

## 31) Strukturelle metamodelldrierte Modellierung

1

Prof. Dr. rer. nat. habil. Uwe Aßmann  
Institut für Software- und  
Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
Version 13-1.1, 08.06.13

- 1) Metamodelle
- 2) Merkmale
- 3) Beziehungen
  - 1) Aggregation und Komposition
- 4) Mehrfachvererbung



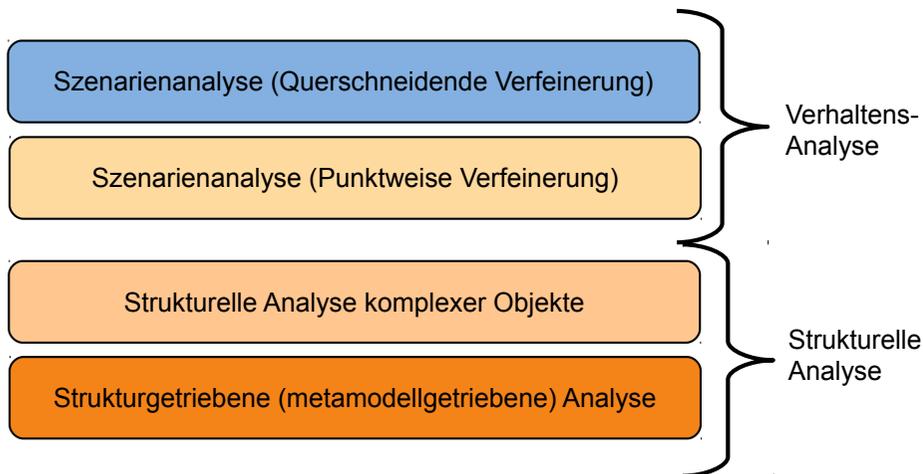
2

- ▶ Zuser, Kap. 7-9
- ▶ Störrle 5.2-5.5
- ▶ Balzert Kap. 6-7, 9-10



## Objektorientierte Analyseverfahren

3



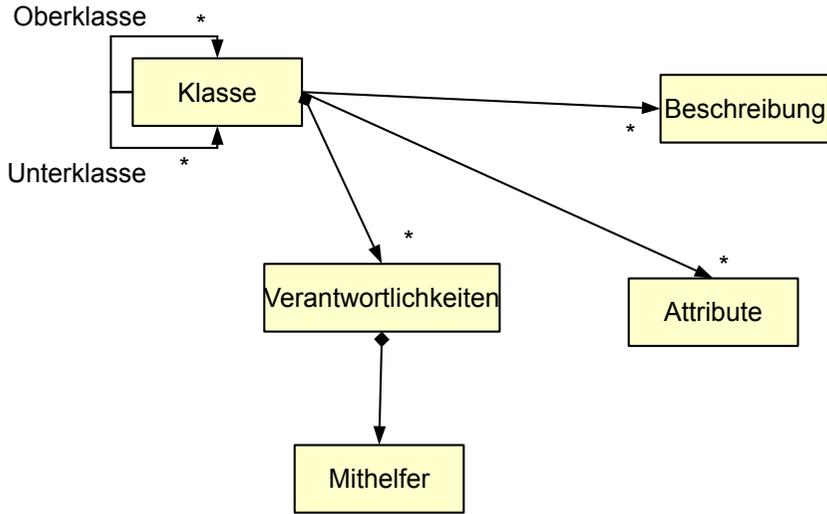
## 31.1 Strukturmodelle (Metamodelle)

4



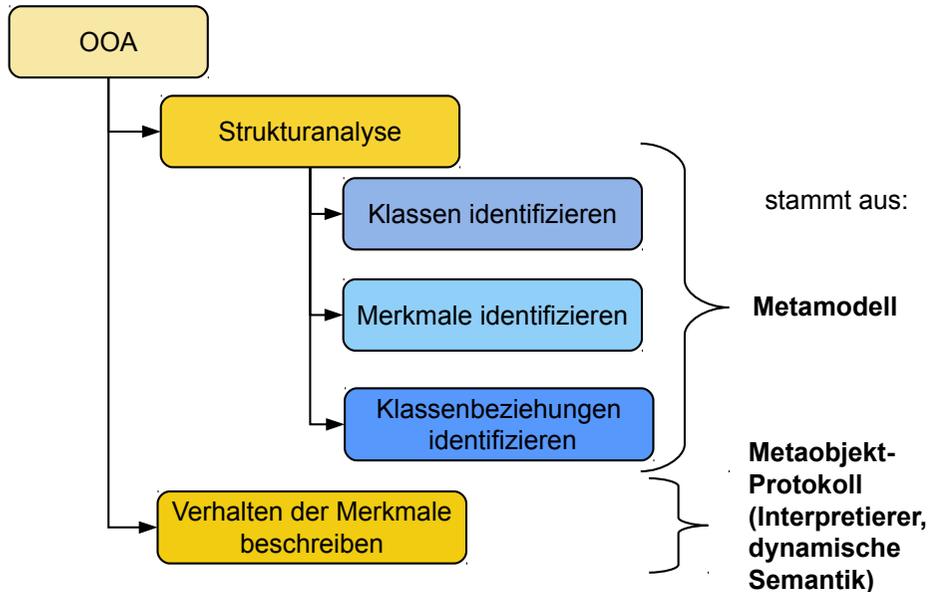
# Metamodelle

- 5
- Ein **Metamodell** beschreibt die *Struktur einer Menge von Modellen* mit Hilfe eines Klassendiagramms
  - Bsp.: Vereinfachtes Metamodell von CRC:



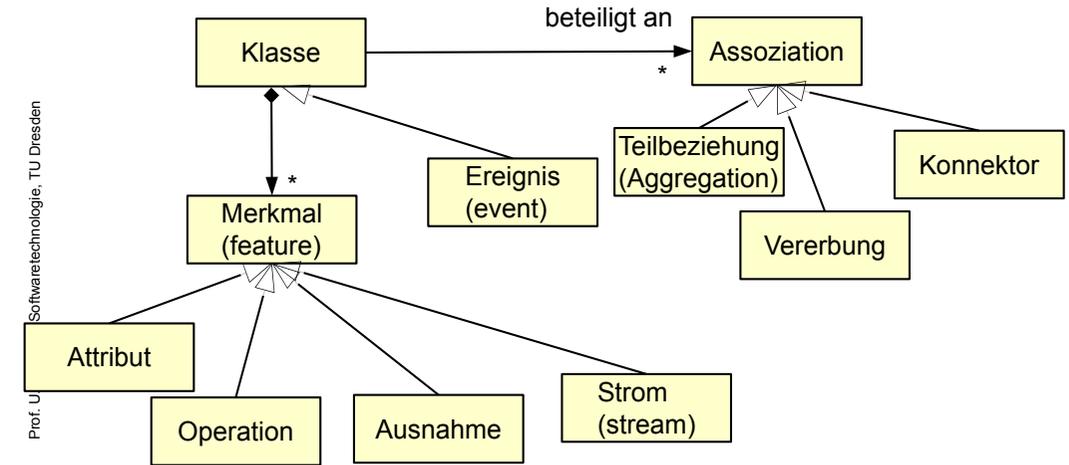
# Schritte der objektorientierten, metamodelldriven Analyse

- 8
- Ähnlich zur CRC Analyse, mit größeren Metamodellen, reicherer Struktur



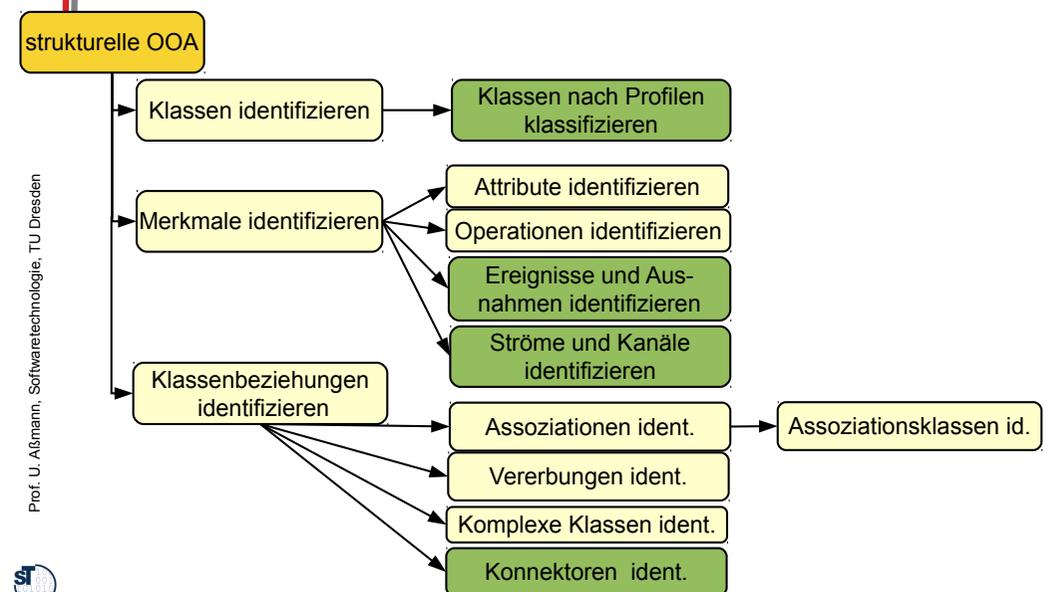
# Metamodelle

- 7
- Vereinfachtes Metamodell von UML:



# Schritte der strukturellen, metamodelldriven Analyse

- 9
- gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur



# Strukturgetriebene Analyse

10

Während einer **strukturgetriebenen Analyse** sucht man im Problembereich des Kunden nach Ausprägungen aller Begriffe des Metamodells.

- ▶ Die Schritte der metamodellgetriebenen Analyse gelten für alle Arten von Analysemodellen in der Produktdefinition
  - Anforderungsspezifikationen
  - Fachliche Modelle
    - Domänenmodell
    - Kontextmodell
    - Top-level-Architektur
- ▶ Im folgenden werden sie hauptsächlich für ein Analysemodell dargestellt, das alle obigen Teilmodelle integriert
  - Man kann aber Klassen den jeweiligen Teilmodellen zuordnen
- ▶ Wir werden noch weitere Analyseschritte für Kontextmodell und Top-level-Architektur kennenlernen.



