Software Engineering

Objektorientiertes Design

Prof. Dr. Peter Jüttner

Hochschule Deggendorf

Inhalt

Objektorientiertes Design (OOD)

- Ziel des Objektorientierten Designs (OOD) ist es, den bereits bestehenden Grobentwurf (Architektur gemäß dem OOA-Modell) durch
  - Vervollständigung der Klassendiagramme (insbes. der Methoden)
  - Vervollständigung der Interaktionsdiagramme und
  - bei Bedarf die Modellierung des internen Verhaltens einer Klasse

  zum Feinentwurf zu detaillieren.
- Das OOD lässt sich (ähnlich wie die OOA) in
  - eine statische Modellierungsphase und
  - eine dynamische Modellierungsphase unterteilen.

  Beide Phasen erfolgen meistens iterativ und ergänzen sich gegenseitig.
- Zusätzlich wird in der Architekturmodellierungs-Phase die logische und physikalische Struktur des Systems festgelegt.

Objektorientierte Architekturmodellierung

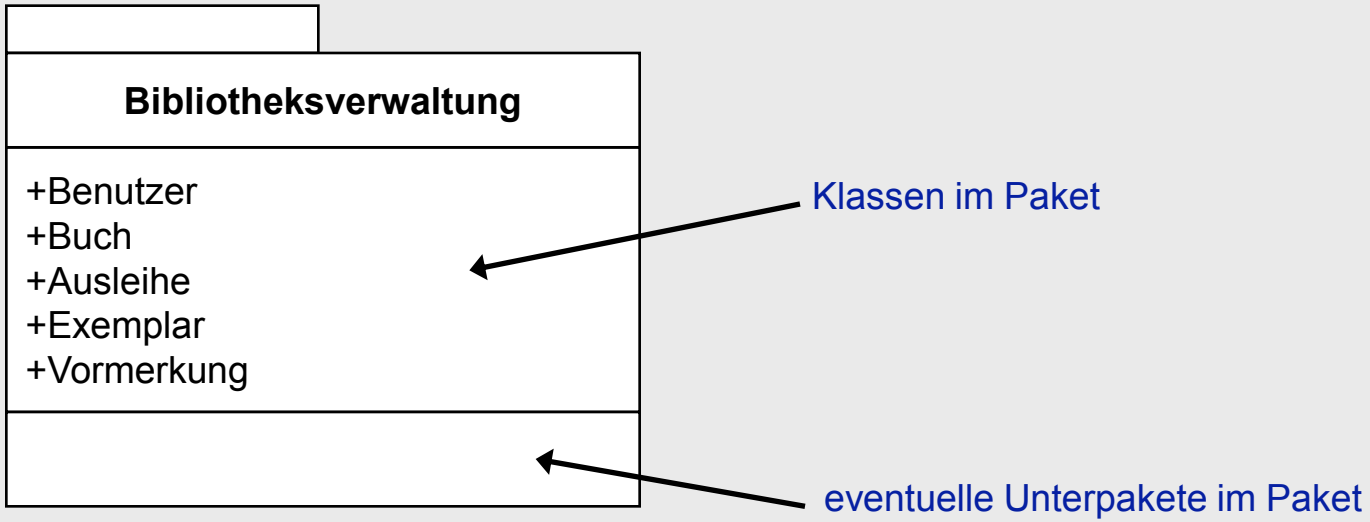
- Bisher fehlt noch eine Modularisierung des Systems, um
  - festgelegte Aufgaben Modulen (Komponenten) zuordnen zu können,
  - die Skalierbarkeit (durch Entfernen und Hinzufügen von Modulen) des Systems zu unterstützen und
  - die Wartbarkeit zu verbessern (indem einzelne Module ausgetauscht bzw. geändert werden können).
- In der Objekt-orientierten Architekturmodellierung unterscheidet man zwischen zwei Teilaspekten:
  - Logische Systemarchitektur: welche Klassen werden zu Paketen und Komponenten zusammengefasst
  - Physikalische Systemarchitektur: wie werden die Komponenten auf mehrere Recheneinheiten verteilt und wie kommunizieren die Komponenten untereinander

Architekturmodellierung: Diagramme

- Logische Sicht:
  - Paketdiagramme
    - zeigen die Verteilung der Klassen auf Pakete, d. h. auf Mengen logisch zusammenhängender Klassen mit eindeutiger Namensgebung (Namensräume)
  - Komponentendiagramme
    - zeigen die Verteilung der Pakete auf in sich abgeschlossene Komponenten
- Physikalische Sicht:
  - Einsatzdiagramme
    - zeigen welche Komponenten auf welchen Rechnern installiert werden

Paketdiagramm

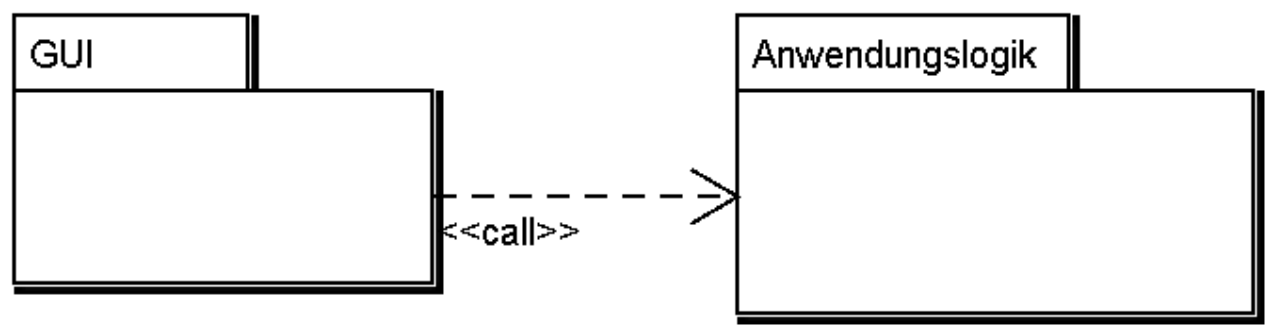
- Paketdiagramme zeigen,
  - welche Klassen bzw. Unterpakete sich innerhalb eines Pakets befinden und
  - Beziehung der Pakete zueinander.
- Beispiel: Bibliotheksverwaltung



Beispiel: Paket Bibliotheksverwaltung

- Die Beziehungen zwischen den Paketen werden als sogenannte Abhängigkeiten (gestrichelter Pfeil) notiert.

- Beispiel: Die Benutzeroberfläche ruft die Anwendungslogik auf.

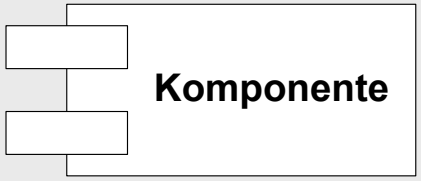


Komponentendiagramm

- Ein Komponentendiagramm zeigt
  - Komponenten sowie
  - deren Beziehungen untereinander.
- Komponenten werden durch eine oder mehrere Klassen realisiert.
- Komponenten
  - benötigen Schnittstellen und
  - bieten Schnittstellen an

Komponentendiagramm

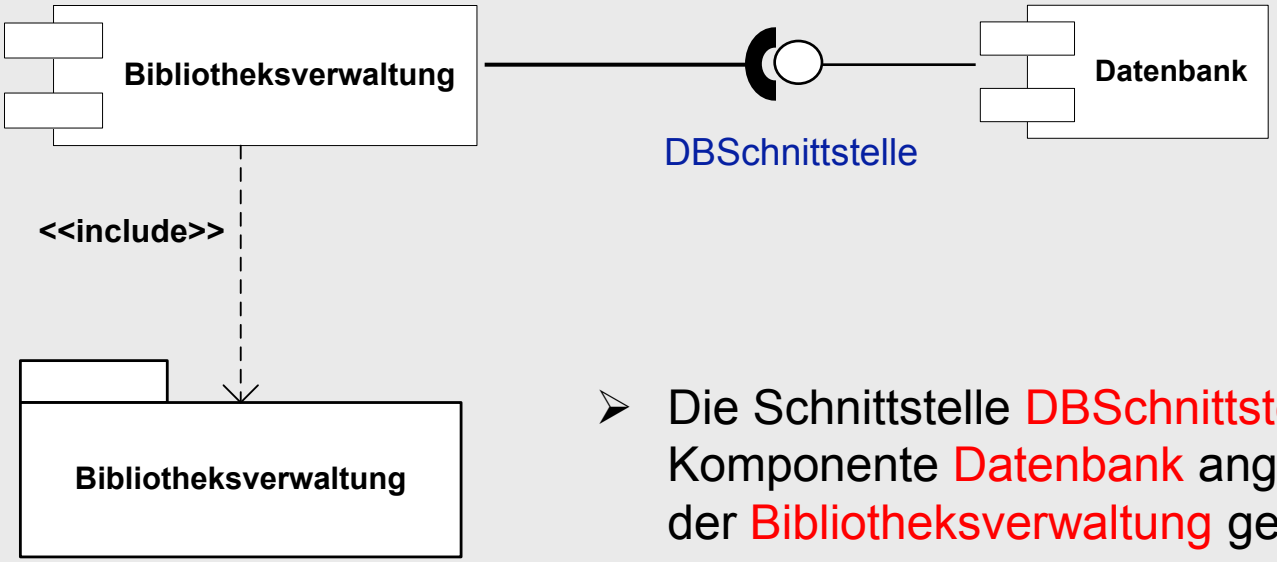
- Komponenten werden wie folgt dargestellt:



- Schnittstellen werden als Kreis dargestellt:

- Zusätzlich können Pakete in einem Diagramm auftauchen
- Beziehungen werden durch Abhängigkeiten ausgedrückt

Beispiel: Bibliotheksverwaltung



- Die Schnittstelle DBSchnittstelle wird von der Komponente Datenbank angeboten und von der Bibliotheksverwaltung genutzt.

