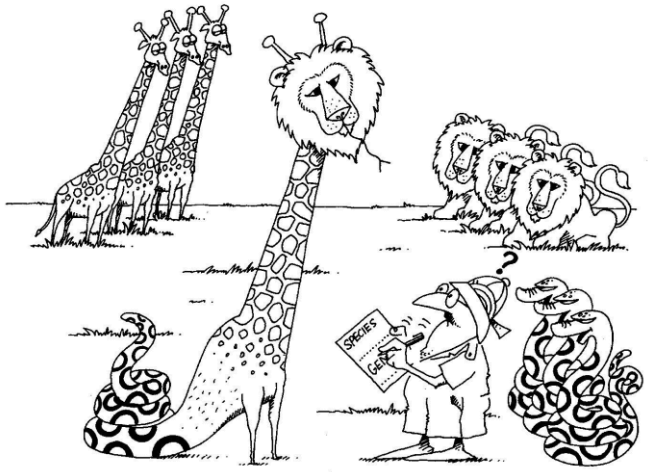
Media Engineering
Objektorientierte Modellierung

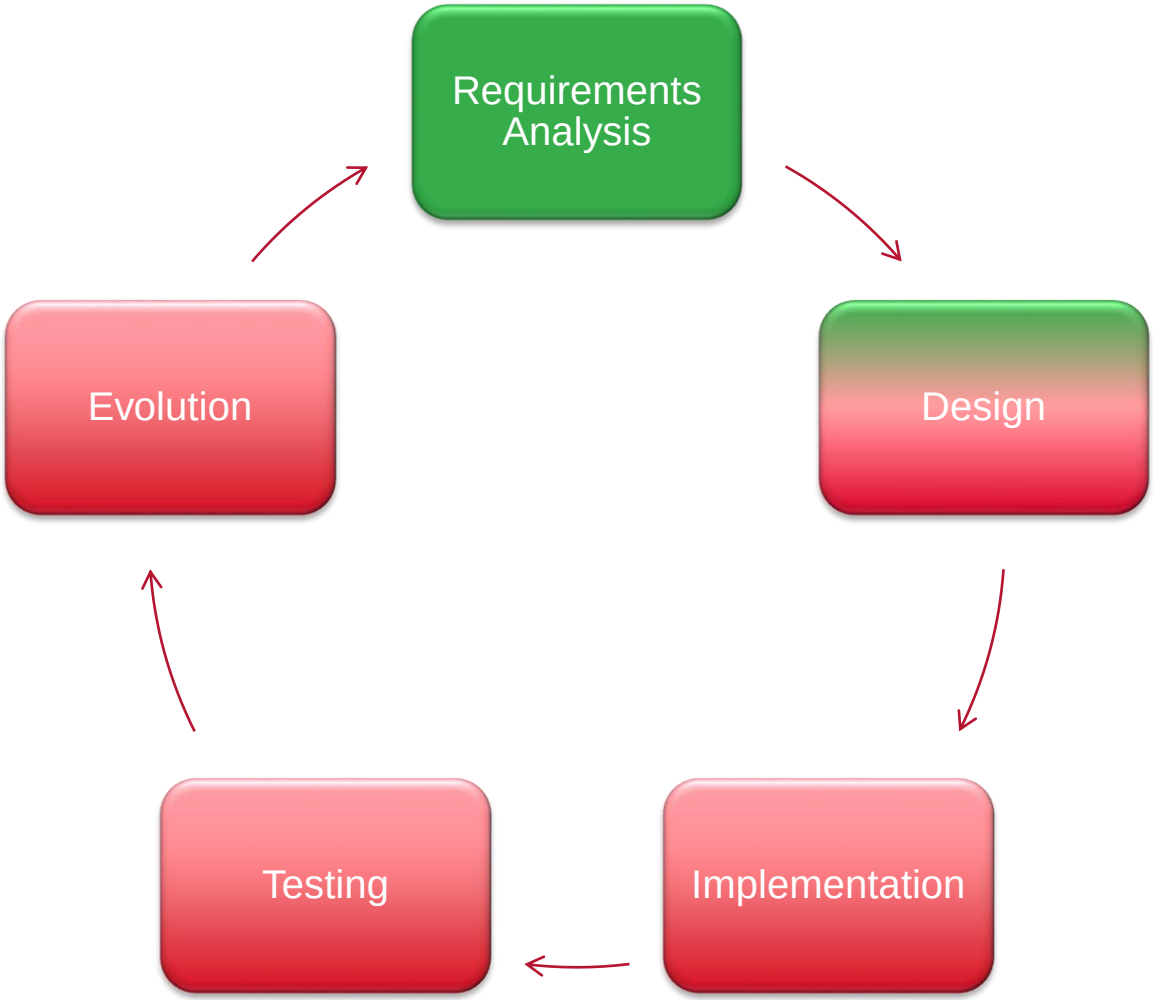


R. Weller

University of Bremen, Germany

cgvr.cs.uni-bremen.de

Der Software Development-Lifecycle



Zur Erinnerung

Was nun?

Modellbildung

„Reale Welt“

„Computer Vorstellung“

Modellierung

Automat:
S-Bank1
1000,- €
800,- €

ort

Auszahlung:
4.3.09
200,- €

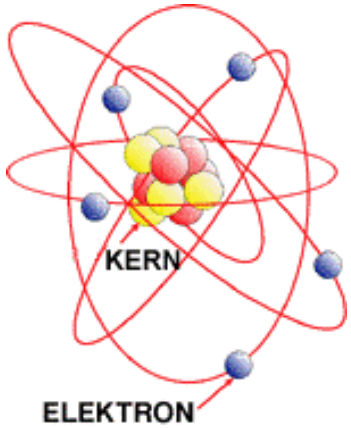
kunde

Konto:
A. Muster
400,- €
200,- €

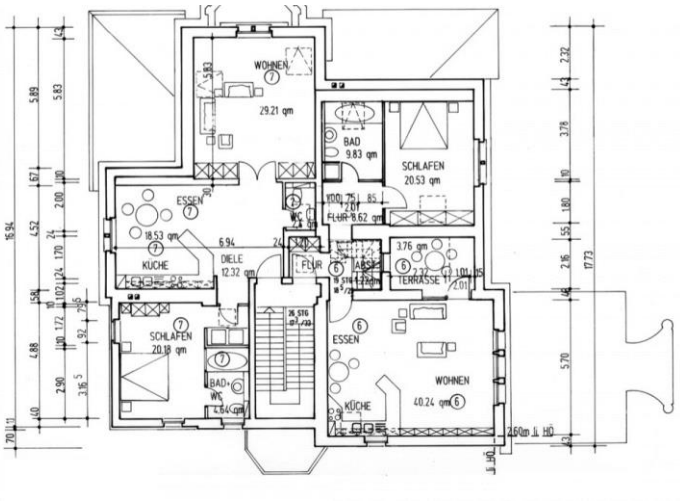
Interpretation

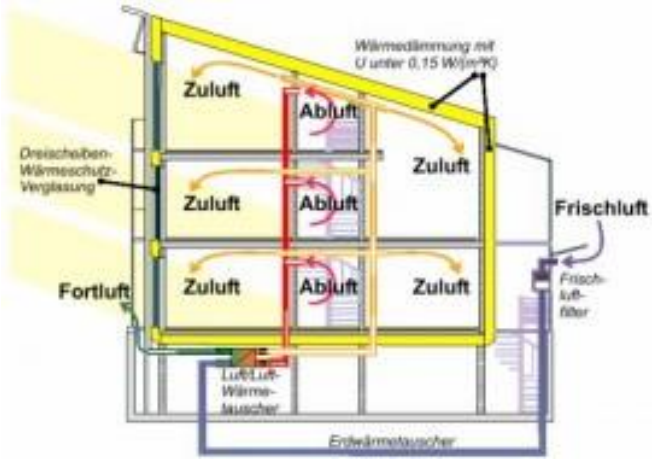
Modellbildung

- Was ist ein Modell?
  - Abbild eines Originals
    - Kann abstrakt sein
    - Kann kleiner (oder größer) sein
- Warum modellieren wir?
  - Um etwas zu verstehen
  - Um Vorhersagen zu machen
  - Um etwas zu dokumentieren
- Wann modellieren wir?
  - Immer
    - Projektplan
    - Pflichtenheft
    - Architektur
    - …



Modelle in der Architektur





Definition

Architektursicht (View)

- Repräsentation eines ganzen Systems aus der Perspektive einer kohärenten Menge von Anliegen (IEEE P1471 2002).

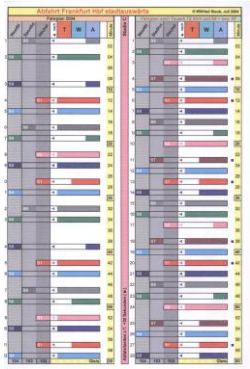
Architekturblickwinkel (Viewpoint)

- Spezifikation der Regeln und Konventionen, um eine Architektursicht zu konstruieren und zu benutzen (IEEE P1471 2002).
- Ein Blickwinkel ist ein Muster oder eine Vorlage, von der aus individuelle Sichten entwickelt werden können, durch Festlegung von
  - Zweck
  - adressierte Betrachter
  - und Techniken für Erstellung, Gebrauch und Analyse.

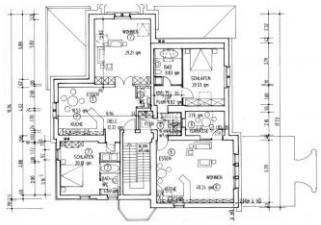
Blickwinkel auf Softwaresysteme

- Externe Perspektive
  - Modell von Kontext und Umgebung des Systems

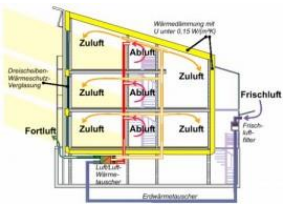
- Interaktive Perspektive
  - Modell der Interaktionen des Systems mit seiner Umgebung
  - Oder zwischen verschiedenen Komponenten des Systems



- Strukturelle Perspektive
  - Aufbau des Systems oder die Struktur der vom System verarbeiteten Daten

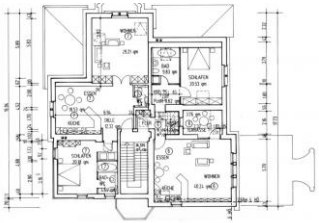


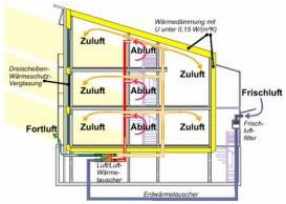
- Verhaltensbasierte Perspektive
  - Modell des dynamischen Verhaltens des Systems und sein Antwortverhalten aus Ereignisse



Wie findet man Modelle für die unterschiedlichen Blickwinkel?

- Grundsätzlich zwei verschiedene Sichten
  - Statische
    - Strukturelle Perspektive
    - Externe Perspektive
  - Dynamische (berücksichtigen auch die Zeit)
    - Interaktive Perspektive
    - Verhaltensbasierte Perspektive

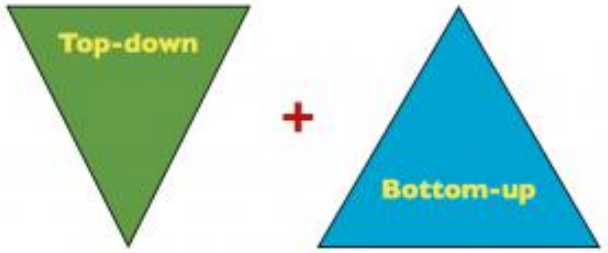




- Systematische Modellierung muss beide Arten berücksichtigen
  - Problem: Sie können kaum unabhängig voneinander betrachtet werden
  - Lösung: Top-Down-Analyse
    - Angelehnt an objektorientierter Programmierung

Objektorientierte Analyse und Design

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases)
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein



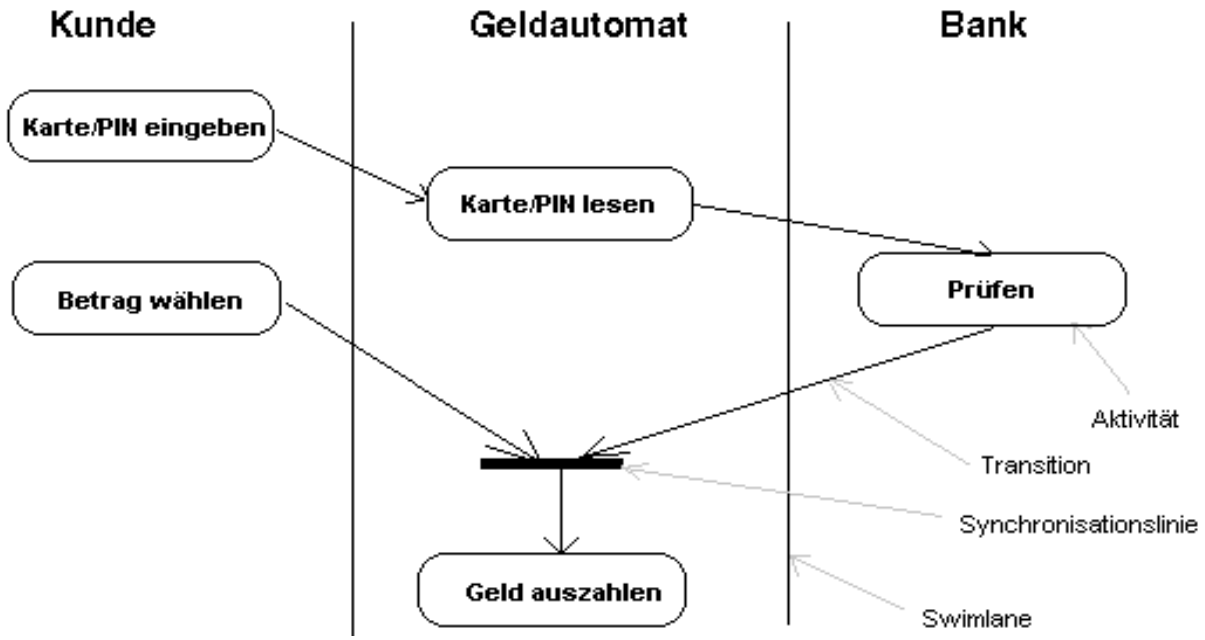
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Wie notiert man Software-Modelle?

