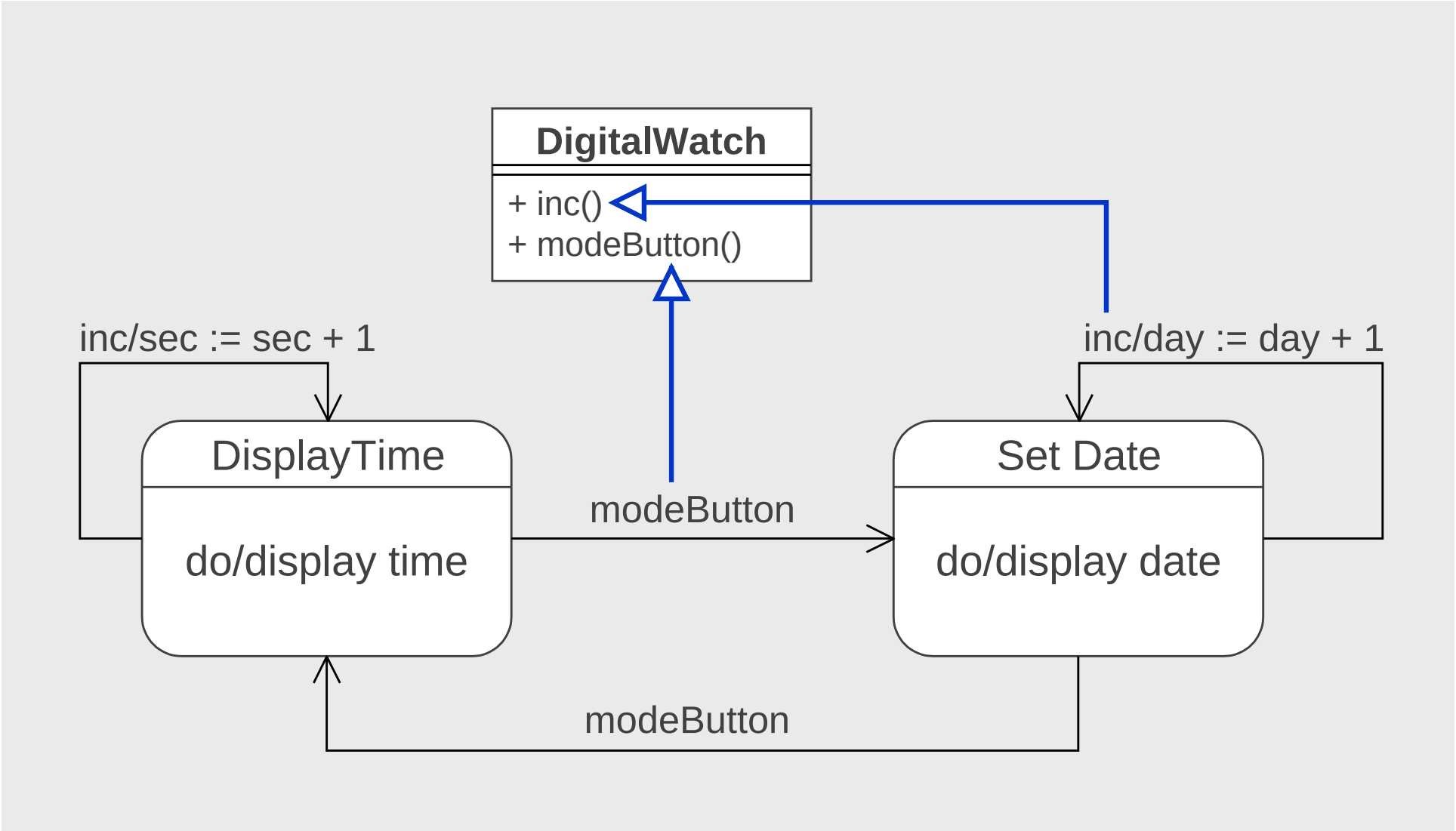