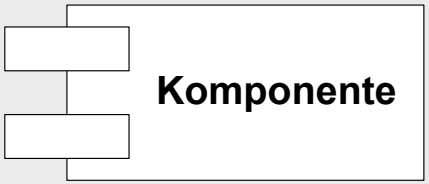
Einsatzdiagramm

- Ein Einsatzdiagramm zeigt
  - die Knoten (Rechner) eines Systems,
  - welche Komponenteninstanzen auf diesen installiert sind und
  - die Beziehungen zwischen den Knoten.
- Das Einsatzdiagramm wird auch Verteilungsdiagramm (Deployment) genannt
- Ziel: statische Sicht auf installiertes System

Einsatzdiagramm

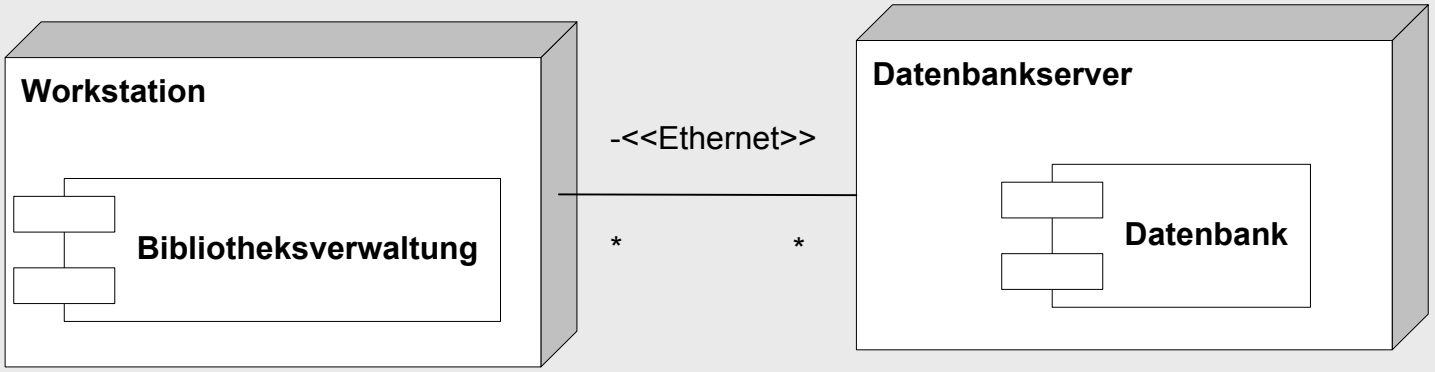
- Ein Knoten (Instanz):

- Komponenteninstanz:



- Beziehungen (z. B. Kommunikationskanäle) können mit Verbindungen (Instanzen von Assoziationen) dargestellt werden

Beispiel: Bibliotheksverwaltung



Inhalt

OOD: Statische Modellierung

Ziel der statischen Modellierung im OOD ist es, die in der OOA entwickelten UML-Klassendiagramme zu präzisieren.

Insbesondere werden

- die bisher ausgewählten Klassen evtl. durch weitere Klassen ergänzt
- die identifizierten Klassen um Operationen und Attribute ergänzt und
- die z.T. bereits ermittelten Assoziationen präzisiert

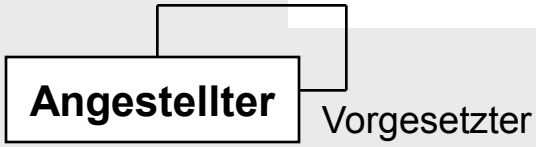
Eigenschaften einer Assoziation

- Name
  - Eindeutige Bezeichnung der Assoziation
- Leserichtung
  - Wenn der Name der Assoziation die Leserichtung nicht eindeutig erkennen lässt, so wird die Leserichtung durch ein schwarzes Dreieck spezifiziert.
  - Eine Assoziation mit Leserichtung von Klasse A nach Klasse B wird als „Klasse A <Name der Assoziation> Klasse B“ gelesen.
- Benutzungsrichtung
  - Wird die Assoziation nur in einer Richtung benutzt, so wird diese durch einen Pfeil gekennzeichnet.
  - Eine Assoziation mit Benutzungsrichtung von Klasse A nach Klasse B bedeutet, dass Klasse A ein Attribut enthält, das eine Referenz auf ein Objekt der Klasse B darstellt.

Beispiel für Leserichtung und gerichtete Assoziation

- Jede Rechnung enthält Rechnungsposten (Leserichtung hier von rechts nach links).
- Beim Verarbeiten einer Rechnung muss auf deren Rechnungsposten zugegriffen werden können. Dies wird durch die gerichtete Assoziation modelliert.

Eigenschaften einer Assoziation (Forts.)



- Rollennamen
  - An beiden Enden einer Assoziation kann zusätzlich die Bedeutung („Rolle“) der Klasse in Bezug auf die genannte Assoziation annotiert werden
  - Die Rollennamen sind für rekursive Assoziationen Pflicht,
  - ansonsten sollten sie eingesetzt werden, um die Verständlichkeit zu erhöhen
- Jede Rolle einer Assoziation hat eine Multiplizität
  - Sie gibt die erlaubte Anzahl an Objekten (Kardinalität) an, die sich in einer vorgegebenen Rolle an einer Assoziation beteiligen können
  - Wird spezifiziert durch
    - einen zugelassenen Zahlenbereich: `Untergrenze..Obergrenze`
    - wobei das *-Symbol eine unbeschränkte Obergrenze symbolisiert
  - Wird keine Multiplizität explizit angegeben, beträgt diese 1..1
  - Abkürzend kann das *-Symbol anstelle von 0..* und 1 anstelle von 1..1 verwendet werden

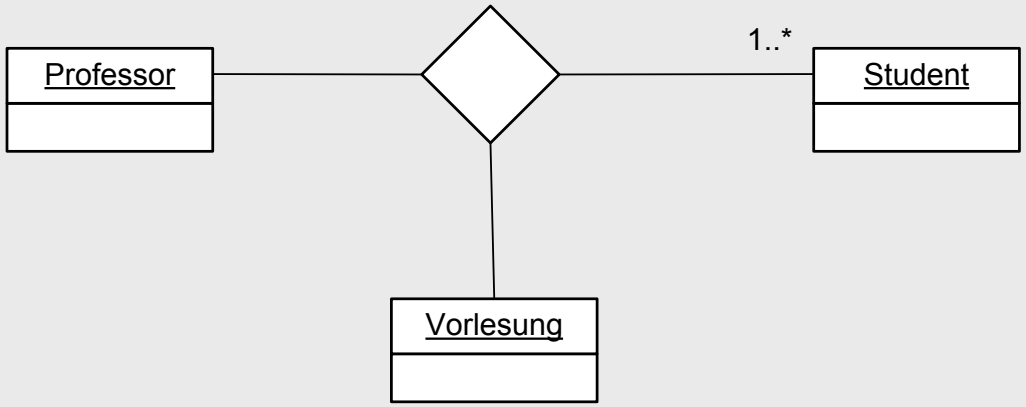
Beispiel für Rollenname und Multiplizität

- Zu den Rollennamen:
  - Ein Benutzer kann in zwei Rollen mit einer Teilnehmergruppe in Beziehung stehen: Als Mitglied und als Besitzer.
- Zu den Multiplizitäten:
  - Ein Benutzer kann beliebig viele Teilnehmergruppen verwalten.
  - Eine Teilnehmergruppe wird von genau einem Benutzer verwaltet.
  - Ein Benutzer kann Mitglied beliebig vieler Teilnehmergruppen sein.
  - Eine Teilnehmergruppe besteht aus mindestens einem Mitglied.

Mehrgliedrige Assoziationen

- Neben Zweierbeziehungen gibt es noch mehrgliedrige Assoziationen, an denen drei oder mehr Klassen beteiligt sind.
- Beispiel für eine ternäre (dreigliedrige oder 3-stellige) Assoziation:

Studenten hören Vorlesung bei einem Professor

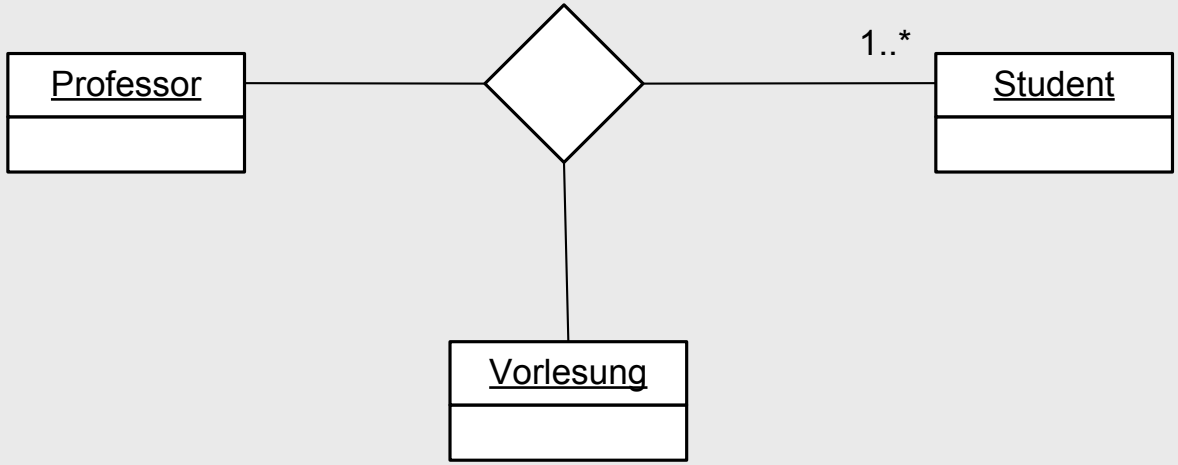


Mehrgliedrige Assoziationen

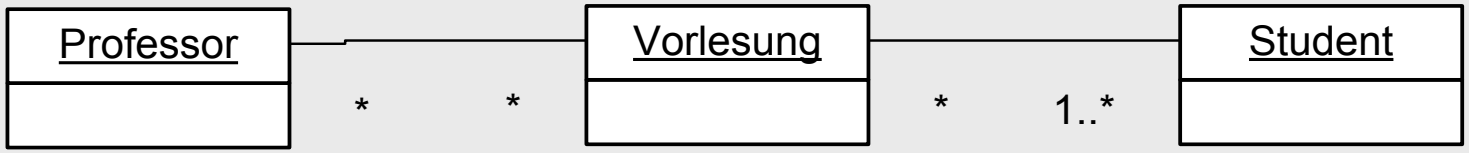
- Mehrgliedrige Assoziationen können in den meisten Programmiersprachen nicht direkt realisiert werden.