- Versuch 1: Natürliche Sprache
- Beispiel Bankautomat:
  - […] Das Kartenlesegerät des Geldautomaten ist mit einer Autorisierungszentrale online verbunden. Diese prüft, ob zu der Karte überhaupt ein Konto gehört, und gleicht sie mit einer Sperrdatei ab. Ist ein entsprechendes Konto vorhanden und liegen keine Sperrungen vor, fordert die Autorisierungszentrale mit Hilfe der Online-Personalisierung von Terminals (OPT) den Kunden zur Eingabe der Geheimzahl auf. Deren Richtigkeit wird unmittelbar geprüft. Diese auch PIN genannte Zahl ist in der Regel vierstellig, aber bei internationalen Kreditkarten kann sie sechsstellig sein. Eine Fehleingabe der PIN kann dem Kunden erst nach der Abfolge weiterer Schritte (beispielsweise nach der gewünschten Geldstückelung) bis unmittelbar vor dem eigentlichen Auszahlungsvorgang mitgeteilt werden. So genannte Offline-Transaktionen, in der Anfangsphase der Geldautomaten einziges Verfahren, gelten als unsicher, wurden durch OPT überflüssig und werden auch international kaum noch praktiziert. Bei dreimaliger Fehleingabe wird die Karte in den meisten Ländern eingezogen. In Österreich erscheint ein Hinweis, dass die Karte beim vierten Mal zur Sicherheit des Kunden eingezogen wird, sie verbleibt dann im Automaten. Die eingezogene Karte gelangt zur First Data Austria (FDA, früher Europay Austria),…
- Probleme:
  - Sehr lang
  - unübersichtlich
  - Änderungen schwierig
  - Keine Granularität

Wie notiert man Software-Modelle? (cont)

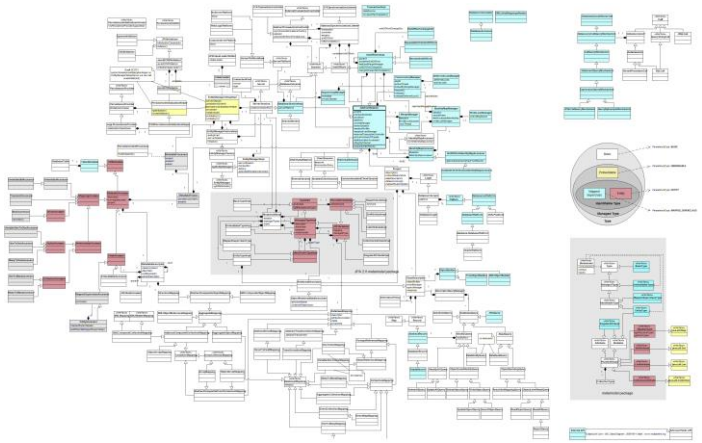
- Versuch 2: Diagramme
- Beispiel Bankautomat:



- Vorteile
  - Deutlich übersichtlicher
  - Änderungen einfach

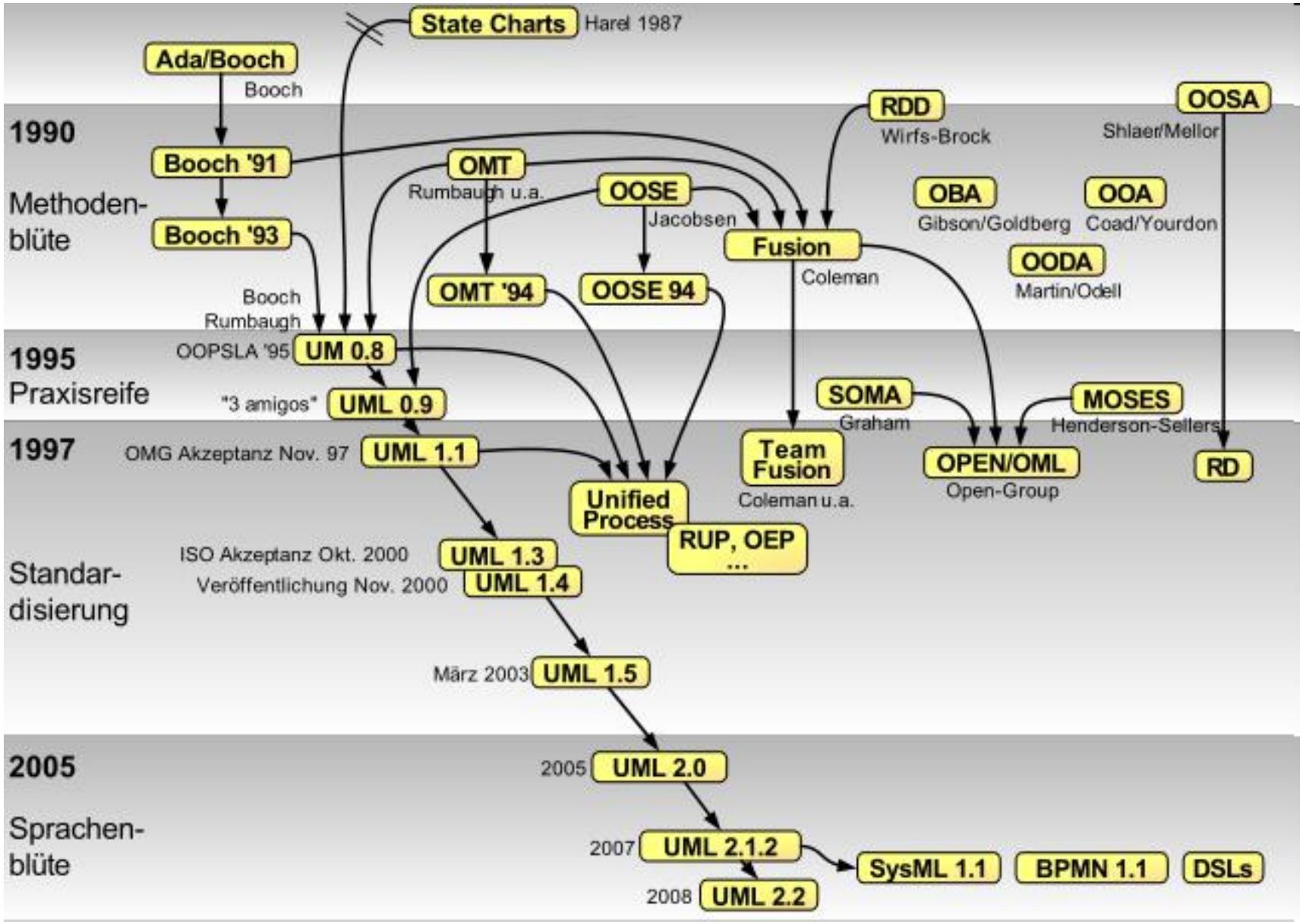
Unified Modeling Language (UML)

- Grafische Modellierungssprache für eine einheitliche Modellierungsnotation für alle Softwaresysteme
  - Konstruktion
  - Visualisierung
  - Dokumentation
- UML bietet verschiedene Diagrammtypen für unterschiedliche Blickwinkel
  - Diese ergänzen sich gegenseitig
- Warum UML?
  - Industriestandard
  - Viele Entwicklungstools unterstützen UML
  - Automatische Code-Generierung



- Aber: UML ist eine Notation, keine Methode zur Modellierung
  - Die Methode ist die objektorientierte Analyse und Design

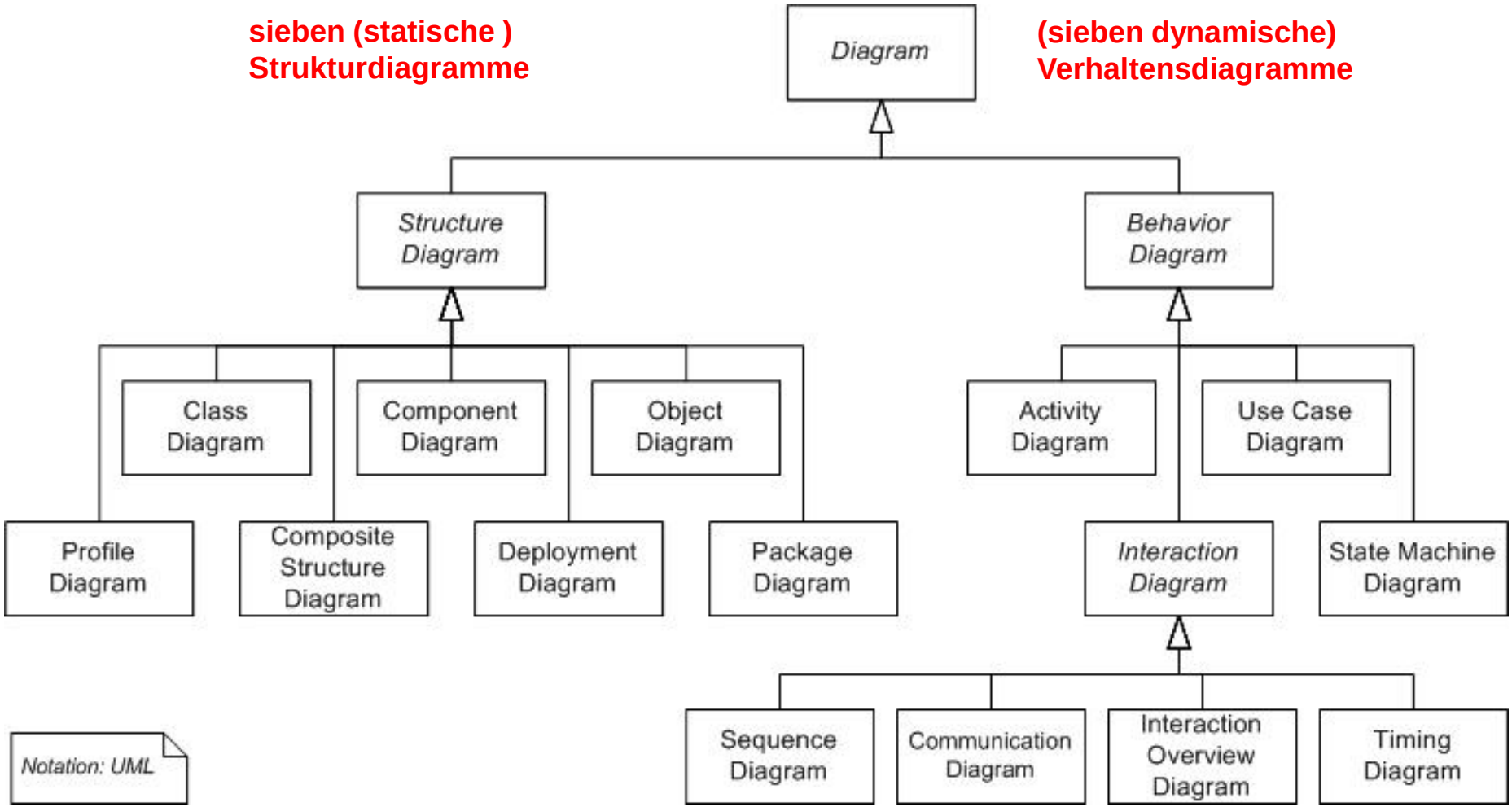
Historisches



Unterstützte Diagrammtypen in UML

sieben (statische) Strukturdiagramme

(sieben dynamische) Verhaltensdiagramme



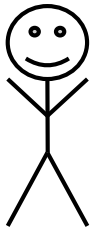
- Keine Bange, wir brauchen nicht alle

Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases)
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Akteure

- Repräsentieren eine kohärente Menge von Rollen, die von Benutzern in der Interaktion mit dem System eingenommen werden können
- Können Menschen, aber auch andere Dinge (z.B. andere automatisierte Systeme sein)
- UML-Symbol:



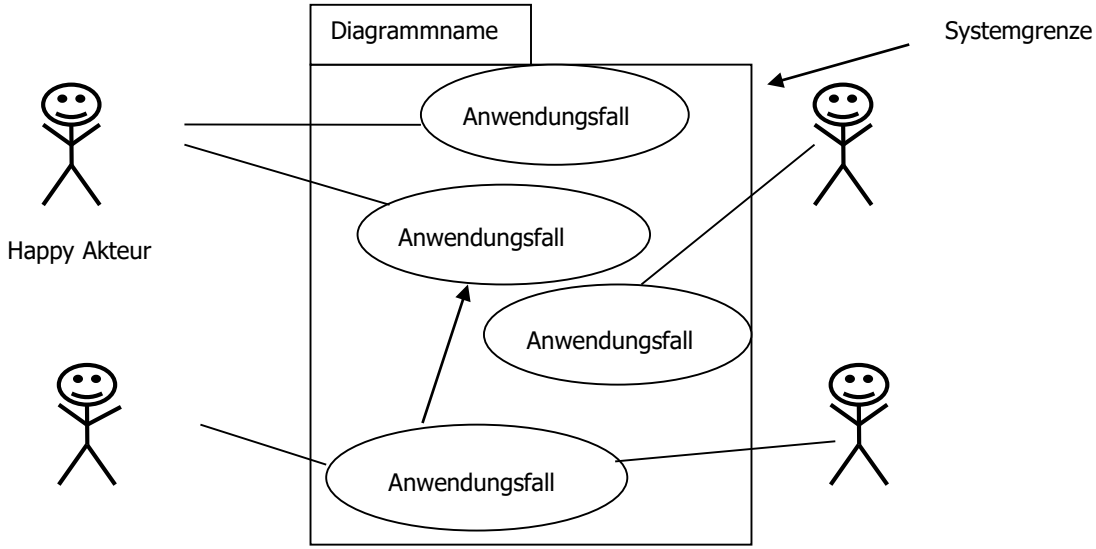
- Sollten schon im Pflichtenheft stehen
- Beispiel: Spieler, Webseitenbesucher, Bankkunde, Systemadministrator, Ausleiher, Bibliothekar,…

Objektorientierte Analyse und Design im Detail

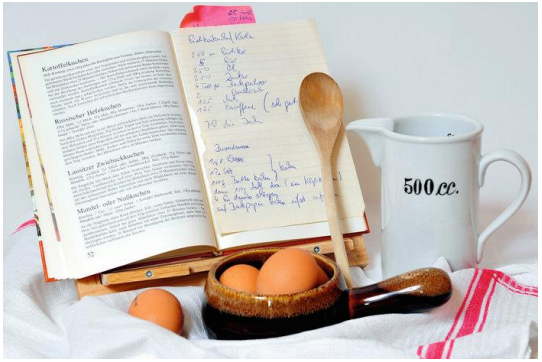
- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases)
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Anwendungsfälle

- Beschreibt eine Menge von Aktionssequenzen (Inkl. Varianten)
- Jede solche Sequenz repräsentiert die Interaktion zwischen externen Akteuren mit dem System
- Folge ist beobachtbares Resultat, relevant für Akteur
- Mehrere Anwendungsfälle bilden einen Geschäftsprozess
- Kennen wir schon aus dem Pflichtenheft