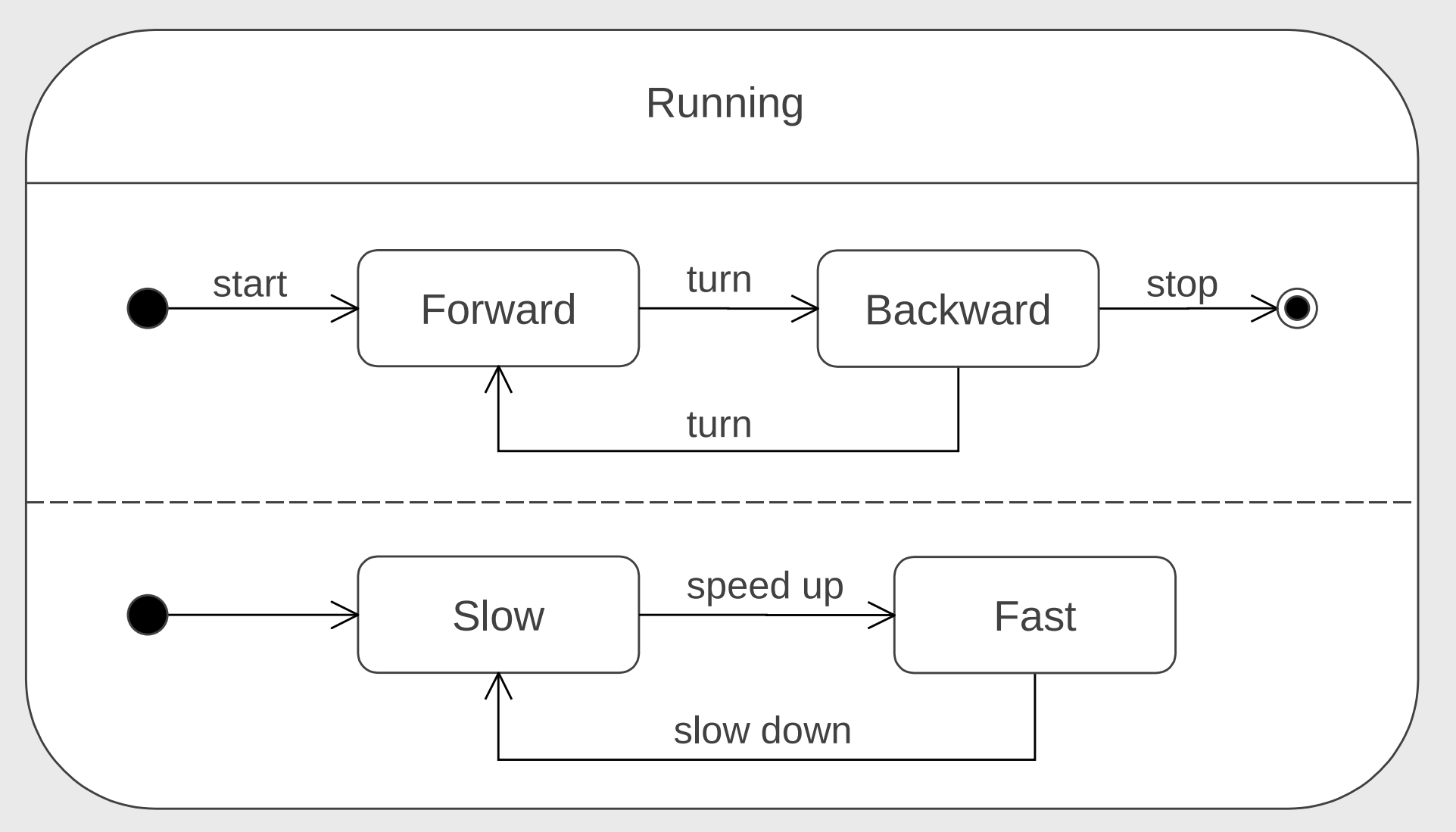