# Verfeinerung von der Analyse zum Entwurf

12

Das **Analysemodell** besteht aus Fragmentgruppen (Fragmenten und generischen Fragmenten) von UML

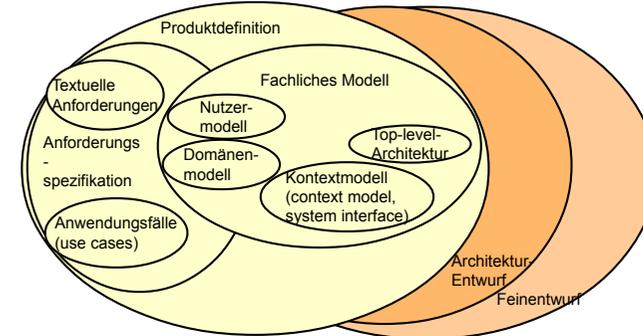
- ▶ Verfeinerungsschritte wurden bereits diskutiert
- ▶ Verfeinerungsschritte vom Analysemodell zum Entwurfsmodell
  - **Vervollständigung** (Ausfüllen, Elaboration) von Fragmenten zu Sätzen
    - Detaillierung von Fragmenten mit optionalen Einzelheiten
  - **Strukturierung**, Restrukturierung
  - **Abflachen** von Fragmenten (Flachklopfen, lowering, **Realisierung**): Ersetzen von ausdrucksstarken Konstrukten durch weniger ausdrucksstarke, implementierungsnähere
  - **Erhöhung Zuverlässigkeit**: Einziehen von qualitätssteigernden Fragmenten, z.B. Typen von Parametern, generische Typen



# Vom Analysemodell zum Entwurfsmodell

11

- ▶ Analysemodelle sind größer als Entwurfsmodelle.
  - Beim Übergang zum Entwurfsmodell wird das Analysemodell *verfeinert*, d.h. *ausgefüllt* und *detailliert*.
  - Das Analysemodell ist sozusagen das *Skelett* des Entwurfsmodells
    - Kontextmodell ergibt Top-Level-Architektur ergibt Architekturmodell ergibt Feinentwurf
    - Domänenmodell ergibt Materialentwurf
    - GUI-Prototyp ergibt GUI



# Was man vom Analysemodell abflachen muss

13

- ▶ Das Analysemodell nutzt verschiedene ausdrucksstarke Sprachkonstrukte (hier aUML), die nicht in der Programmiersprache (hier Java) vorhanden sind
- ▶ Beim Übergang vom Analysemodell zum Entwurfsmodell und Implementierungsmodell muss man diese abflachen (*Realisieren*, *Flachklopfen*, *lowering*)
- ▶ Strukturelle Eigenschaften
  - Relationen:
    - n-stellige Assoziationen, bidirektionale Assoziationen
    - Aggregationen, Kompositionen
    - Konnektoren
  - Objekte:
    - Mehrfachvererbung von Code
    - Aktive Objekte
    - Komplexe Objekte wie Unterobjekte (Rollen, Facetten, Teile)
    - Ströme und Kanäle
- ▶ Verhalten
  - States
  - Activities
  - Ereignisse auffangen und behandeln



## 31.2 Analyse von Merkmalen

14

(Wdh.)

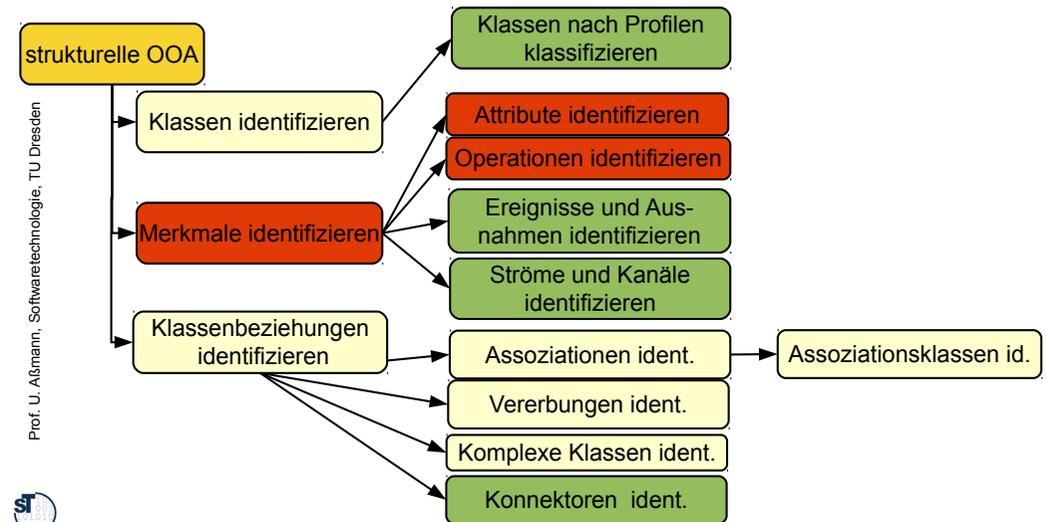
Softwaretechnologie, © Prof. Uwe Aßmann  
Technische Universität Dresden, Fakultät Informatik



## Schritte der strukturellen, metamodelldriven Analyse

15

- ▶ Idee: Laufe das Metamodell ab und finde Ausprägungen aller Begriffe
- ▶ gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur



Prof. U. Aßmann, Softwaretechnologie, TU Dresden



## Aufgabe der Analyse von Klassen

16

- ▶ im Domänenmodell:
  - Klassen können sowohl Objekte als Begriffe repräsentieren
  - Vererbung zwischen Begriffen bilden Begriffshierarchien (Taxonomien)
  - Assoziation
  - Aggregation und Komposition (Ganz-Teile-Beziehungen)
- ▶ in Kontextmodell und Top-level-Architektur:
  - Klassen repräsentieren Daten, die auf Kanälen fließen
  - Prozesse, die die Daten bearbeiten
- ▶ in funktionalen Anforderungen:
  - funktionale Anforderungen werden Operationen, die Klassen zugeordnet sind

Prof. U. Aßmann, Softwaretechnologie, TU Dresden



## Botschaften, Operation, Methode

17

- ▶ Eine **Botschaft (Message, Nachricht)** ist eine Nachricht eines Senders an ein Empfänger-Objekt, eine Operation auszuführen
- ▶ Eine **Rückmeldung** ist eine Botschaft, die ein Sender einer Botschaft als Antwort erhält
- ▶ Eine **Operation** koppelt eine Botschaft mit einer Reaktion und einer Rückmeldung: "a service that can be requested from an object to effect behaviour" (UML-Standard)
- ▶ Eine **Methode (method)** "is the implementation of an operation" (das "Wie" einer Operation)
  - "In den Methoden wird all das programmiert, was geschehen soll, wenn das Objekt die betreffende Botschaft erhält." (Middendorf/Singer)
  - *synchrone Operation / Methode*: der Sender wartet auf die Beendigung des Service
  - *asynchrone Operation*: ein Service mit Verhalten aber ohne Rückgabe, d.h. der Sender braucht nicht zu warten
- ▶ Ein **Push** ist eine Botschaft mit einem Datum an ein Objekt zur Ablage
- ▶ Ein **Pull** ist eine Botschaft an ein Objekt zur Ablage, das als Rückmeldung ein Datum mitführt

Prof. U. Aßmann, Softwaretechnologie, TU Dresden



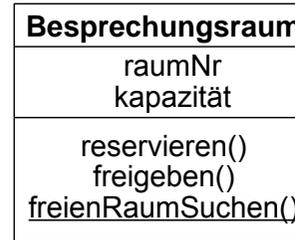
# Klassenattribute (Statische Attribute)

- 18
- ▶ Ein **Klassenattribut** beschreibt ein Datenelement, das genau einen Wert für die gesamte Klasse annehmen kann.
    - Es ist also ein Attribut des Klassenprototypen
  - ▶ **Notation:** Unterstreichung
  - ▶ **Implementierung:**
    - Implementierungsmuster Singleton
    - Klassenattribute und -operationen: Schlüsselwort **static**



# Klassenoperation (Statische Operation)

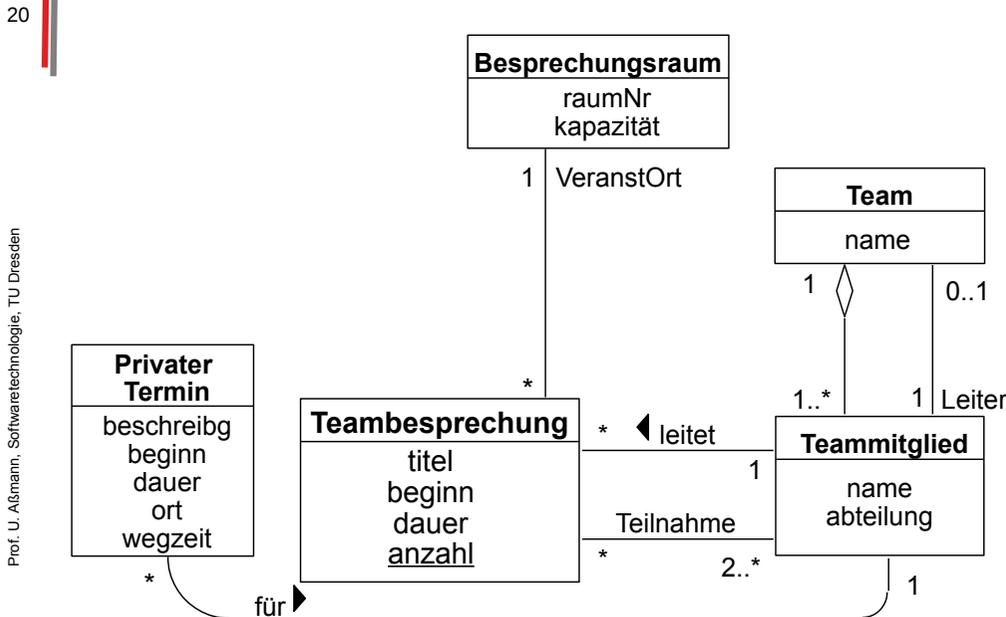
- 19
- ▶ **Definition** Eine **Klassenoperation** A einer Klasse K ist die Beschreibung einer Aufgabe, die nur unter Kenntnis der aktuellen Gesamtheit der Instanzen der Klasse ausgeführt werden kann. Gewöhnliche Operationen heißen auch **Instanzoperationen**.
  - ▶ **UML Notation:** Unterstreichung analog zu Klassenattributen.
  - ▶ **Java:** Die Methode main() ist statisch, und kann vom Betriebssystem aus aufgerufen werden



```

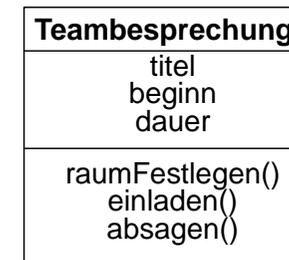
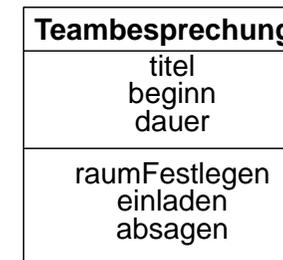
class Steuererklärung {
    public static main (String[] args) {
        Steuerzahler hans =
            new Steuerzahler();
        ...
    }
}
    
```

# Beispiel: Analysemodelle beginnen mit unvollständiger Information (Fragmente)



# Operationen

- 21
- ▶ **Def.:** Eine **Operation** einer Klasse K ist die Beschreibung einer Aufgabe, die jede Instanz der Klasse K ausführen kann.