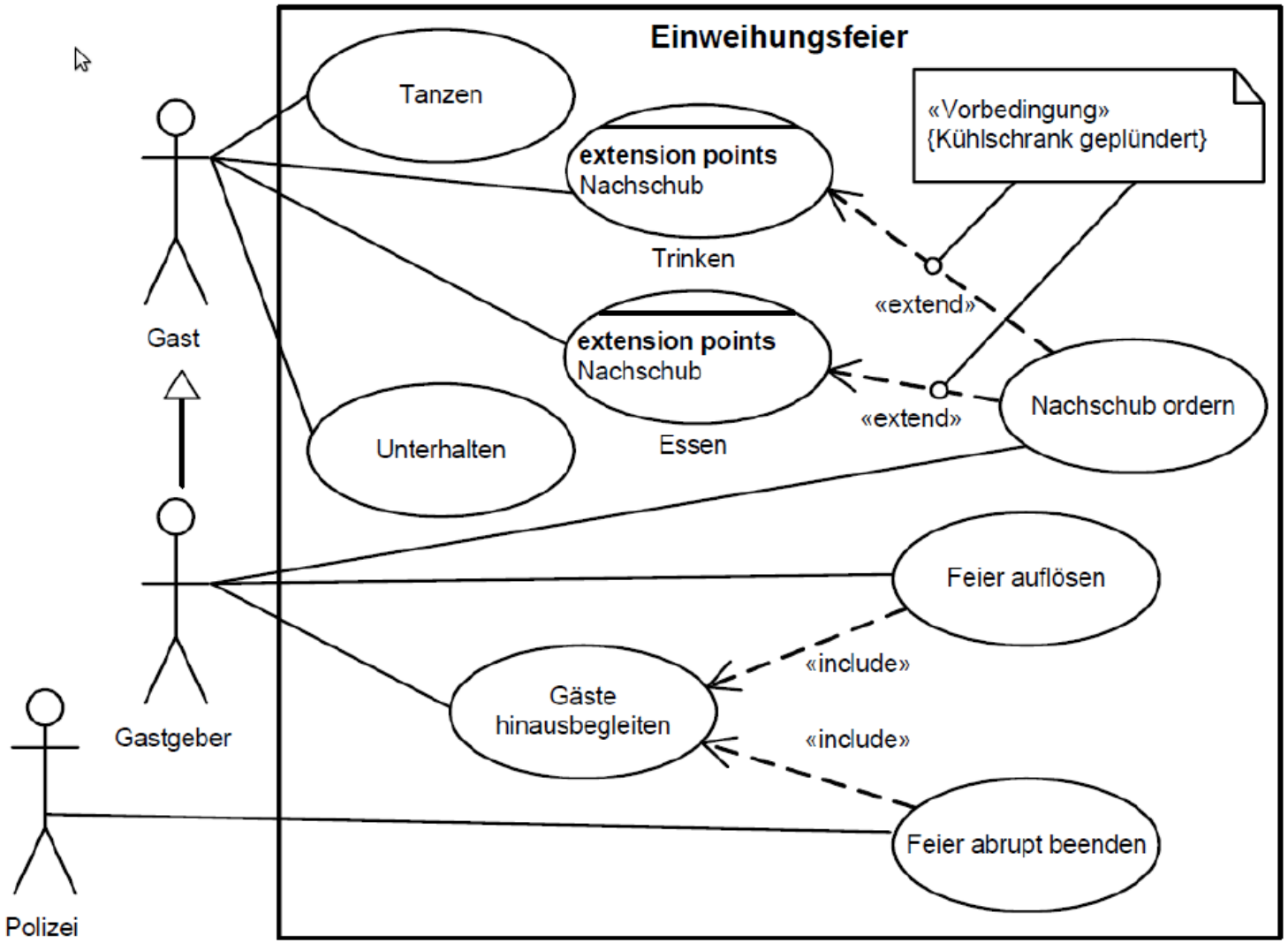


Rezept zur Identifikation von Anwendungsfällen

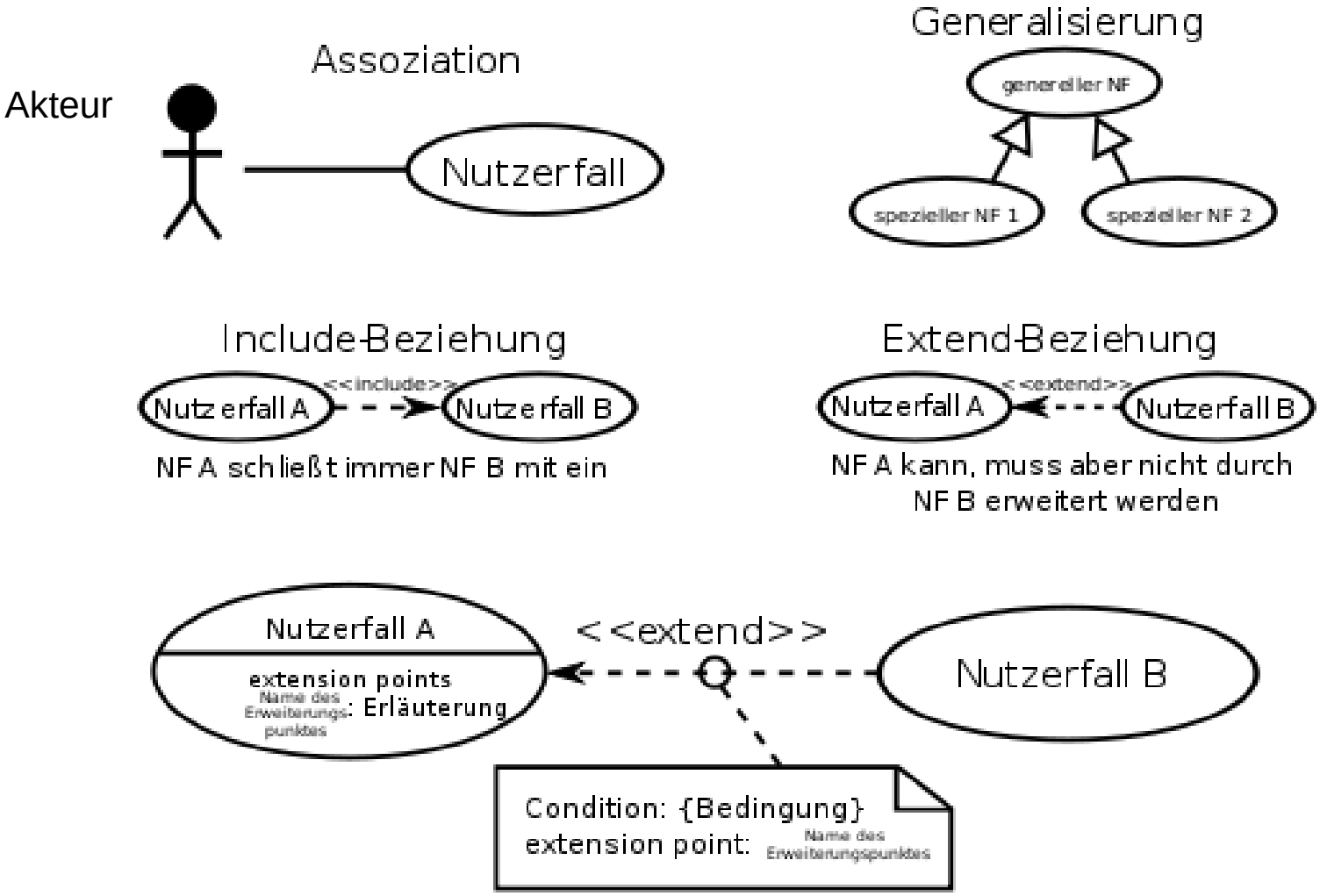


1. Idenzifiziere Akteure
2. Betrachte System aus Sicht der Akteure
3. Bestimme Anwendungsfälle für Akteure
   - Liefert möglicherweise neue Akteure
4. Goto 2., bis keine neuen Akteure und Anwendungsfälle mehr gefunden werden
5. Bestimme Zusammenhänge zwischen Anwendungsfällen und Akteuren
   - Bestimme hierarchische Zusammenhänge und fasse entsprechend zusammen

Beispiel: Einweihungsfeier

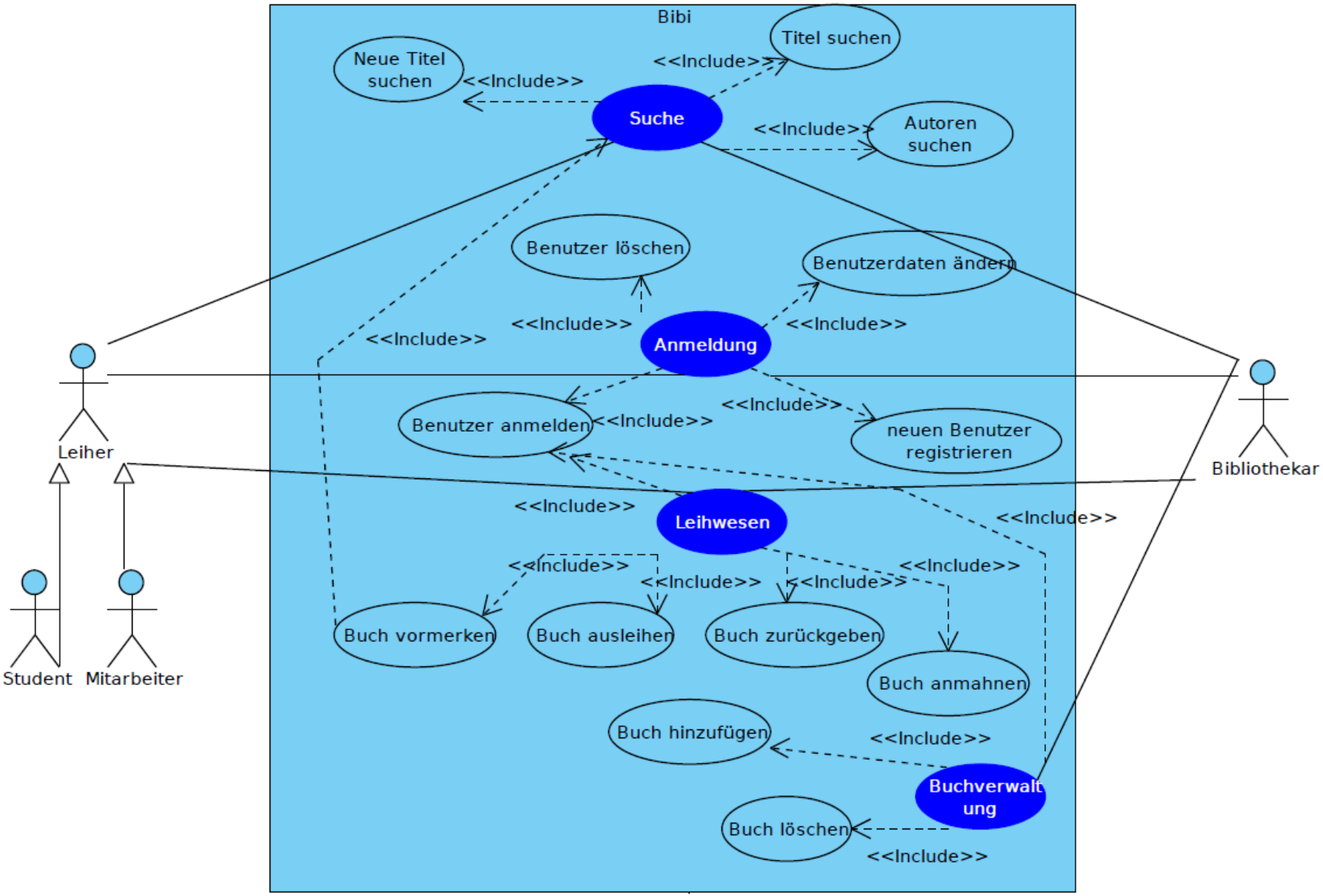


UML-Notation für Use-Case-Diagramme



- Hinweis: Zeitliche Reihenfolge fehlt

Noch ein Beispiel: Bibliothek



Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Identifikation von Objekten

- Definitionsversuch Objekt:
  - „Ding“ (Gegenstand, Entität) mit eigener Identität
- π* -Regel zum Finden von Objekten:
  - Suche nach Substantiven in Anwendungsfällen

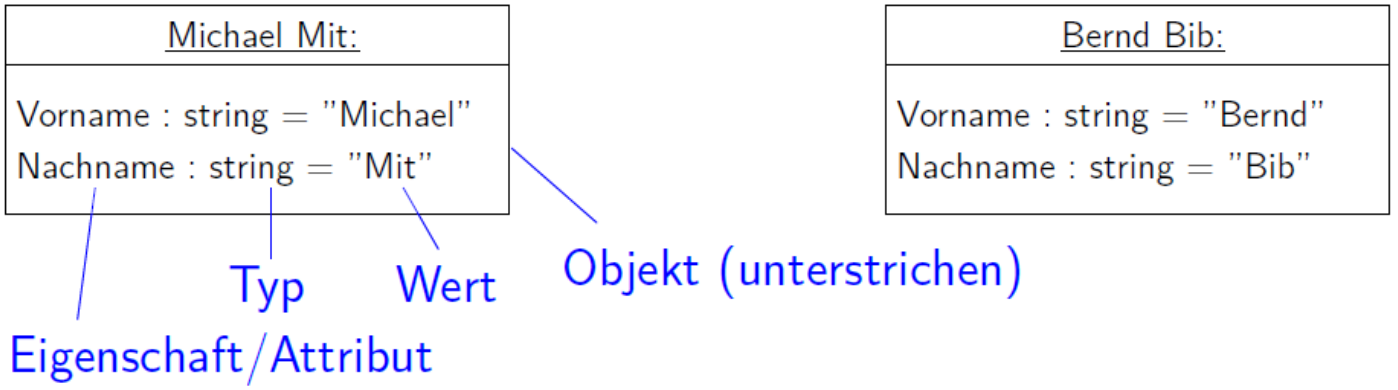
- Beispiel Bibliothek: Vormerkung und Ausleihe
  - Mitarbeiter *Michel Mit* möchte Buch ausleihen
  - *Michel Mit* sucht online nach einem Buch mit Titel *Softwaretechnik*
  - *Michel Mit* reserviert das gefundene Buch
  - *Michel Mit* geht zum Bibliothekar *Bernd Bib* der Universität Bremen
  - *Bernd Bib* sucht nach Reservierung
  - *Bernd Bib* hält Ausleihe fest
  - *Michel Mit* geht mit Buch von dannen