Diagramme verbinden



Geschachtelte Zustände



Was ist Entwurf?

- Architekturentwurf
  - » Umfassende Architektur bestimmen
  - » Identifikation von Subsystemen
  - » Erste Effizienzüberlegungen
- Detailentwurf
  - » Analyse + Implementierungsdetails
  - » Effizienz sicherstellen
  - » Wiederverwendung und Wartung garantieren

Software-Architektur

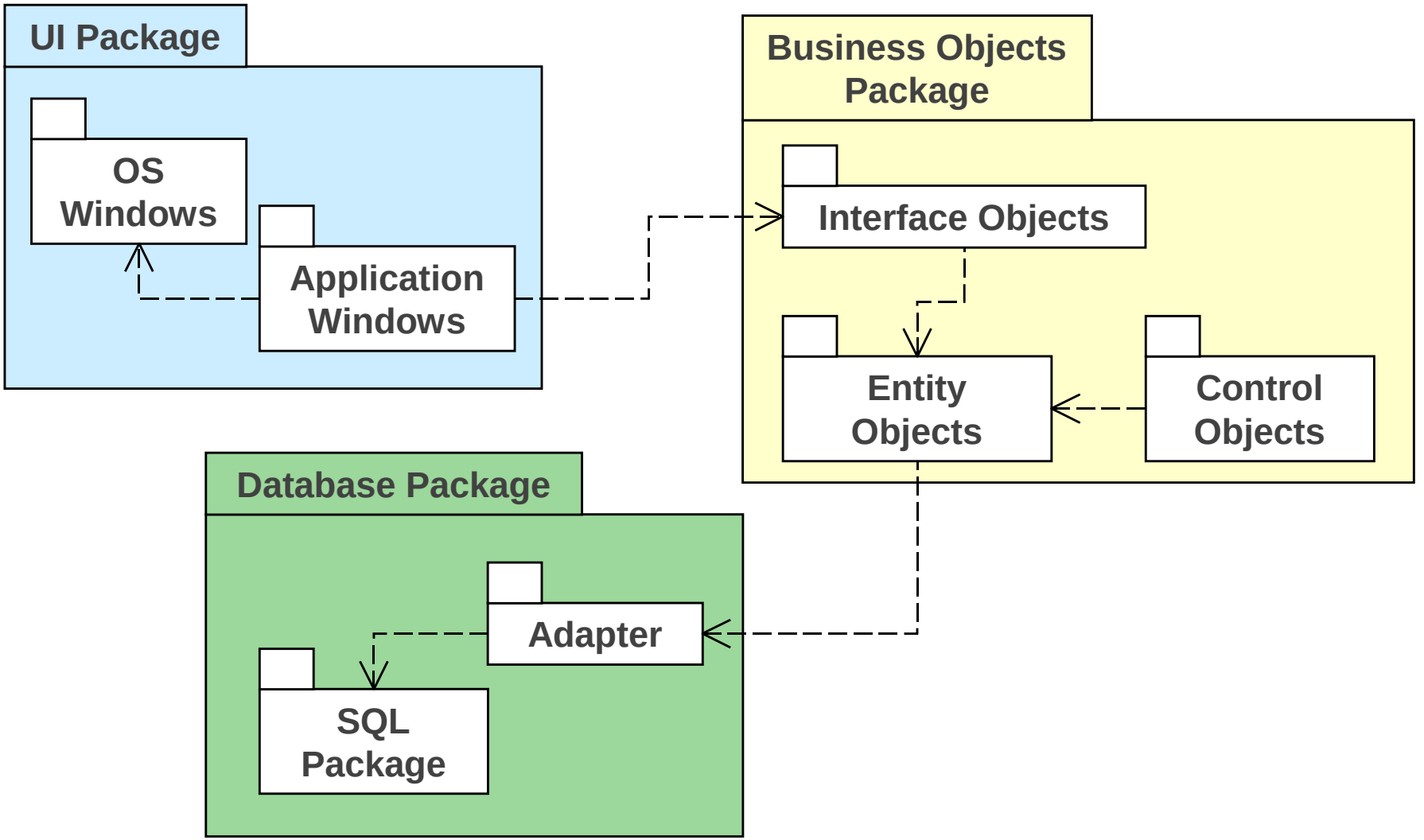
A software architecture is a description of the subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system. The software architecture of a system is an artifact. It is the result of the software design activity.

Buschman et al. (1996)

