Technische Universität
Kaiserslautern

o1o1seda o1o1oo

software engineering dependability

Software Entwicklung 2

UML in der Analyse

Inhalt

- Objektorientierte vs. klassische Softwareentwicklung
  - Imperative Programmierung
  - Objektorientierte Programmierung
  - Grundprinzipien der objektorientierten Denkweise
  - Elemente in der objektorientierten Programmierung
  - Besonderheiten objektorientierter Softwareentwicklung
- Objekte
- Klassen
- Operationen
- Assoziationen
  - Kardinalität
  - Assoziationsnamen und Rollen
  - Sonderfälle von Assoziationen
  - Assoziative Klassen
  - Aggregationen und Kompositionen
- Vererbung
- Pakete
- Botschaften
- Szenarios

Lernziele

- Grundlegende Eigenschaften der nicht-objektorientierten, imperativen Softwareentwicklung kennen
- Grundlegende Eigenschaften der objektorientierten Softwareentwicklung kennen
- Wesentliche Unterschiede zwischen diesen beiden Alternativen kennen
- Die Begriffe im Zusammenhang mit der Objektorientierung erläutern können (Objekt, Klasse, Attribut, Operation, Objektdiagramm, Vererbung, Pakete, Botschaften, Szenarios, usw.)
- Assoziationen erläutern und anwenden können (Kardinalitäten, Assoziationsnamen und Rollen, Sonderfälle von Assoziationen, Assoziative Klassen, Aggregationen und Kompositionen)
- Die UML-Darstellungen für diese Elemente anwenden können
- Die UML zur Beschreibung in der Analysephase verwenden können

Imperative Programmierung

- Trennung von Daten und Verarbeitung
- Getrennte Hierarchien für Daten und Prozeduren
- Änderung der globalen Datenstruktur bewirkt weitreichende Änderungen der prozeduralen Struktur
- Ungünstiges, weil kompliziertes Paradigma für die Problemanalyse (Bindung, Kopplung)

Objektorientierte Programmierung

- Keine Trennung von Daten und Prozeduren
- Klare Richtlinie für die Identifikation von Objekten und Klassen
- Natürlicher Mechanismus zur Identifikation von Objekten
- Zusammenfassung von Daten und den dazugehörigen Prozeduren zu Klassen bzw. Objekten
- Gewünschtes Systemverhalten kommt zustande durch Zusammenwirken verschiedener Objekte und Klassen über Botschaften oder Nachrichten
- Natürlicher Mechanismus zur Abbildung der Gruppenzugehörigkeit aus der realen Welt durch Bildung von Klassen
- Unterschiedliche Reaktionsmöglichkeiten auf Nachrichten durch Objekte

Grundprinzipien der objektorientierten Denkweise

- Objekte vereinigen Daten und Prozeduren
- Objekte tauschen Nachrichten aus und reagieren auf diese
- Objekte erben ihre Eigenschaften und Fähigkeiten durch die Zugehörigkeit zu Objektklassen
- Objekte verschiedener Klassen reagieren auf gleiche Nachrichten unterschiedlich

Elemente in der objektorientierten Programmierung

- Objekt
  - Eindeutig benannte Abstraktion eines in sich geschlossenen Elements der realen Welt
  - Stellt die relevanten Aspekte einer Entität dar
  - Enthält Daten - sogenannte Attribute und hat ggf. Teile (Aggregation)
  - Befindet sich in einem Zustand
  - Enthält Prozeduren - sogenannte Methoden, die die Fähigkeiten des Objekts realisieren und Daten modifizieren
  - Attribute können (im Idealfall) nur über die objekteigenen Methoden abgefragt und modifiziert werden (*Information Hiding*, Datenkapsel)
  - Methoden werden durch Nachrichten (Botschaften) aktiviert

Elemente in der objektorientierten Programmierung

- Klasse
  - Klassen beschreiben Objekte derselben Art.
  - Klassen sind die ”Baupläne“ der Objekte (Abstrakter Datentyp)
  - Ein Objekt ist eine Instanz einer Klasse (vgl. Vereinbarung einer Variablen eines bestimmten Datentyps in imperativen Programmiersprachen)
  - Eigenschaften, die für alle Objekte einer Klasse identisch sind, können der Klasse zugeordnet werden (Klassenvariablen). Alle Objekte einer Klasse haben auf diese Variablen Zugriff
  - Klassen sind ebenfalls Objekte. Daher können Klassen auch selbst Methoden besitzen (Klassenmethoden), z.B. zur Initialisierung der Klassenattribute

Elemente in der objektorientierten Programmierung

- Attribut
  - Attribute beschreiben die Eigenschaften von Objekten bzw. Klassen (lokale Daten der Objekte)
- Methode
  - Methoden realisieren die Funktionalität der Objekte, bzw. der Klassen
  - Fähigkeiten (Methoden) die zu allen Objekten einer Klasse hinzugefügt werden müssen, werden lediglich der Klasse hinzugefügt
- Botschaft
  - Botschaften (Nachricht, Message) dienen zur Aktivierung von Methoden in Objekten
  - Botschaften können verschiedene passende Methoden aktivieren (Polymorphismus)

Besonderheiten objektorientierter Softwareentwicklung

- Vererbung
  - Neue Klassen können aus vorhandenen Klassen abgeleitet werden
  - Der Bezug zwischen Oberklasse und Unterklasse ist streng hierarchisch
  - Die Suche nach ausführbaren Methoden erfolgt in der Klassenhierarchie streng hierarchisch von unten nach oben
    (Empfänger, Oberklasse, deren Oberklasse, ...)
  - Klassen auf einer Ebene werden nicht durchsucht
  - Methoden in abgeleiteten Klassen überschreiben gleichnamige Methoden in Oberklassen (Polymorphismus)
- Mehrfachvererbung
  - Abgeleitete Klassen können mehr als eine Oberklasse besitzen
  - Die Vererbungsbeziehungen bilden keine baumartige Struktur mehr

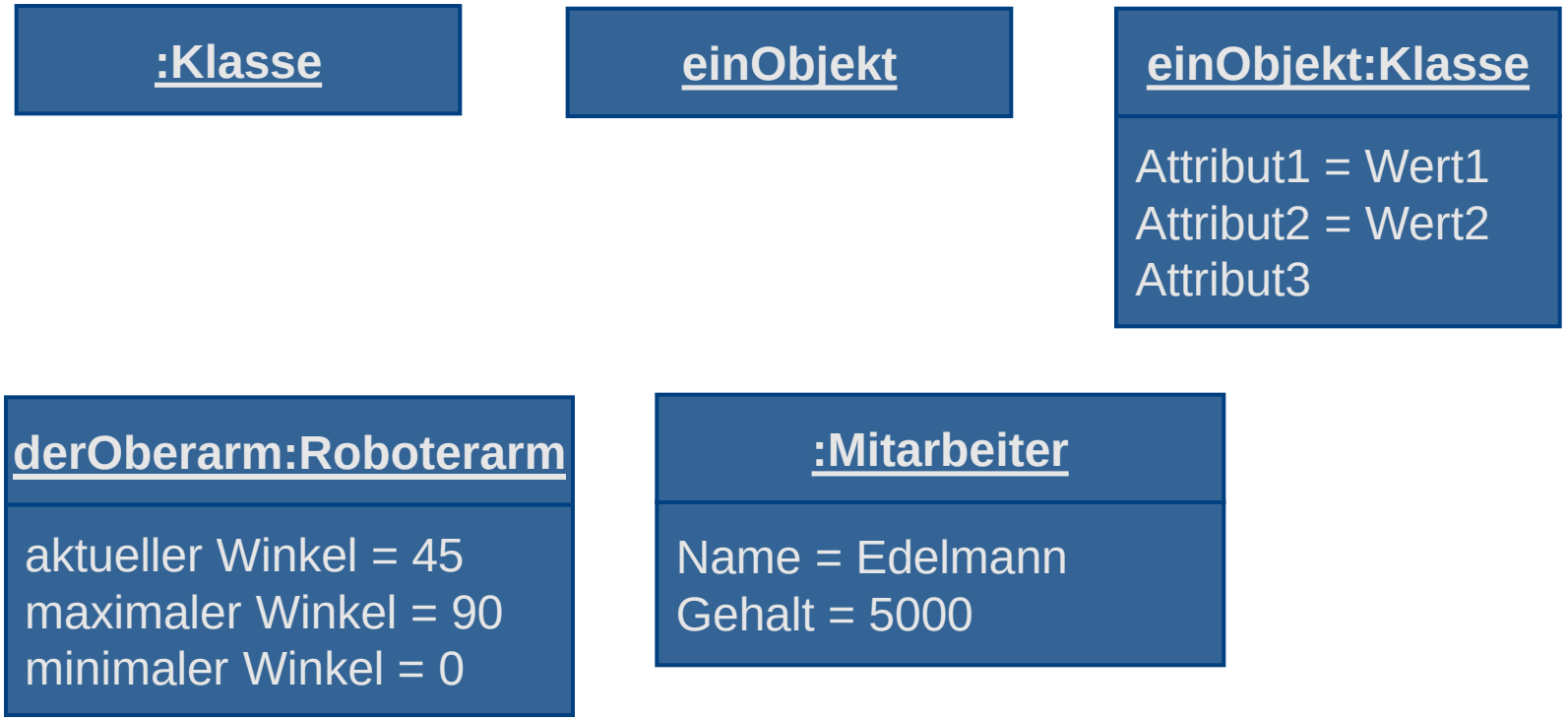
Besonderheiten objektorientierter Softwareentwicklung

- Binden
  - Zuordnung zwischen Aufrufmechanismus (Methodenname) und Implementierung
  - Statisches Binden (typ. imperative Programmierung)
    - Zuordnung geschieht bereits zum Zeitpunkt der Übersetzung
    - Statische Fehleranalyse zur Übersetzungszeit möglich
    - Höhere Ausführungsgeschwindigkeit des Programmes
    - Nicht flexibel
  - Dynamisches Binden (typ. objektorientierte Programmierung)
    - Zuordnung zwischen aufrufender Botschaft und auszuführendem Code geschieht zur Laufzeit des Programmes
    - Flexiblere Behandlung von Aufrufen, höhere Laufzeit, statische Fehlerprüfung nicht möglich

- Abstrakte Klassen
  - Werden verwendet, um einen Klassenbaum zu strukturieren
  - Können zur Vereinbarung von Methodenschnittstellen verwendet werden (falls Methoden erst später implementiert werden)
  - Sind nicht instanziierbar
  - Auch Klassen sind Objekte, d.h. auch Klassen sind Instanzen von Klassen. Derartige Klassen heißen Metaklassen

Objekte

- UML-Notation Objekt



Objekte

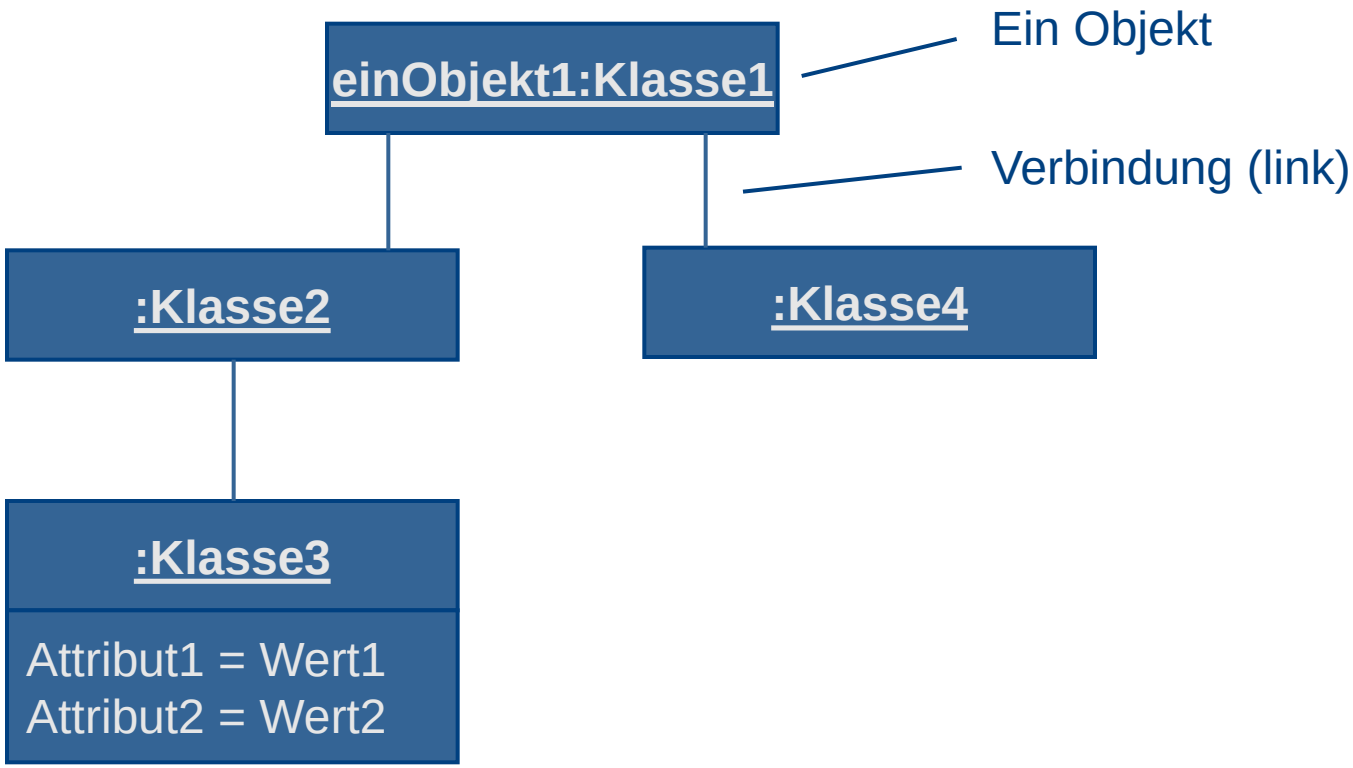
- Objekte der Seminarorganisation

:Kunde
Name = Hans Schulz Adresse = Dortmund Kommunikation = 0231/234789 Geburtsdatum = 31.12.1960 Funktion = Berater Umsatz = 6600,- [€] Kunde seit = 15.3.1995

:Kunde
Name = Sabine Wenzel Adresse = Bochum Kommunikation = 0234/76424 Geburtsdatum = 29.2.1972 Funktion = Systemanalytiker Umsatz = 2500,- [€] Kunde seit = 30.6.1998