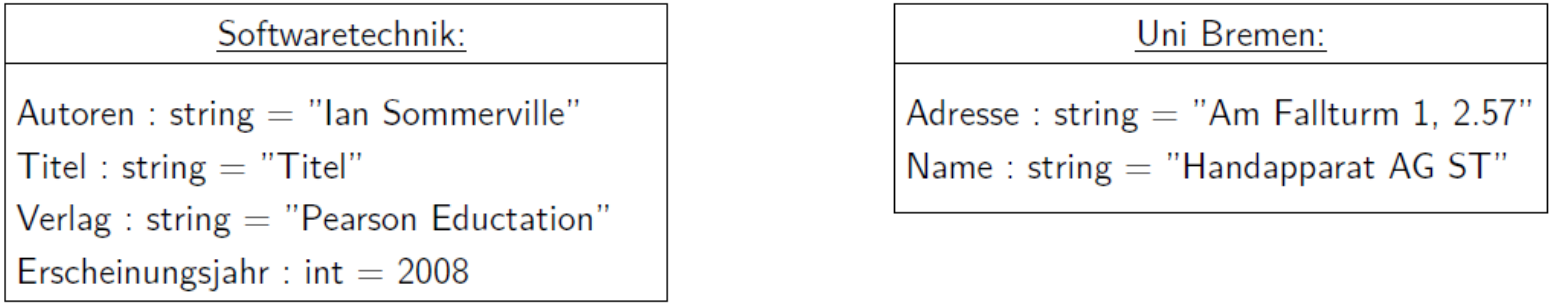
Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Eigenschaften von Objekten

- Eigenschaften (auch Attribute genannt) haben
  - Typ (z.B. Datentypen der Programmiersprache wie int, float, string,…)
  - Und einen Wert





Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

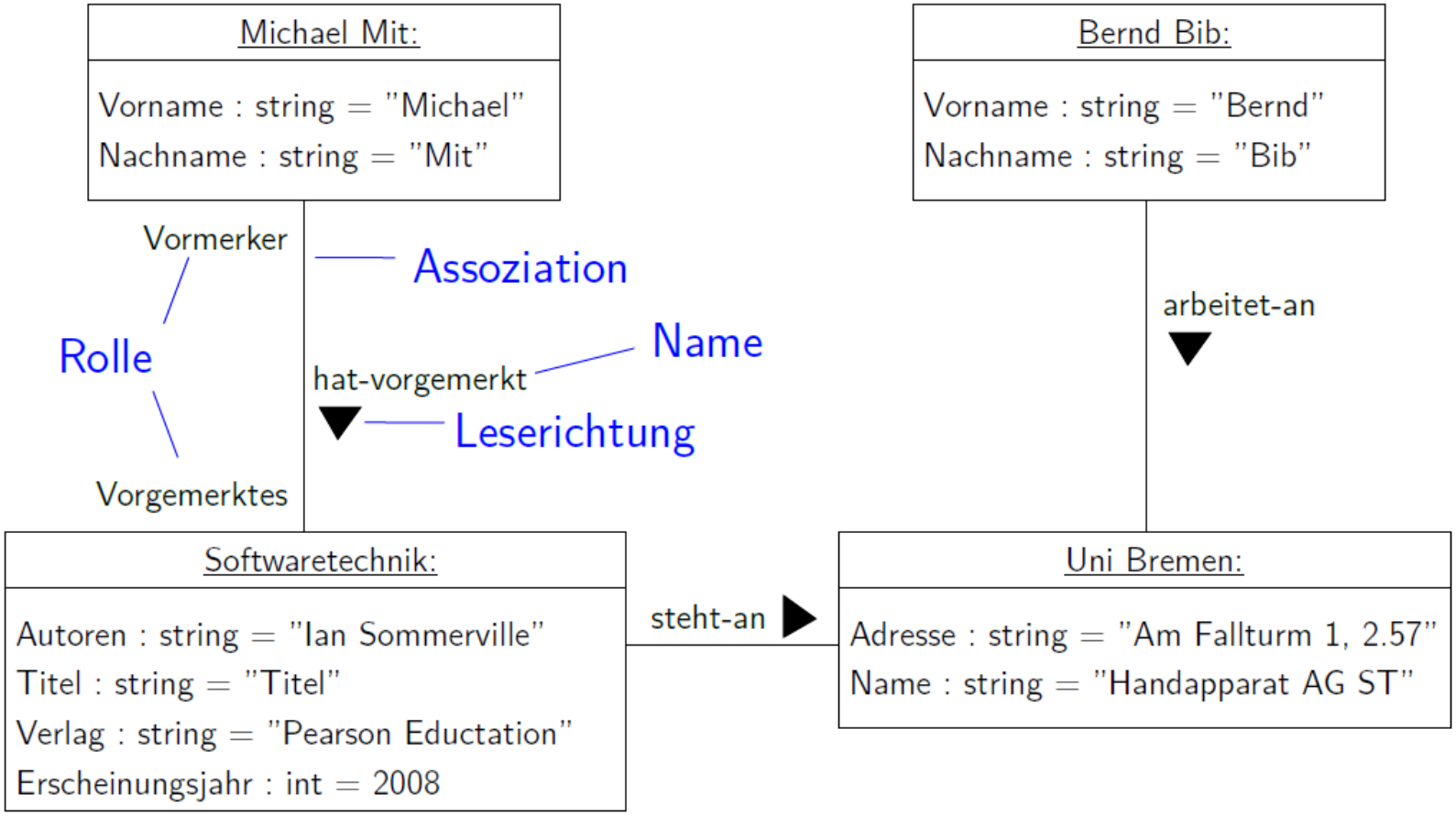
Assoziationen zwischen Objekten

- Definitionsversuch Assoziationen
  - Logische Beziehungen zwischen Objekten
- π* -Regel zum Finden von Assoziationen:
  - Suche nach Verben in Anwendungsfällen

- Beispiel Bibliothek: Vormerkung und Ausleihe
  - Mitarbeiter *Michel Mit* möchte Buch ausleihen
  - *Michel Mit* sucht online nach einem Buch mit Titel *Softwaretechnik*
  - *Michel Mit* reserviert das gefundene Buch
  - *Michel Mit* geht zum Bibliothekar *Bernd Bib* der Universität Bremen
  - *Bernd Bib* sucht nach Reservierung
  - *Bernd Bib* hält Ausleihe fest
  - *Michel Mit* geht mit Buch von dannen

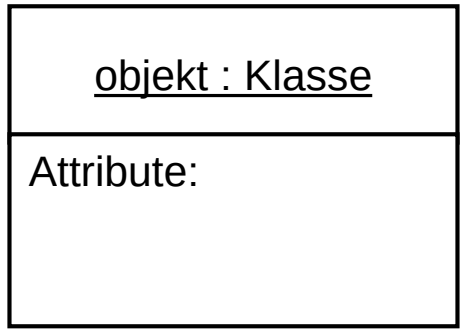
Assoziationen zwischen Objekten

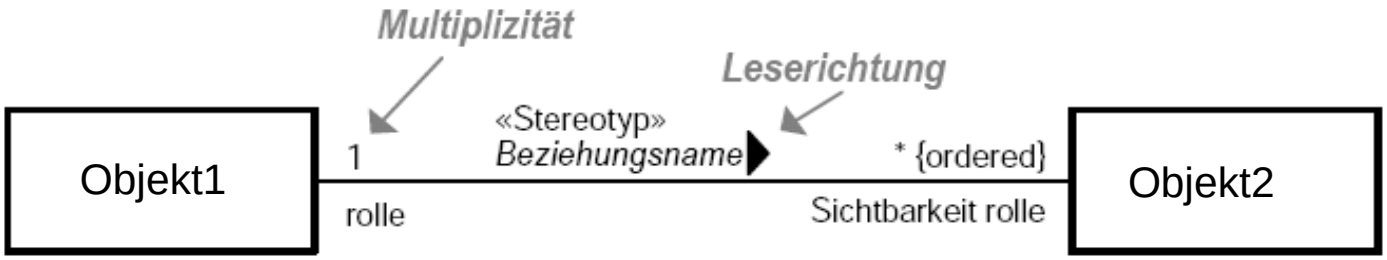
- Jedes Objekt nimmt eine Rolle in der Beziehung ein
- Beziehungen haben einen Namen und eine Richtung



UML-Notation für Objektdiagramme

- UML-Objektdiagramm = Objekte + Assoziationen
- Objekte werden durch Rechtecke visualisiert
- Die Rechtecke sind zweigeteilt
  - Oben: Objektname (eventuell Klassenname)
  - Unten: Attribute
- Der Name wird unterstrichen
- Assoziationen zwischen Objekten:
  - Pfeil gibt Leserichtung vor
  - Optional: Multiplizität (dazu später mehr)



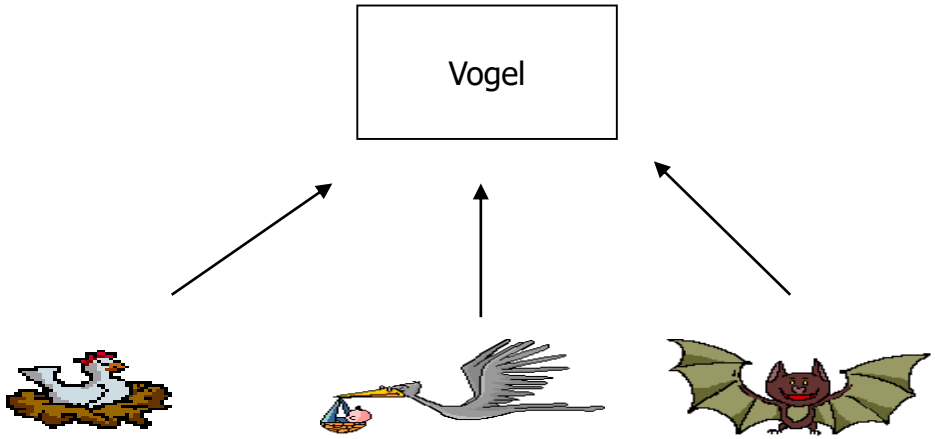


Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte) => Objektdiagramm
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Klassen

- Abstraktes Modell von strukturell gleichartigen Objekten mit gemeinsamen Eigenschaften
  - „Gleichartig“ bedeutet oft (aber nicht immer): Gleiche Attribute



- Außerdem haben alle Elemente einer Klasse die gleichen Beziehungen

Beispiel: Vom Objekt zur Klasse

Michael Mit
Vorname: string = Michael Nachname: string = Mit

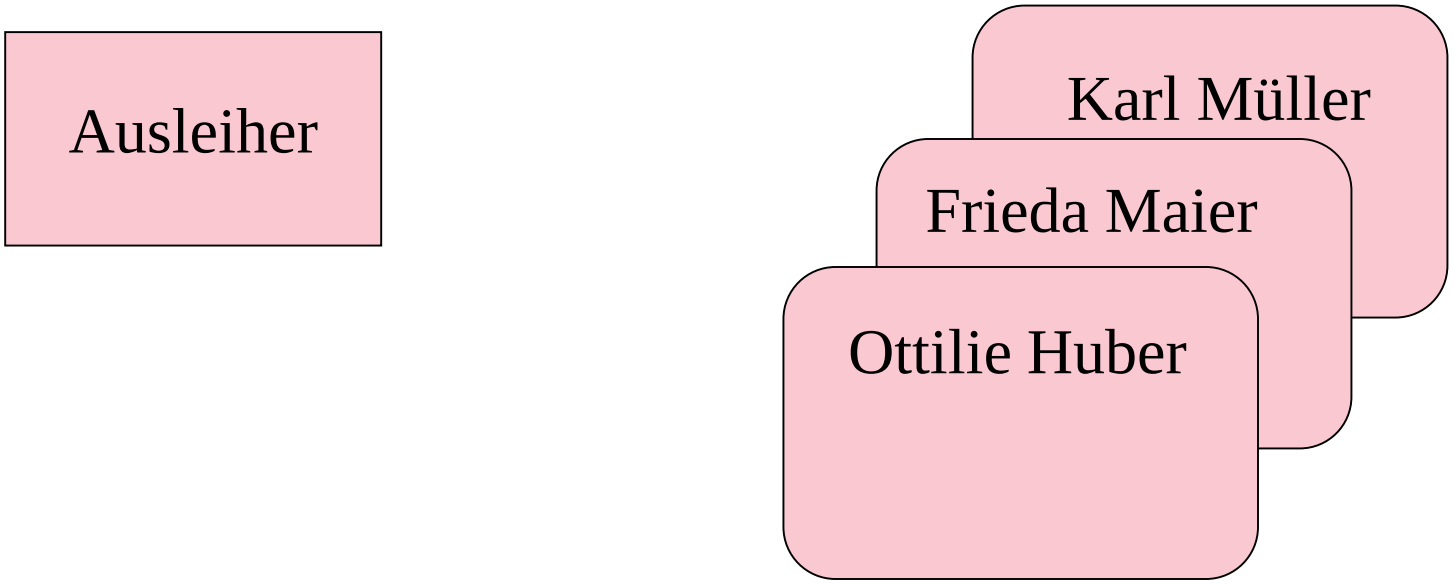
Stefanie Stud
Vorname : string = Stefanie Nachname: string = Stud

Ausleiher
Vorname: string Nachname: string

gemeinsame Attribute

Terminologie: Objekt vs. Klasse

- Klasse
  - Abstrakt
  - Bestimmt Typ eines Objekts
- Objekt (auch Instanz genannt)
  - Konkrete Ausprägung einer Klasse