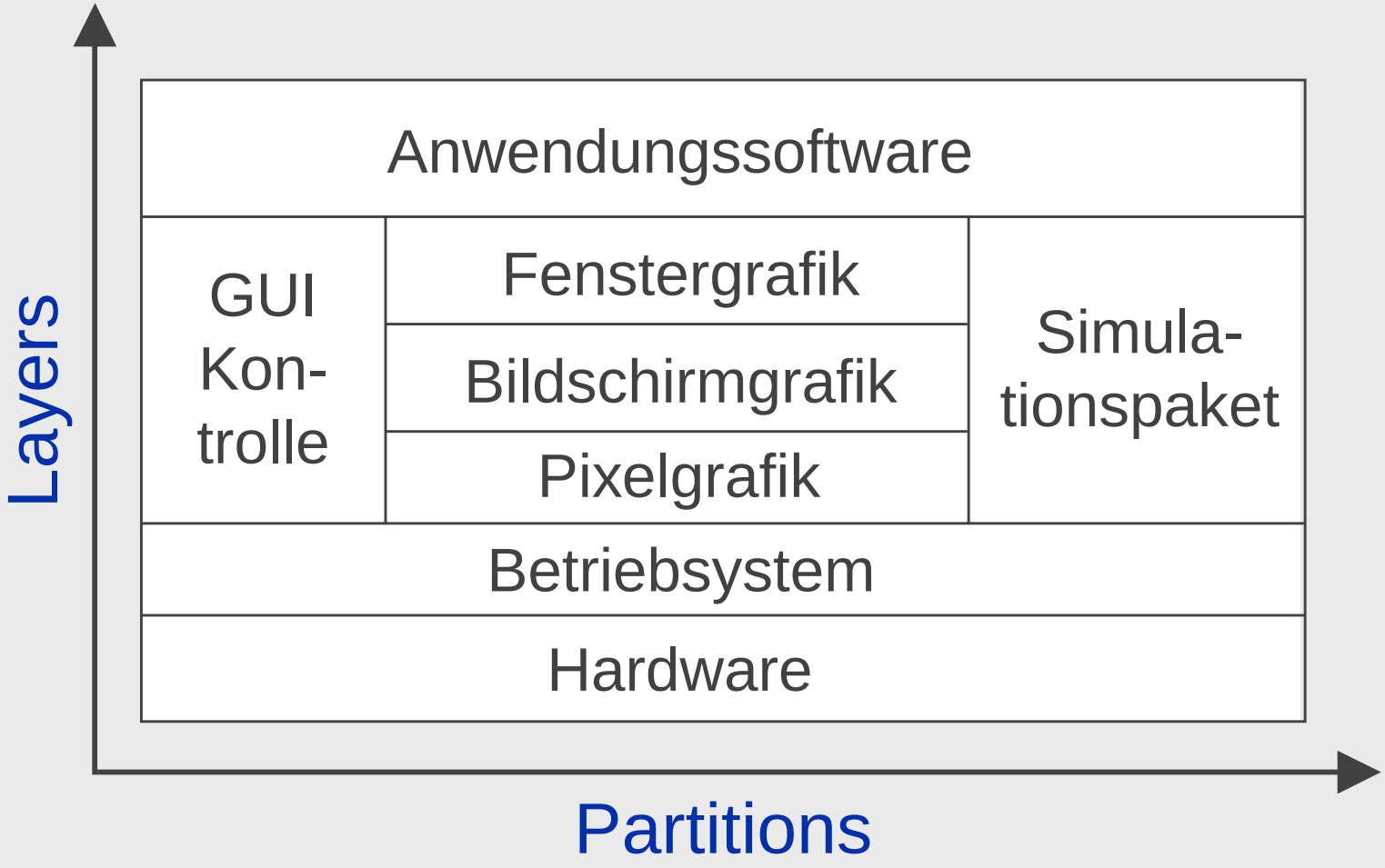
Systementwurf

- Architektur (Systemtopology)
  - » MVC, Pipes&Filters, Blackboard
- Subsysteme (Systemhierarchie)
  - » layers & partitions
- Nebenläufigkeit (Systemthreads)
  - » unabhängige Subsysteme