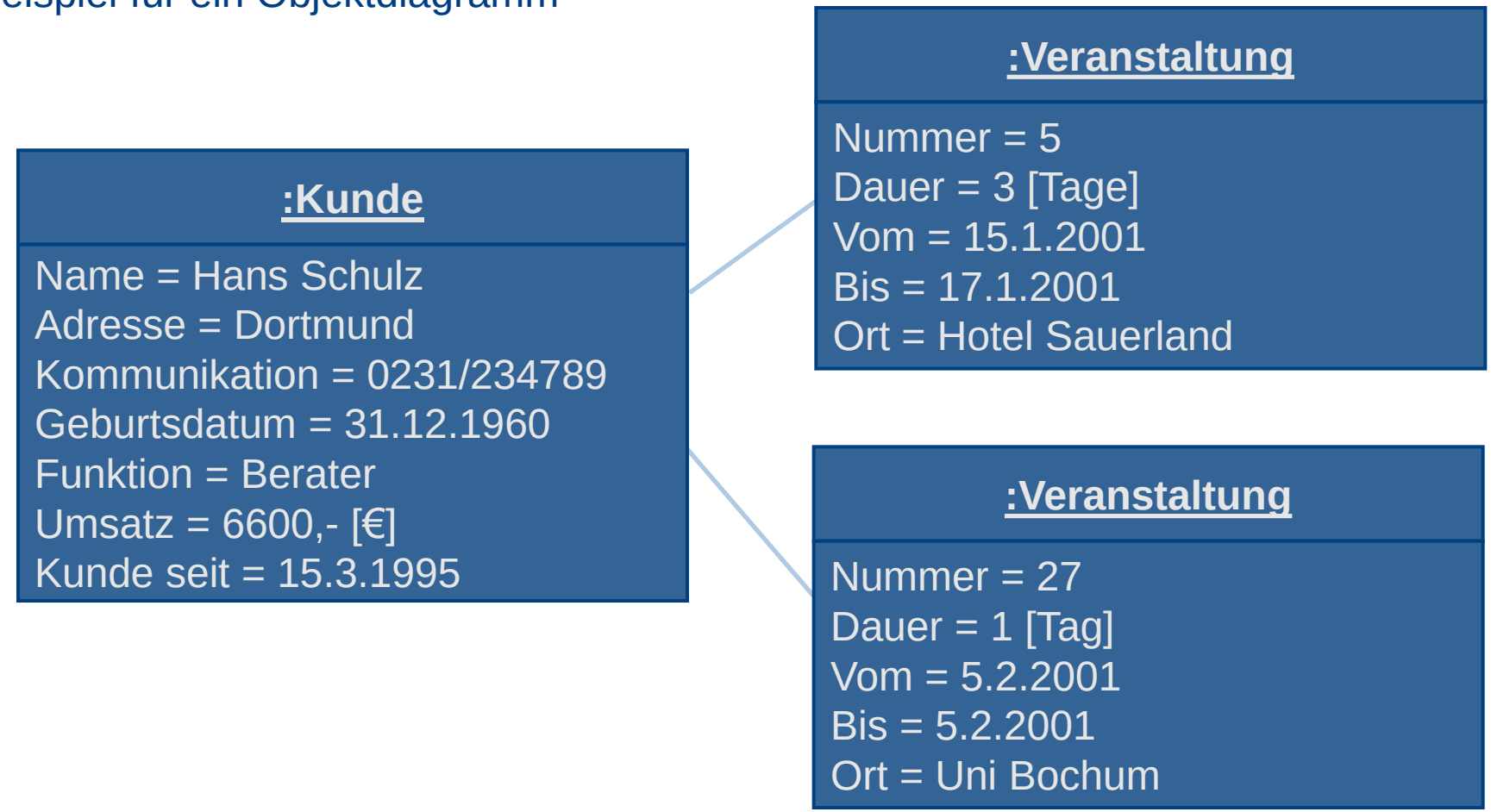
Objekte

- UML-Objektdiagramm



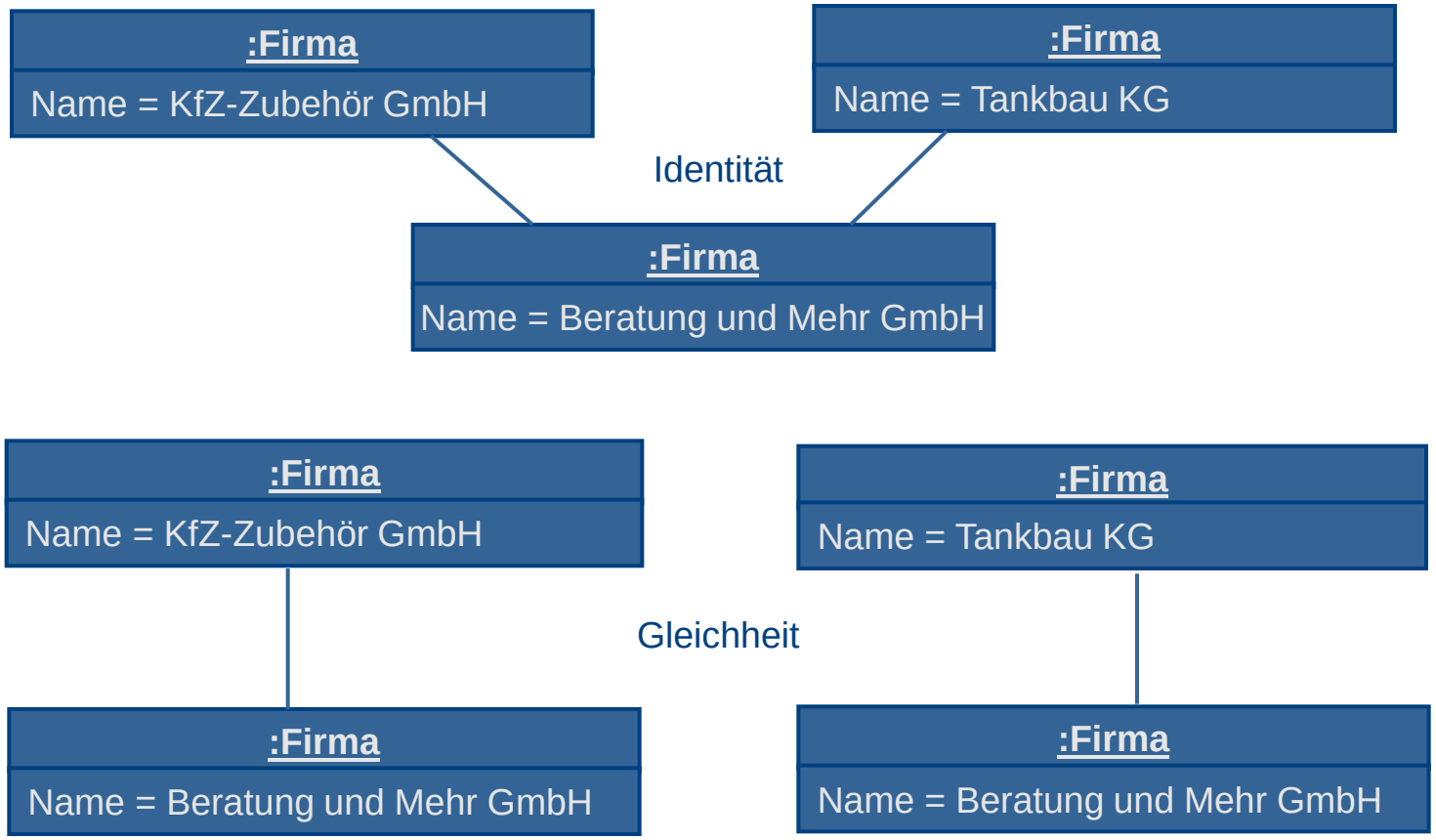
Objekte

- Beispiel für ein Objektdiagramm



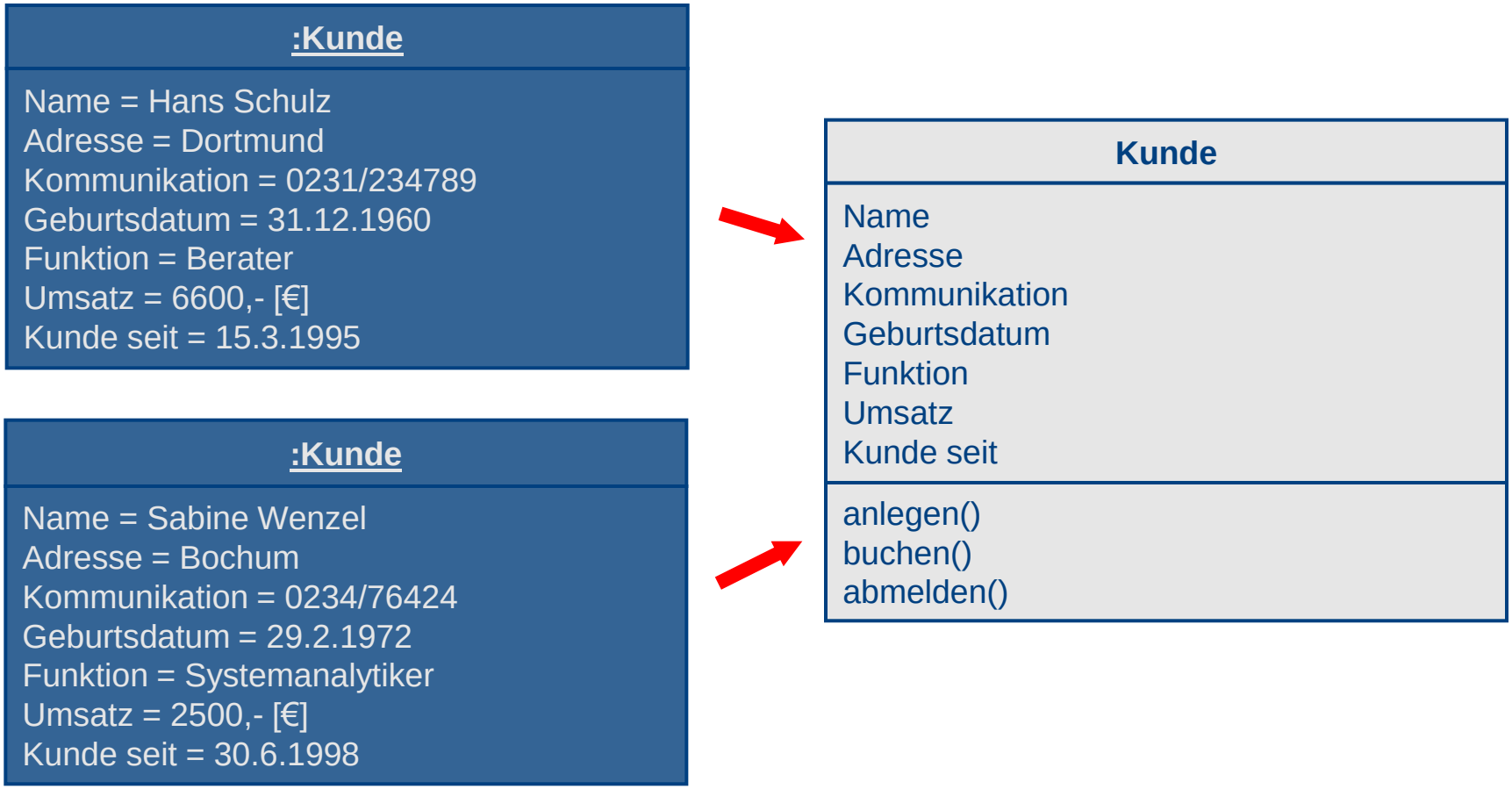
Objekte

- Identität und Gleichheit von Objekten



Klassen

- Beispiel



Klassen

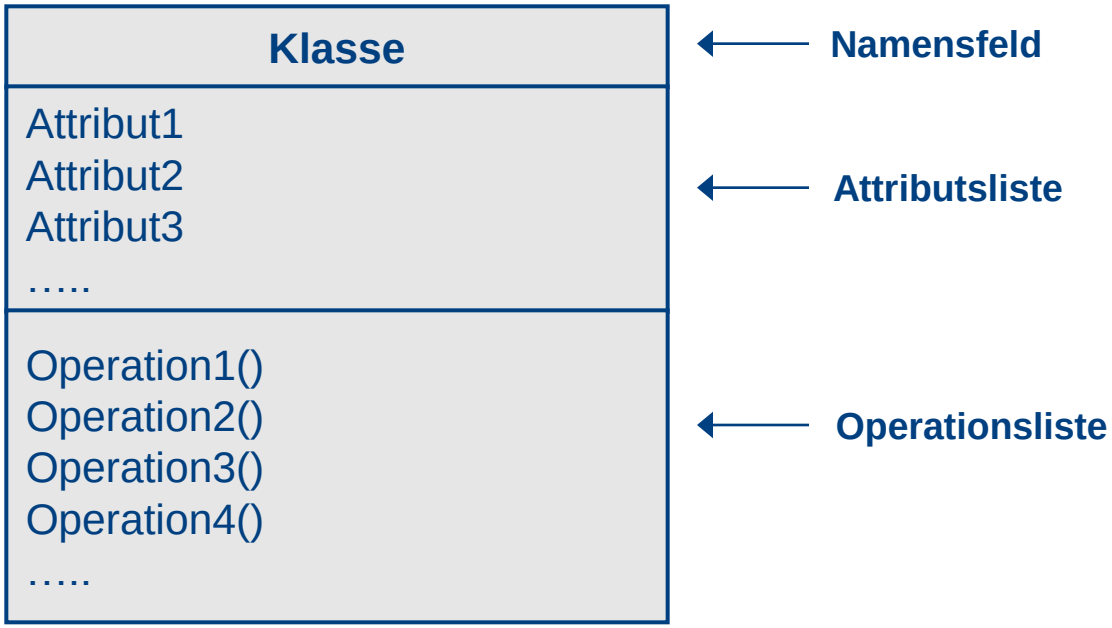
- UML-Notation Klasse

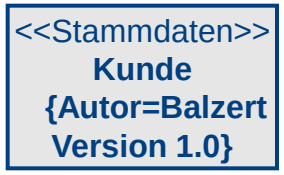
Nur Klassennamen

Nur Attribute

Nur Operationen

Ausführliche Darstellung





Klassen

- UML-Notation für Attribute

Klasse
Attribut1: Typ = Anfangswert1 Attribut2: Typ Klassenattribut: Typ /abgeleitetes Attribut: Typ