- Es sollte daher stets geprüft werden, ob die mehrgliedrige Assoziationen ersetzt werden können:
  - durch mehrere binäre Assoziationen oder
  - durch eine Assoziationsklasse

Mehrere binäre Assoziationen

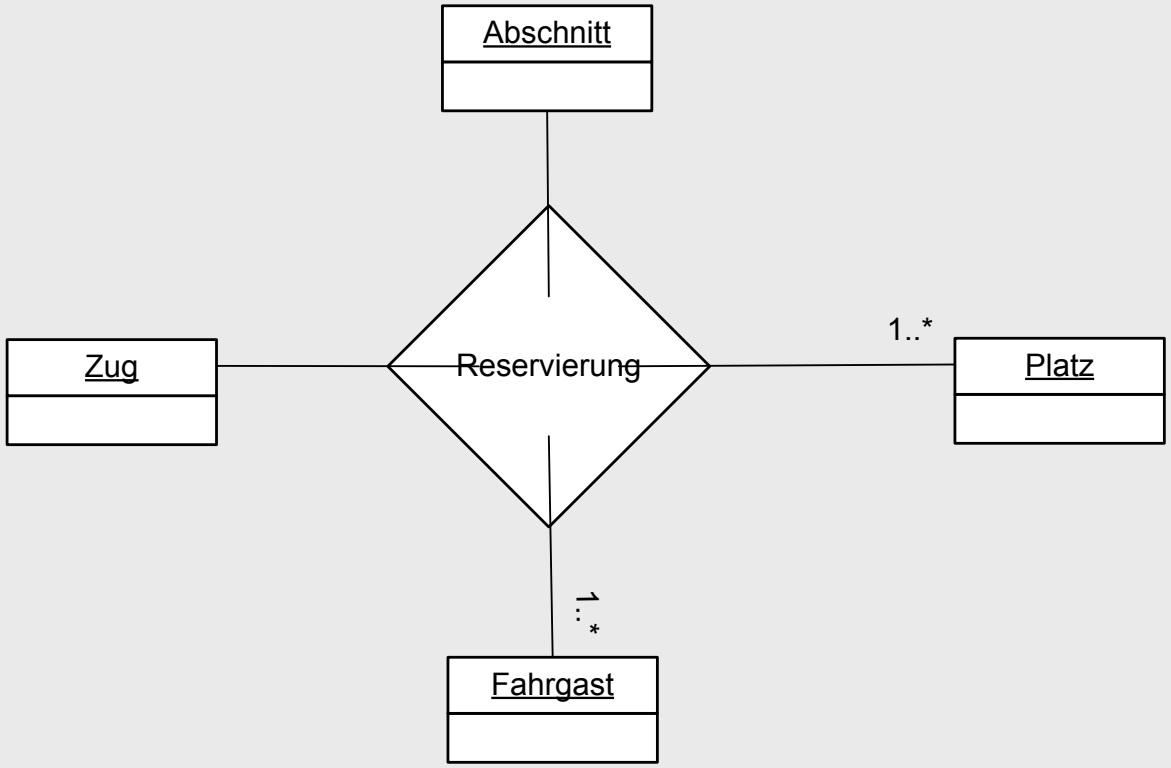


- Diese Assoziation kann auch mittels zweier binärer Assoziationen ausgedrückt werden:



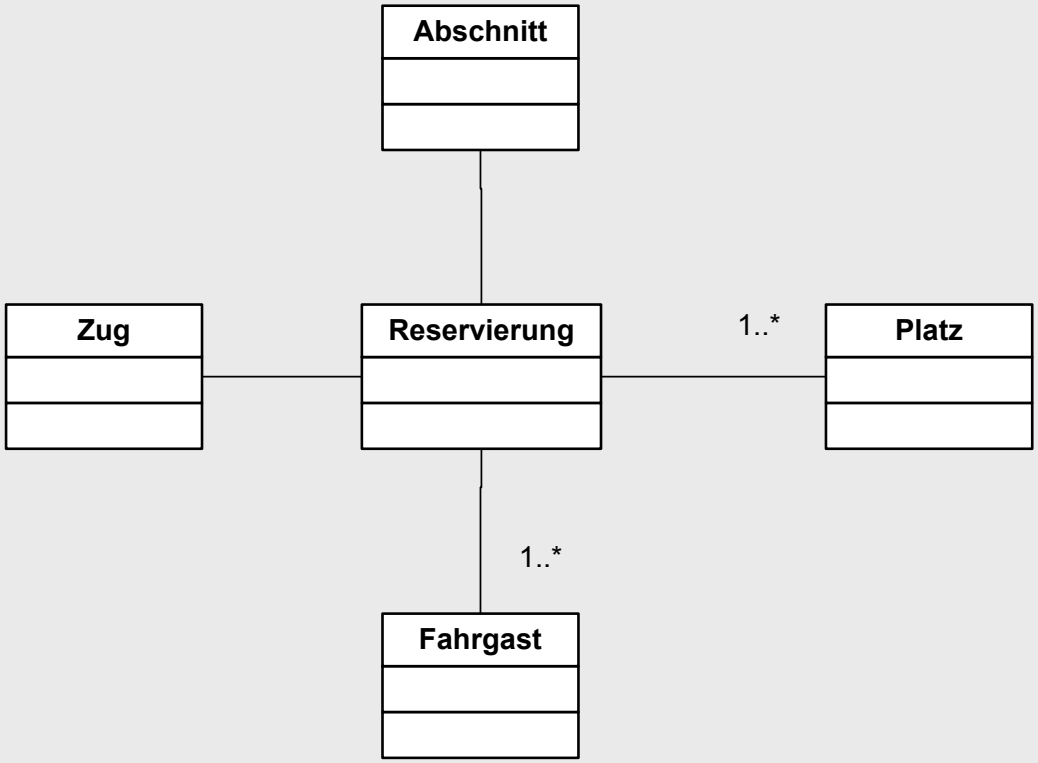
Beispiel für mehrgliedrige Assoziation

- Beispiel für eine viergliedrige Assoziation: *Fahrgäste reservieren Plätze* für (Fahr-) *Abschnitte* von *Zügen.*



Assoziationsklasse

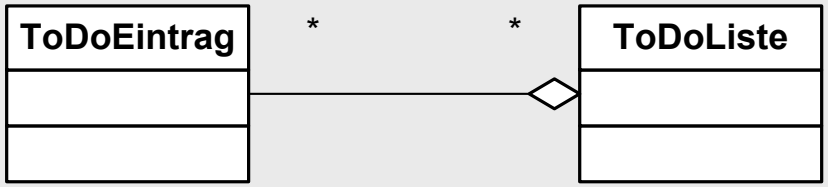
- Die viergliedrige Assoziation **`Reservierung`** kann durch eine Klasse des gleichen Namens (sog. Assoziationsklasse) ersetzt werden:



Aggregation

- Die Aggregation ist ein Spezialfall der Assoziation:
  - asymmetrische Beziehung zwischen nichtgleichwertigen Partnern, die unabhängig voneinander existieren können
  - wobei ein Objekt der einen der beiden Klassen (der sog. Aggregatklasse) eine Menge von Objekten der anderen Klasse darstellt.
- Sie wird oft auch als "Teile-Ganzes-Beziehung" oder "Besteht-aus-Beziehung" bezeichnet wird.
- Aggregationen werden in UML durch eine Raute dargestellt, die jenes Ende der Assoziationskante markiert, das zur Aggregatklasse, also "zum Ganzen" hinführt.

Beispiele für Aggregation: Listen, etc.