- ▶ "Leere Klammern":
  - In vielen Büchern (und den Unterlagen zur Vorlesung) zur Unterscheidung von Attributnamen: raumFestlegen(), einladen(), absagen() etc.
  - Auf Analyseebene gleichwertig zu Version ohne Klammern

## Parameter und Datentypen für Operationen

- 22
- ▶ Detaillierungsgrad in der Analysephase gering
    - meist Operationsname ausreichend
    - Signatur kann angegeben werden
    - Entwurfsphase und Implementierungsmodell: vollständige Angaben der Typen ist nötig, um Fehler in der Programmierung früher zu erkennen (frühe oder statische Typisierung)
  - ▶ **Notation:**  
**Operation (Art Parameter:ParamTyp=DefWert, ...): ResTyp**
    - *Art* (des Parameters): **in**, **out**, oder **inout** (weglassen heißt in)
    - *DefWert* legt einen Default-Parameterwert fest, der bei Weglassen des Parameters im Aufruf gilt.
  - ▶ **Beispiel** (Klasse Teambesprechung):  
 raumFestlegen (in wunschRaum: Besprechungsraum): Boolean



## Sequentielle und parallele OO Sprachen

- 24
- ▶ Objekte kommunizieren mit Nachrichtenaustausch und reagieren auf den Empfang einer Nachricht mit dem Ausführen einer (oder mehrerer) Methode(n)
  - ▶ In einer **sequentiellen objekt-orientierten Sprache** setzt sich ein Aufruf an ein Objekt mit der Anfrage, eine Operation (einen Dienst) auszuführen, zusammen aus:
    - einer Aufruf-Nachricht (Botschaft, message),
    - einer synchronen Ausführung einer (oder mehrerer) Methoden und
    - einer Aufruf-Fertigmeldung (mit Rückgabe)
  - ▶ In einer **parallelen objekt-orientierten Sprache** setzt sich ein Aufruf an ein Objekt mit der Anfrage, eine Operation (einen Dienst) auszuführen, zusammen aus:
    - einer Aufruf-Nachricht (Botschaft, message),
    - einer *asynchronen* Ausführung von Methoden (der Sender kann parallel weiterlaufen)
    - einer Aufruf-Fertigmeldung (mit Rückgabe), die vom Sender ausdrücklich abgefragt werden muss



## Spezifikation von Operationen

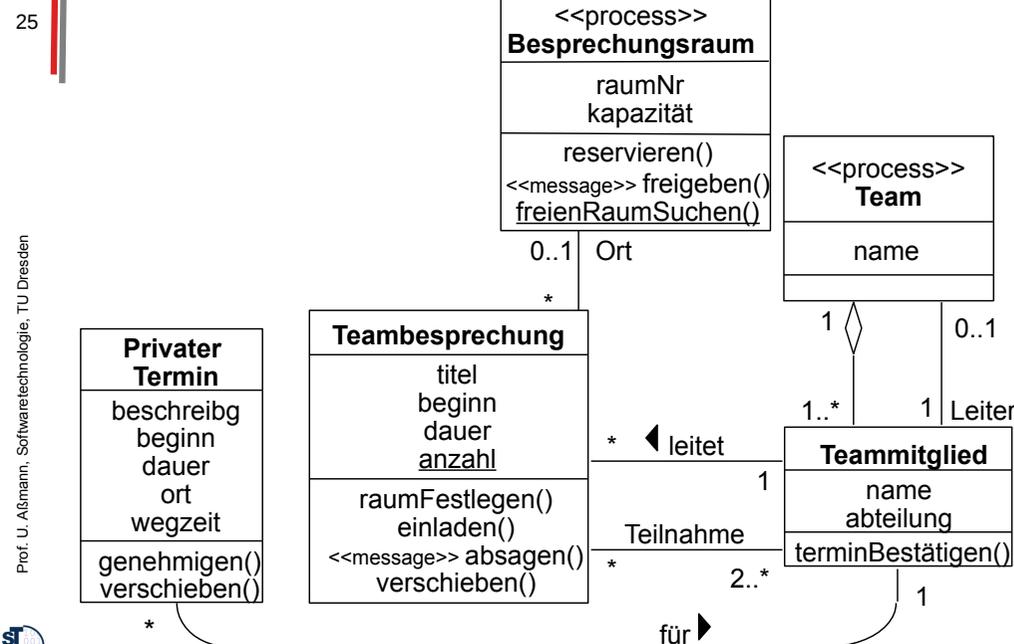
- 23
- ▶ **Definition** Die *Spezifikation* einer Operation legt das Verhalten der Operation fest, ohne einen Algorithmus festzuschreiben.

Es wird das **"Was"** beschrieben und noch nicht das **"Wie"**.

- ▶ Häufigste Formen von Spezifikationen:
  - Signaturen (Typen der Parameter und Rückgabewerte)
  - Text in natürlicher Sprache (oft mit speziellen Konventionen)
    - Oft in Programmcode eingebettet (Kommentare)
    - Werkzeugunterstützung zur Dokumentationsgenerierung, z.B. "javadoc"
  - Vor- und Nachbedingungen (Verträge, contracts)
  - Tabellen, spezielle Notationen
  - "Pseudocode" (Programmiersprachenartiger Text)
  - Zustandsmaschinen, Aktivitätendiagramme



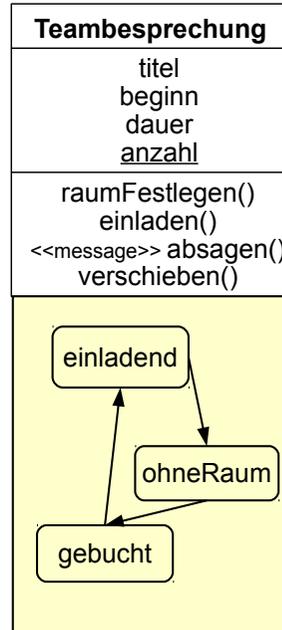
## Beispiel: Parallelität und verschiedene Arten von Operationen im Analysemodell



## Weitere Bemerkungen

26

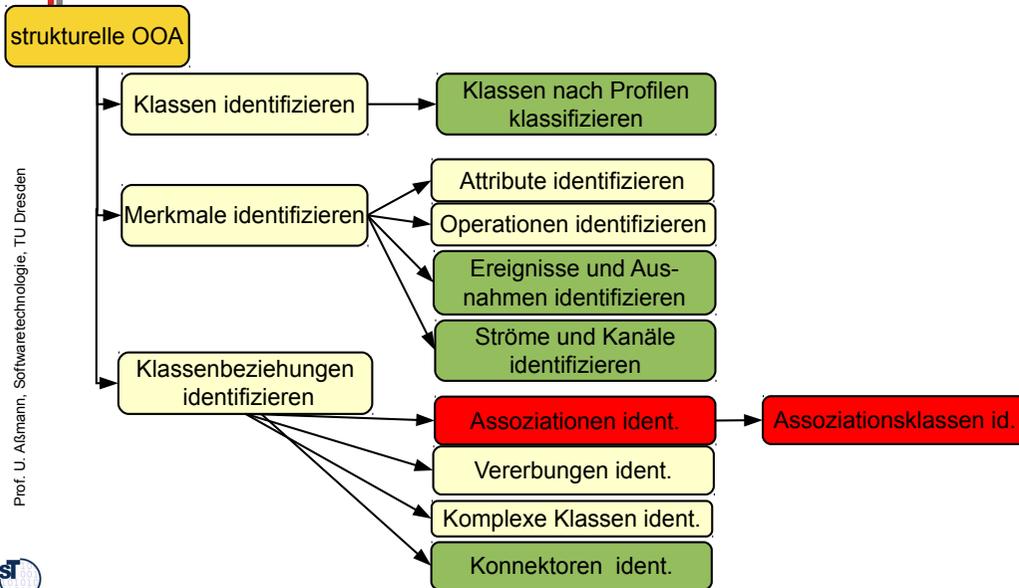
- ▶ Weitere Abteile (compartments) in Klassen:
  - Neben dem Attribut- und Operationsabteil können weitere Abteile angehängt werden
  - Verhaltensmodelle können in Abteilen hinzukommen (Statecharts, Aktivitätendiagramme)
  - Mit solchen Verhaltensbeschreibungen können *Objektlebenszyklen* beschrieben werden (siehe später)



## Schritte der strukturellen, metamodelgetriebenen Analyse

28

- ▶ gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur



## 31.3 Analyse von Klassenbeziehungen

27

## Assoziation (Klassenbeziehung)

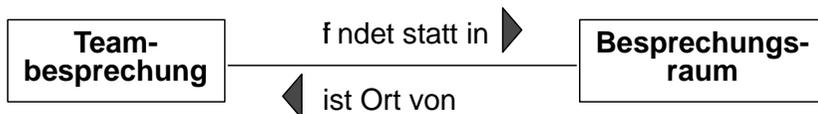
29

- ▶ Assoziationen stellen Relationen, Graphen oder Hypergraphen dar, d.h. bilden Abstraktionen von Referenzen
- ▶ **Definition:** Eine (binäre) **Assoziation (Beziehung, relationship) AS** zwischen zwei Klassen  $K1$  und  $K2$  beschreibt, daß die Instanzen der beiden Klassen in einer Beziehung zueinander stehen.
- ▶ **Semantik:** Für jedes Objekt  $O1$  der Klasse  $K1$  gibt es eine individuelle, veränderbare und endliche Menge AS von Objekten der Klasse  $K2$ , mit dem die Assoziation AS besteht. Analoges gilt für Objekte von  $K2$ .
- ▶ Mathematisch ist dies eine Relation zwischen dem Extent von  $K1$  und dem Extent von  $K2$

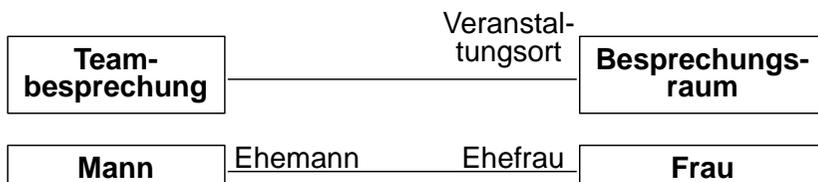


# Leserichtung und Assoziationsenden

- 30 Für Assoziationsnamen kann die **Leserichtung** angegeben werden.