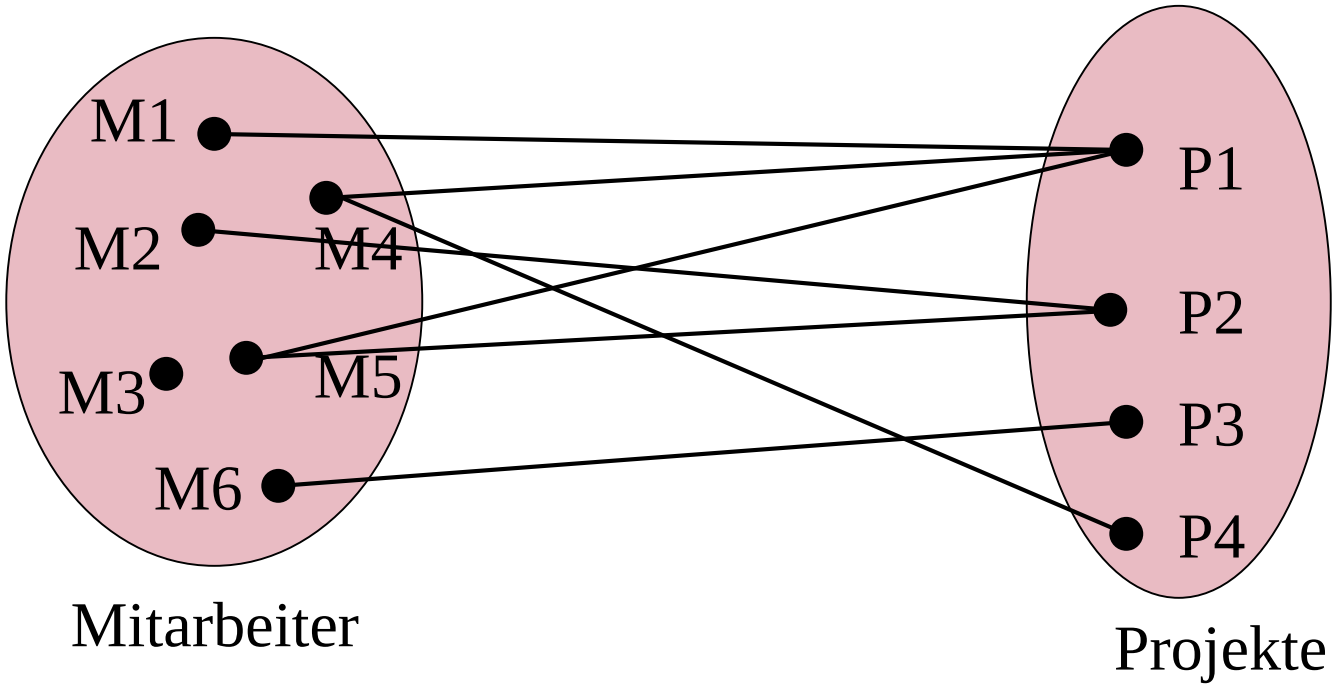


Objektorientierte Analyse und Design im Detail

- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte) => Objektdiagramm
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

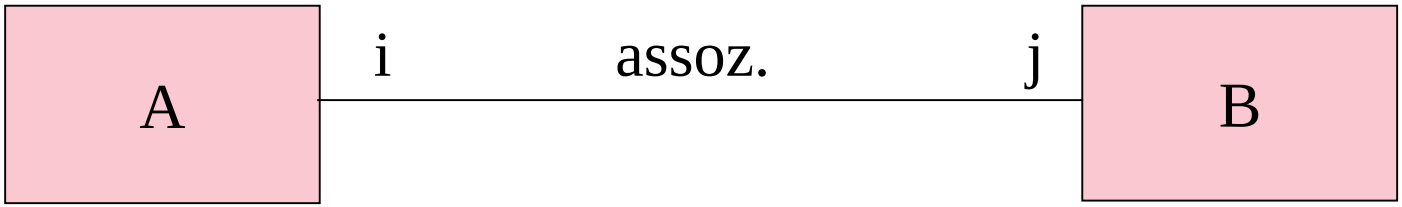
Multiplizität

- Die Multiplizität einer Assoziation drückt aus, wie viele Objekte des jeweiligen Typs in Beziehung stehen können
  - Verwandt mit dem Entity-Relationship-Modell der Datenbanken
  - Dort ist aber eher der Begriff Kardinalität gebräuchlich
  - Dient dazu, dass Modell auf Konsistenz und Integrität zu prüfen



Multiplizität von Assoziationen

- Spezifikation der Zuordnungswertigkeit einer Beziehung

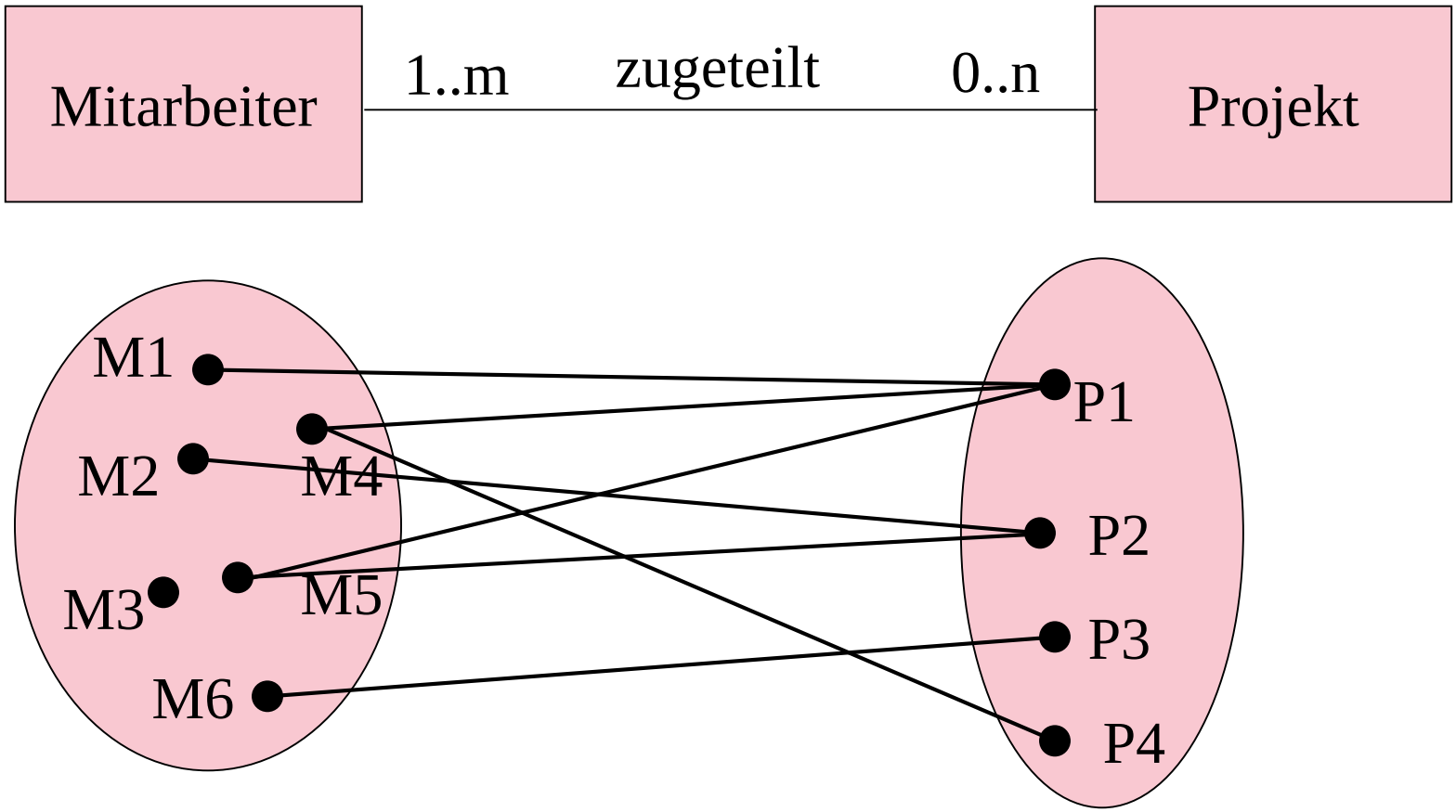


i: Anzahl der Instanzen der Klasse A, die mit einer Instanz der Klasse B in Beziehung stehen können.