Attribute:
{ Merkmal1, Merkmal2, … }

Person
Geburtsdatum /Alter

Operationen

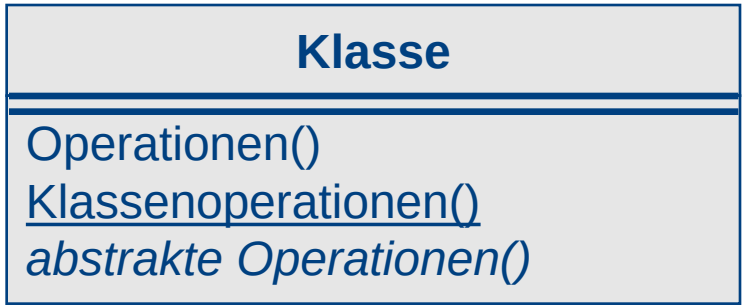
- Drei Klassen von Operationen
  - Objektoperationen (kurz: Operationen)
  - Konstruktoroperationen (kurz: Konstruktoren)
  - Klassenoperationen

Mitarbeiter
Personal-Nr Name Geburtsdatum Einstellungsdatum Beurteilung Anzahl
einstellen() entlassen() erstelle Arbeitsbescheinigung() notiere Beurteilung() berechne Dienstzeit() erstelle Geburtstagsliste() versetze in Abteilung() erstelle Zeugnis()

Abteilung
Bezeichnung Ort Leiter Abteilungsprofil
gib Abteilungsprofil()

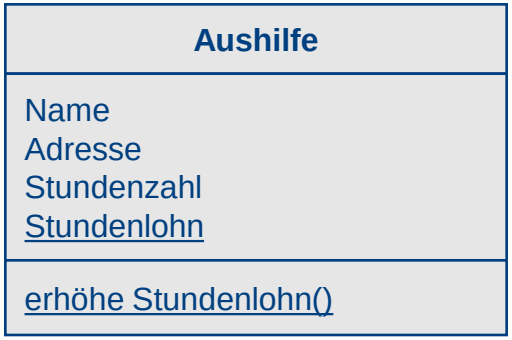
Operationen

- UML-Notation für Operationen



Operation()
{Merkmal1, Merkmal2, … }

- Klassenoperationen



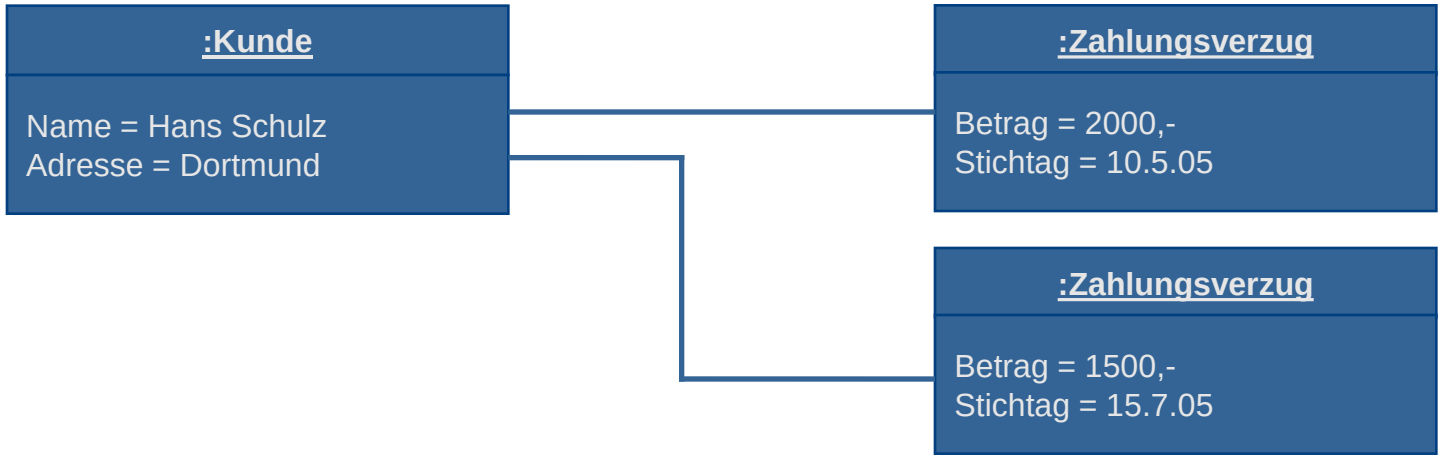
Assoziation

- Modelliert Beziehungen zwischen Objekten gleichrangiger Klassen
- Auch zwischen Objekten derselben Klasse zulässig (reflexive Assoziation)
- Im Gegensatz dazu verknüpft eine Vererbung Klassen miteinander, nicht Objekte der Klassen
- In der Systemanalyse: bidirektional
- UML: binäre und höherwertige Assoziationen

Assoziation

- Fallstudie »Seminarorganisation«

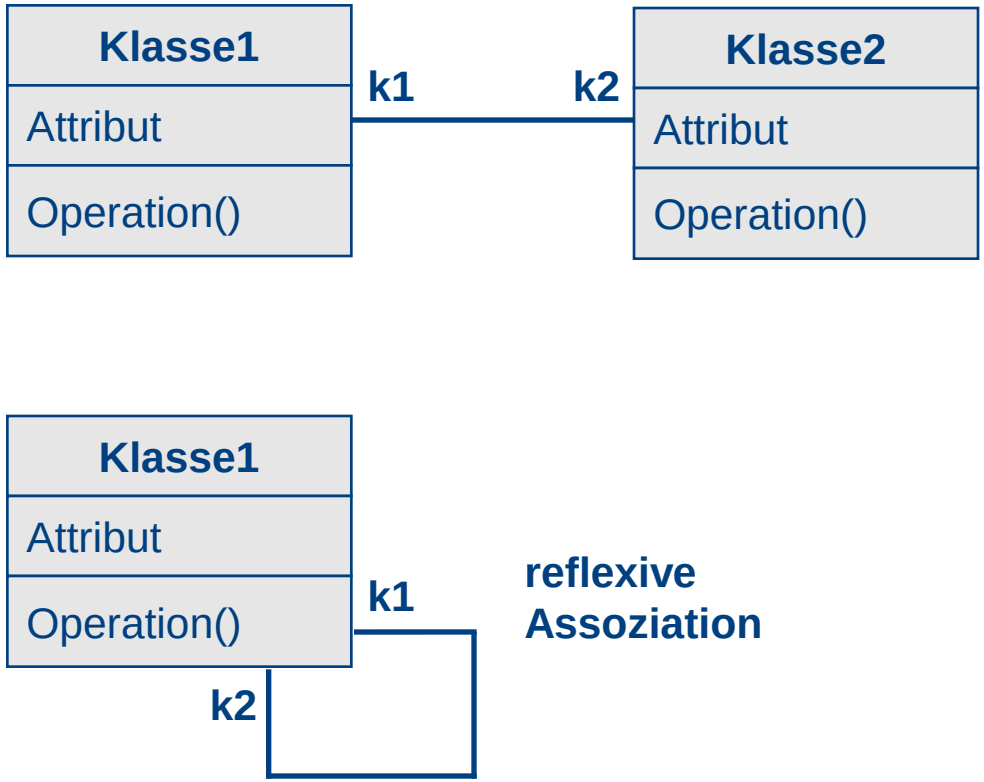
Objektdiagramm



Klassendiagramm

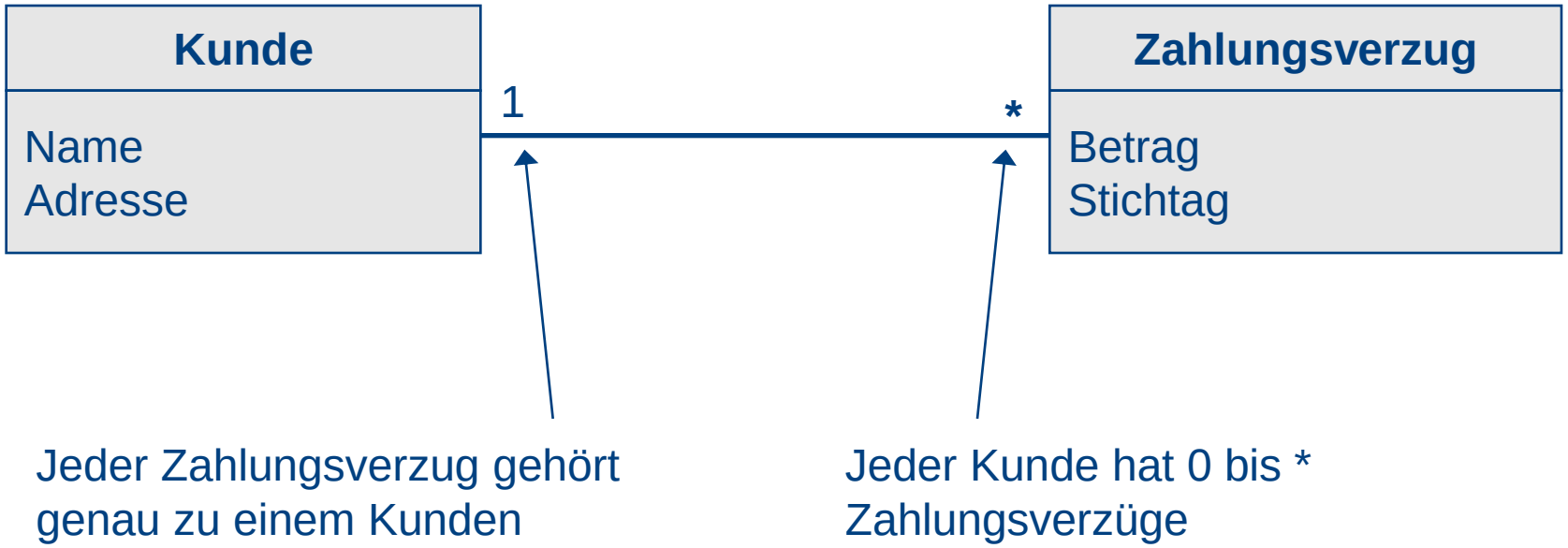
Assoziation

- UML-Notation



Assoziation Kardinalität

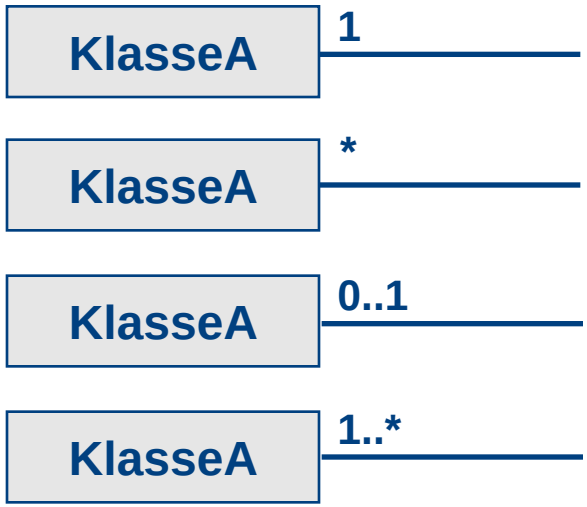
- Kardinalität *(multiplicity)*



Assoziation Kardinalität

- UML-Notation

Anzahl der Assoziationen:



Zwischen einem beliebigen Objekt und einem Objekt der Klasse A gibt es:

genau eine Beziehung
(Muss-Beziehung)

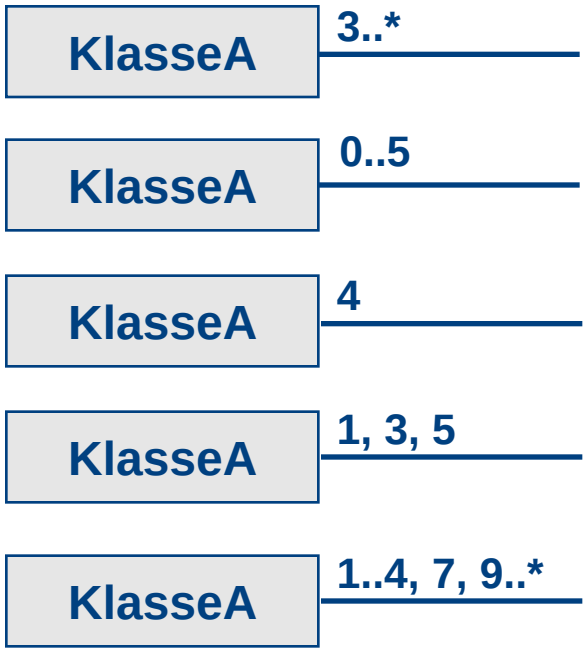
viele Beziehungen (null, eine oder mehrere)
(Kann-Beziehung)

null oder eine Beziehung
(Kann-Beziehung)

eine oder mehrere Beziehungen
(Muss-Beziehung)

Assoziation Kardinalität

- UML-Notation



drei oder mehr als drei Beziehungen (Muss-Beziehung)

null bis fünf Beziehungen (Kann-Beziehung)

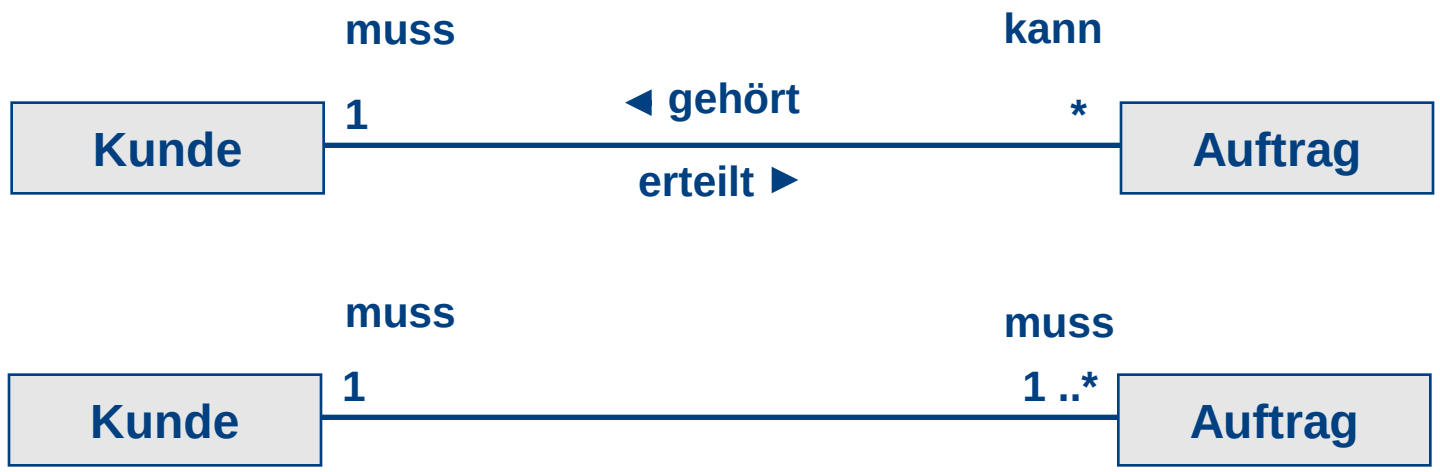
genau vier Beziehungen (Muss-Beziehung)

eine, drei oder fünf Beziehungen (Muss-Beziehung)

nicht keine, fünf, sechs oder acht Beziehungen (Muss-Beziehung)