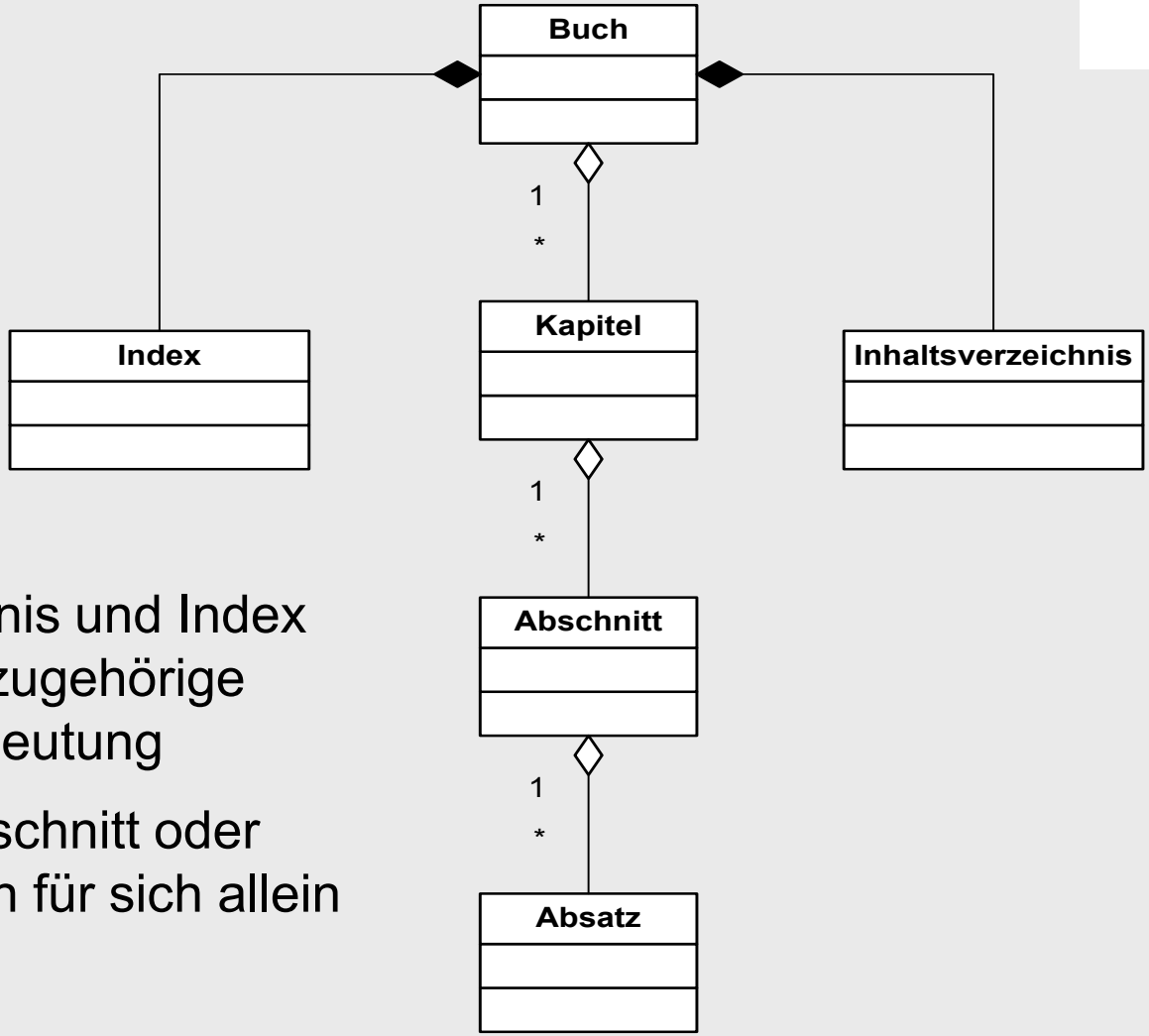


- Eine Liste besteht aus einer beliebig großen (evtl. auch leeren) Menge von Einträgen.
- Die Klasse ToDoListe ist somit die Aggregatklasse.
- Weitere Beispiele:
  - Ein PKW hat die Teile Motor, Getriebe, Räder, Karosserie,
  - Ein Kapitel besteht aus Unterkapiteln
  - Ein Unternehmen besteht aus Abteilungen
  - Eine Workstation besteht aus Monitor, Rechnereinheit, Tastatur, Maus, ...

Komposition

- Ein Sonderfall der Aggregation ist die Komposition.
- Eine Komposition liegt dann vor, wenn das Einzelteil vom Aggregat existenzabhängig ist — also nicht ohne das Aggregat existieren kann.
- Ein Teil kann immer nur von *einem* Aggregat existenzabhängig sein.
- Kompositionen werden in UML durch eine gefüllte Raute dargestellt, die jenes Ende der Assoziationskante markiert, das zur Aggregatklasse, also "zum Ganzen" hinführt.

Beispiel für Aggregation und Komposition



- Inhaltsverzeichnis und Index sind ohne das zugehörige Buch ohne Bedeutung
- Ein Kapitel, Abschnitt oder Absatz hat auch für sich allein einen Inhalt

Übung

Übungsaufgabe Objektorientierte Design UML: Assoziationen, Aggregationen

Beschreibung:
In UML werden Beziehungen zwischen Klassen durch "Assoziationen" dargestellt.

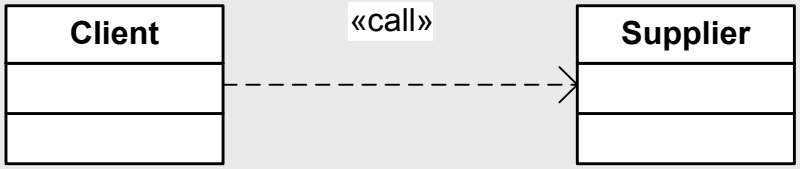
Aufgabe:
Sie befinden sich auf einer Cocktailparty. Stellen Sie Assoziationen zwischen den Klassen "Person", “Zutat“, "Frucht" und "Cocktail" dar mit Bezeichnung der Rollen und der Multiplizitäten.
Stellen Sie eine Aggregation zwischen den Klassen "Cocktail" und "Zutat" her.
Erläutern Sie an Hand dieses Beispiels den Unterschied zwischen Assoziation und Aggregation.
Warum ist die "Frucht" keine Aggregation zum "Cocktail" ?

Tipp:
Eine "Rolle" kann auch eine spezielle Bedeutung einer Klasse für die assoziierende Klasse sein.

Zeit:
15 Minuten, arbeiten Sie ggf. zusammen mit einem Partner

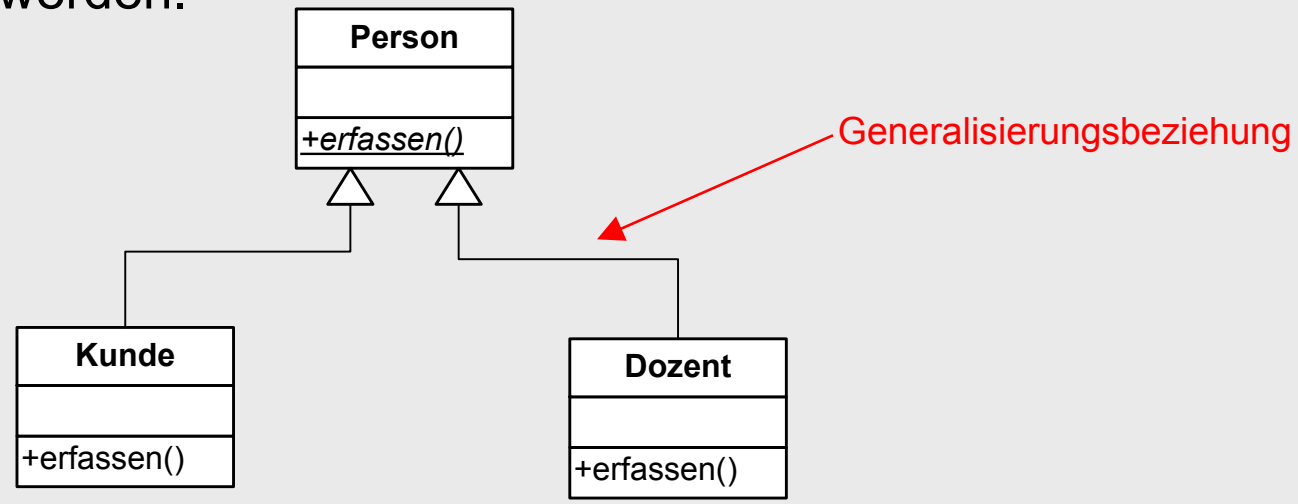
Abhängigkeiten

- Eine Abhängigkeit zeigt an, dass eine Klasse eine andere benötigt.
- Abhängigkeiten sind immer gerichtet.
- Abhängigkeiten werden in der UML als gestrichelter Pfeil mit einer offenen Spitze dargestellt.
- Um die Bedeutung der Abhängigkeit näher zu spezifizieren, können sogenannte Stereotypen textuell an der betroffenen Abhängigkeitskante in doppelten spitzen Klammern annotiert werden.



Abstrakte Klasse

- Eine abstrakte Klasse implementiert nur einen Teil der Operationen, die sie deklariert. Somit kann eine abstrakte Klasse nicht instanziiert werden.
- Abstrakte Klassen und nichtimplementierte Operationen werden im Klassendiagramm kursiv dargestellt.
- Die restlichen Operationen müssen über eine Unterklasse implementiert werden.



Übung

Übungsaufgabe Objektorientierte Design Abstrakte Klassen

Beschreibung:
In UML werden Beziehungen zwischen Klassen durch "Assoziationen" dargestellt.