"Three-Tier" Architektur



Layers & Partitions

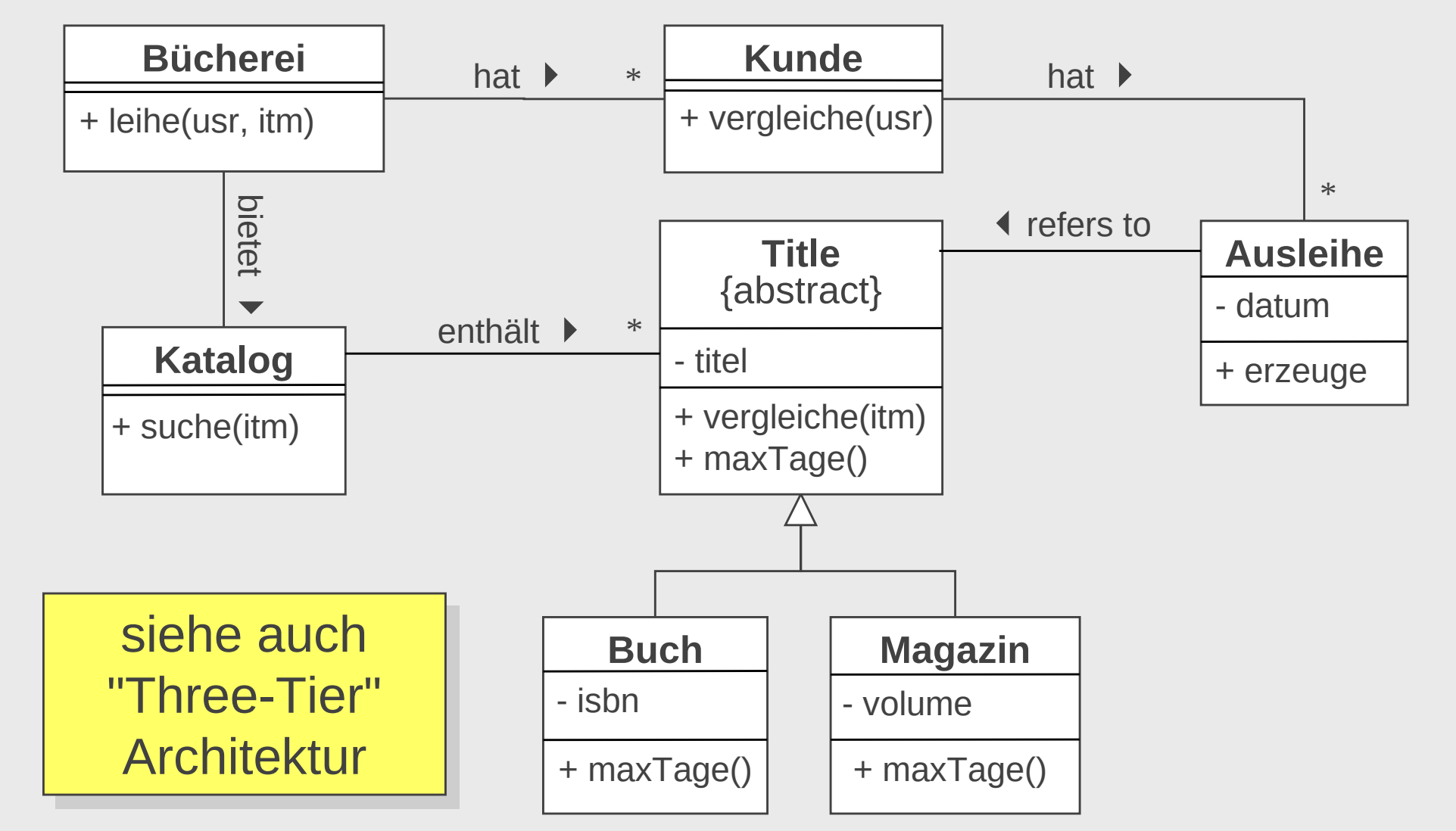


Detailentwurf

Erweiterungen einplanen...