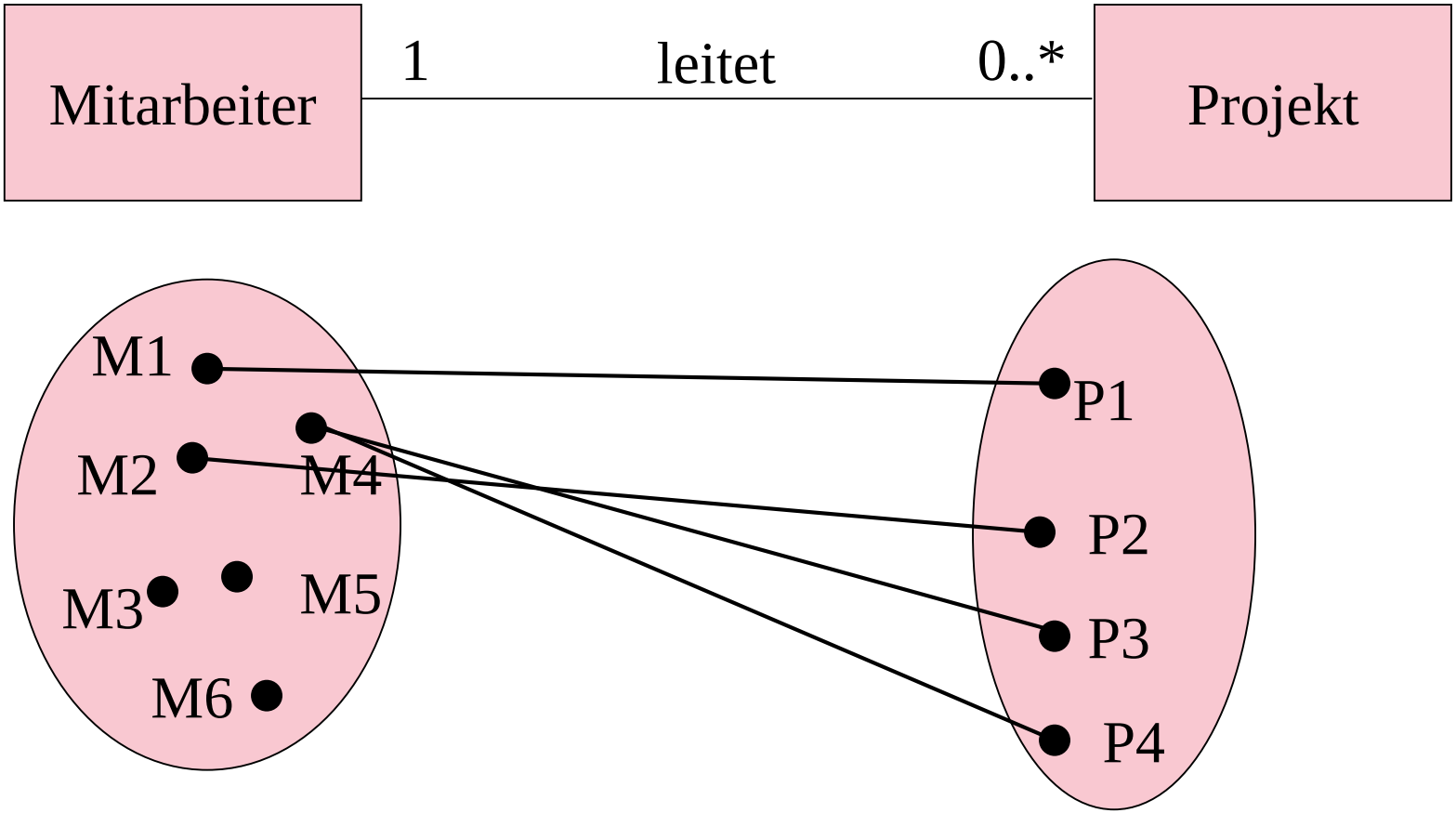
Angabe: Zahl, Intervall, *, Kombination

Bsp: 1; 0..5; *; 0..3, 7..9, *21, 5..*

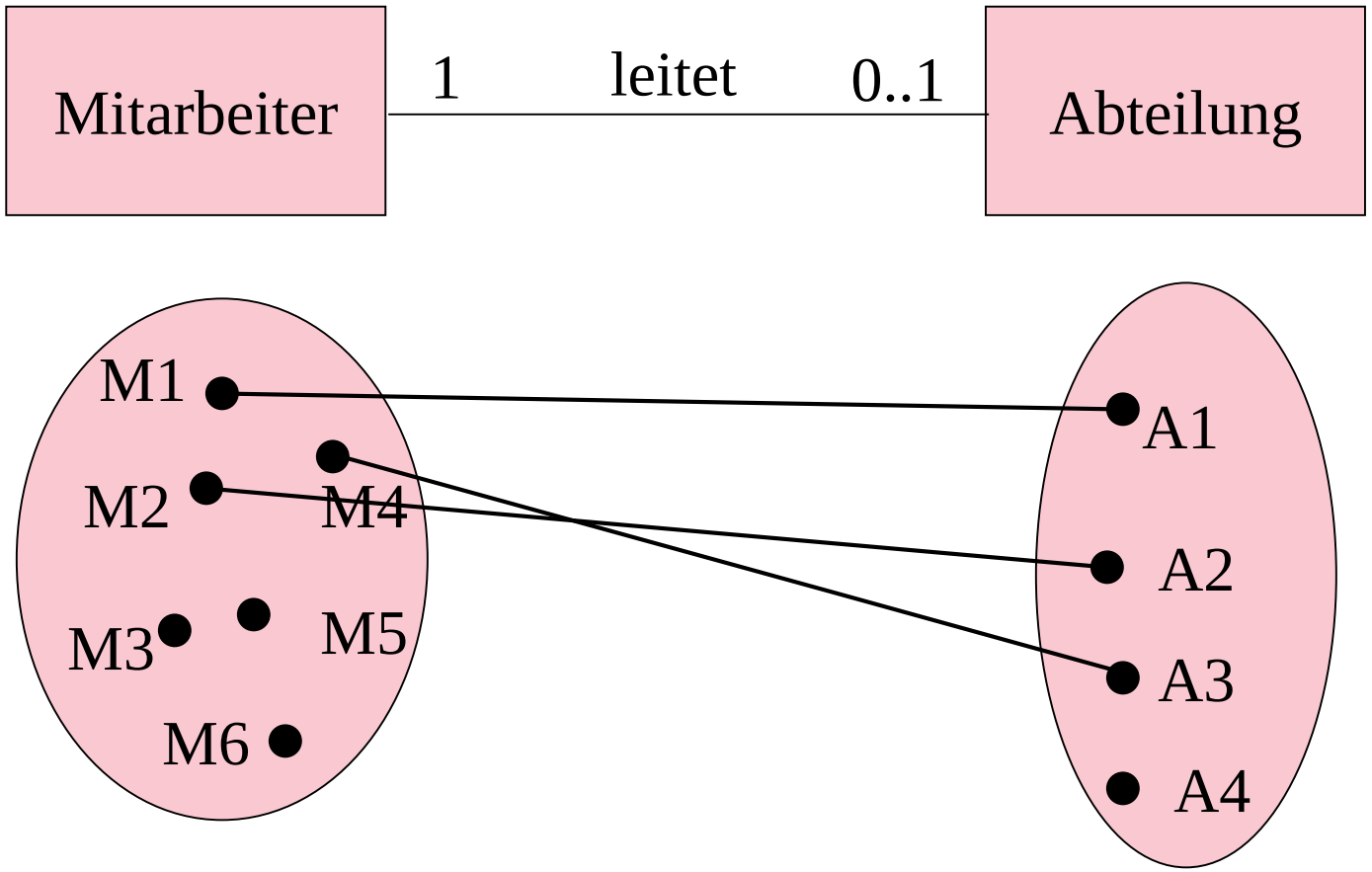
Multiplizität - m : n



Multiplizität - 1:n



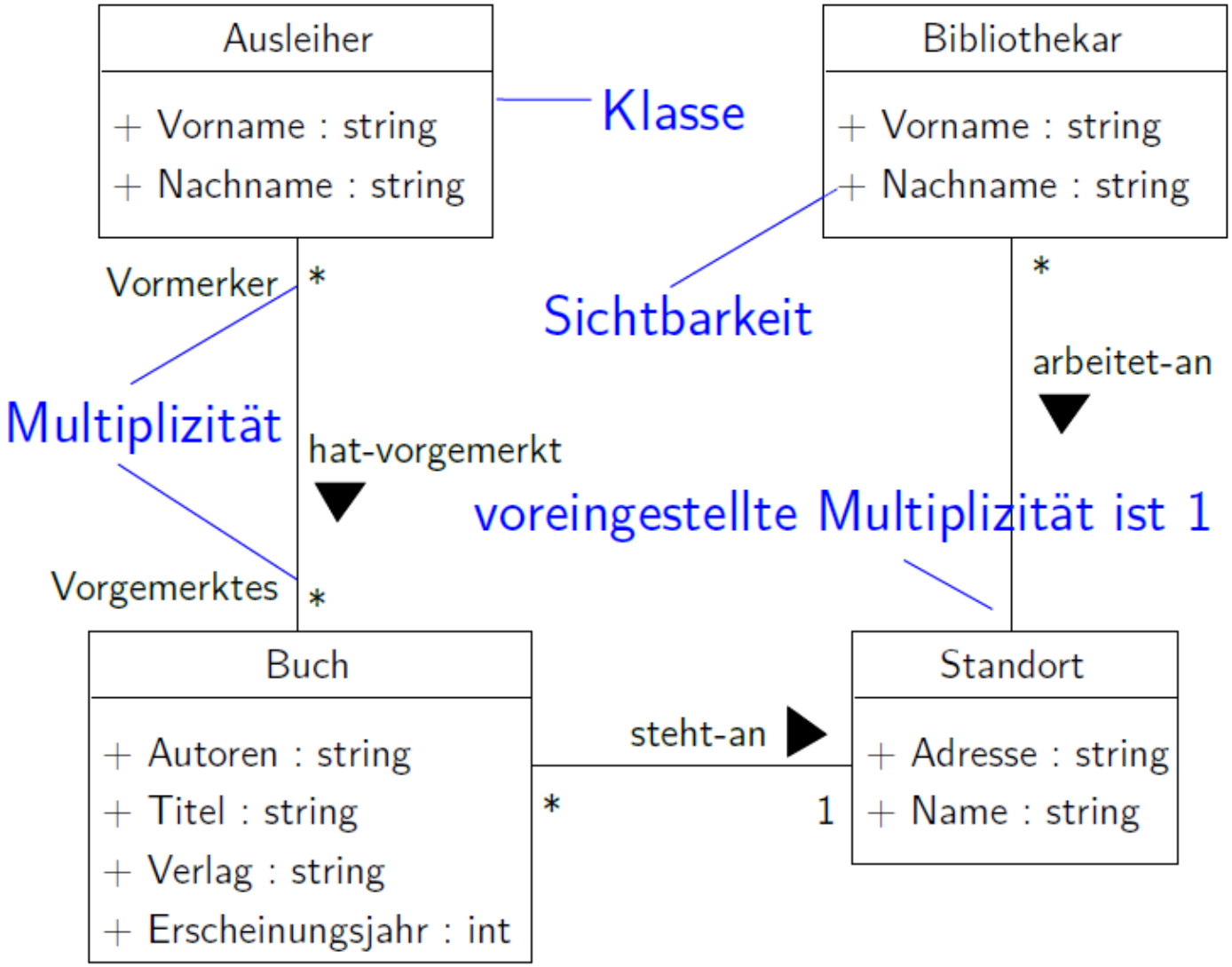
Multiplizität - 1 : 1



Sichtbarkeit

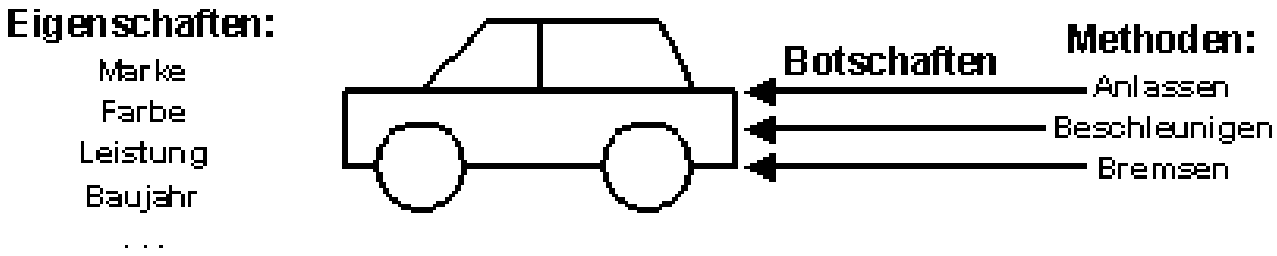
- Für Attribute kann man die Sichtbarkeit festlegen
- Diese dienen der besseren Datenkapselung
- Mögliche Werte für die Sichtbarkeit sind public, private, protected (und package)
  - public (UML-Zeichen +): Das Attribut ist für jeden sichtbar
  - private (UML-Zeichen -): Das Attribut ist nur innerhalb der Klasse sichtbar
  - protected (UML-Zeichen #): Das Attribut ist innerhalb der Klasse und in Spezialisierungen der Klasse (auch abgeleiteten Klassen) sichtbar, dazu später mehr
  - package (UML-Zeichen ~): Das Attribut ist innerhalb eines Packages (im Prinzip eine Zusammenfassung mehrerer Klassen) sichtbar
- Die Werte korrespondieren zu Sichtbarkeiten, wie man sie aus Programmiersprachen wie C++ oder Java kennt
  - Package nur von einigen Sprachen wie Java oder C# unterstützt

Beispiel


Ausleiher
+ Vorname : string
+ Nachname : string
Klasse
Bibliothekar
+ Vorname : string
+ Nachname : string
Vormerker
*
Sichtbarkeit
*
arbeitet-an
Multiplizität
hat-vorgemerkt
voreingestellte Multiplizität ist 1
Vorgemerktes
*
Buch
+ Autoren : string
+ Titel : string
+ Verlag : string
+ Erscheinungsjahr : int
steht-an
*
1
Standort
+ Adresse : string
+ Name : string

Objektorientierung

- Wichtigstes Grundprinzip der Objektorientierung:
  - Fasse Software als Sammlung diskreter Objekte auf, die sowohl zur
  - Speicherung der Daten (und Datenstrukturen) dienen
  - als auch Methoden (oder Funktionen) zur Manipulation oder zum Zugriff auf die Daten zur Verfügung stellen

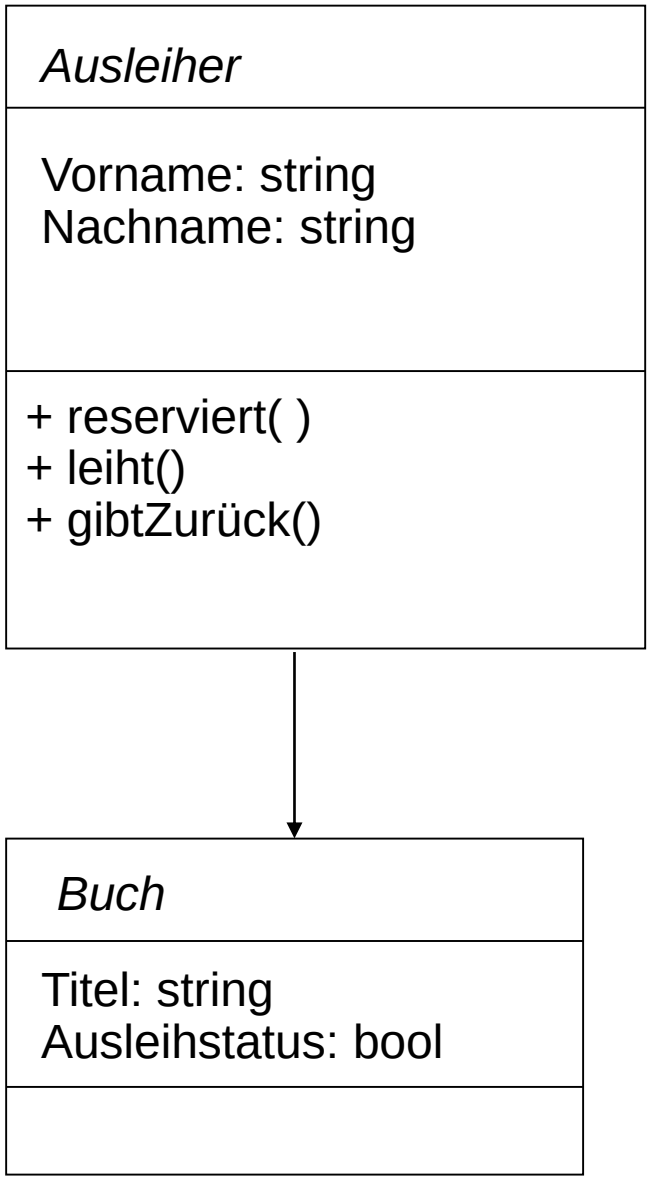


- Kurz: Objektorientiert = Daten + Methoden

Warum Objektorientierung?

- Datenkapselung
  - Abschottung der internen Implementierung vor direktem externen Zugriff.
  - Zugriff nur über fest definierte Schnittstellen
- Wiederverwendbarkeit
  - Klassen können in verschiedenen Systemen verwendet werden
- Denken in Terminologien der realen Welt
  - Weniger Abstraktion notwendig
- Reduktion vom Komplexität
  - Klassen kapseln Details
  - Bieten Struktur

Beispiel: Klassen und Funktionen

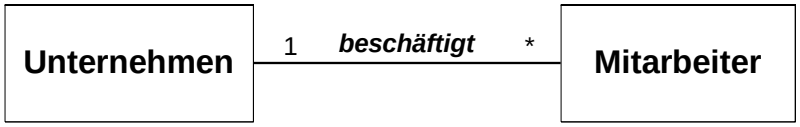


Objektorientierte Analyse und Design im Detail

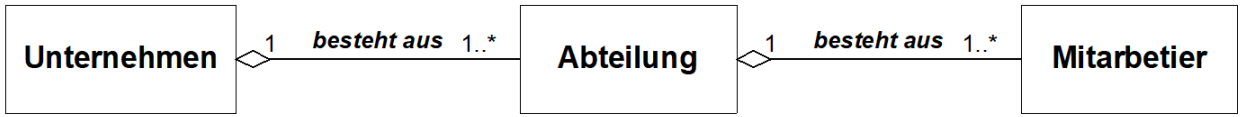
- Identifiziere Akteure
- Beschreibe Anwendungsfälle (Use Cases) => Use-Case-Diagramm
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen der Objekte) => Objektdiagramm
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Funktionen und Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

Verschiedene Assoziationen zwischen Klassen

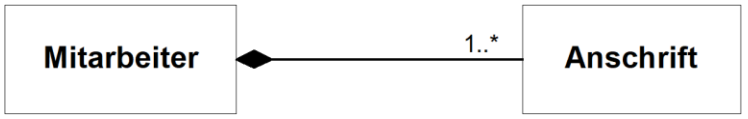
- Assoziation
  - Benutzt-Beziehungen zwischen gleichrangigen Klassen



- Aggregation
  - Teile/Ganzes-Beziehung
  - Zusammensetzung eines Objekts aus Einzelteilen

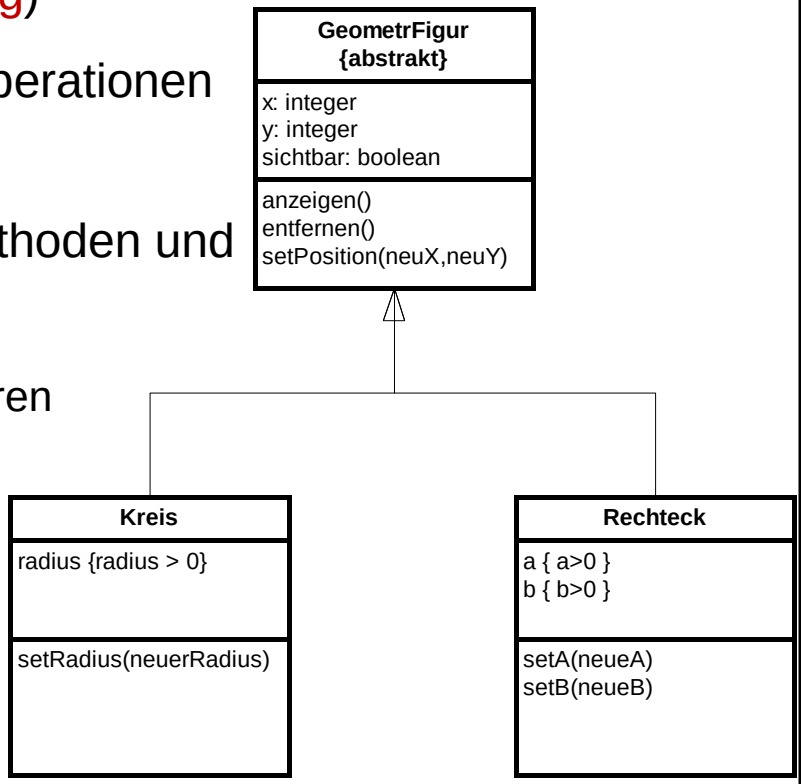


- Komposition
  - Strenge Form der Aggregation
  - Teile vom Ganzen sind existenzabhängig