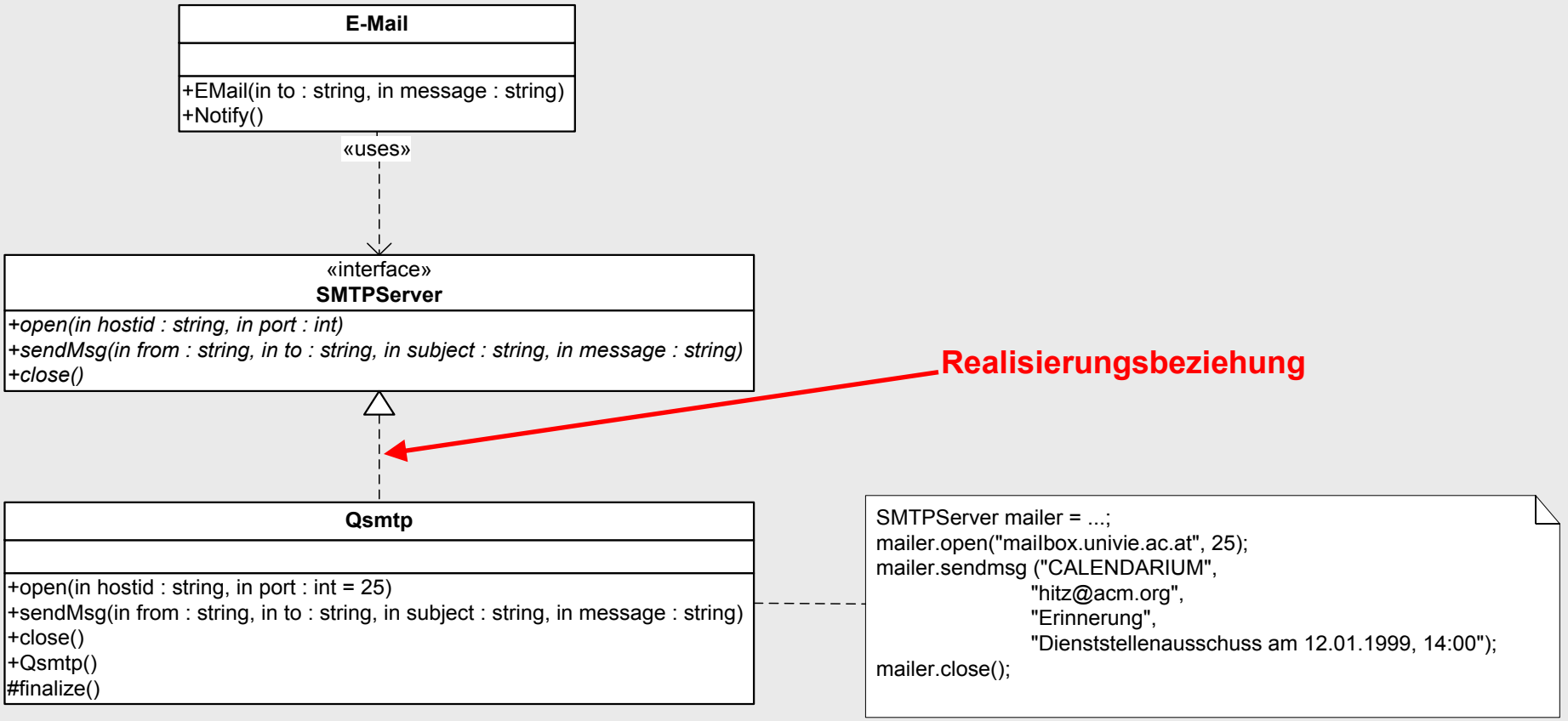
Aufgabe:
Erstellen Sie ein UML-Diagramm für die Klassen "BankAccount", "CheckingBankAccount" und SavingsBankAccount" mit den wesentlichen Methoden.

Zeit:
15 Minuten, arbeiten Sie ggf. zusammen mit einem Partner

Interface

- Erst im OOD sind die Klassen um noch fehlende Operationen bzw. Details bereits identifizierter Operationen ergänzt worden; damit können nun die Schnittstellen im Detail beschrieben werden.
- Zur detaillierten Beschreibung von Schnittstellen werden typischerweise besondere abstrakte Klassen (sog. Interfaces) verwendet, die durch den Stereotyp «interface» gekennzeichnet werden.
- Ein Interface ist eine abstrakte Klasse (also nicht instanziierbar) mit
  - keiner implementierten Methode und
  - keinen Attributen.
- Sie enthält lediglich eine Liste von Operationsdeklarationen (Signatur).

Beispiel für Interface: Versenden von E-Mails

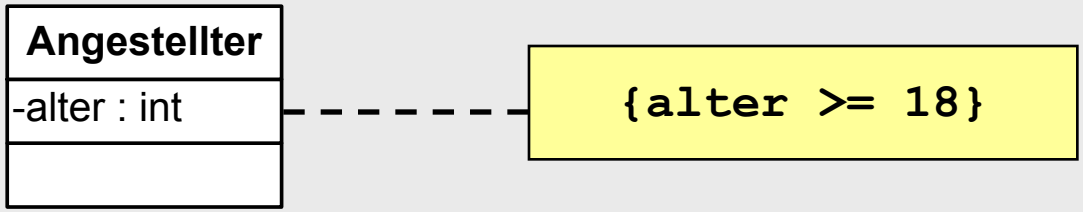


Zusicherungen

- Mittels OCL (Object Contraint Language) können Zusicherungen über Klassen formuliert werden.
- Diese können als Kommentare in geschweiften Klammern in den Diagrammen angegeben werden:

```
{context Angestellter inv:
        alter >= 18}
```

- Wenn der Kontext klar ist (etwa durch Zuordnung eines Kommentarfeldes), kann die explizite Angabe des Kontextes im OCL-Ausdruck weggelassen werden:



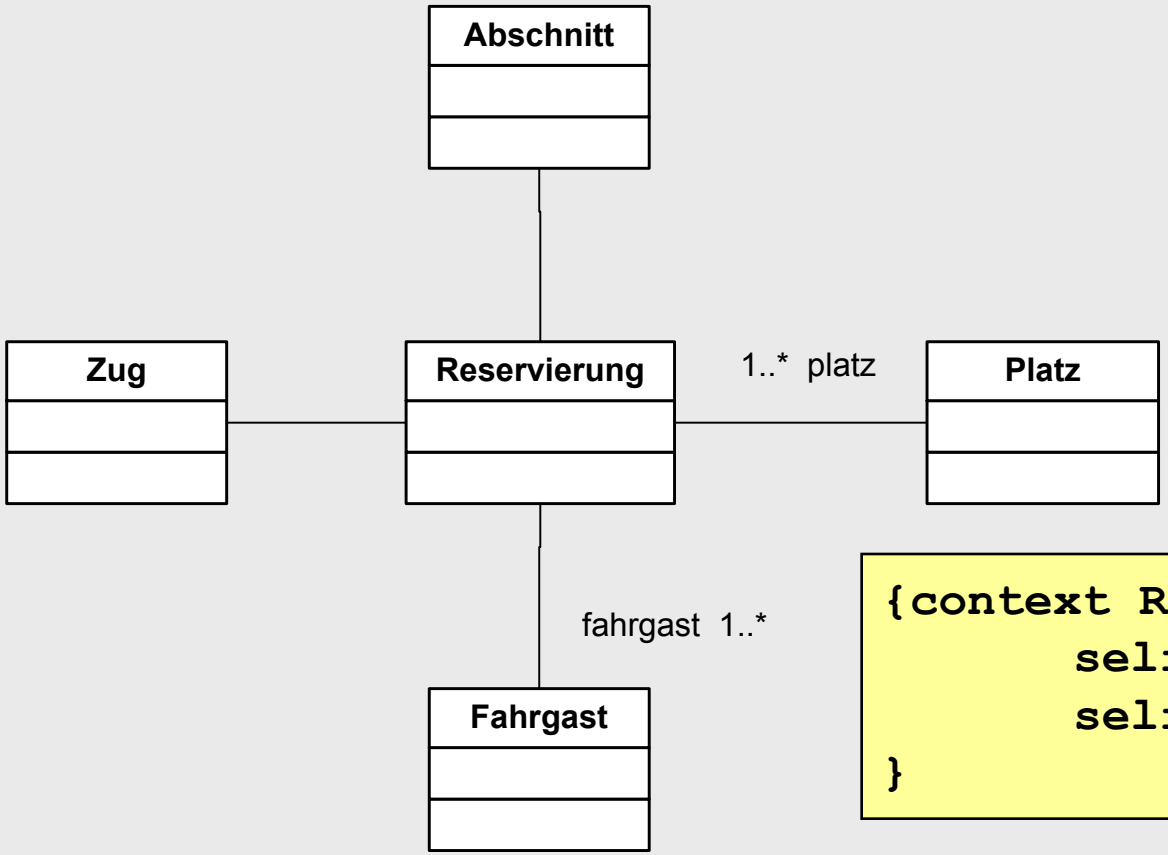
Beispiel für Zusicherungen

```
{context Projekt inv:
      projektmitglied->includes (leiter) }
```

- Der Projektleiter muss zur Menge der Projektmitglieder gehören.

Beispiel für Zusicherungen

- Die Anzahl der reservierten Plätze muss mit der Anzahl der reservierenden Fahrgäste übereinstimmen




```
{context Reservierung inv:
        self.platz->size () ==
        self.fahrgast->size()
}
```

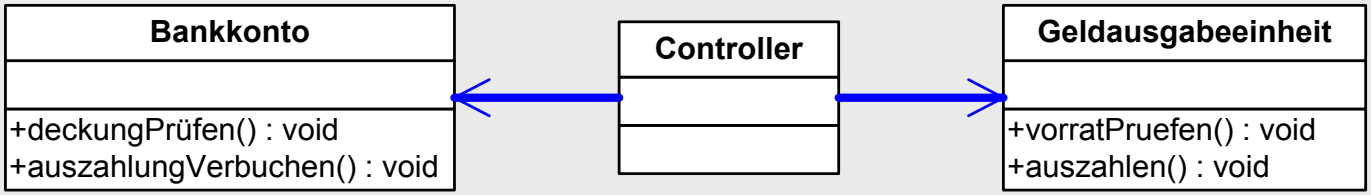
OOD: Detaillierte Klassendiagramme

- Mit vorgestellten Notationselemente lassen sich nun die noch unvollständigen Klassendiagrammentwürfe aus der OOA verfeinern.