- Ein Name für ein **Assoziationsende** bezeichnet die Assoziation aus der Sicht einer der teilnehmenden Klassen.
  - Ein Assoziationsende beschreibt die **Rolle** einer Klasse in einer Assoziation



# Multiplizität bei Assoziationen

- 32 **Definition** Die **Multiplizität (Kardinalität)** einer Klasse  $K1$  in einer Assoziation  $AS$  mit einer Klasse  $K2$  begrenzt die Anzahl der Objekte der Klasse  $K2$ , mit denen ein Objekt von  $K1$  in der Assoziation  $AS$  stehen darf. (**Weite** der Relation)

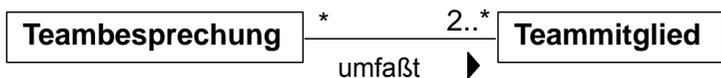


Multiplizität *Mult*:

- $n$  (genau  $n$  Objekte der Klasse  $K2$ )
- $n..m$  ( $n$  bis  $m$  Objekte der Klasse  $K2$ )
- $n1, n2$  ( $n1$  oder  $n2$  Objekte der Klasse  $K2$ )

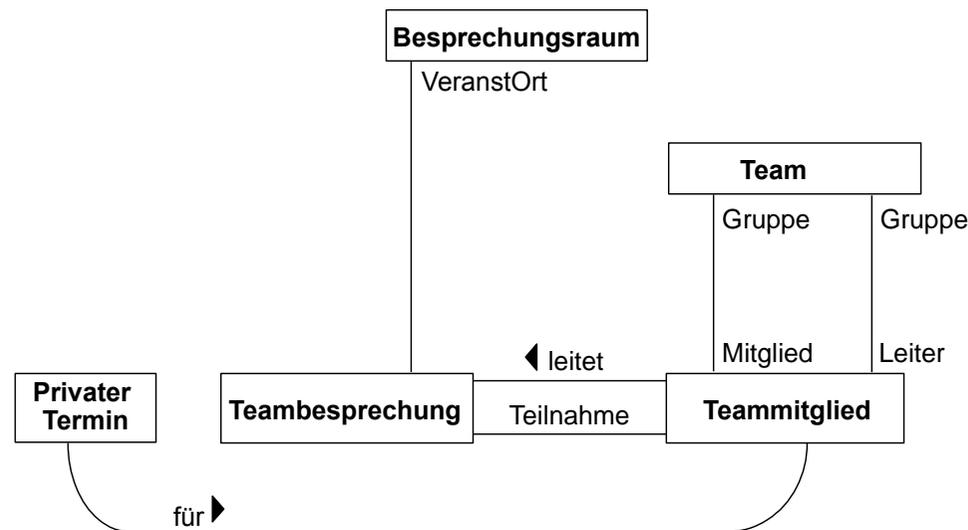
Zulässig für  $n$  und  $m$ :

- Zahlenwerte (auch 0)
- \*
- (d.h. beliebiger Wert, einschließlich 0)



# Beispiel: Assoziationen

- 31 Die Assoziationen eines Analysediagramms können "gelesen" werden:
  - Ein Team besitzt als Mitglieder die Teammitglieder, als Leiter ein Teammitglied. Teammitglieder gehören zur Gruppe des Teams.



# Multiplizitäten bestimmen durch "Vorlesen"

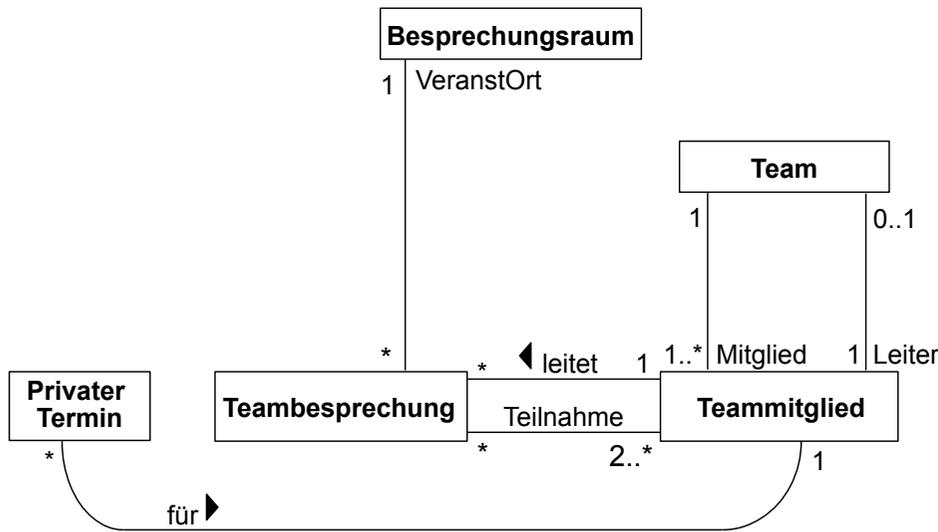


- Von links nach rechts:
  - "Jede Teambesprechung findet statt in (**wie vielen?**) Besprechungsräumen."
  - "Jede Teambesprechung findet statt in genau 1 Besprechungsraum."
- Von rechts nach links:
  - "Jeder Besprechungsraum ist Ort von (**wie vielen?**) Teambesprechungen."
  - Jeder Besprechungsraum ist Ort von 0 oder mehreren Teambesprechungen."
- Die Multiplizitätsbeschränkung steht an der Klasse, für die die Anzahl der Teilnehmer an der Assoziation beschränkt werden.



## Beispiel: Multiplizitäten

34



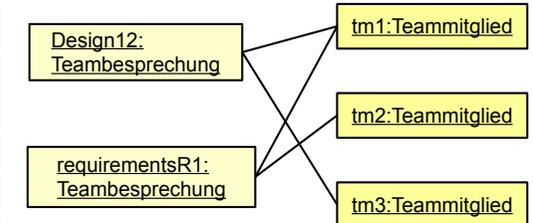
Prof. U. Alßmann, Softwaretechnologie, TU Dresden

## Semantik (bidirektionaler) Assoziationen

35

- Ein **Extent** einer Assoziation (auch *Relation, Graph*) ist ähnlich einer *Tabelle*:

Teilnahme-Assoziation	
Teambesprechung	Teammitglied
design12	tm1
design12	tm3
requirementsR1	tm1
requirementsR1	tm2

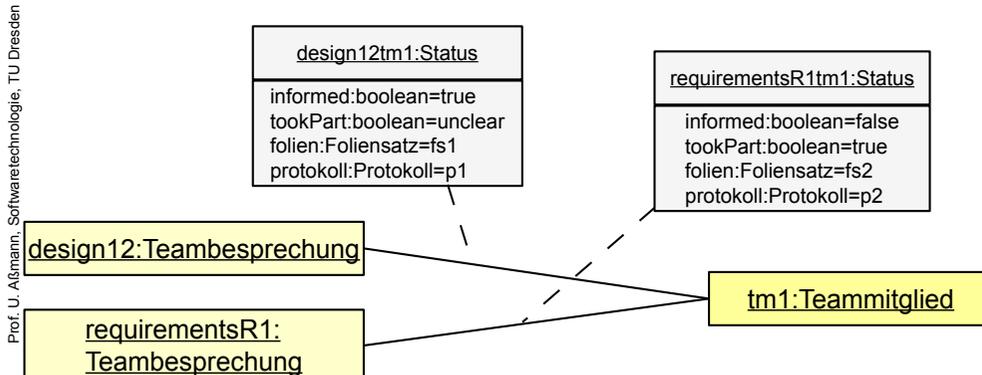


- Von einem beteiligten Objekt aus betrachtet, gibt eine Assoziation eine *Menge* von assoziierten Objekten an (**Nachbarmenge**):
  - Objekt **design12: Teambesprechung**  
Teammitglied-Objekte in Teilnahme-Assoziation: {**tm1**, **tm3**}
  - Objekt **tm1: Teammitglied**  
Teambesprechung-Objekte in Teilnahme-Assoziation: {**design12**, **requirementsR1**}

Prof. U. Alßmann, Softwaretechnologie, TU Dresden

## Assoziationsattribute

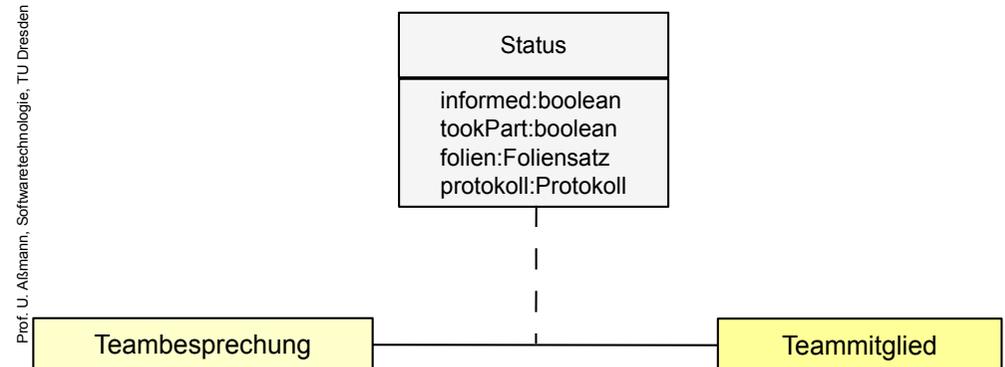
- Oft tragen Assoziationen (Tupel der Relationen, Kanten des Graphen) *Kantenattribute*
  - Diese werden durch *Kantenobjekte* modelliert:



Prof. U. Alßmann, Softwaretechnologie, TU Dresden

## Assoziationsklassen

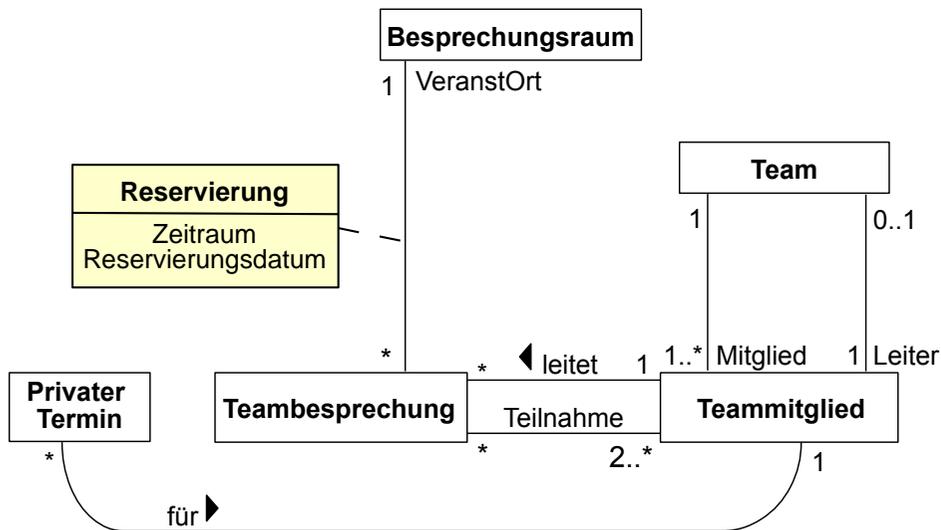
- Definition:** Eine *Assoziationsklasse* beschreibt die Kantenobjekte einer Assoziation.
- Assoziationsklassen werden benötigt, wenn Wissen darstellt werden soll, das für jede Kante (Tupel) *unterschiedlich* ist, d.h. Wissen über die Assoziation der Objekte darstellt



Prof. U. Alßmann, Softwaretechnologie, TU Dresden

# Beispiel: Reservierung als Assoziationsklasse

38



Prof. U. Alßmann, Softwaretechnologie, TU Dresden

# 21.3.2 Realisierung von Assoziationen

39

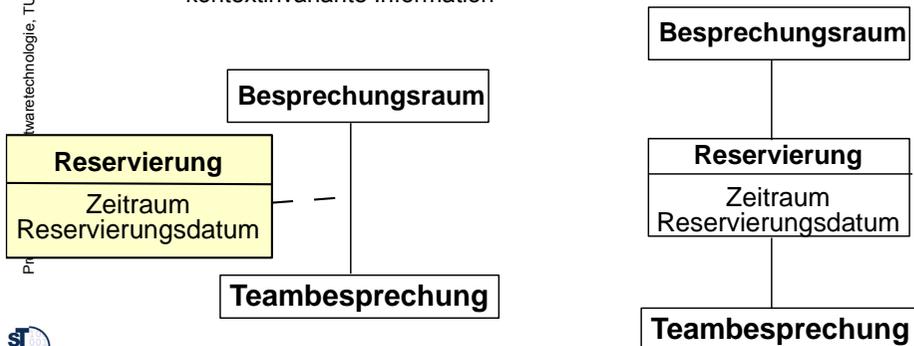
- ▶ 1) Realisierung durch Assoziationsklassen bzw. Zwischenklassen, die Assoziationen repräsentieren:
    - Geeignete Datenstrukturen erforderlich
      - "Beidseitige" Abfragemöglichkeit
      - Abflachung zu normalen Klassen nötig
  - ▶ 2) Realisierung durch Graphen einer Graph-Bibliothek (siehe Kap. 23)
  - ▶ 3) Realisierung von Assoziationen durch explizite Rollenklassen
  - ▶ 4) Realisierung durch zwei gerichtete Assoziationen
    - Redundanz: zusätzlicher Speicheraufwand, Gefahr von Inkonsistenzen
    - Aber: schnelle Navigation
  - ▶ 5) Realisierung nur in einer Richtung:
    - Gibt nicht die volle Semantik des Modells wieder
    - Abhängig von Benutzung (Navigation) der Assoziation
- ▶ **Genauere Entscheidung erst im Entwurf !**

Prof. U. Alßmann, Softwaretechnologie, TU Dresden

## 21.3.2.1 Realisierung von Assoziationsklassen mit "Zwischenklassen"

40

- ▶ Assoziationsklassen können im Implementierungsmodell in normale relationale „Zwischen“-Klassen abgeflacht werden:
  - Attribut "Reservierungsdatum" für eine Raumreservierung wird für Assoziation benötigt (z.B. um Reservierungskonflikte aufzulösen).
  - Die Klasse "Reservierung" wird in die bestehende Assoziation eingefügt und "zerlegt" sie in zwei neue Assoziationen.
  - Die Klasse "Reservierung" trägt die kontextspezifische Information, "Besprechungsraum" und "Teambesprechung" fokussieren sich auf die kontextinvariante Information



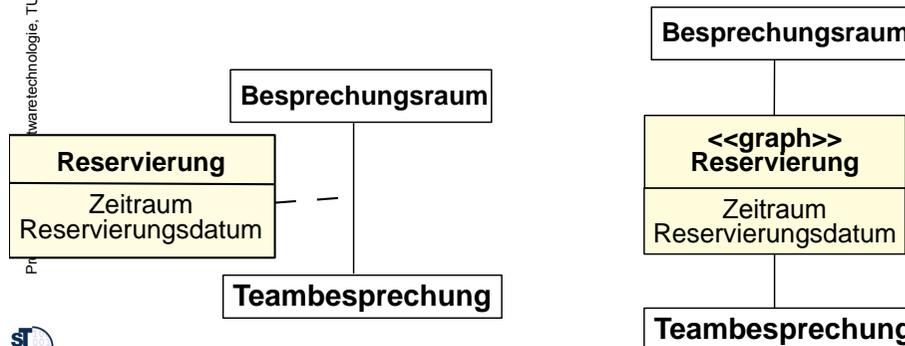
Softwaretechnologie, TU Dresden

## Realisierung von Assoziationsklassen

41

- ▶ Assoziationsklassen können im Implementierungsmodell in Graphklassen abgeflacht werden:

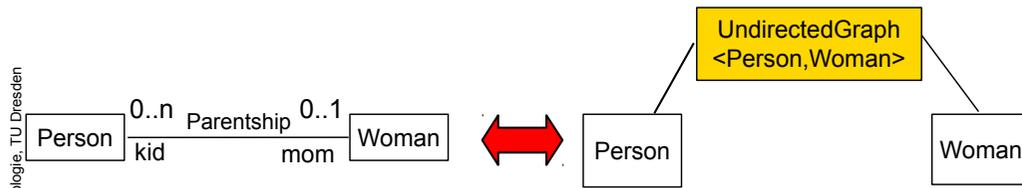
Softwaretechnologie, TU Dresden



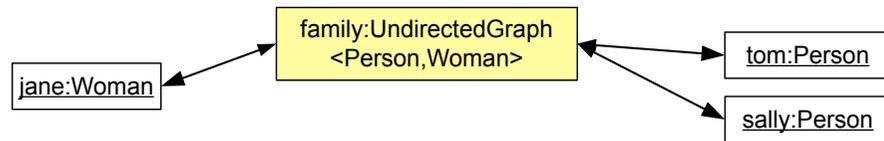
Softwaretechnologie, TU Dresden

## 21.3.2.2 Realisierung von bidirektionalen Assoziationen durch Graphklassen

- 42
- Auch bidirektionale Assoziationen können durch Graphklassen realisiert werden

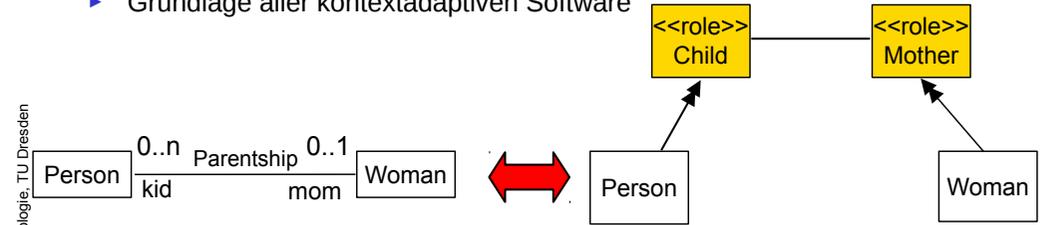


Laufzeit: zwei Referenzen zwischen Graphobjekt und den assoziierten Objekten

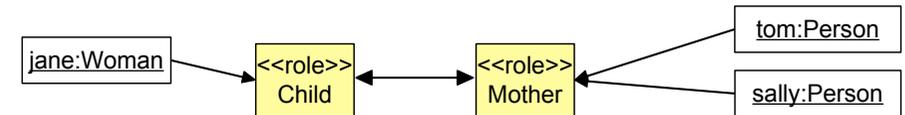


## 21.3.2.3 Realisierung von bidirektionalen Assoziationen durch Rollenklassen

- 43
- Assoziationen können durch Rollenklassen realisiert werden, die das kontextspezifische Verhalten tragen
  - Person, Woman tragen das kontextinvariante Verhalten
  - Grundlage aller kontextadaptiven Software

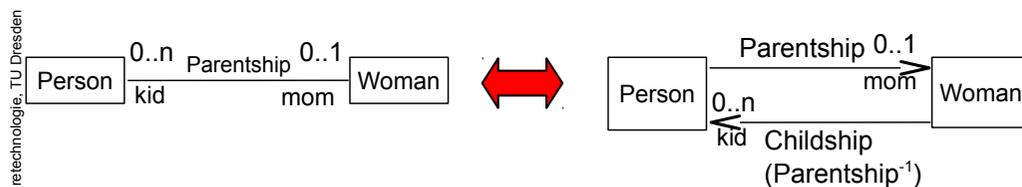


Laufzeit: Referenzen zwischen Rollenobjekten und den assoziierten Objekten

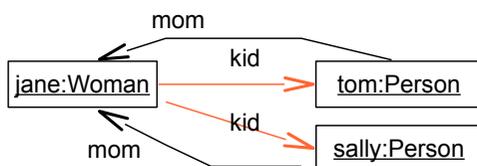


## 21.3.2.4 Realisierung von bidirektionalen Assoziationen durch unidirektionale

- 44
- Bidirektionale Assoziationen können, falls sie keine Attribute tragen, in gerichtete Assoziationen umgewandelt werden
  - Realisierung in jUML durch Einführung von *gerichteten* Assoziationen