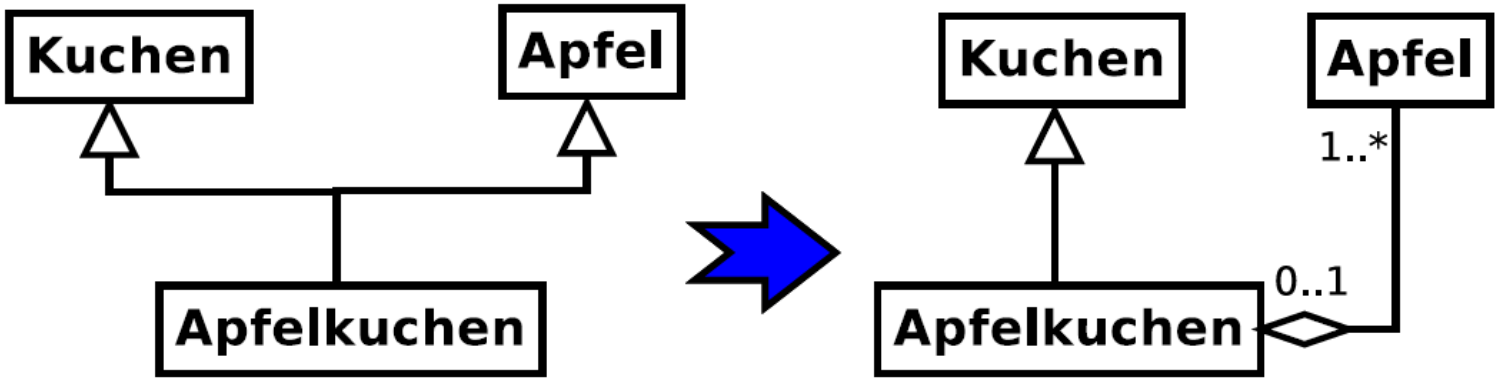
Eine spezielle Assoziation: Vererbung

- Vererbung
  - Oberbegriff-Beziehung (Ist-ein-Beziehung)
  - Die Unterklasse erbt die Attribute und Operationen der Oberklasse
  - Unterklassen können neue Attribute, Methoden und Assoziationen definieren
    - Aber auch vererbte Elemente neu redefinieren
  - Generalisierung – Spezialisierung
  - Vererbung über mehrere Oberklassen möglich (Mehrfachvererbung)



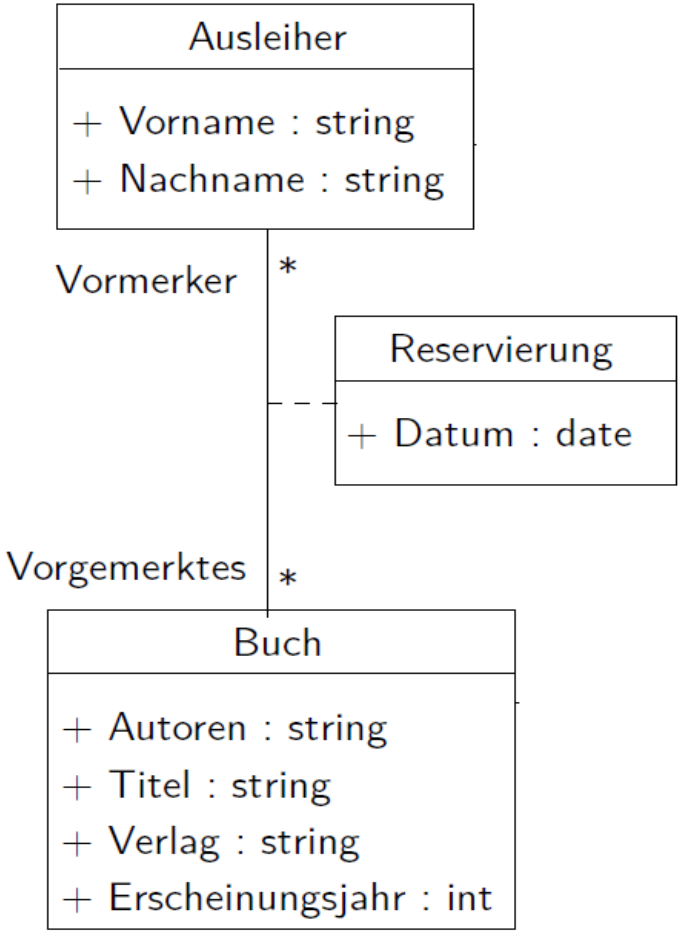
Liskovs Substitutionsprinzip (1988)

- Wie entscheidet man, ob eine Klasse eine Spezialisierung einer anderen Klasse ist?
- Liskovs Substitutionsprinzip: Jede Instanz einer Unterklasse kann immer überall dort eingesetzt werden, wo Instanzen der Oberklasse auftreten



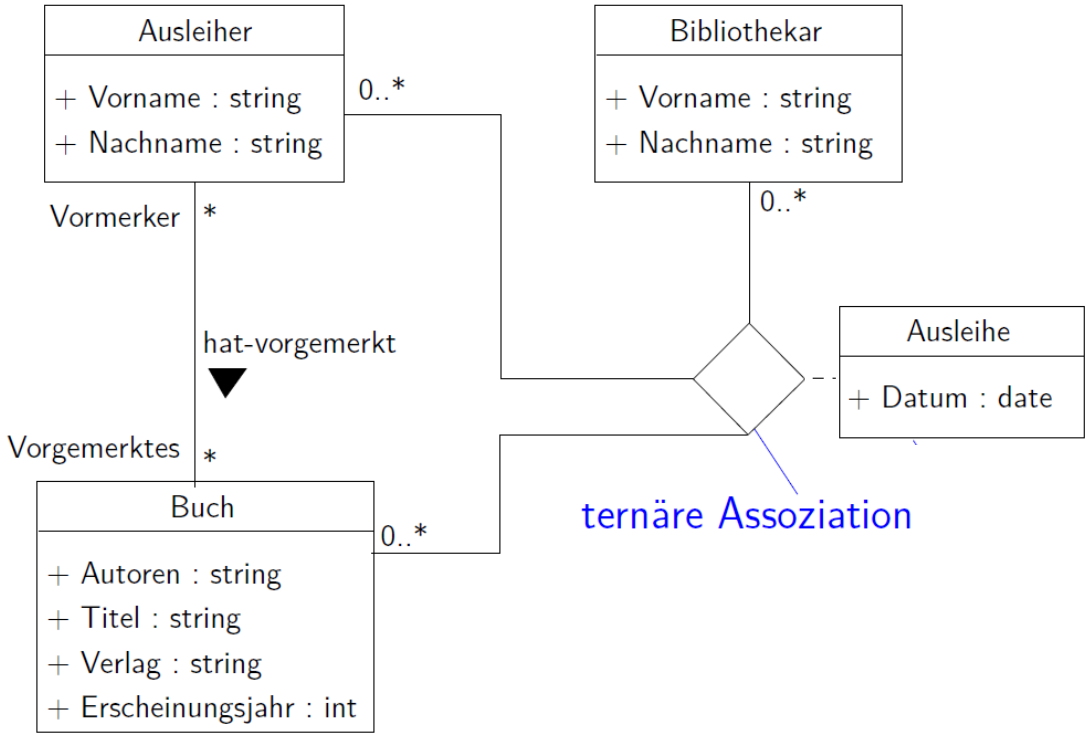
Weitere Assoziationen

- Assoziationsklassen
  - Wenn Assoziationen eigene Attribute haben, kann man eine Assoziationsklasse erstellen
  - Eigenes Klassensymbol mit Attribut

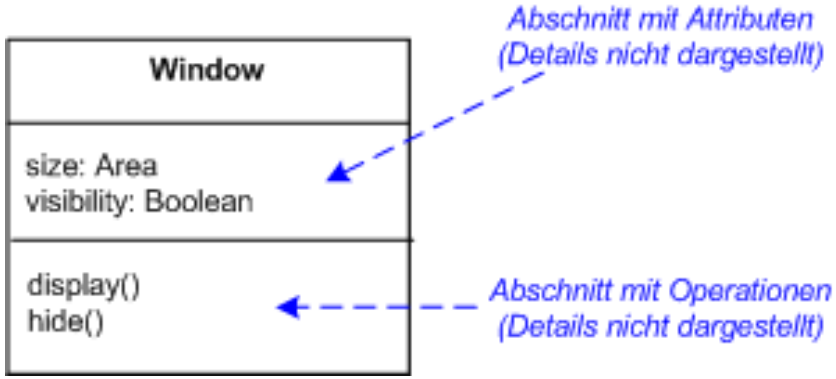


Weitere Assoziationen: Mehrstellige Assoziationen

- n-äre Assoziationen
  - Assoziation zwischen mehreren Klassen
  - Eventuell mit zugeordneter Assoziationsklasse



Notation von Klassendiagrammen in UML

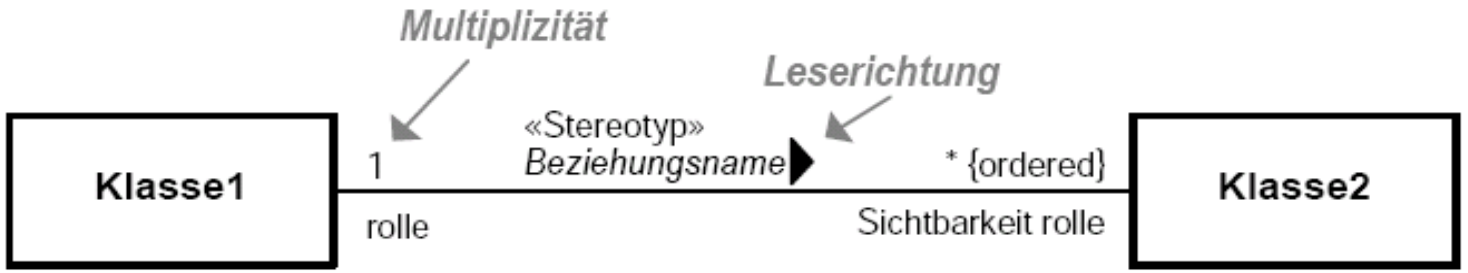


Sichtbarkeit

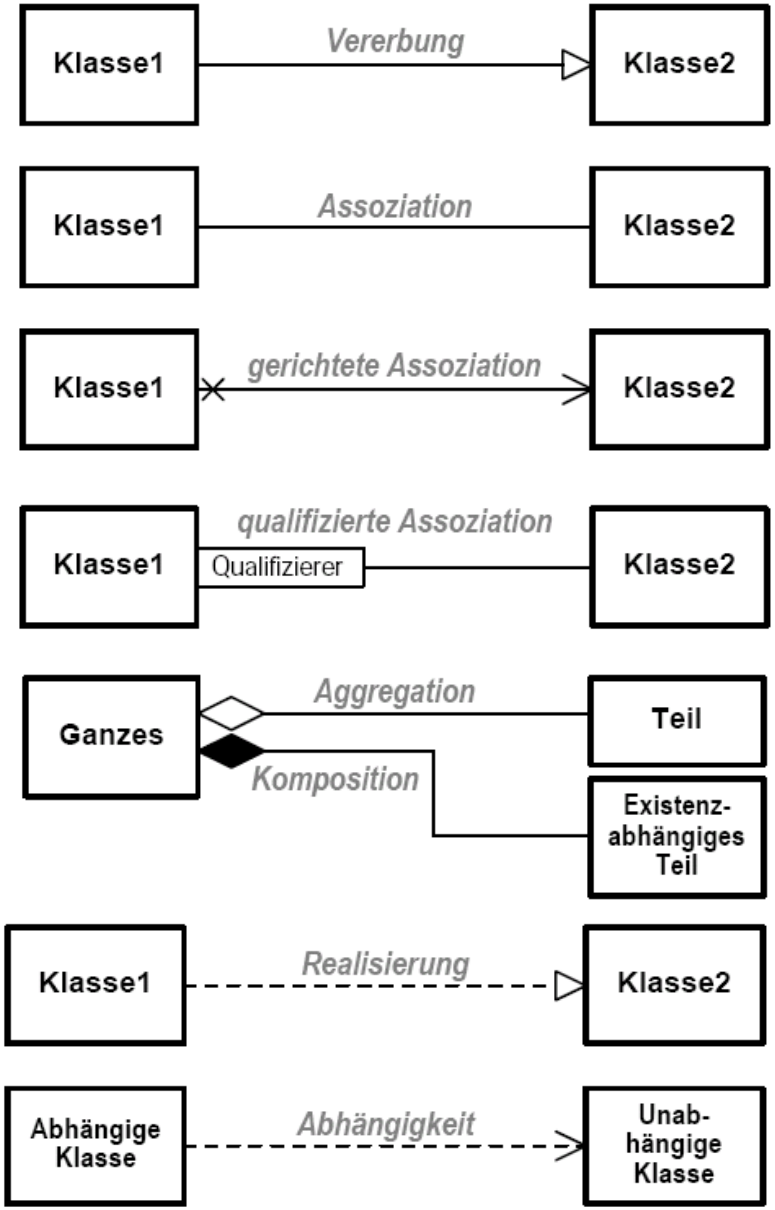
+ für public
# für protected
~ für package
− für private

Abstrakt: kursiv geschriebener Klassennamen.
Oder „{abstract} Klassenname“.

Assoziationen:



Notation von Klassendiagrammen in UML (cont)



Klasse1 erbt von Klasse2
(Klasse1 spezialisiert Klasse2)

Beziehung zwischen Klasse 1 und Klasse 2

Gerichtete Assoziation
Navigierbar: von Klasse 1 nach 2
nicht navigierbar: von Klasse 2 nach 1

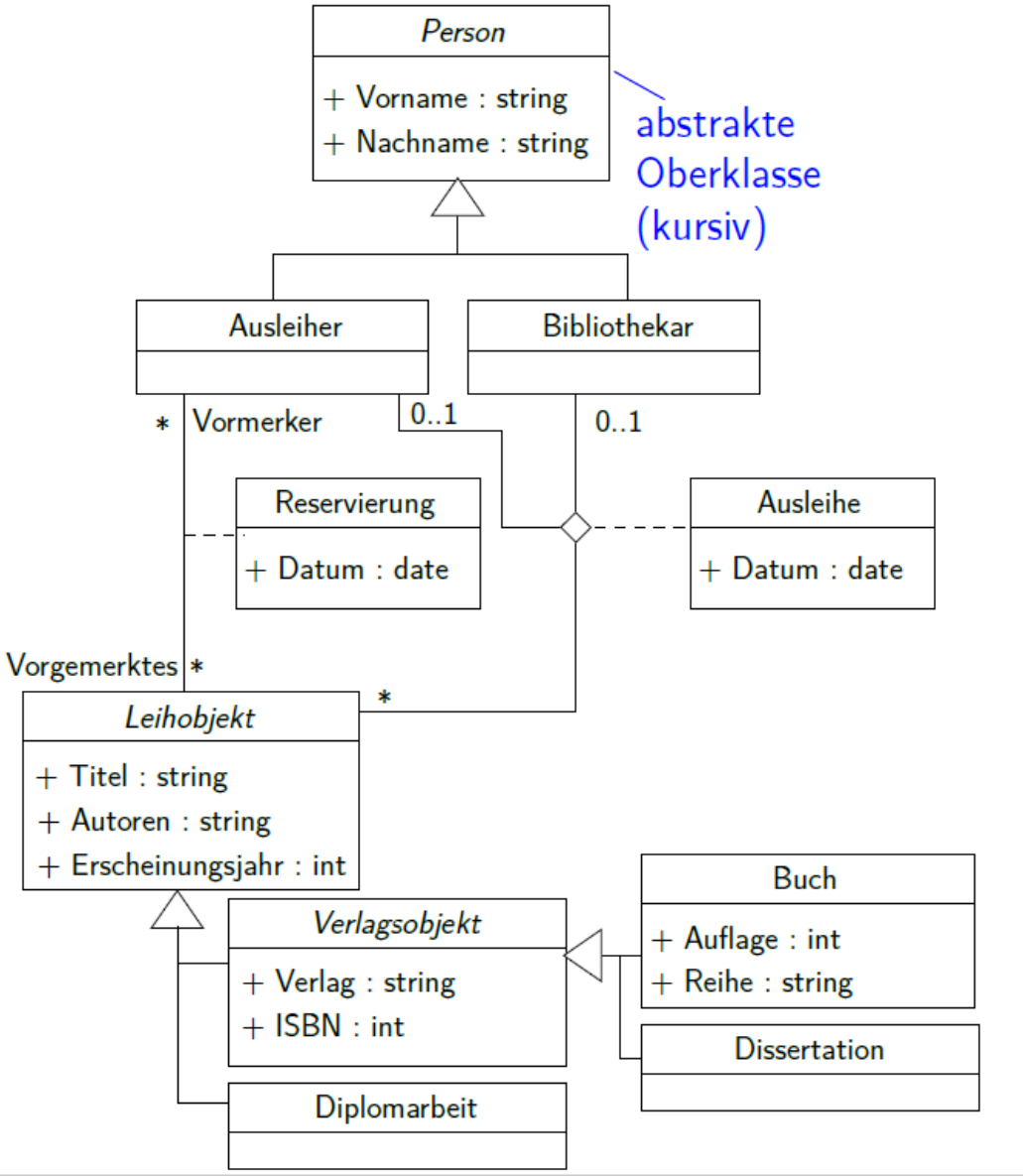
Beziehung zwischen Klasse 1 und Klasse 2

Aggregation:
Teil des Ganzen.
Der Teil kann aber auch alleine bestehen.