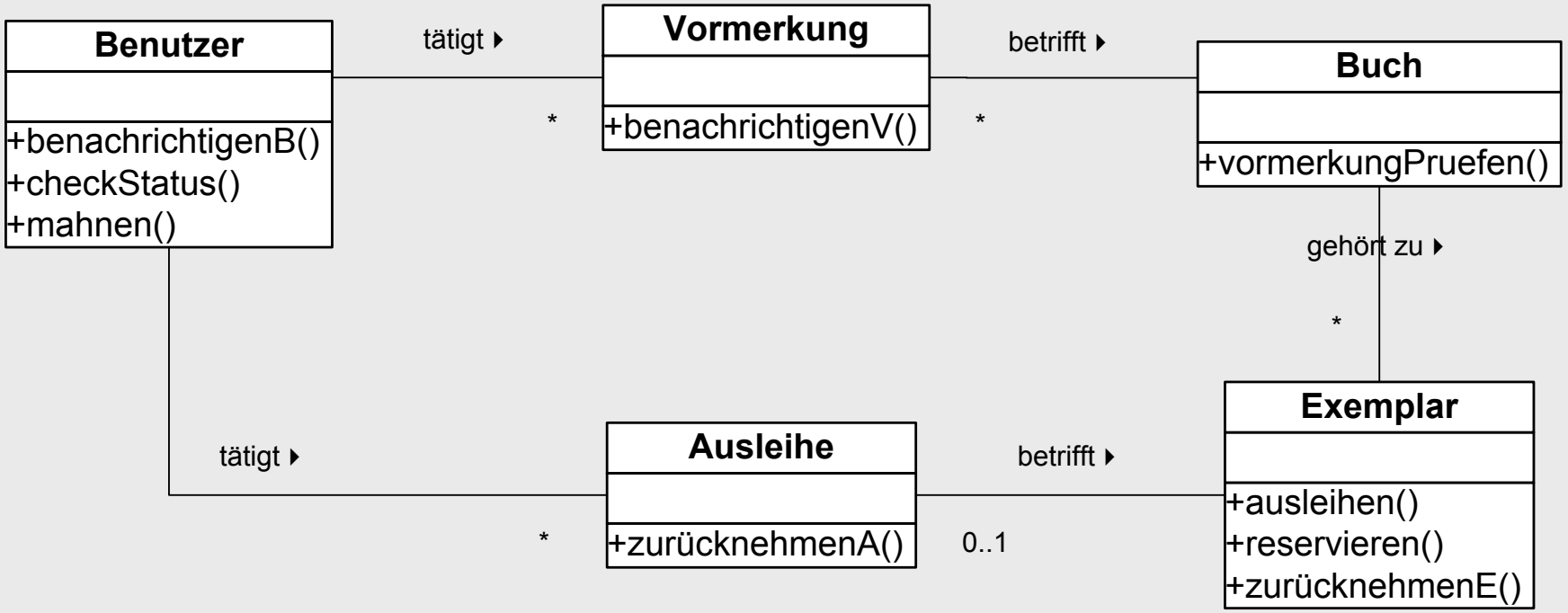
- Wo nötig, wird dieser Prozess von der dynamischen Modellierung unterstützt.

Bsp. für vervollständigtes Klassendiagramm: Geldautomat

- Das in der OOA ermittelte unvollständige Klassendiagramm wird durch Angabe der Methoden vervollständigt.



Bsp. für vervollständigtes Klassendiagramm: Bibliothek



Übung

Übungsaufgabe Objektorientierte Design Klassendiagramm für eine "Tagung"

Aufgabe:
Identifizieren Sie anhand der folgenden Beschreibung Klassen, Attribute, Operationen und Assoziationen und zeichnet sie in ein Klassendiagramm ein.
Eine Tagung (z.B. Softwaretechnik-Tagung in Hamburg) ist zu organisieren. Für jeden Teilnehmer der Tagung werden Name, Adresse und der Status (Student, Mitglied, Nichtmitglied) gespeichert. Jeder Teilnehmer kann sich für mehrere halbtägige Tutorien, die zusätzlich zum normalen Tagungsprogramm angeboten werden, anmelden. Für jedes Tutorium werden dessen Nummer, Bezeichnung sowie das Datum gespeichert. Alle Tutorien kosten gleich viel.
Damit ein Tutorium stattfindet, müssen mind. 10 Anmeldungen vorliegen. Jedes Tutorium wird von genau einem Referenten angeboten. Für jeden Referenten werden dessen Name und Firma gespeichert. Ein Referent kann sich für ein oder mehrere Tutorien - anderer Referenten - anmelden und kann bei diesen kostenlos zuhören. Diese Anmeldungen zählen bei der Ermittlung der Mindestanmeldungen nicht mit.
Ein Teilnehmer kann nicht gleichzeitig Referent sein. Ein Referent kann mehrere Tutorien anbieten. An einem Tutorium können mehrere Referenten kostenlos teilnehmen. Ein Teilnehmer kann sich in der Tagungsanmeldung auch für einige Rahmenprogramme (z.B. Besuch eines Musicals) eintragen lassen. Für jedes Rahmenprogramm werden dessen Bezeichnung, das Datum, die Zeit, der Ort und die Kosten gespeichert.
Zeit:
45 Minuten, arbeiten Sie ggf. zusammen mit einem Partner

Zum Schluss dieses Abschnitts …

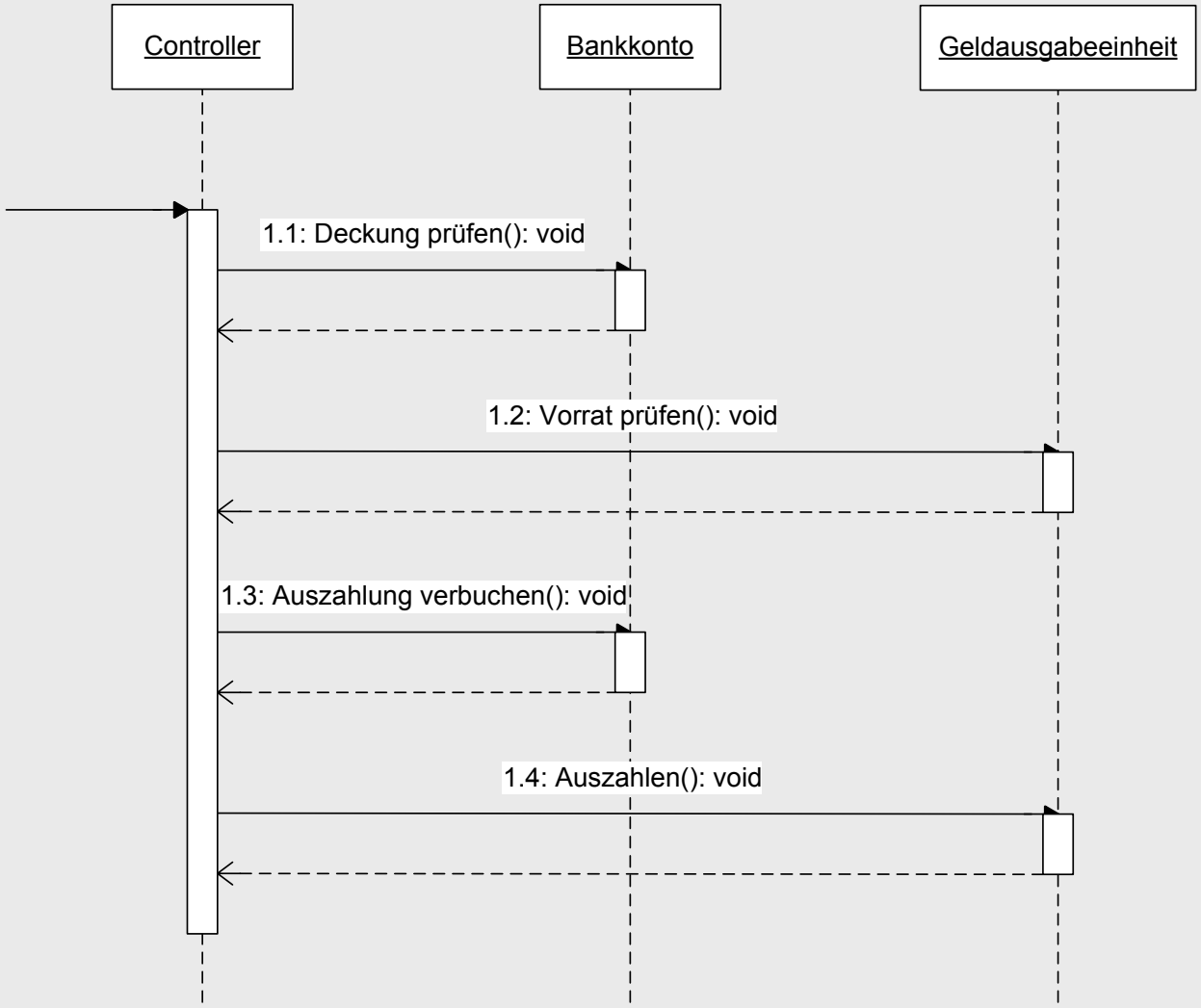
Noch Fragen ??