Assoziation Kardinalität

- Kann- und Muss-Assoziationen

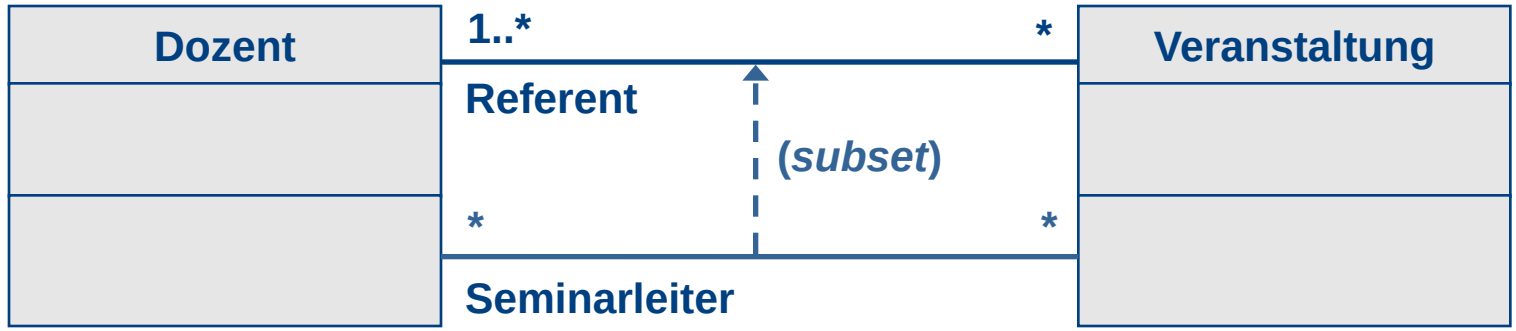


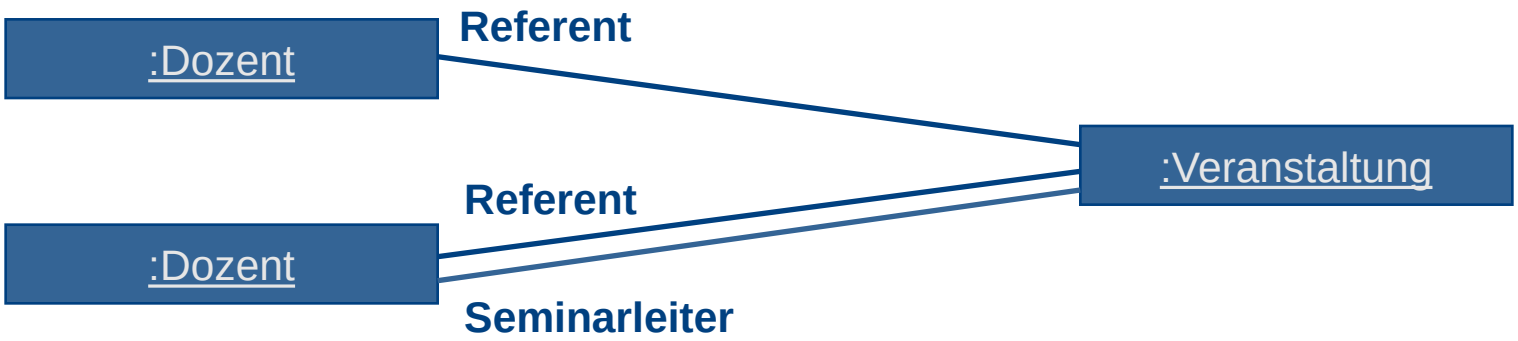
Assoziation
Assoziationsnamen und Rollen

- Rolle
  - Beschreibt, welche Funktion ein Objekt in einer Assoziation innehat
  - Beispiel

Assoziation
Assoziationsnamen und Rollen

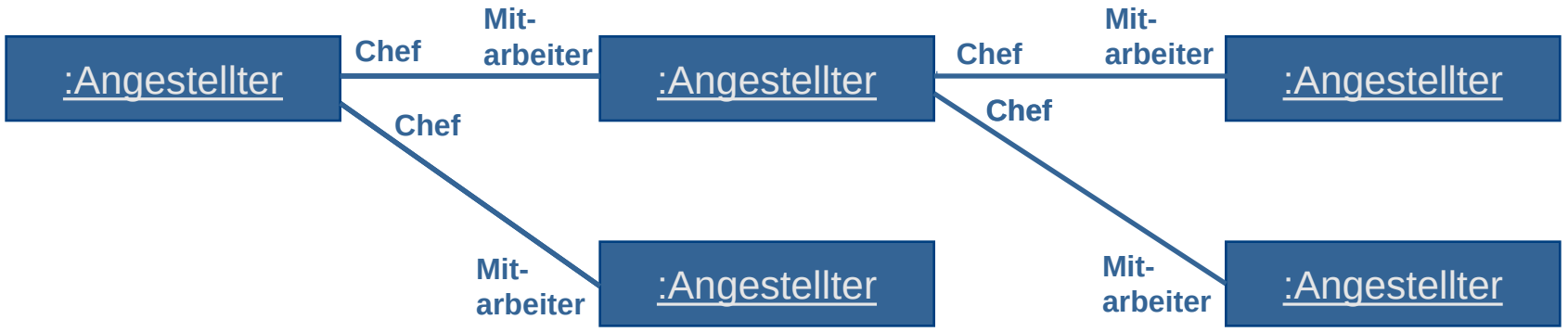
- Rollennamen bei mehr als einer Assoziation



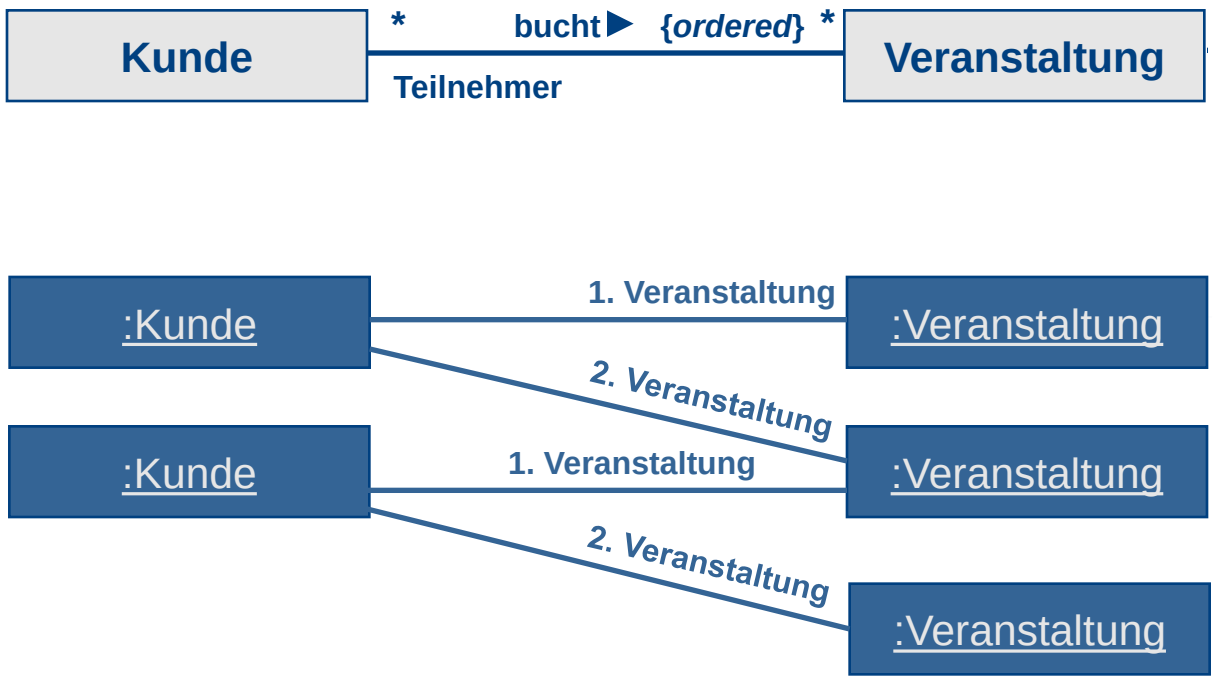


Assoziation
Assoziationsnamen und Rollen

- Rollennamen in einer reflexiven Assoziation



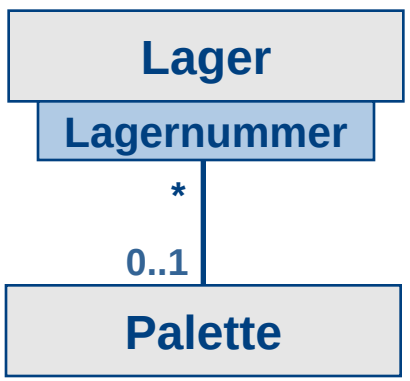
- Geordnete Assoziation
  - Definition einer Ordnung auf einer Assoziation

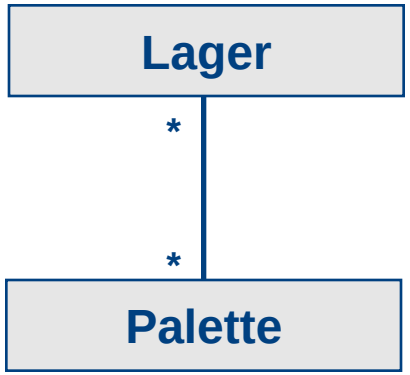


- Restriktionen *(constraints)*

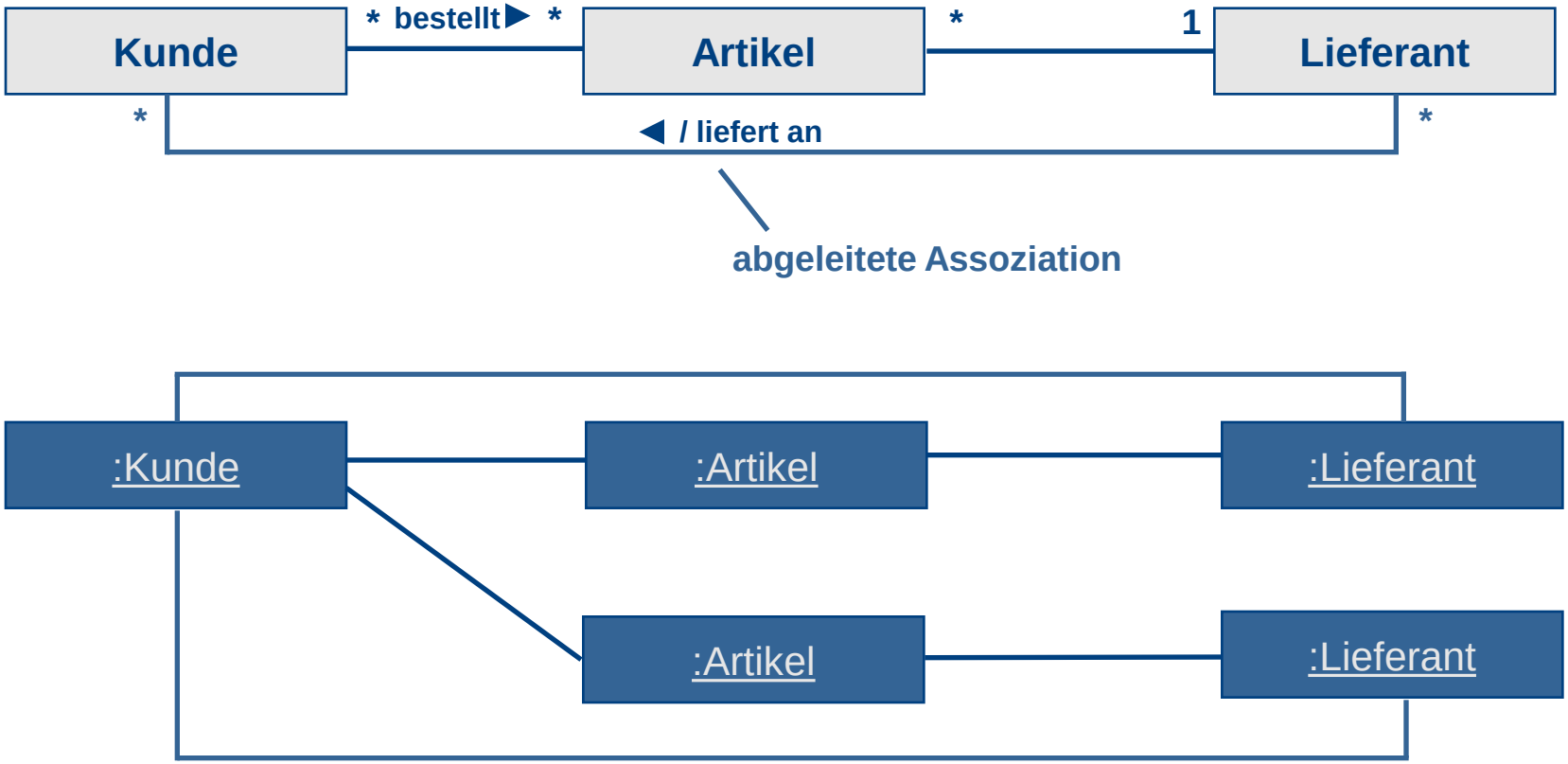
{ Zugfahrzeug.Anhängerlast >= Anhänger.Gewicht }