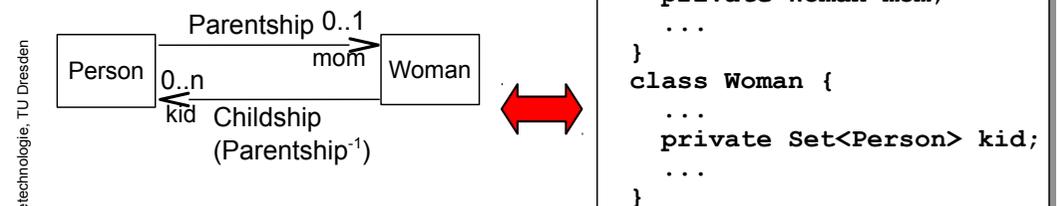


Laufzeit:

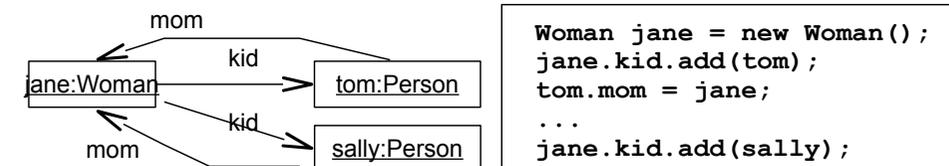


## Realisierung von bidirektionalen Assoziationen durch unidirektionale

- 45
- Realisierung von jUML in Java durch Arrays oder Collections (s. Kap. 21-collections)



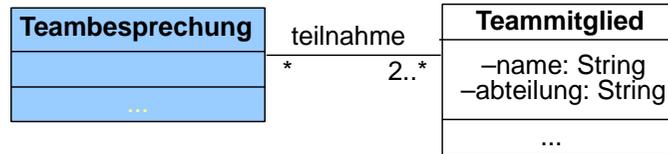
```
class Person {
    ...
    private Woman mom;
    ...
}
class Woman {
    ...
    private Set<Person> kid;
    ...
}
```



```
Woman jane = new Woman();
jane.kid.add(tom);
tom.mom = jane;
...
jane.kid.add(sally);
```

## Realisierung von bidirektionalen Assoziationen durch unidirektionale: Beispiel UML/Java

- 46 ▶ Achtung: fixe Multiplizitäten müssen in Java durch Programmierung kontrolliert werden, z.B. in Konstruktoren (s. Kap. 21-collections)



```

class Teambesprechung {
    private Teammitglied[] teilnahme;
    ...
    public Teambesprechung (
        Teammitglied[] teilnehmer) {
        this.teilnahme = teilnehmer;
    }
}

class Teammitglied {
    private String name;
    private String abteilung;
    private Teambesprechung[] teilnahme;
    public Teammitglied (
        Teambesprechung[] teilnahme) {
        if (teilnahme.size() < 2) error();
        this.teilnahme = teilnahme;
    }
}
    
```

## Aggregation (has-a)

- 48 ▶ **Definition:** Wenn eine Assoziation den Namen „hat-ein“ oder "besteht-aus" tragen könnte, handelt es sich um eine **Aggregation** (Ganzes/Teile-Relation).
- Eine Aggregation besteht zwischen einem *Aggregat*, dem *Ganzen*, und seinen *Teilen*.
  - Die auftretenden Aggregationen bilden auf den Objekten immer eine transitive, antisymmetrische Relation (einen gerichteten zyklensfreien Graphen, *dag*).
  - Ein Teil kann zu mehreren Ganzen gehören (*shared*), zu einem Ganzen (*owns-a*) und exklusiv zu einem Ganzen (*exclusively-owns-a*)



Lies: „Team hat ein Teammitglied“

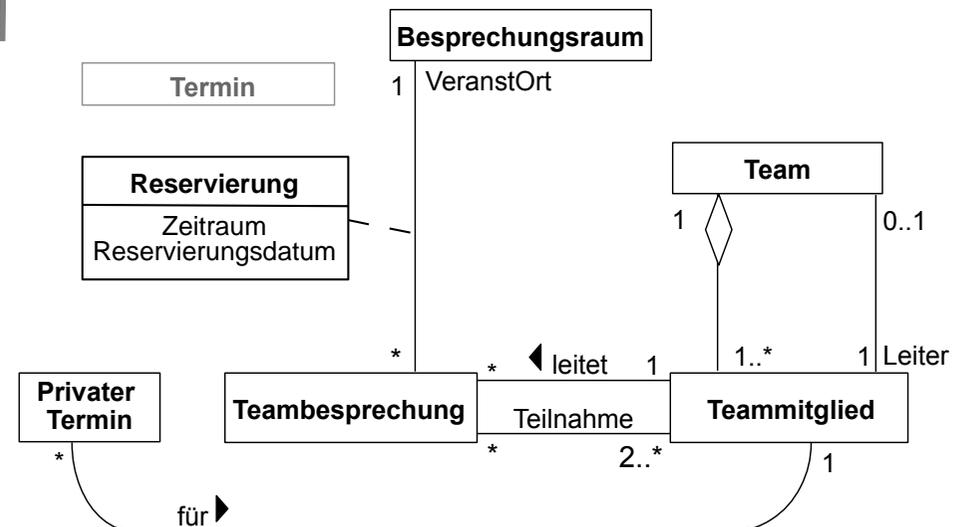
## 31.3.2 Aggregation, Komposition, Rollenspiel als Integrationsoperationen

47

Softwaretechnologie, © Prof. Uwe Alßmann  
Technische Universität Dresden, Fakultät Informatik

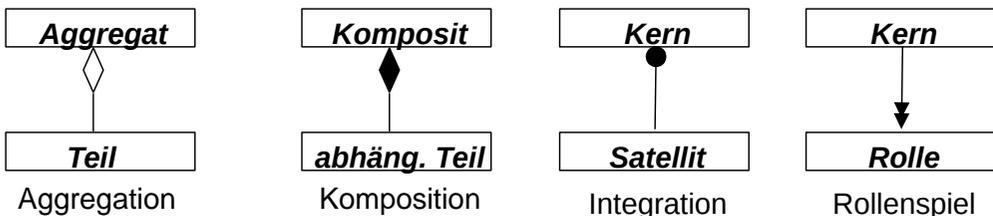
## Beispiel: Assoziationen und Aggregationen

49



# Komposition (owns-a)

- 50
- ▶ **Definition:** Ein Spezialfall der Aggregation ist die **Komposition** zwischen einem *Komposit* und seinen *Teilen*.
    - Ein Objekt kann Teil höchstens eines Komposits sein (Eigentums-Relation *exclusively-owns-a*).
    - Das Teil ist *abhängig* vom Komposit (*dependent part*).
      - Das Komposit hat die alleinige Verantwortung für Erzeugung und Löschung seiner Teile (gleiche Lebenszeit)
  - ▶ Def: Aggregation, Komposition und Rollenspiel sind Spezialfälle der **Integration** von Unterobjekten (integrates-a)

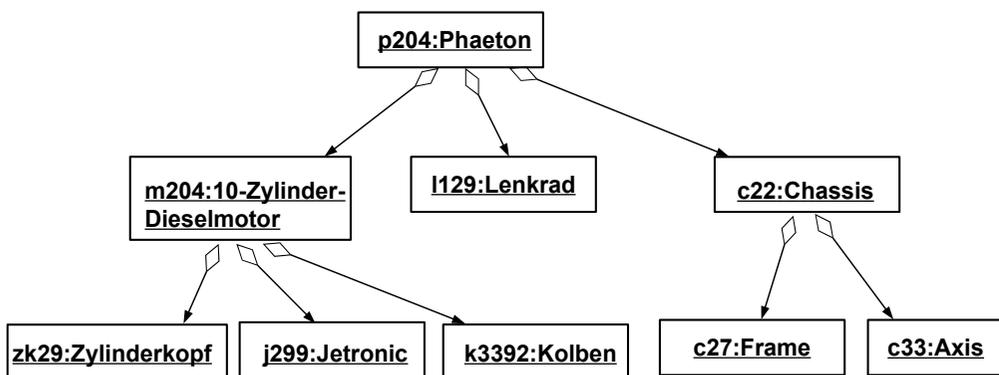


# Komposite Objekte in Anwendungen

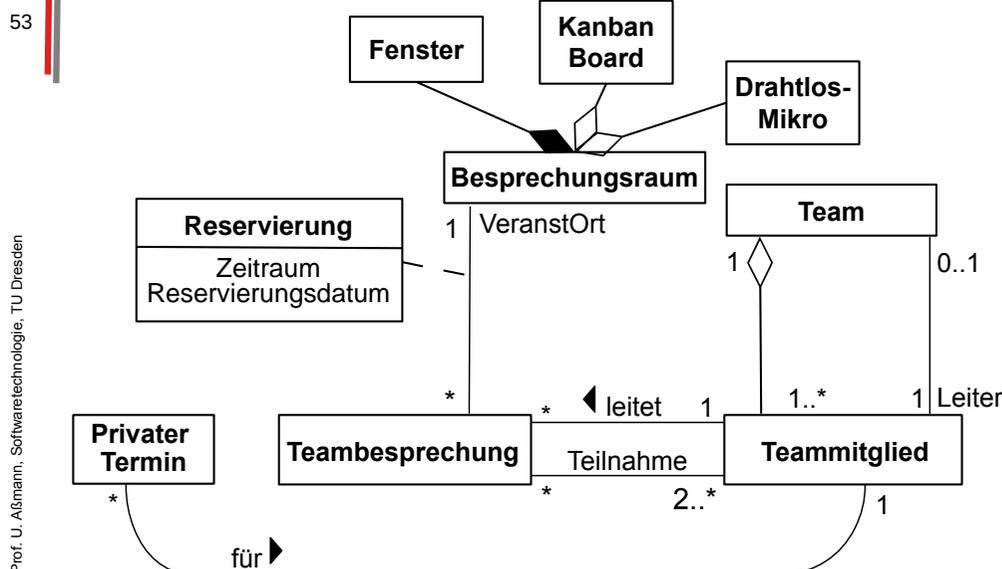
- 51
- ▶ Viele Daten, die in Anwendungen gehandhabt werden, sind **komposite Objekte**
  - ▶ Typischerweise werden *Materialien* von Anwendungen mit kompositen Objekten darstellt
    - Materialien gehören zur Datenhaltungsschicht
    - Materialien sind oft komplex
  - ▶ Beispiele:
    - Produktionsplanungssysteme verwalten
      - *Produkte*. Die Teile eines Produkts werden in Stücklisten (eigentlich Stückbäume) verwaltet
      - *Fabriken*. Die Teile und Maschinen einer Fabrik werden modelliert
    - Geschäftsprozesssoftware verwaltet *Dokumente von Geschäftsvorgängen* (Bestellungen, Rechnungen, Löhne, Mitarbeiter...), die ebenfalls komposite Objekte darstellen
    - *Geschäftsobjekte (business objects)*, Objekte des Domänenmodells, sind oft komposit
      - Komposition ist wichtig im Domänenmodell und Datenhaltung!