Komposition:
„Existenzabhängiges Teil“ kann nur in Verbindung mit dem Ganzen existieren.

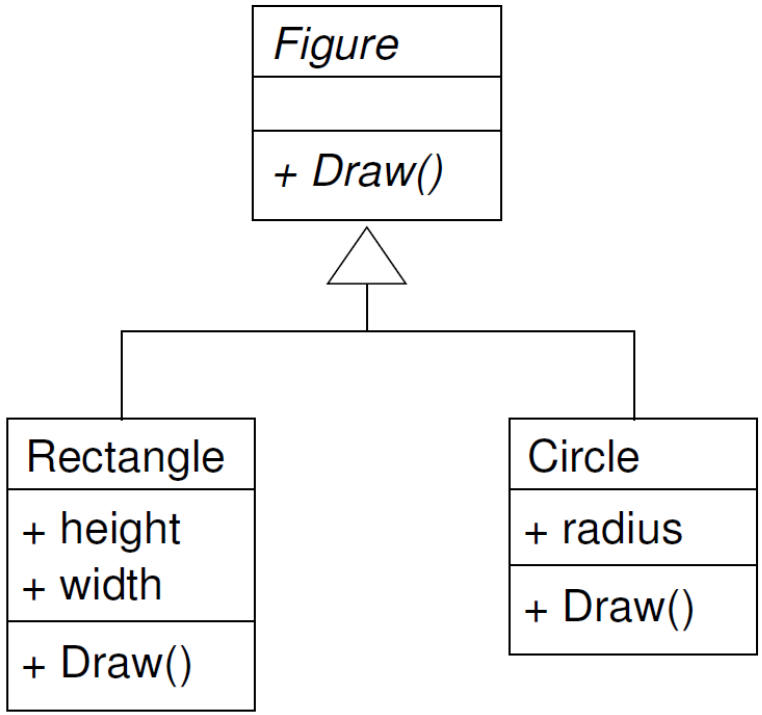
Klasse1 realisiert Klasse2

Beispiel Klassendiagramm



Warum das Ganze?

- Graphische Darstellung des späteren Codes
  - Automatische Übersetzung UML -> Programmiersprache




```
class Figure
{
   public virtual void draw();
}

class Rectangle : Figure
{
   public void draw();
   public int height;
   public int width;
}

class Circle : Figure
{
   public void draw();
   public int radius;
}
```

Noch ein Beispiel aus der Praxis

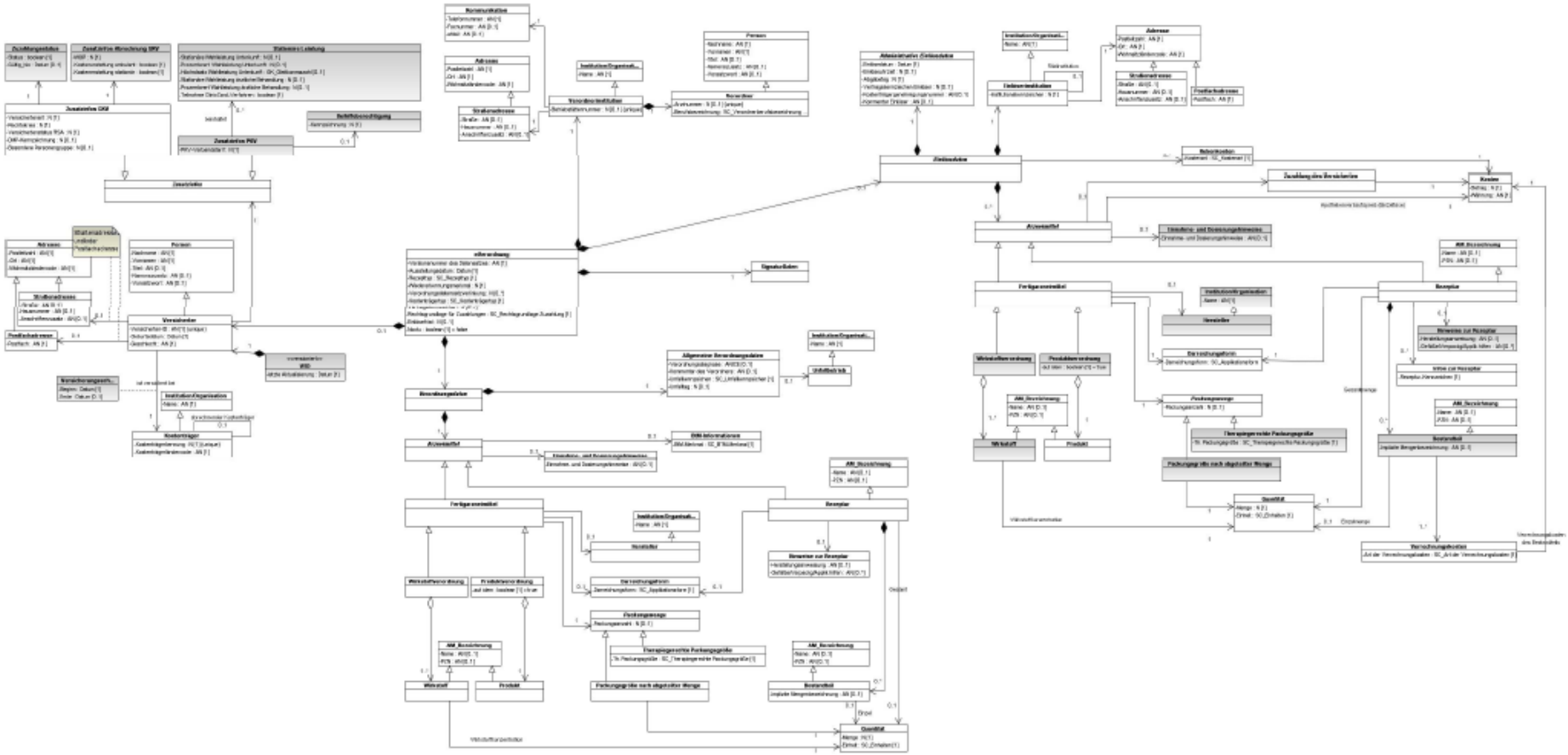
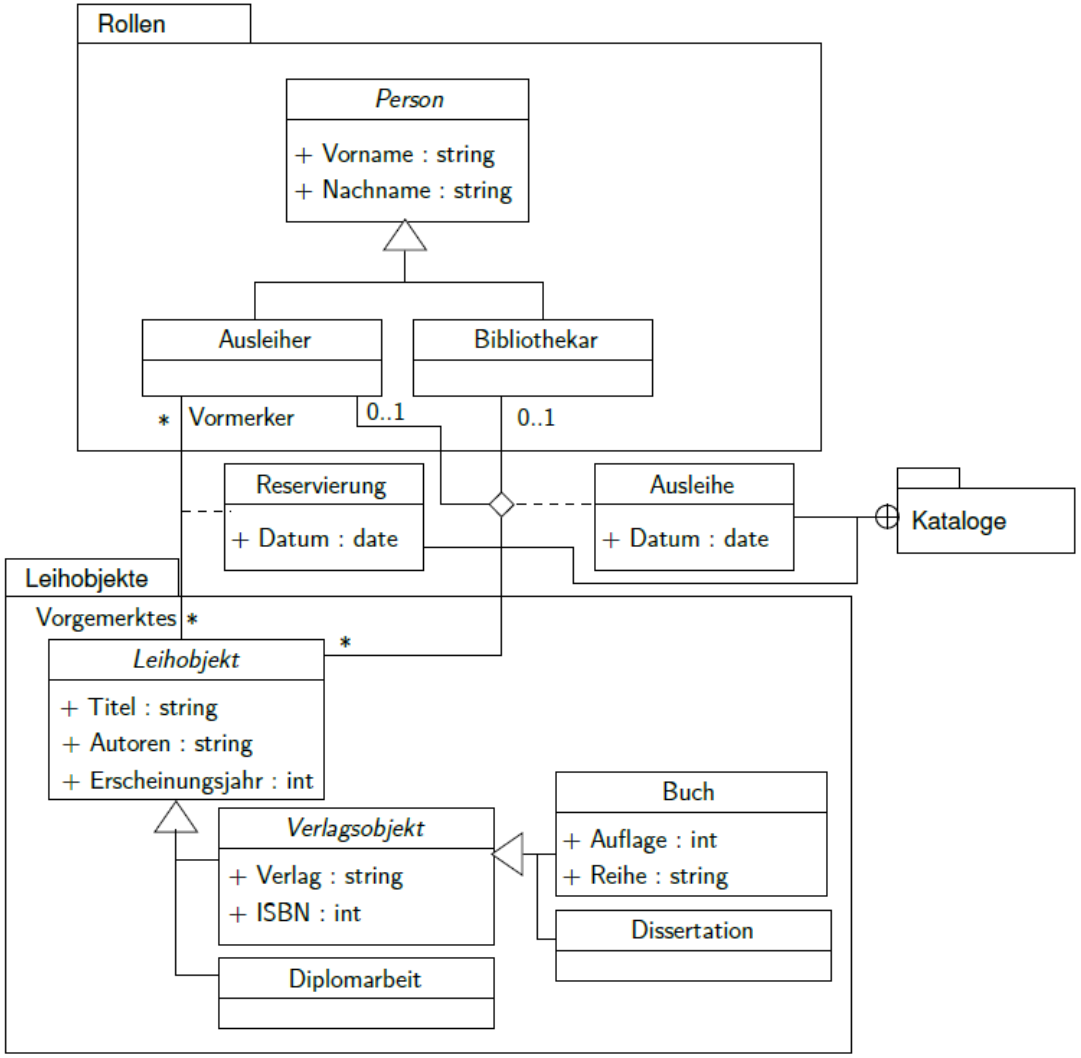
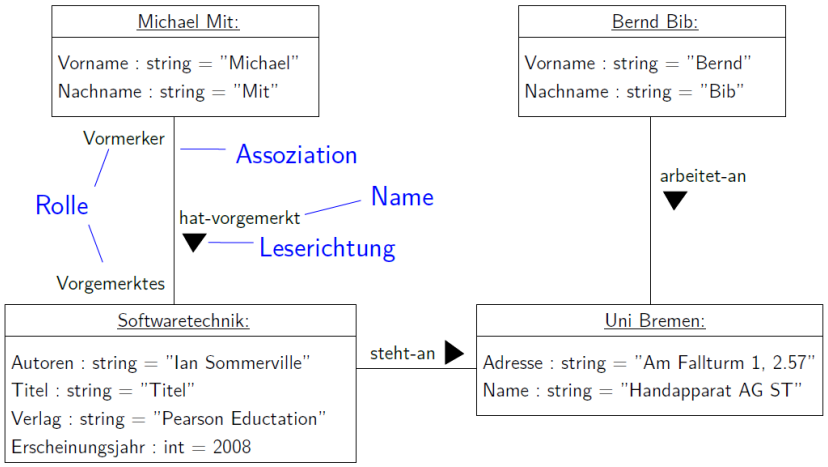
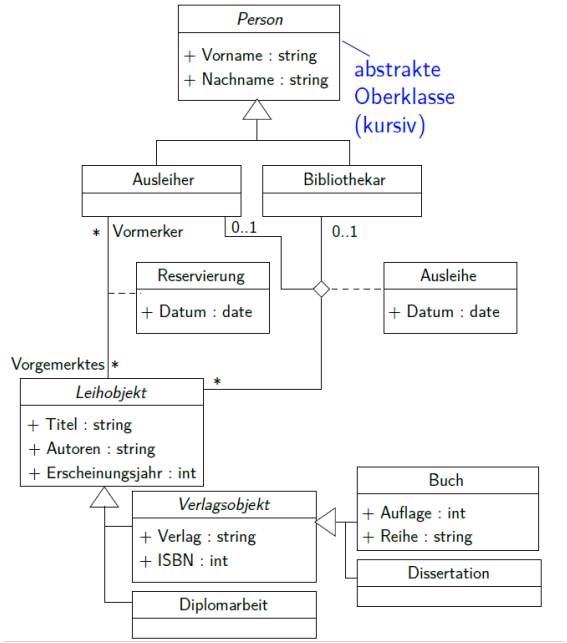


Abbildung 8 – Infomodell VODM zu Release 2

Strukturierung großer UML-Modelle

- Paketdiagramme
  - Zusammenfassung von Modellelementen zu Gruppen
  - Modellelemente sind meist Klassen, kann aber auch für Anwendungsfälle verwendet werden
  - Programmiersprachen-äquivalent: Namespaces oder Packages



Resultat der statischen Modellierung

- UML-Diagramme
  - Objektdiagramme
  - Klassendiagramme
- Zusammen beschreiben diese das statische Datenmodell, also statische Konzepte und ihre Beziehungen zueinander