- Qualifizierte Assoziation



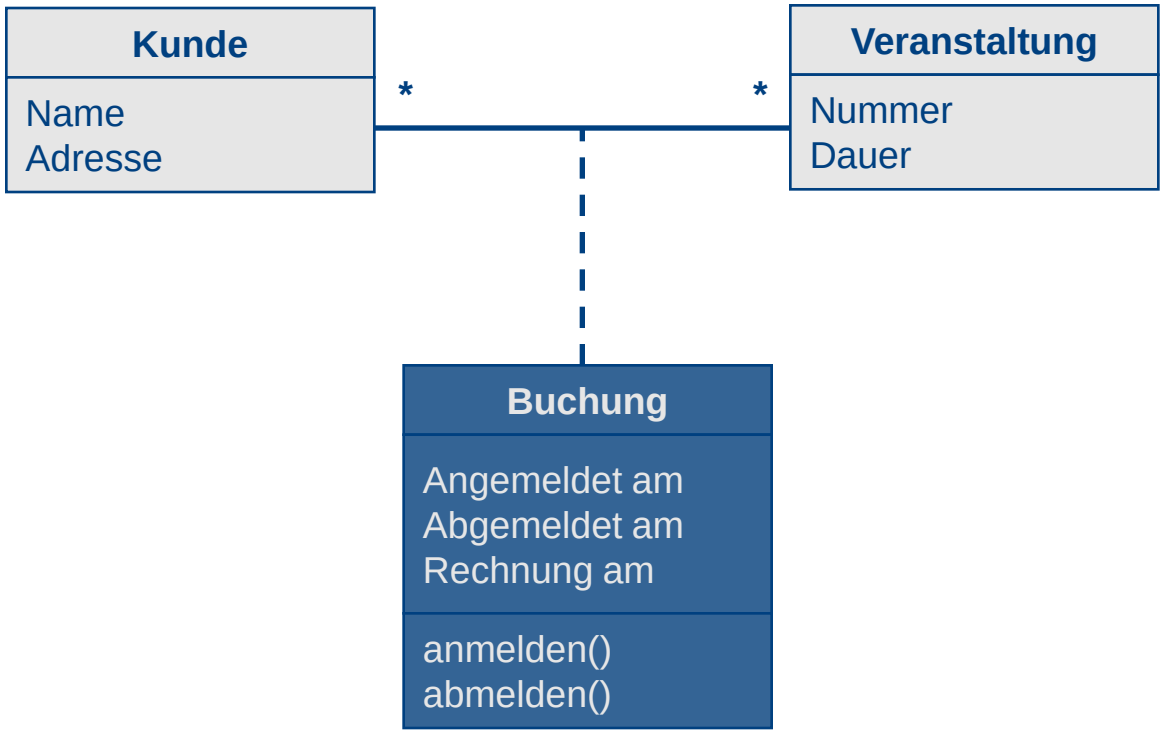


- Abgeleitete Assoziation



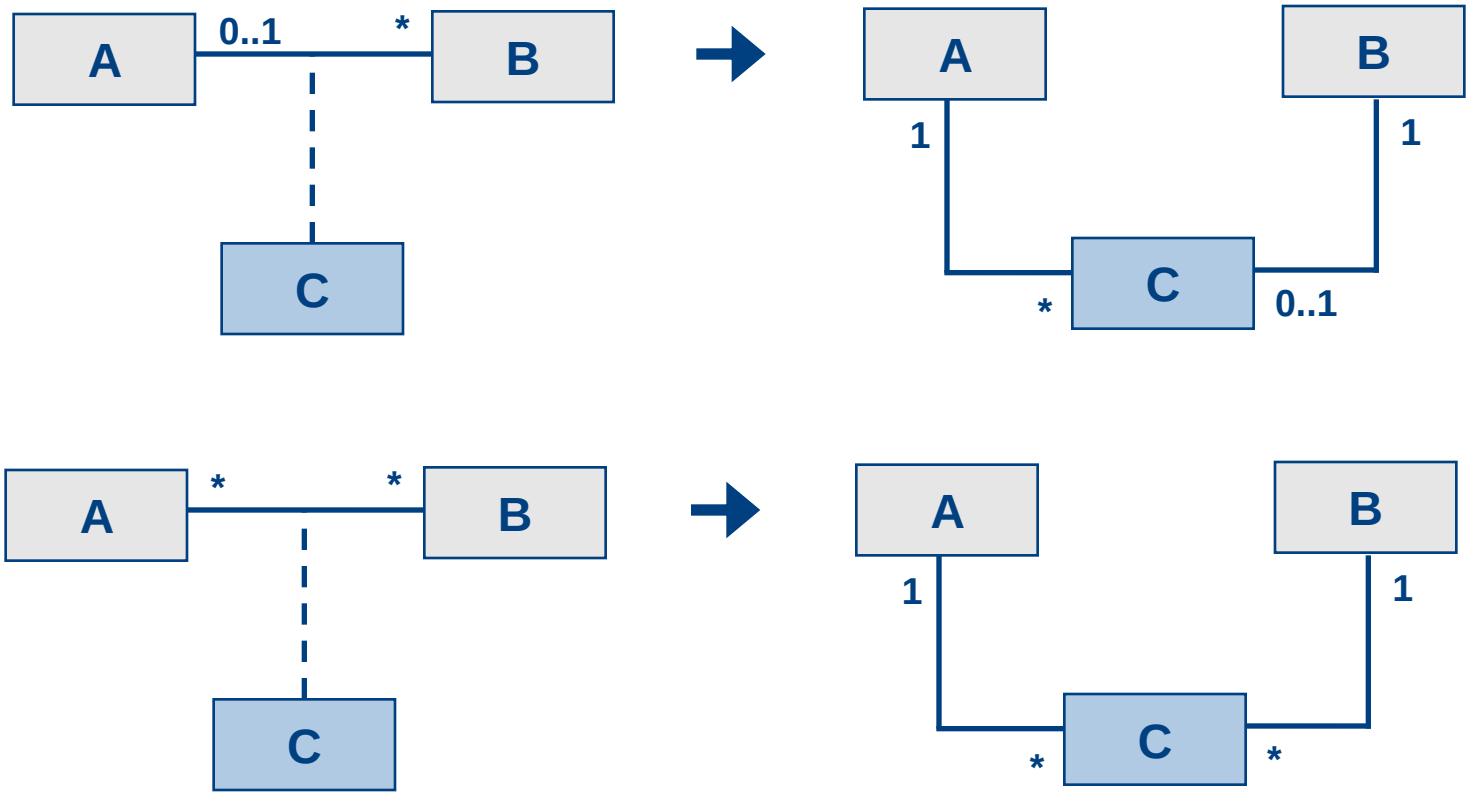
Assoziation
Assoziative Klassen

- Assoziation kann Eigenschaften einer Klasse besitzen



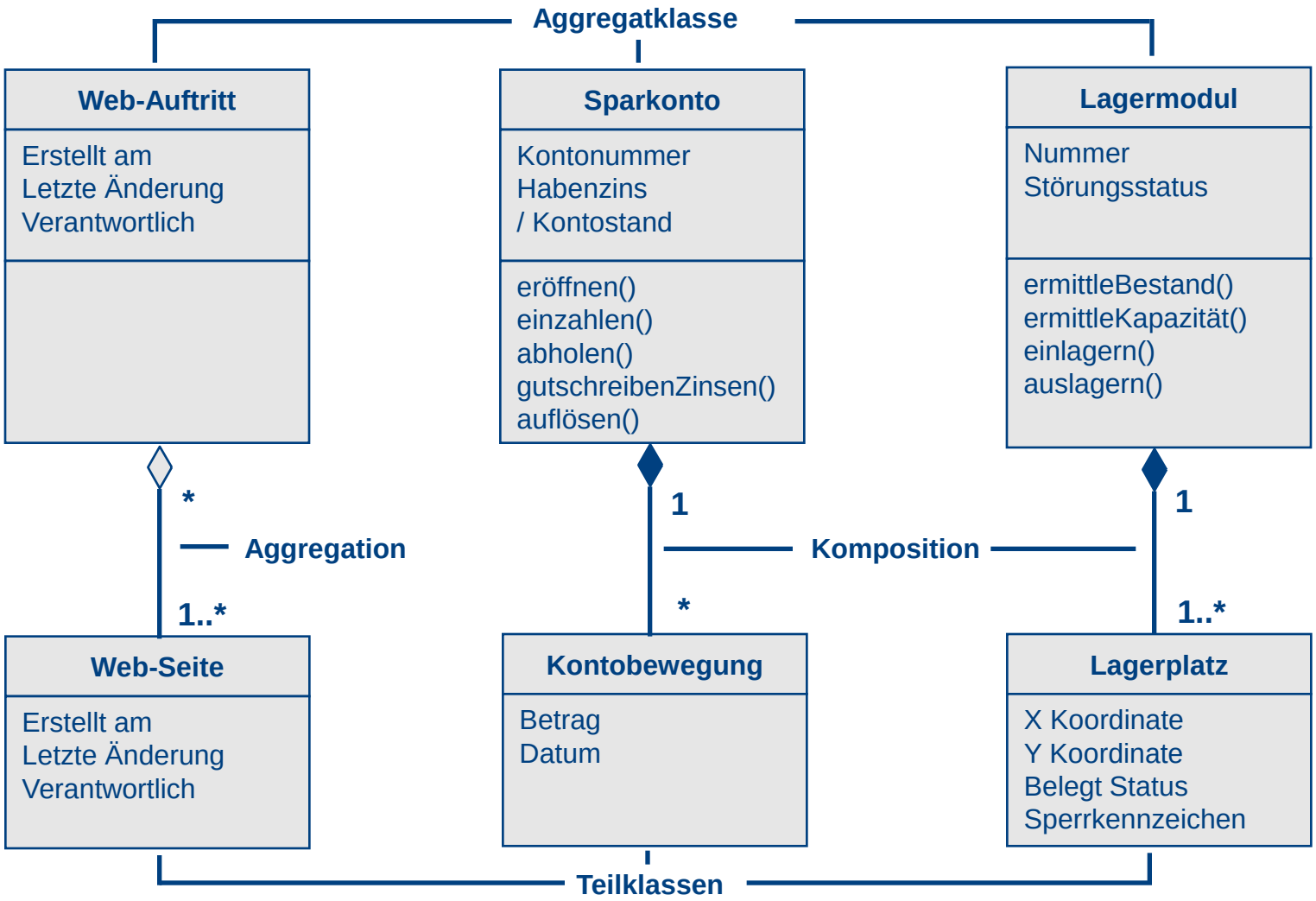
Assoziation
Assoziative Klassen

- Auflösen einer assoziativen Klasse



- Sonderfall der Assoziation
- Liegt vor, wenn zwischen den Objekten der beteiligten Klassen (kurz: den beteiligten Klassen) eine Rangordnung gilt, die sich durch »ist Teil von« bzw. »besteht aus« beschreiben lässt
  - Man spricht vom Ganzen und seinen Teilen
- Die Objekte der Aggregation bilden einen gerichteten azyklischen Graphen
  - Wenn B Teil von A ist, dann darf A nicht Teil von B sein
  - Shared aggregation (weak ownership)
    - Ein Teilobjekt kann mehreren Aggregatobjekten zugeordnet werden

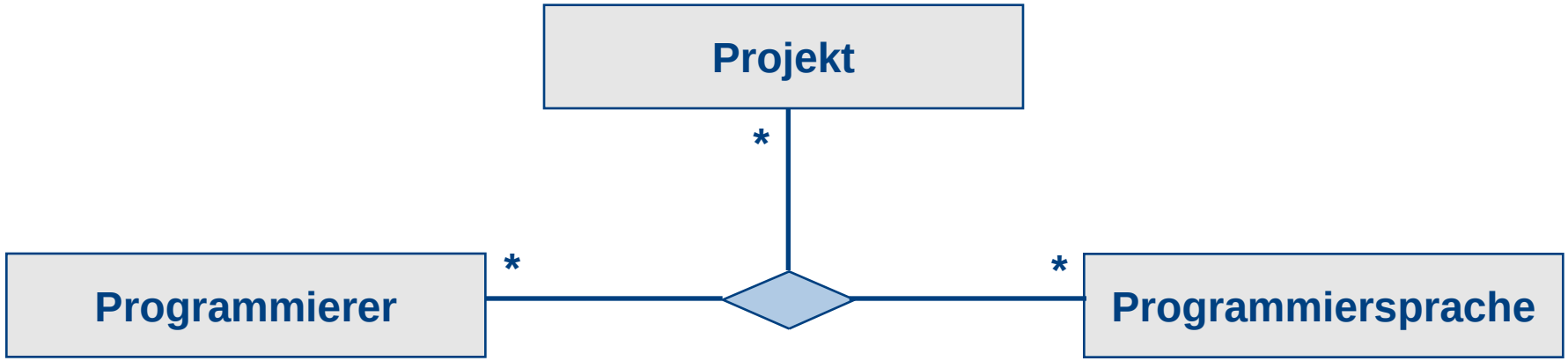
- Starke Form der Aggregation
- Zusätzlich muss gelten
  - Jedes Objekt der Teilklasse kann – zu einem Zeitpunkt – nur Komponente eines einzigen Objekts der Aggregatklasse sein, d. h., die bei der Aggregatklasse angetragene Kardinalität darf nicht größer als eins sein (*unshared aggregation, strong ownership*)
  - Ein Teil darf jedoch – zu einem anderen Zeitpunkt – auch einem anderen Ganzen zugeordnet werden
  - Die dynamische Semantik des Ganzen gilt auch für seine Teile (*propagation semantics*)
    - Wird beispielsweise das Ganze kopiert, so werden auch seine Teile kopiert
  - Wird das Ganze gelöscht, dann werden automatisch seine Teile gelöscht (*they live and die with it*)
    - Ein Teil darf jedoch zuvor explizit entfernt werden



Assoziation
Aggregationen und Kompositionen

- Höherwertige Assoziationen

Programmierer	Projekt	Programmiersprache
Meier	A	C++
Schröder	B	Java
Ludwig	A	Java