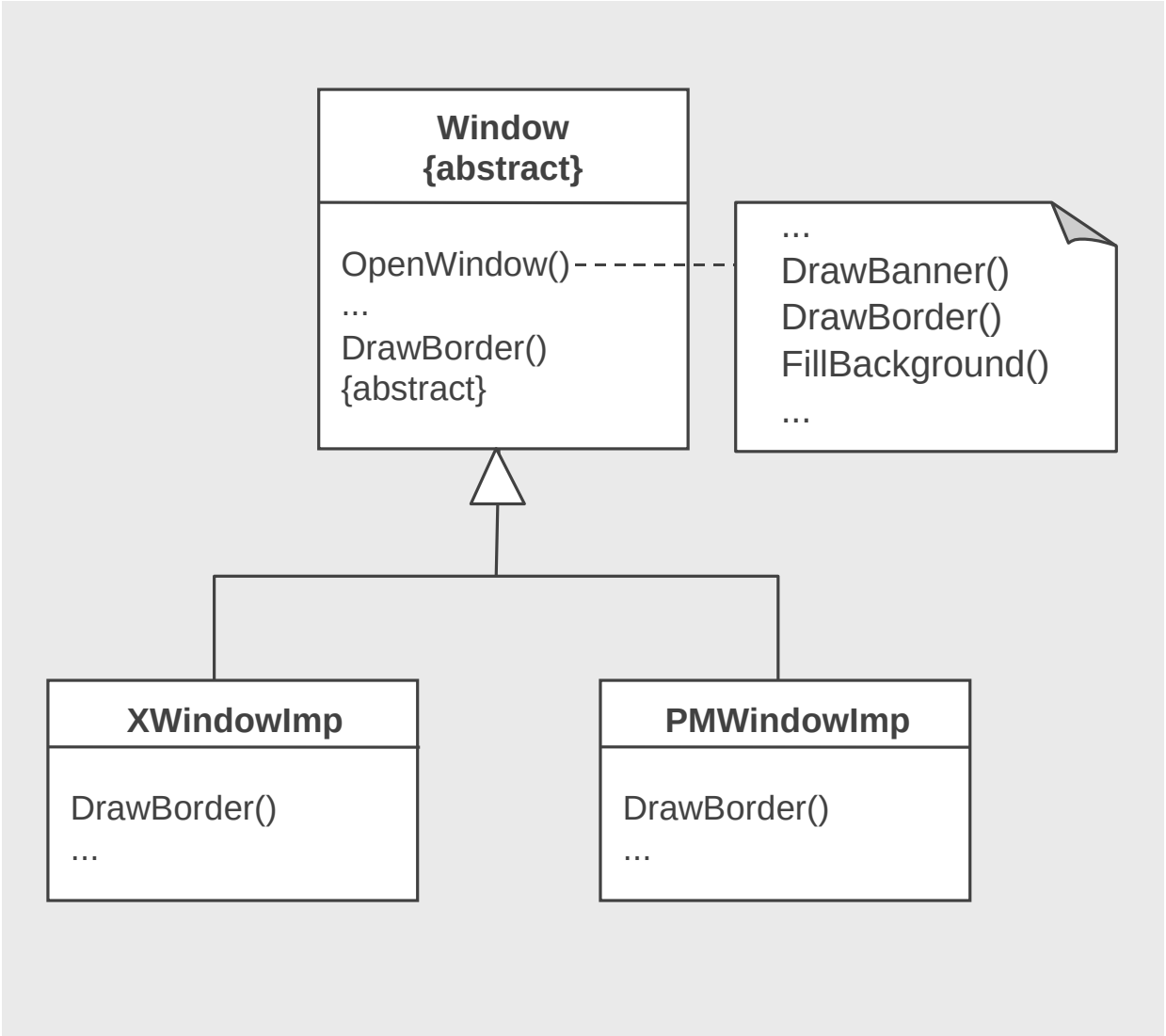
- Vererbung einführen
  - » allgemeines Verhalten abstrahieren
- Assoziationen umsetzen
  - » Zeiger, doppelte Zeiger, explizites Objekt
- Module identifizieren
  - » information hiding
- Algorithmenentwurf
  - » konstruktive & effiziente Lösungen finden

(Früher) Bücherei Entwurf



Verhaltensabstraktion

- Verantwortlich-keiten delegieren
- gemeinsames Verhalten konzentrieren
- Erweiterungen ermöglichen
- Methoden-namen unifizieren



Datenkapselung

- Private Attributes & Operations
- Assoziationsnavigierbarkeit einschränken
- Kopplung verringern
  - » Auf Schnittstellen programmieren (nicht auf Implementierungen)
- Kohesion erhöhen
  - » Eine Klasse nur für eine Verantwortlichkeit
  - » keine "eierlegenden Wollmilchsäue"

Algorithmenentwurf

- konstruktive Beschreibungen finden
- effiziente Lösungen finden
  - » angemessene Datenstrukturen
  - » effiziente Strategien
- Allerdings: Optimierung gefährdet Erweiterbarkeit
  - » 1. Gesetz: Optimiere nicht
  - » 2. Gesetz: Wenn es sein muß, dann spät
  - » 3. Gesetz: Nur da wo es sich lohnt (Profiling)

Zusammenfassung

Analyse

- Anforderungen
  - » Was soll das System machen?
- Modellierung
  - » Was sind die Domänen-konzepte?

Entwurf

- Architektureentwurf
  - » "Angriffsplan"
- Detailentwurf
  - » Detaillierte Basis für die Implemen-tierung

Zusammenfassung

Statische Sicht

- Use Cases
  - » Systemfunktionalität
- Konzeptmodell
  - » Konzepte
  - » Beziehungen

Dynamische Sicht

- Interaktionen
  - » Sequenzdiag.
  - » Kollaborationsdiag.
- Aktivitätsdiagramme
- Zustandsdiagramme

"Just Do It?"

Analysis means “understand the problem”,
Design means “plan the solution”.

Are you saying that you work faster
when you don't understand the problem,
and have no particular solution in mind?

John DiCamillo