# Beispiel: Komposite Objekte als Stücklisten

- 52
- ▶ Produktionsplanungssysteme (PPS) verwalten Produkte.
    - Die Teile eines Produkts werden in *Stücklisten* (eigentlich *Stückbäume*) verwaltet
    - Stückliste eines Phaeton (alle Teile sind lebenslang nummeriert, um verfolgbar zu sein)

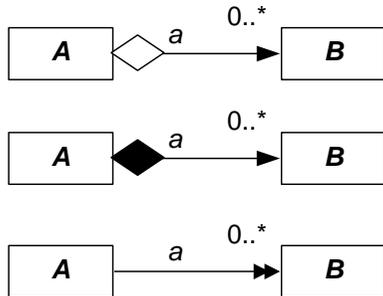


# Beispiel: Assoziationen und Aggregationen



# Realisierung von Aggregation und Komposition in Java

- 54
- Aggregationen stellen azyklische Graphen dar und können daher genau wie Assoziationen abgebildet werden
  - Komposition und Rollen werden in gleicher Weise abgebildet
    - Daher gehen in Java die Analyseinformationen verloren!
  - Überlege auch den Einsatz des Entwurfsmusters Composite

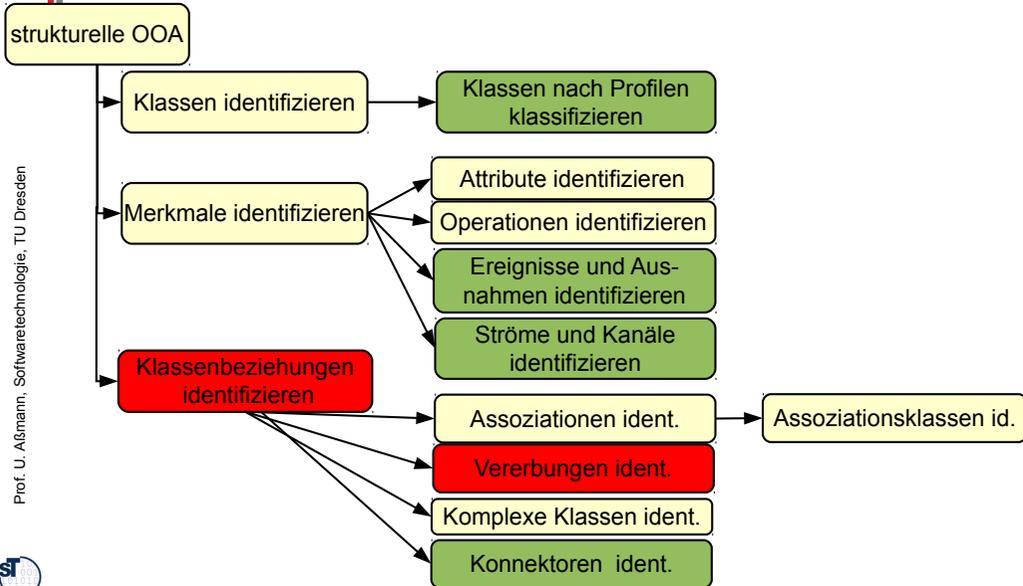


```
class A {
    ...
    Collection<B> a;
    // B[] a;
    ...
}
```



# Schritte der strukturellen, metamodelgetriebenen Analyse

- 56
- gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur



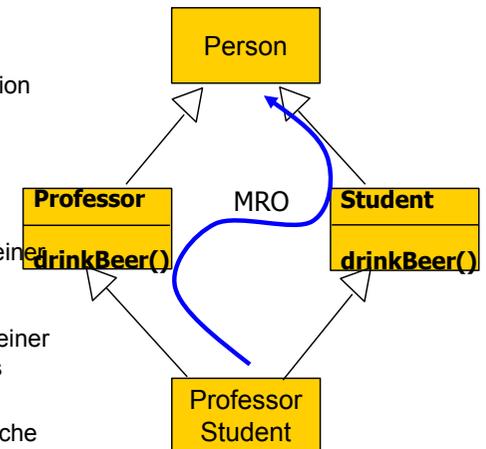
# 31.4 Mehrfachvererbung zwischen Klassen

55



# Mehrfachvererbung (Multiple Inheritance)

- 57
- Mehrfachvererbung
    - Eine Klasse kann mehrere Oberklassen besitzen
    - Dadurch wird die Vererbungsrelation ein gerichteter azyklischer Graph (dag)
  - Eine Klasse kann ein Merkmal mehrmals erben
    - Daher muss die Merkmalsuche einer Strategie gehorchen, der Merkmalsauflösungsordnung (method-resolution-order, MRO), einer Navigationsstrategie, die aufwärts nach Merkmalen sucht
    - Oft links-nach-rechts-aufwärts-Suche

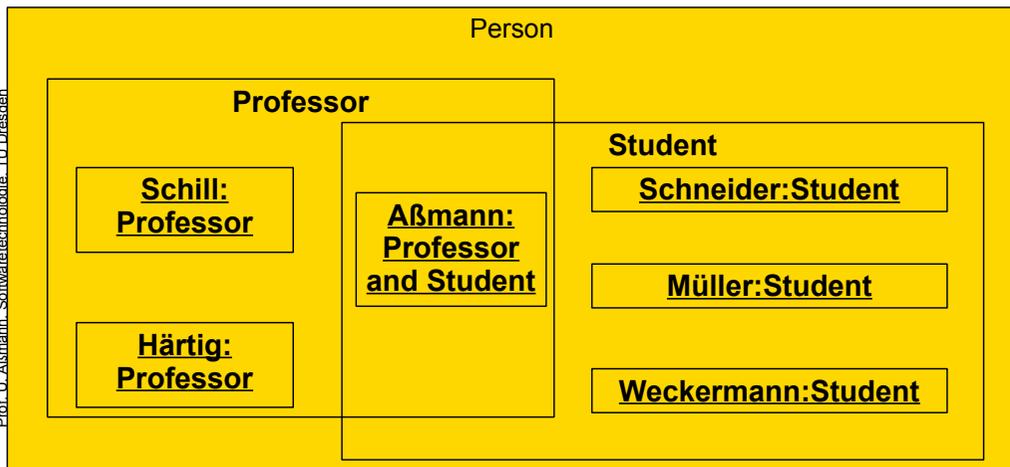


Von wo wird drinkBeer() geerbt?



# Mehrfachvererbung als Venn-Diagramm

- 58 ▶ Betrachtet man Klassen als Mengen, bildet sich Unterklassenbeziehung auf Teilmengenbeziehung ab
- ▶ Mehrfach-Vererbung ergibt ein Venn-Diagramm mit Überschneidungen

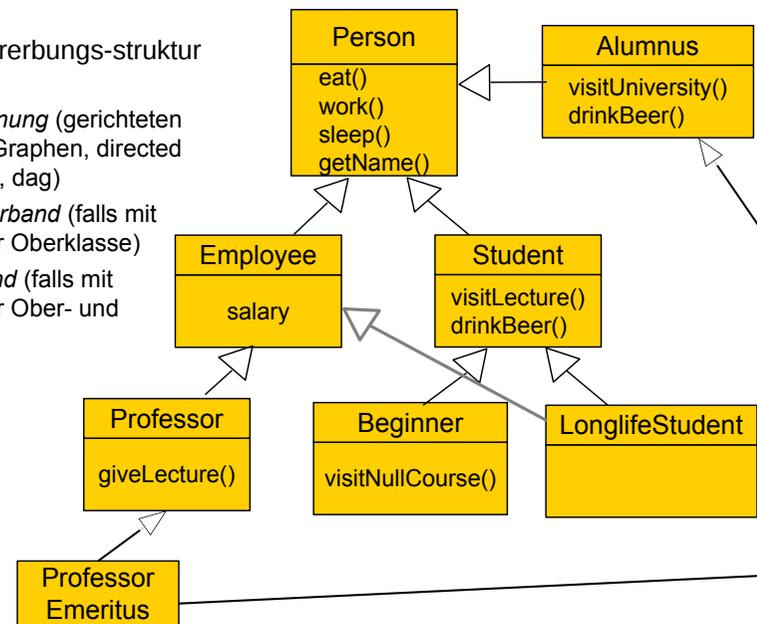


Prof. U. Alßmann, Softwaretechnologie, TU Dresden



# Ein grosser Mehrfachvererbungsverband

- 59 ▶ Eine Mehrfachvererbungs-struktur bildet
  - eine *Halbordnung* (gerichteten azyklischen Graphen, directed acyclic graph, dag)
  - einen *Halverband* (falls mit gemeinsamer Oberklasse)
  - Einen *Verband* (falls mit gemeinsamer Ober- und Unterklasse)

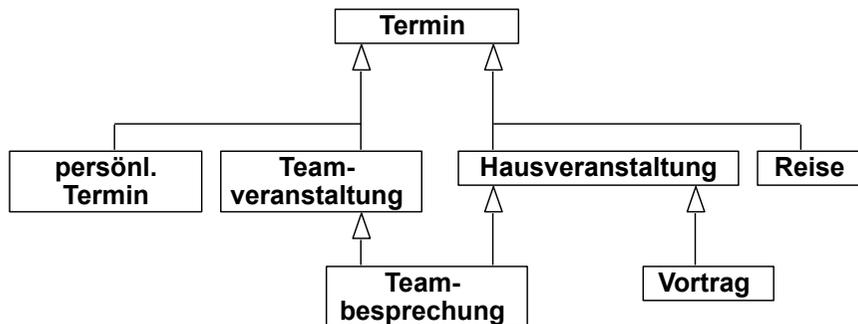


Prof. U. Alßmann, Softwaretechnologie, TU Dresden



# Realisierung von Code-Mehrfachvererbung in Java

- 60 ▶ In UML ist es prinzipiell möglich, daß eine konkrete Klasse von mehreren Klassen erbt:



Prof. U. Alßmann, Softwaretechnologie, TU Dresden

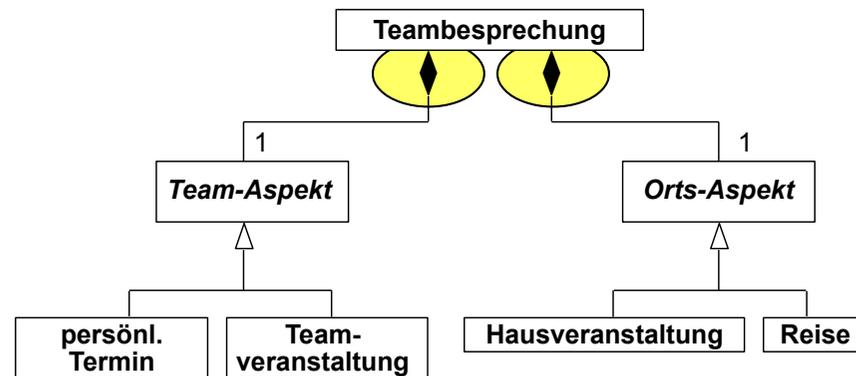


- ▶ Java unterstützt Mehrfachvererbung von konkreten Klassen **nicht**, nur von Schnittstellen

```
class Teambesprechung
    extends Teamveranstaltung, Hausveranstaltung
```

# Realisierung von Mehrfachvererbung durch Komposition

61



Prof. U. Alßmann, Softwaretechnologie, TU Dresden



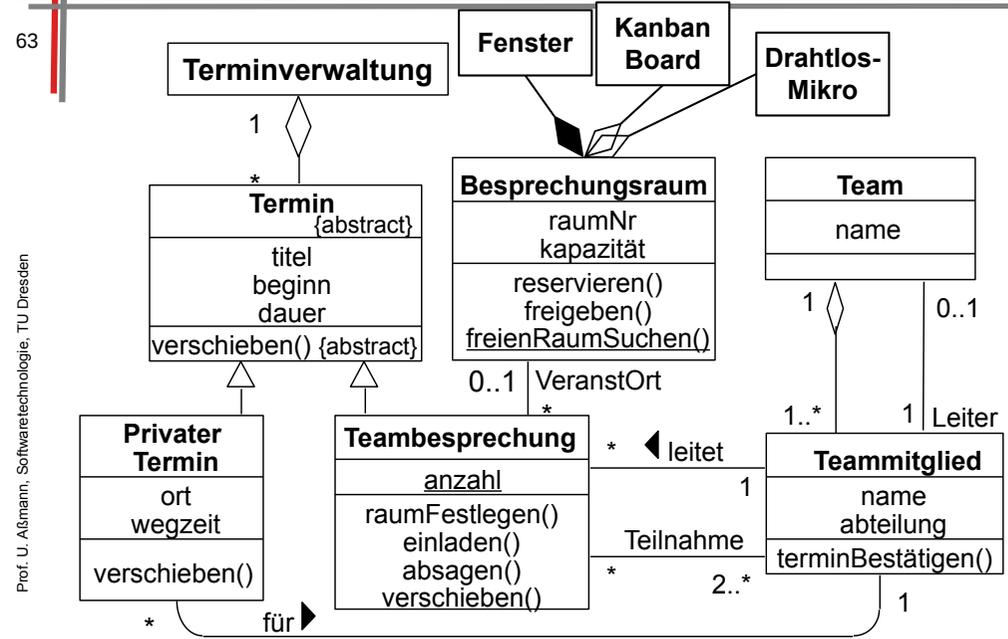
```
class Teambesprechung {
    private Teamaspekt ta;
    private Ortsaspekt oa;
    ...
}
```

```
abstract class Teamaspekt {
}
abstract class Teamveranstaltung
    extends Teamaspekt {
    ...
}
```

# Verschiedene Ähnlichkeitsrelationen in Analysemodellen (Similarity Relationships)

- 62
- ▶ *is-a*: zeigt Ähnlichkeit an.
    - In Mengenhierarchien ist die Untermengenrelation gemeint (subset)
    - In Begriffshierarchien Unterkonzept-Relation (subconcept)
    - *is-a* ist azyklische Relation, bei einfacher Vererbung baumförmig
  - ▶ *is-structured-like*: zeigt ähnliche Struktur an (strukturelle Ähnlichkeit oder Gleichheit)
  - ▶ *behaves-like*: Verhaltensähnlichkeit
    - *always-behaves-like*: Konformanz (conformance), Ersetzbarkeit (substitutability)
    - *sometimes-behaves-like*: gelegentlich verhaltensgleich
    - *restrictedly-behaves-like*: im allgemeinen konformant, aber nicht in speziellen Situationen (extravagance, restriction inheritance)
    - Achtung: *is-a*, *is-structured-like*, *behaves-like* werden alle *Vererbung* genannt
  - ▶ *instance-of*: A ist aus einer Schablone S gemacht worden

# Beispiel: Analyse-Klassendiagramm



# Ausblick: Verfeinerung von Analyse- zum Entwurfsmodell

64

Analyse-Modell (Fragmente)	Entwurfs-Modell (Mehr Struktur & mehr Details)
Notation: aUML Objekte: Fachgegenstände Klassen: Fachbegriffe, parallele Prozesse Attribute ohne Typen und Sichtbarkeiten Operationen: ohne Typen, Parameter und Rückgabewerte Assoziationen: partiell, bidirektional i. Allg. ohne Datentypen Aggregationen, Kompositionen Leserichtung, partielle Multiplizitäten Vererbung: Begriffsstruktur Annahme perfekter Technologie Funktionale Essenz Völlig projektspezifisch Grobe Strukturskizze	Notation: dUML Objekte: Softwareeinheiten Klassen: mit Abstrakt, Interface, Stereotyp Attribute: Sichtbarkeiten, Ableitung, Klassenattribute Initialisierung, weitere spezielle Eigenschaften Operationen: voll typisiert, mit Parameter, Rückgabewert, Klassenoperation Unidirektionale Assoziationen mit voller Multiplizität, Navigation, qualifizierte A. Vererbung: Programmableitung Annahme perfekter Technologie Erfüllung konkreter Rahmenbedingungen Gesamtstruktur des Systems Ähnlichkeiten zwischen verwandten Projekten (zwecks Wiederverwendung) Genaue Strukturdefinition

# Was haben wir gelernt?

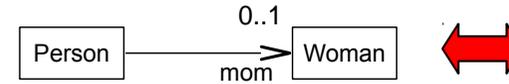
- 65
- ▶ Strukturelle Analyse spürt die Struktur von objektorientierten Anwendungen auf
  - ▶ Strukturgetriebene Analyse verwendet das Metamodell, um die Elemente von Modellen aufzuspüren
    - Strukturelle Analyse mittels CRC und UML-Klassendiagramme sind beides Beispiele für strukturgetriebene Analyse
  - ▶ Das UML-Metamodell gibt Klassen, Merkmale, und Beziehungen als Strukturelemente vor
    - Diese müssen beim Übergang ins Entwurfsmodell abgeflacht werden
    - Mehrfachvererbung kann durch Komposition ausgedrückt werden

# The End

- 66
- Einige Folien sind eine überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. Used by permission.

# Einseitige einstellige Assoziationen in Java und jUML (Wdh.)

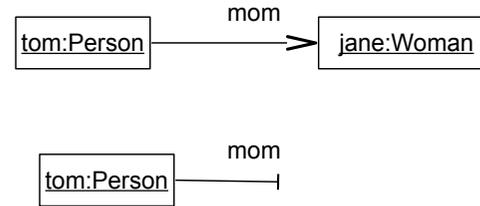
- 67
- Ein Kind kann höchstens eine Mutter haben



```

class Person {
    ...
    private Woman mom;
    ...
}
    
```

Laufzeit:

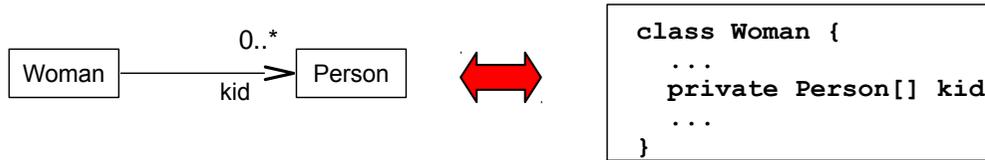


```

Person tom = new Person();
tom.mom = jane;
...
tom.mom = null;
    
```

# Einseitige mehrstellige Assoziationen (Wdh.)

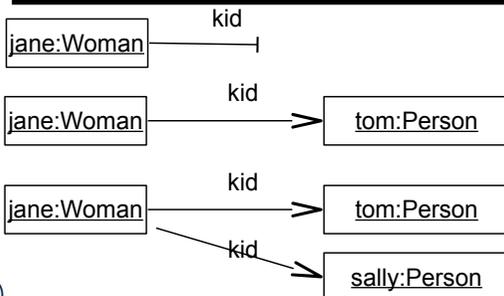
- 68
- Eine Mutter kann aber viele Kinder haben
    - Annahme: Die Obergrenze der Anzahl der Child-Objekte spätestens bei erstmaliger Eintragung von Assoziationsinstanzen bekannt und relativ klein. (Allgemeinere Realisierungen siehe später.)



```

class Woman {
    ...
    private Person[] kid;
    ...
}
    
```

Laufzeit:



```

Woman jane = new Woman();
jane.kid[0] = tom;
...
jane.kid[1] = sally;
    
```

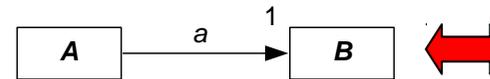
# Optionale und notwendige Assoziationen

- 69
- Untere und obere Schranken von unidirektionalen Assoziationen können durch die Einführung von Argumenten in Konstruktoren eingehalten werden
    - Analog z.B. für Multiplizitäten 0..\* und 1..\*



```

class A {
    ...
    private B a;
    ...
}
    
```



```

class A {
    ...
    private B a;
    ...
    public A (B a, ...) {
        this.a = a; ...
    }
}
    
```