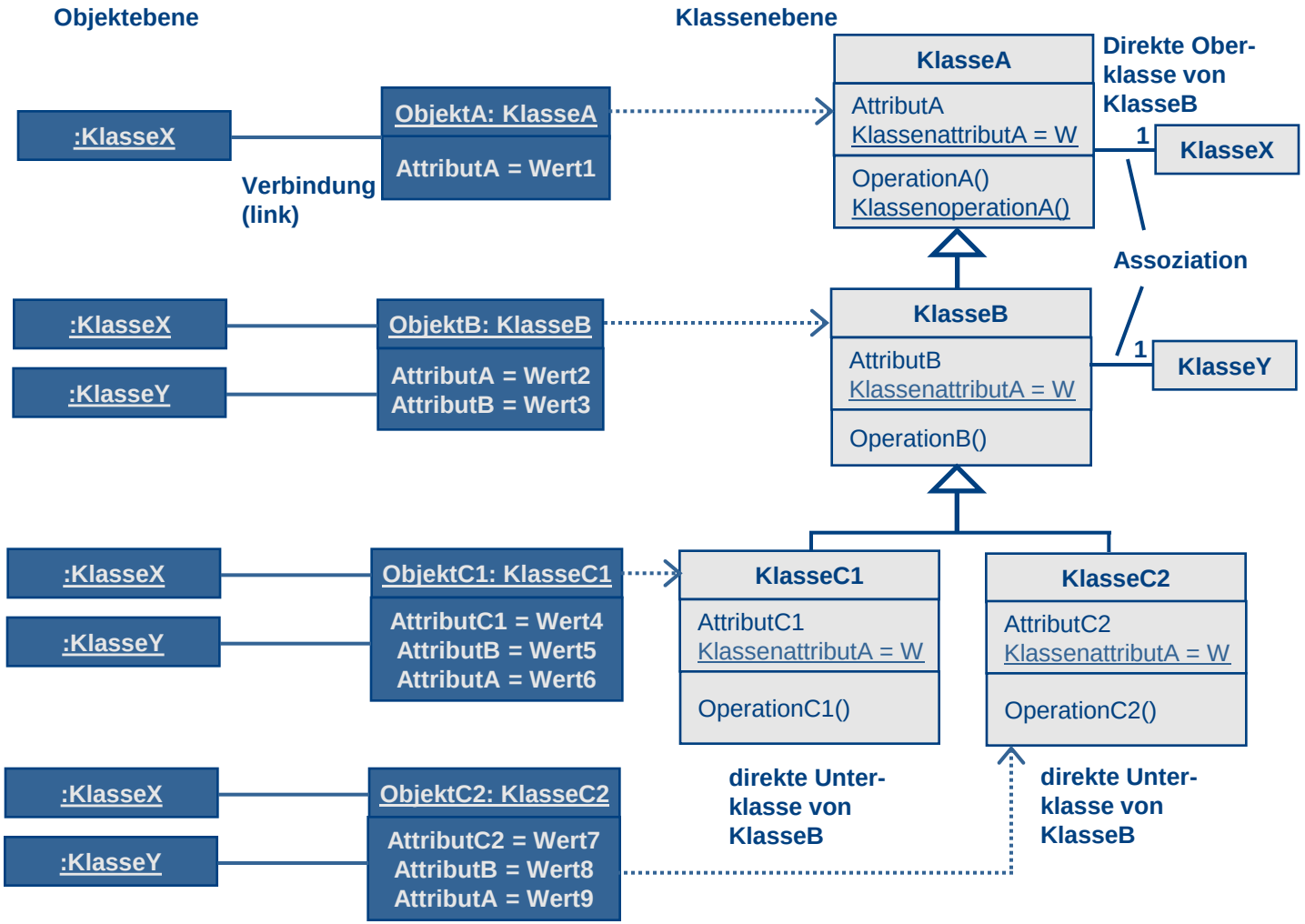


Vererbung

- Vererbung *(generalization)*
  - Eine spezialisierte Klasse (Unterklasse, *subclass*, abgeleitete Klasse) verfügt über
    - die Eigenschaften
    - das Verhalten und
    - die Assoziationen einer oder mehrerer allgemeiner Klassen (Oberklassen, *superclasses,* Basisklassen)

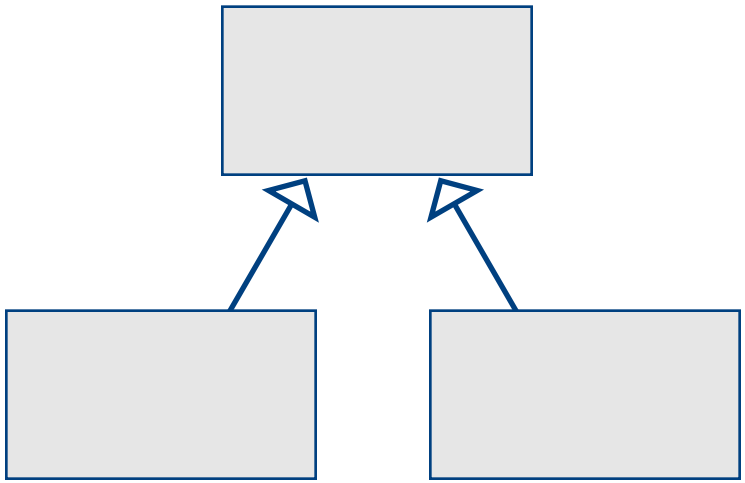
Vererbung

- Beispiel für den Vererbungs-mechanismus

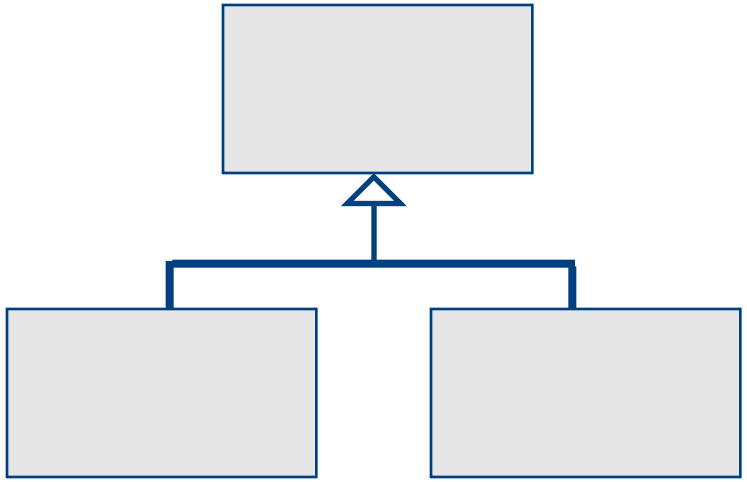


Vererbung

- UML-Notation



alternativ



Vererbung

- Unterklassen »unsichtbar«
  - Jede Klasse »kennt« nur ihre eigenen Attribute, Operationen und Assoziationen und die ihrer Oberklassen, sofern diese für sie sichtbar sind
  - Im Allgemeinen unterscheidet man drei verschiedene Sichtbarkeitsstufen
    - Außerhalb der Klasse nicht sichtbar *(private)*
      (in der UML: -), auch nicht für Unterklassen
    - Für alle Unterklassen sichtbar *(protected)*
      (in der UML: #)
    - Für alle anderen Klassen sichtbar, d.h. öffentlich *(public)*
      (in der UML: +)

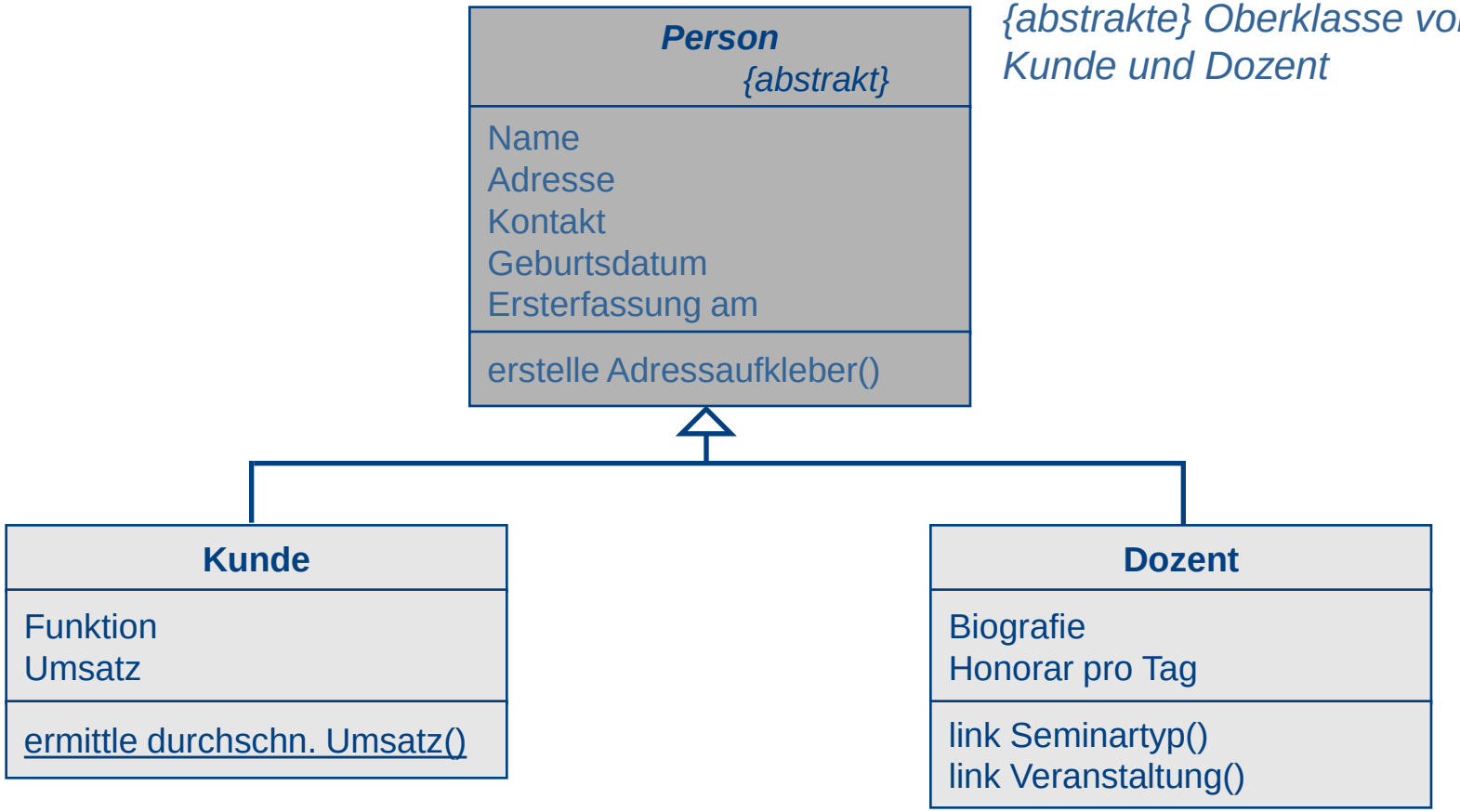
Vererbung

- Fallstudie »Seminarorganisation«

Kunde
Name Adresse Kontakt Geburtsdatum Funktion Umsatz Kunde seit
erstelle Adressaufkleber() ermittle durchschn. Umsatz()

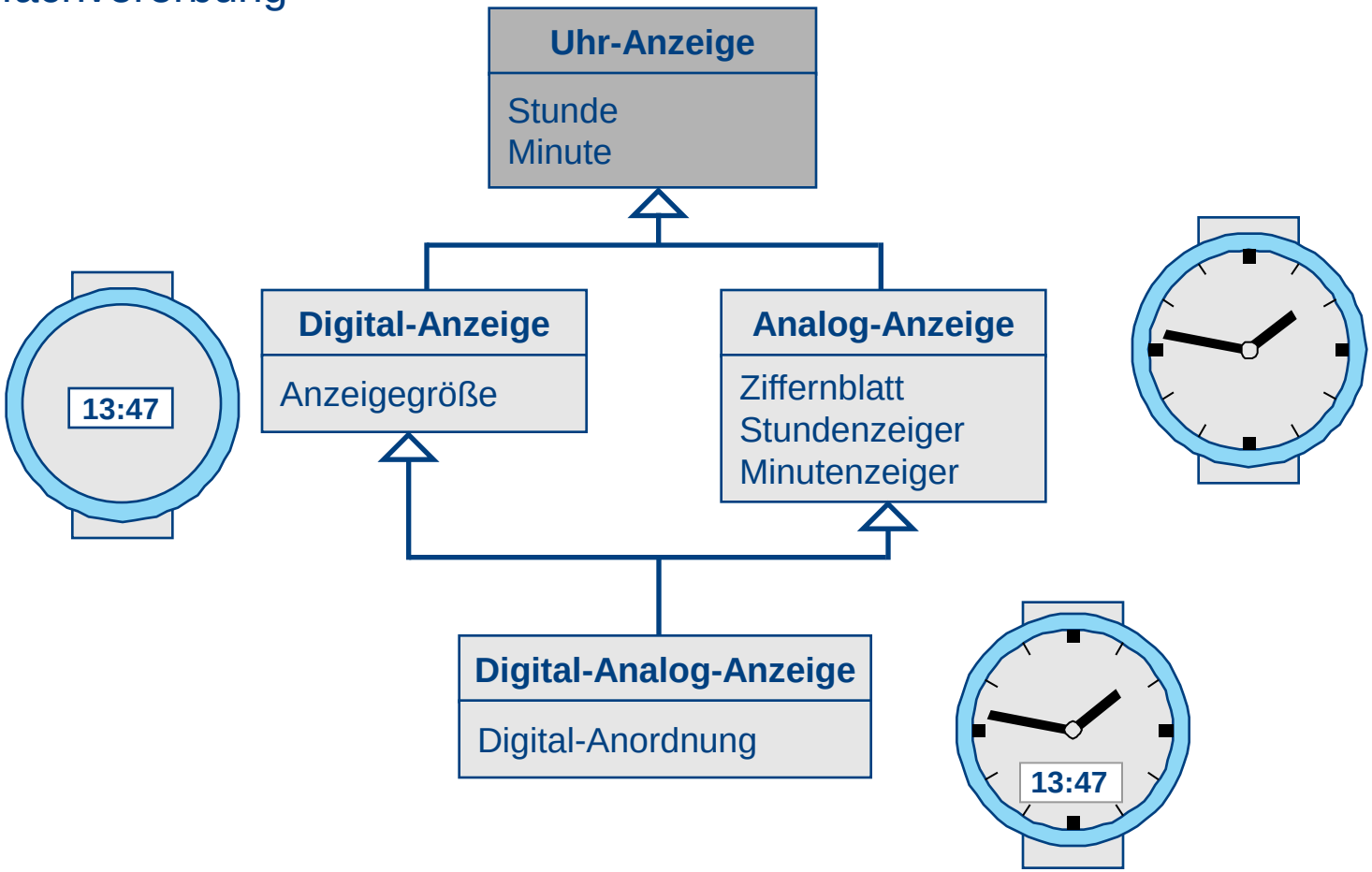
Dozent
Name Adresse Kontakt Geburtsdatum Biografie Honorar pro Tag Dozent seit
erstelle Adressaufkleber() link Seminartyp() link Veranstaltung()