Inhalt

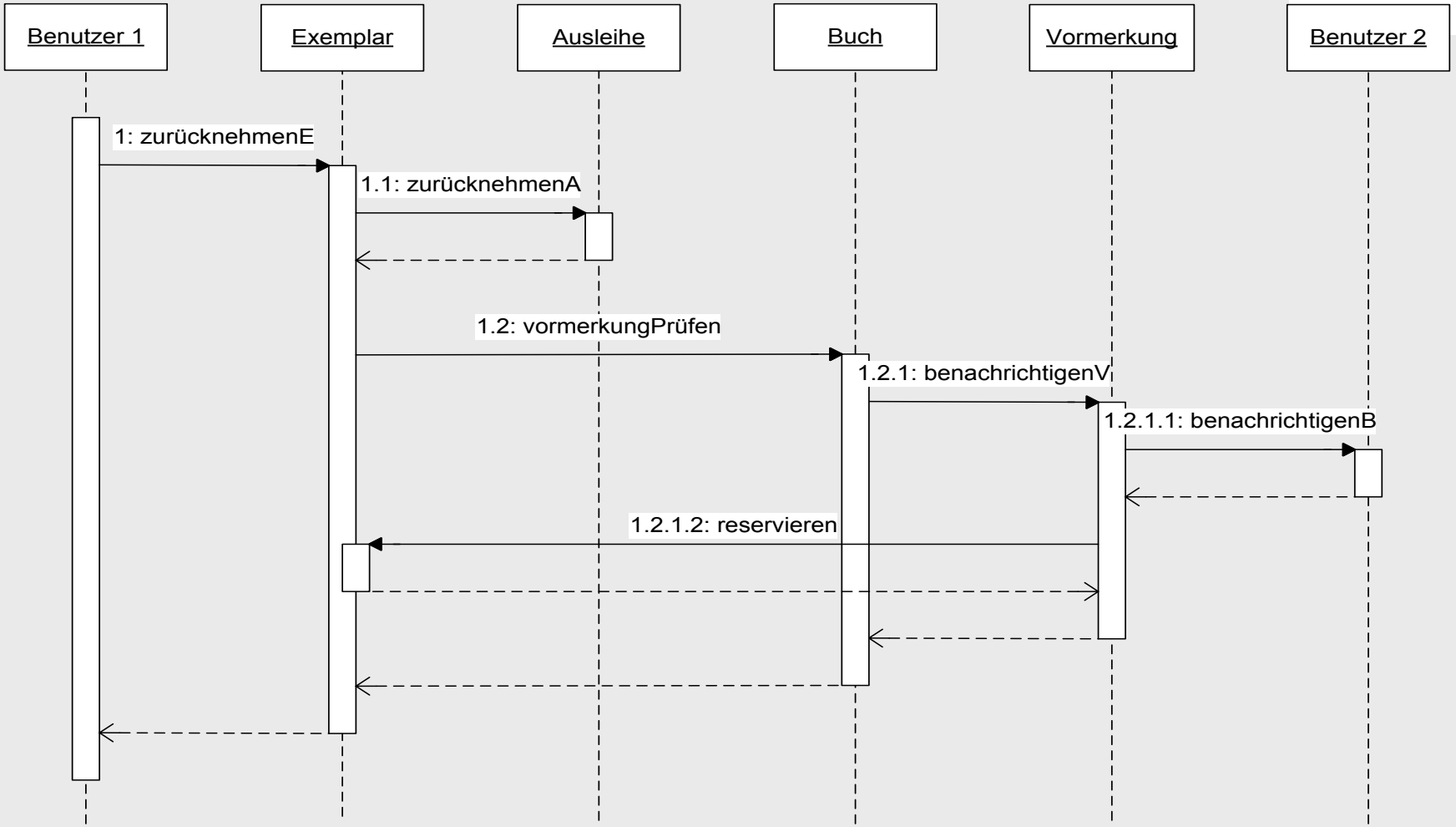
OOD: Dynamische Modellierung

- Die bei der OOA entstandenen, zum Teil noch unvollständigen Interaktionsdiagramme, werden nun weiter verfeinert:
  - Die Verfeinerung der Interaktionsdiagramme führt zur Präzisierung der Operationen bzw. Identifizierung weiterer Operationen in den Klassen.
  - Zu den auftretenden Interaktionen werden nun die bisherigen z.T. noch textuellen Angaben durch einzelne Operationsaufrufe detailliert.

Beispiel für verfeinertes Sequenzdiagramm: Geldautomat



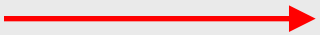
Beispiel für Sequenzdiagramm: Bibliothek



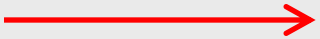
Nachrichtenübermittlung

- Bei der Modellierung von Interaktionsdiagrammen (Sequenz- und Kollaborationsdiagramm) unterscheidet man zwischen

- synchronen Nachrichten,
  die im Folgenden durch einen Pfeil mit ausgefüllter Pfeilspitze dargestellt werden, und

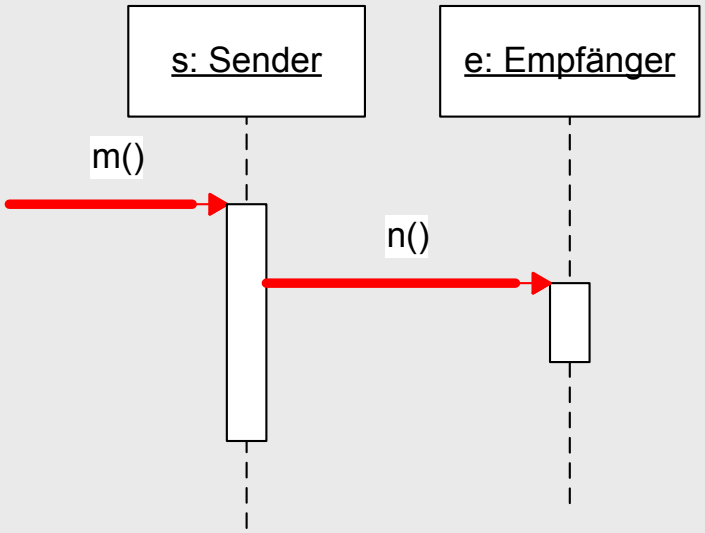


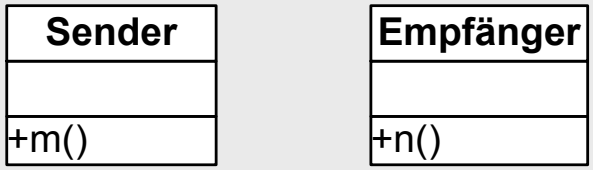
- asynchronen Nachrichten,
  die im Folgenden durch einen Pfeil mit offener Pfeilspitze dargestellt werden.



Synchrone Nachrichtenübermittlung

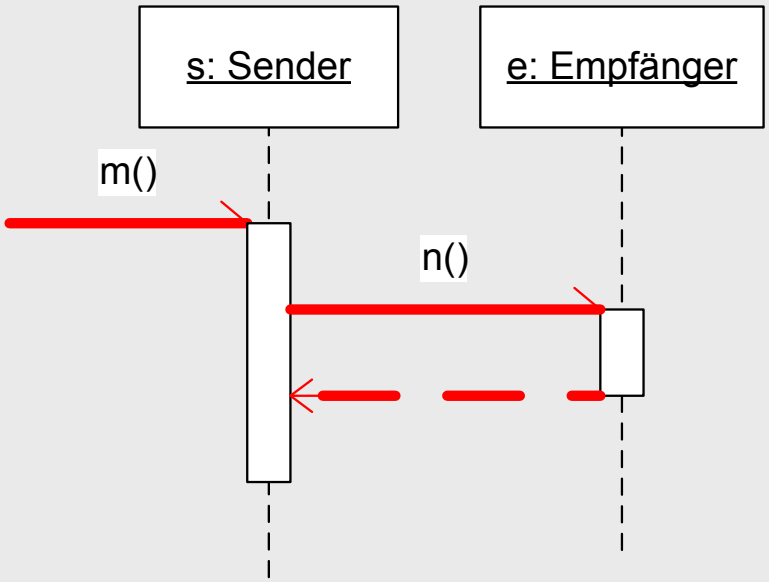
- Synchrone Nachrichten
  - Blockieren den Sender solange, bis die Nachricht vom Empfänger vollständig verarbeitet ist (und das Ergebnis vorliegt).
  - Die Implementierung erfolgt durch **Methodenaufrufe**.
  - Daraus ergibt sich, dass jede synchrone Nachricht eines Interaktions-diagramms als Methode des Empfängerobjekts verfügbar sein muss.





Asynchrone Nachrichtenübermittlung

- Asynchrone Nachrichten
  - werden verschickt, ohne auf das Ende der Verarbeitung durch den Empfänger zu warten.
  - Die Übermittlung des Ergebnisses erfolgt ggf. wiederum durch eine asynchrone Nachricht in der Gegenrichtung.
  - Die Implementierung kann beispielsweise über Threads erfolgen.



Asynchrone vs. synchrone Nachrichtenübermittlung

- Synchrone Nachrichtenübermittlung
  - Dies ist der Normalfall bei der Modellierung von Objekt-Interaktionen.
  - Die synchrone Nachrichtenübermittlung entspricht dem natürlichen zeitlichen Ablauf bei Methodenaufrufen.
- Asynchrone Nachrichtenübermittlung
  - Wird nur bei besonderen Anforderungen verwendet.
  - Mögliche Gründe sind Zeitgewinn durch Parallelisierung von Abläufen oder Irrelevanz einer Rückantwort.
  - Beispiele:
    - Interaktion mit Hardware
      (etwa beim Öffnen der Lade eines CD-Spielers wird lediglich das Signal zum Öffnen versendet, nicht aber auf das Ende der Aktion gewartet)
    - Initiieren nebenläufiger Berechnungen

Modellierung durch Zustandsdiagramm

- Bei komplexem Verhalten einer Klasse, vor allem, wenn das Verhalten der Methoden vom aktuellen Zustand der Klasse abhängt, erlaubt das Zustandsdiagramm eine eindeutige Darstellung der möglichen Zustandsübergänge in Abhängigkeit von den Methodenaufrufen.

- Erst mit Hilfe dieser Information lässt sich die Realisierung der einzelnen Methoden so weit verfeinern, dass sich die Codierung der Methoden direkt ableiten lässt.

Zustandsdiagramm

- Ein Zustandsdiagramm ist die graphische Repräsentation eines Zustandsautomaten.
- Es beschreibt das dynamische Verhalten von Objekten anhand von ereignisgesteuerten Übergängen (Transitionen) zwischen diskreten Zuständen.
- Damit erhält man eine programmiersprachenunabhängige Verhaltensbeschreibung, die anschließend in der Implementierungsphase in die Zielsprache umzusetzen ist.