- Fallstudie »Seminarorganisation«

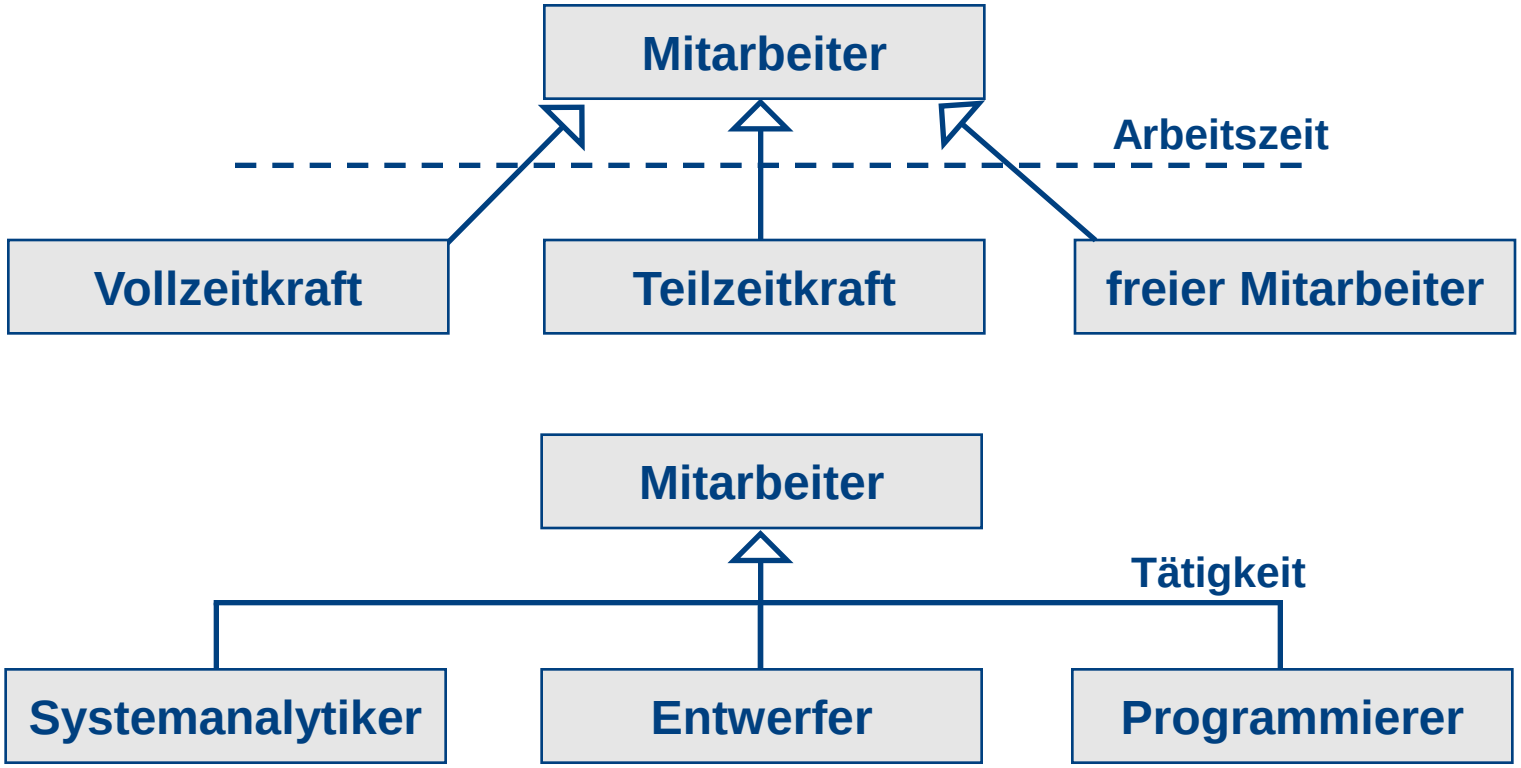


Vererbung

- Mehrfachvererbung



- Vererbungsstruktur mit Diskriminator

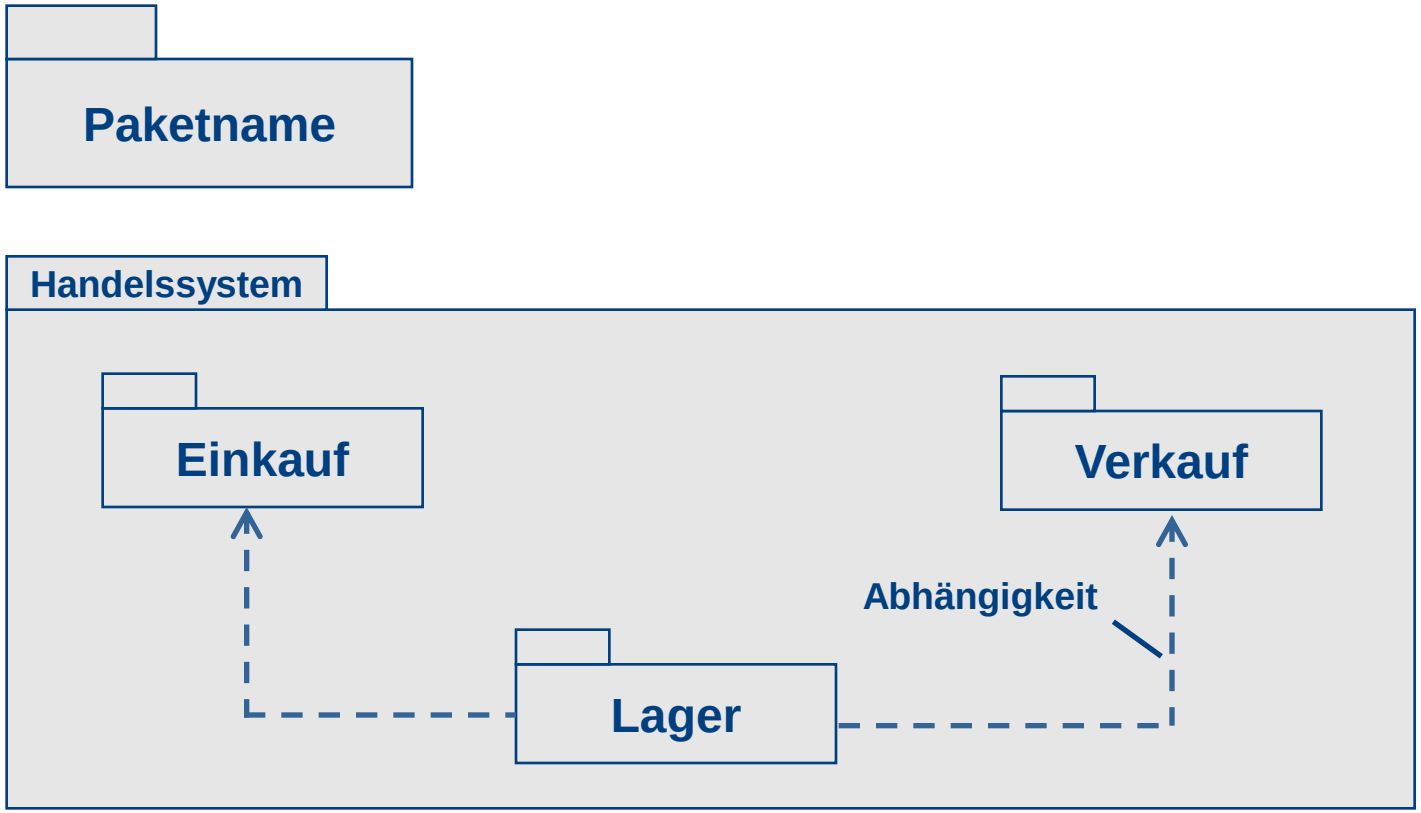


Pakete

- Pakete (*packages*)
  - Strukturierungsmechanismus
  - Erlauben es, Komponenten zu einer größeren Einheit zusammenzufassen
  - Paketbegriff allerdings nicht einheitlich definiert
  - Analoge Begriffe: Subsystem, *subject, category*
  - UML
    - Ein Paket gruppiert Modellelemente (z. B. Klassen) und Diagramme
    - Ein Paket kann selbst Pakete enthalten

Pakete

- UML-Notation



Pakete

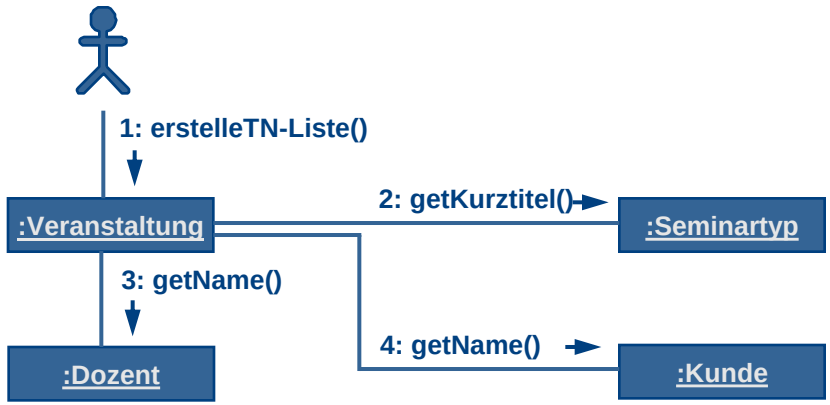
- Kriterien
  - Das Paket soll eine logische Einheit bilden, d. h. es soll so strukturiert sein, dass es
    1. den Leser durch das Modell führt
    2. einen Themenbereich enthält, der für sich allein betrachtet und verstanden werden kann
    3. Klassen enthält, die logisch zusammengehören, z. B. `Artikel, Lieferant und Lager`
    4. für sich entworfen und eventuell implementiert werden kann, wobei eine wohldefinierte Schnittstelle zur Umgebung vorhanden ist
- Die Schnittstelle soll
  1. Vererbungsstrukturen nur in vertikaler Richtung schneiden, d. h. zu jeder Unter-klasse sollen alle Oberklassen in dem Paket enthalten sein
  2. keine Aggregation durchtrennen
  3. möglichst wenig Assoziationen enthalten
- Die Kopplung zwischen den Klassen – innerhalb eines Pakets – soll möglichst groß und zwischen Paketen möglichst gering sein.
  - Jedes potenzielle Paket prüft man auf seine Kopplungseigenschaft

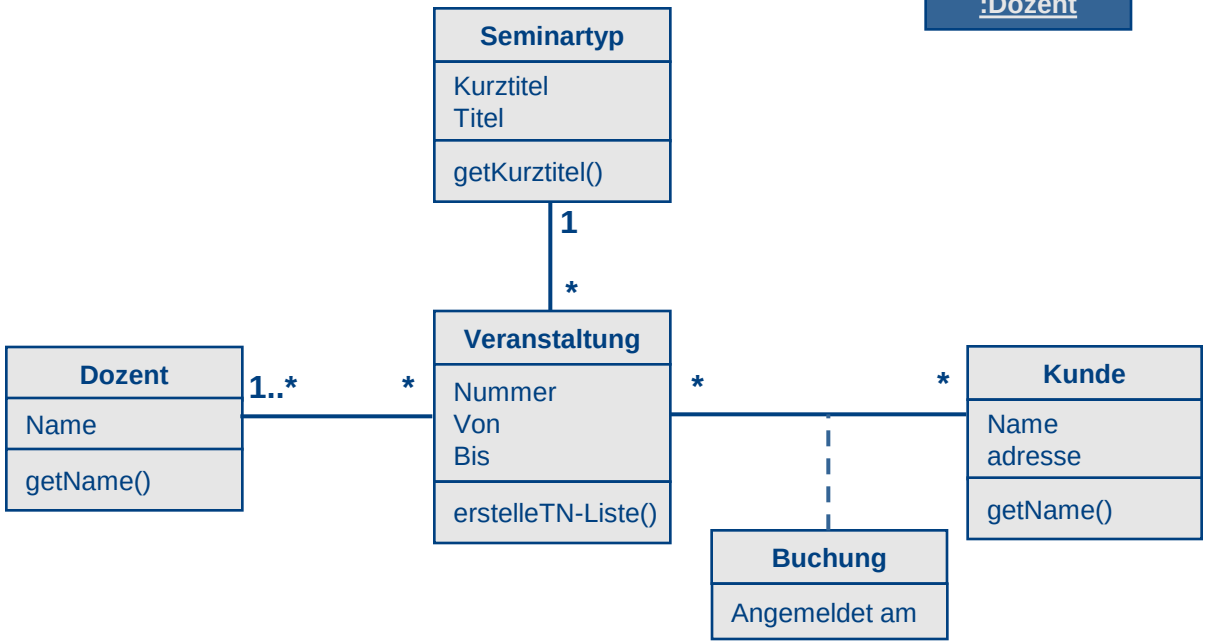
Botschaften

- Botschaft (*message*)
  - Aufforderung eines Senders (*client*) an einen Empfänger (*server, supplier*), eine Dienstleistung zu erbringen
  - Der Empfänger interpretiert diese Botschaft und führt eine Operation aus
  - Der Sender der Botschaft weiß nicht, wie die entsprechende Operation ausgeführt wird
  - Die Menge der Botschaften, auf die Objekte einer Klasse reagieren können, wird als Protokoll (*protocol*) der Klasse bezeichnet
  - Eine Botschaft löst eine Operation gleichen Namens aus

Botschaften

- Fallstudie »Seminarorganisation«





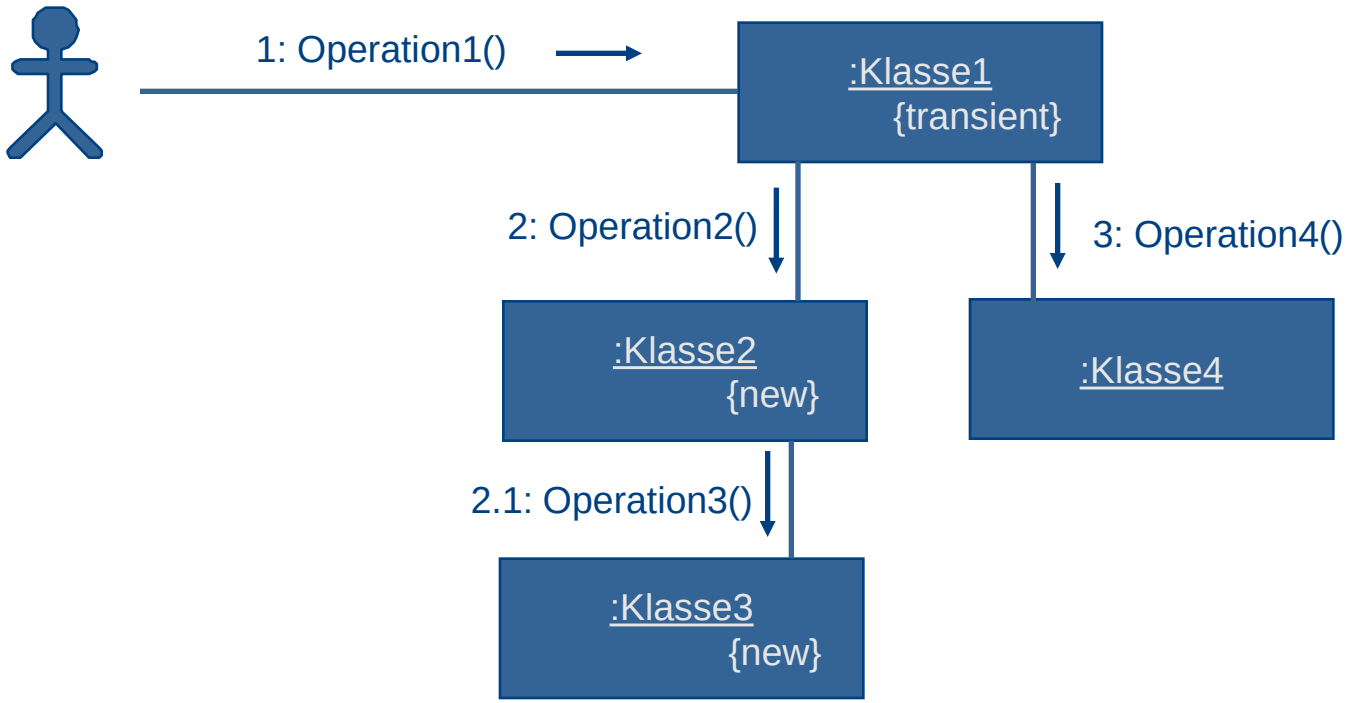
Szenarios

- Sequenz von Verarbeitungsschritten, die unter bestimmten Bedingungen auszuführen ist
- Diese Schritte sollen das Hauptziel des Akteurs realisieren und ein entsprechendes Ergebnis liefern
- Sie beginnen mit dem auslösenden Ereignis und werden fortgesetzt, bis das Ziel erreicht ist oder aufgegeben wird
- Ein Geschäftsprozess kann durch eine Kollektion von Szenarios dokumentiert werden
- Jedes Szenario wird durch eine oder mehrere Bedingungen definiert, die zu einem speziellen Ablauf des jeweiligen Geschäftsprozesses führen

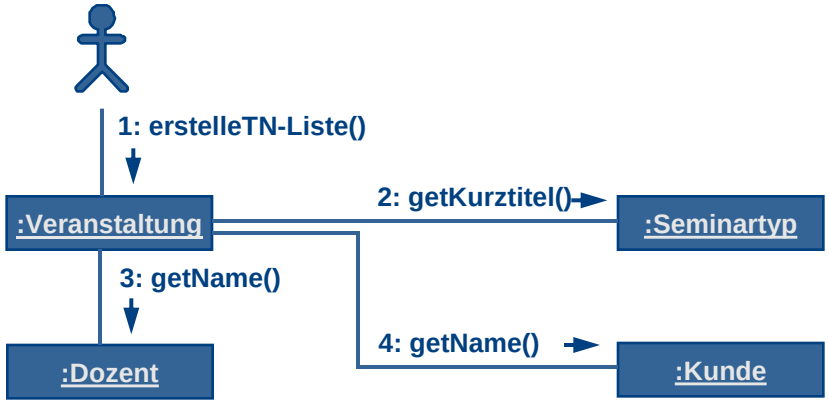
Szenarios
UML-Darstellung dynamischer Abläufe

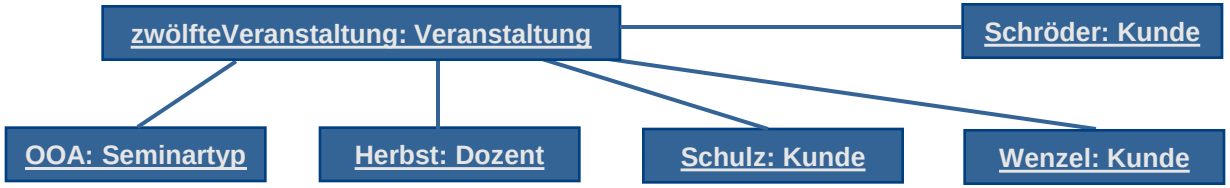
- Objektdiagramme *(object diagrams)*
- Kommunikationsdiagramme *(communication diagrams)*
- Sequenzdiagramme *(sequence diagrams)*

- Kommunikationsdiagramm

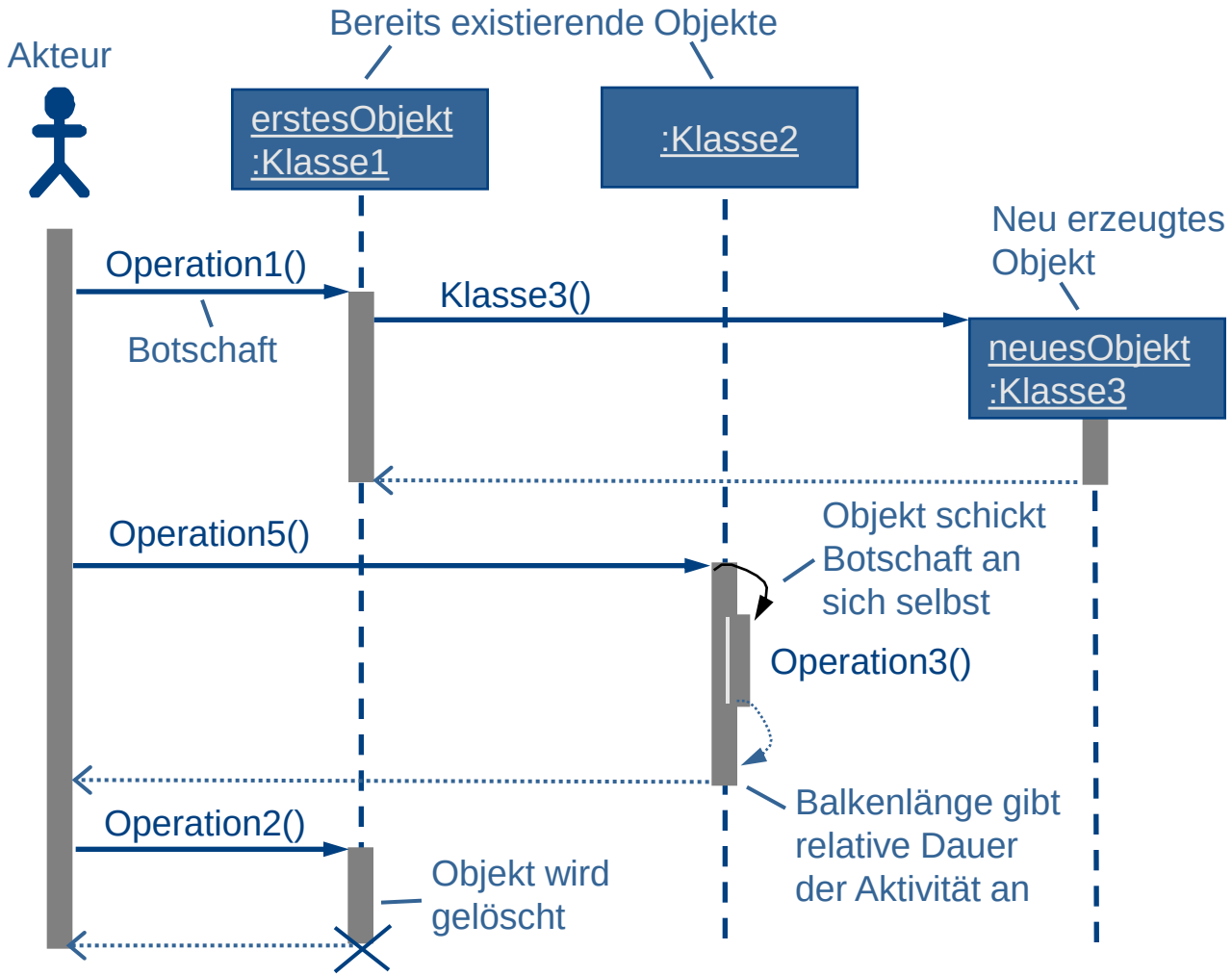


- Kommunikations- vs. Objektdiagramm

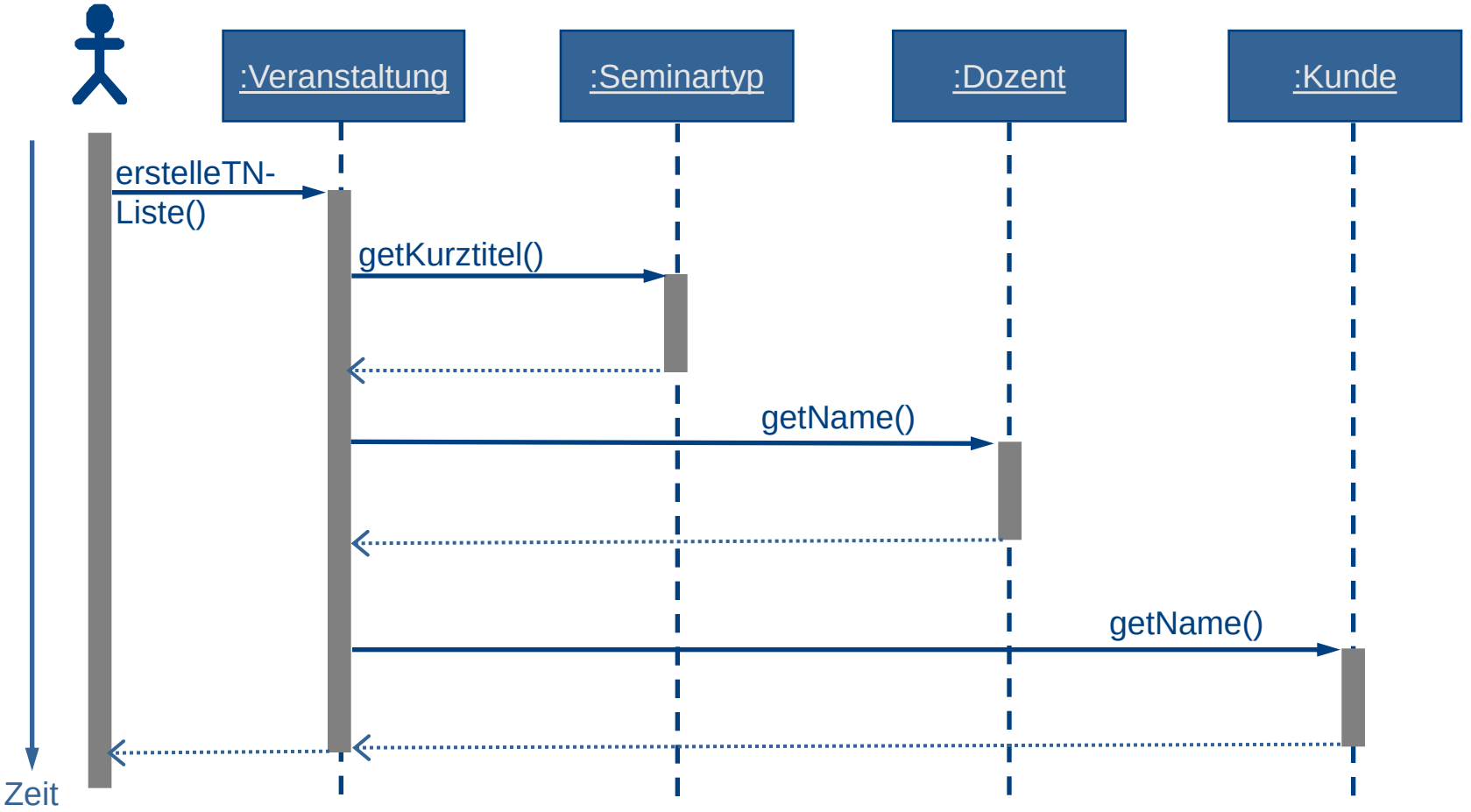




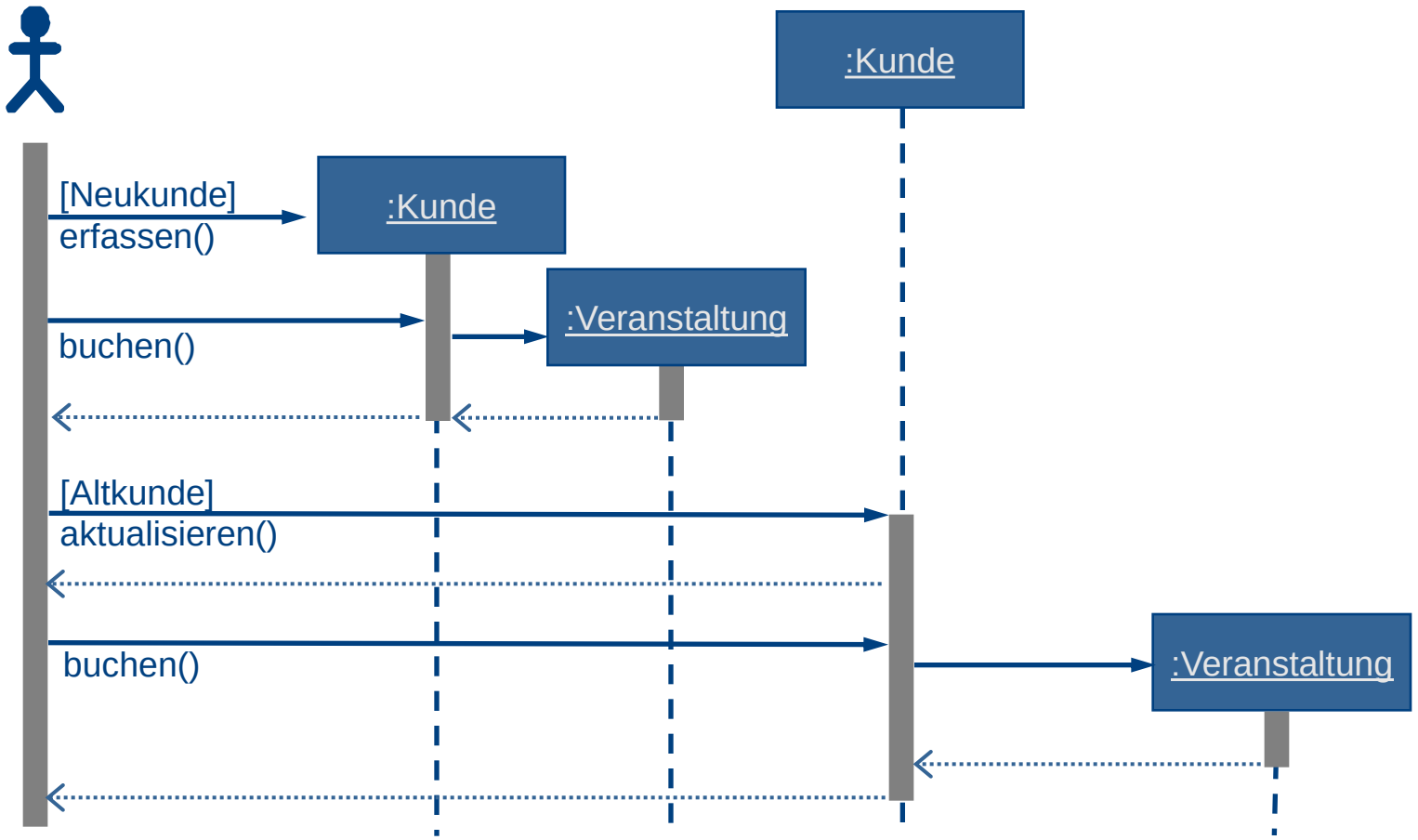
- Sequenz-diagramm



- Sequenzdiagramm: Beispiel



- Sequenzdiagramm: Beispiel mit Alternativen



- Sequenz- vs. Kooperationsdiagramm