Zustände

- Ein Startzustand wird als ausgefüllter Kreis dargestellt:

- Ein Zustand wird als Rechteck mit abgerundeten Ecken dargestellt:

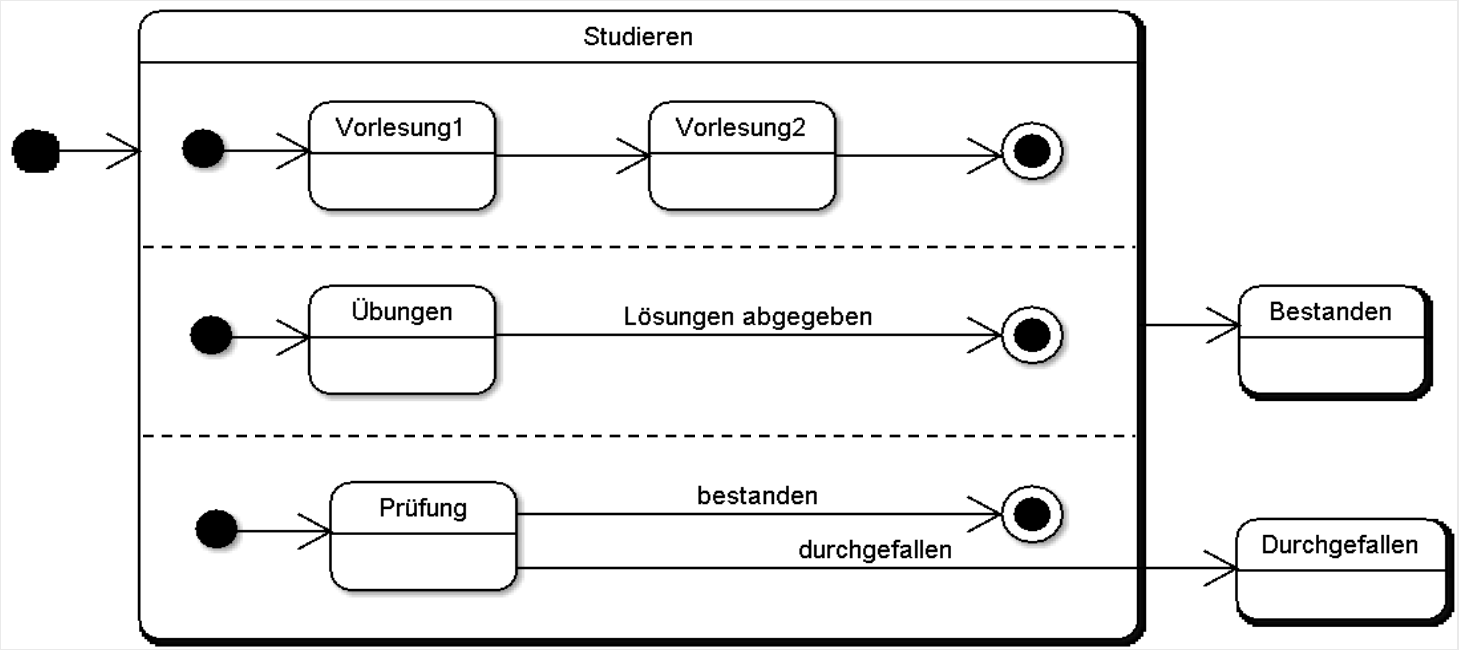
- Ein Endzustand wird als ausgefüllter Kreis mit einem weiteren äußeren Kreis dargestellt:

Transitionen

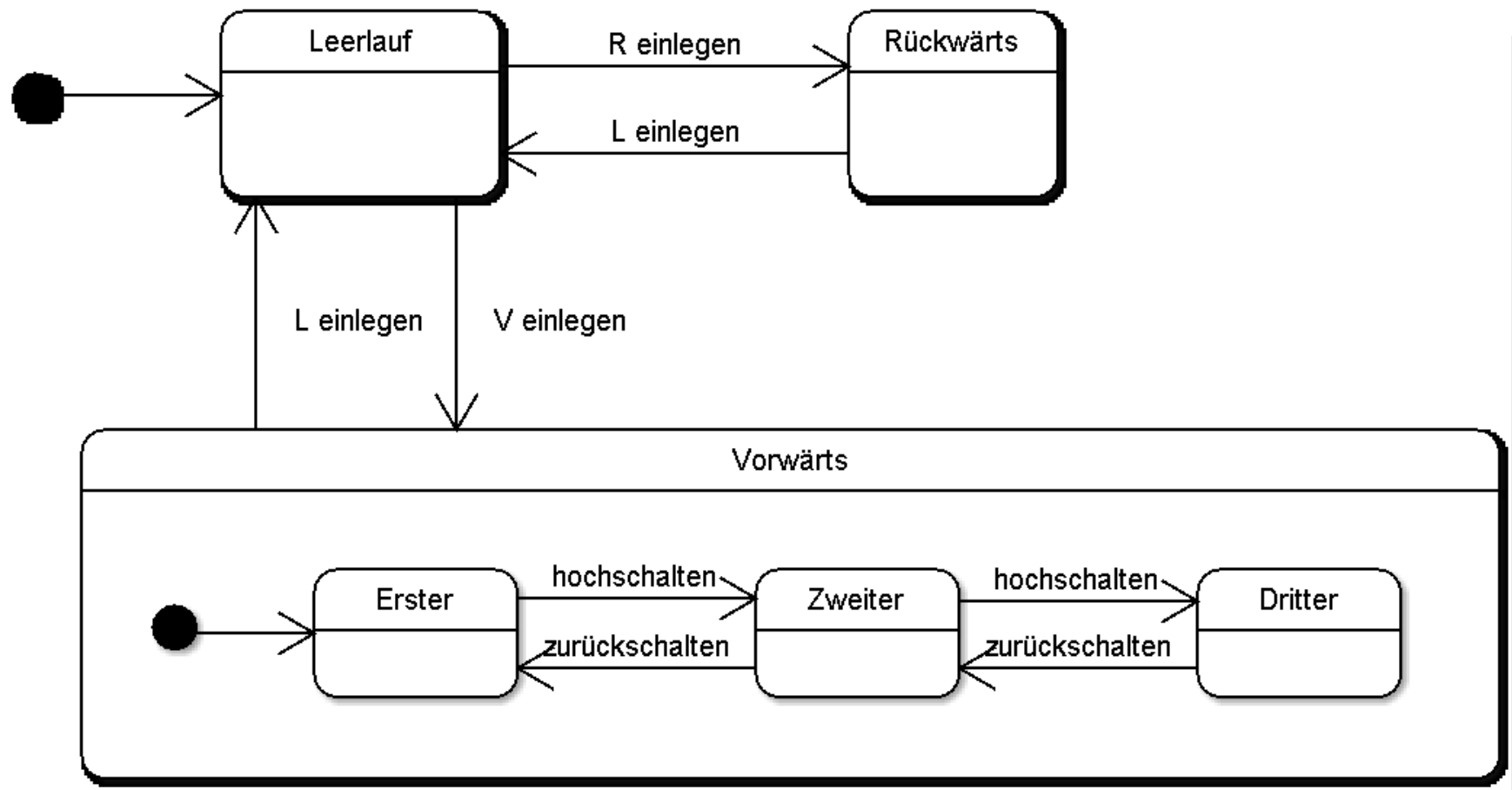
- Transitionen werden als Pfeile mit offenen Pfeilspitzen dargestellt: 
- Die UML bietet die Möglichkeit, eine Transition mit folgenden Angaben zu versehen:
  - dem auslösenden Ereignis (Trigger), der die Transition initiiert
  - dem ausgelösten Ereignis (Effect), das beim Zustandsübergang stattfindet.
  - einer Wächterbedingung (Guard), die erfüllt werden muss, damit die Transition schalten kann
- Darstellungen:

Nebenläufige Zustandsautomaten

- Ein Zustand kann durch mehrere parallele Automaten verfeinert werden.
- Der Zustand Bestanden wird erreicht, wenn alle 3 Automaten in Studieren den Endzustand erreicht haben.
- Der Zustand Durchgefallen wird erreicht, wenn die Transition durchgefallen geschaltet wird (die anderen inneren Automaten werden dann bedeutungslos).



Beispiel für Zustandsdiagramm: Getriebe



Übung

Übungsaufgabe Objektorientierte Design Zustandsautomat für Vorlesung und Prüfung

Beschreibung:

Für eine SW Engineering Vorlesung seien die Teilnahme an der Vorlesung, die Teilnahme am parallelen Praktikum und die Abgabe eines Leistungsnachweises am Ende des Praktikums als Zulassung zu Prüfung verbindlich. Ein nicht bestandener Leistungsnachweis erfordert eine erneute Praktikumsteilnahme inkl.Leistungsnachweis. Eine nicht bestandene Prüfung erfordert die erneute Teilnahme an Vorlesung und Praktikum inkl. Leistungsnachweis.

Aufgabe:

Erstellen Sie einen Zustandsautomaten, der den Weg bis zur bestandenen Prüfung für diese Vorlesung darstellt. Bauen Sie Nebenläufigkeiten ein, wo es sinnvoll ist.

Zeit:

15 Minuten, arbeiten Sie ggf. zusammen mit einem Partner

Ergebnis von OOA + OOD

Am Ende des objektorientierten Entwurfs sollten verfügbar sein:

- Vollständige Klassendiagramme mit
  - sämtlichen Klassen
    (incl. deren Attribute und vollständigen Operationsbeschreibungen)
  - sämtlichen Assoziationen
    (incl. Kardinalitäten, Rollen usw.)
  - sämtliche Vererbungsbeziehungen
- Vollständige Interaktionsdiagramme
  - Sequenz- bzw. Kollaborationsdiagramme
- Bei komplexem Verhalten von Objekten
  - ausführliche Zustandsdiagramme
- Logische und physikalische Modularisierung
  - Paketdiagramme und Komponentendiagramme
  - Einsatzdiagramme

Zum Schluss dieses Abschnitts …

Noch Fragen ??