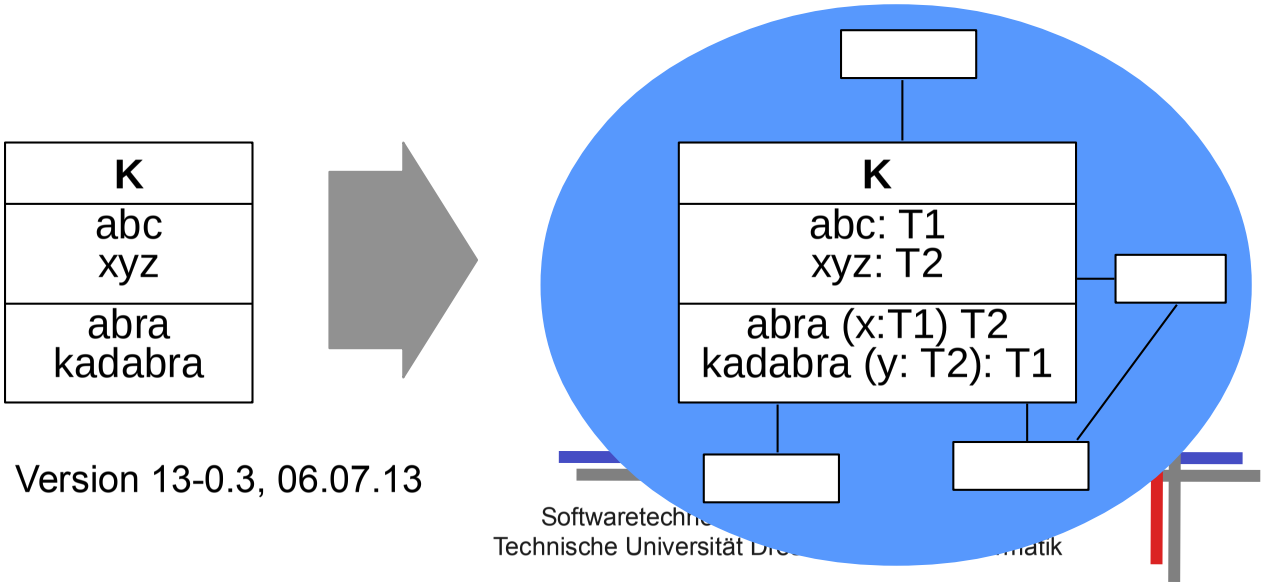
Objektorientierter Entwurf
42) Verfeinerung des Klassenmodells

Zum Selbststudium



Version 13-0.3, 06.07.13

42.1 Verfeinerung von Assoziationen

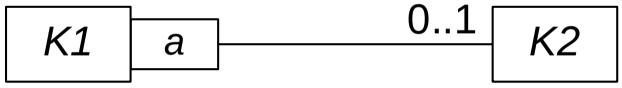
Zum Selbststudium

Qualifizierte Assoziation

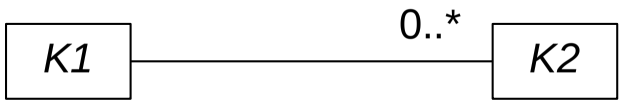
- **Definition:** Eine *Qualifikation (Qualifier)* ist ein Attribut für eine Assoziation zwischen Klassen K1 und K2, durch das die Menge der zu einem K1-Objekt assoziierten K2-Objekte *partitioniert* wird.

  Zweck der Qualifikation ist direkter Zugriff unter Vermeidung von Suche (Beschleunigung)

Notation:

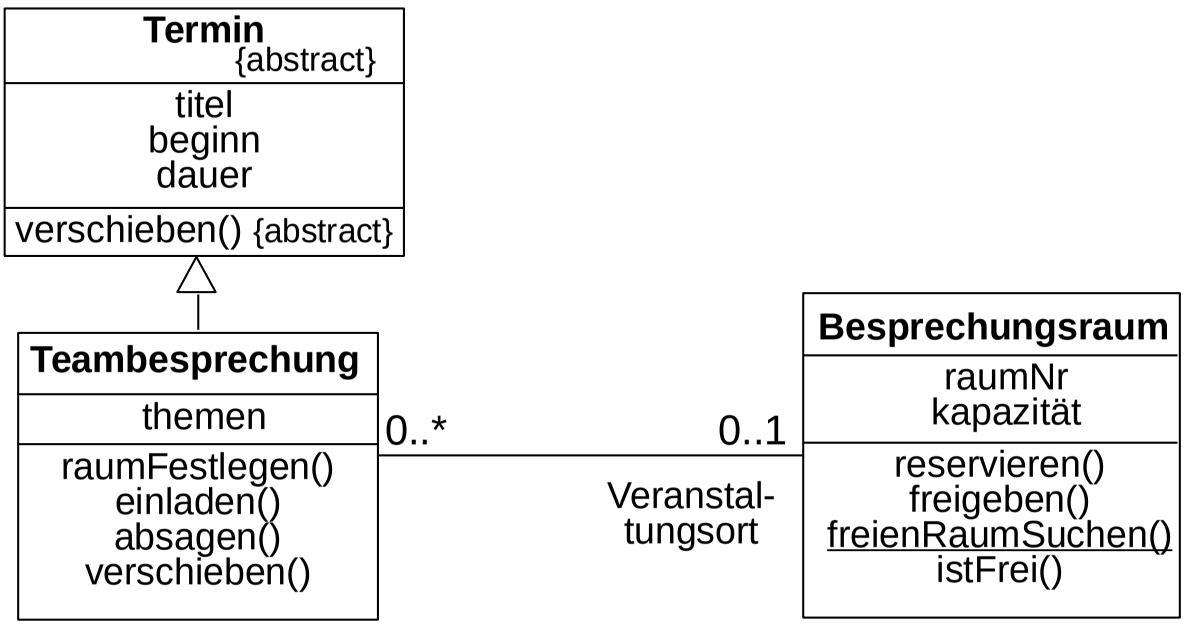


statt:



Hinweis: Qualifizierte Assoziationen werden von vielen UML-Werkzeugen nicht oder nur schlecht unterstützt.
Bedeutung vor allem im Zusammenhang mit Datenbanken (Indizes), aber auch mit geeigneten Datenstrukturen nach Java abbildbar.

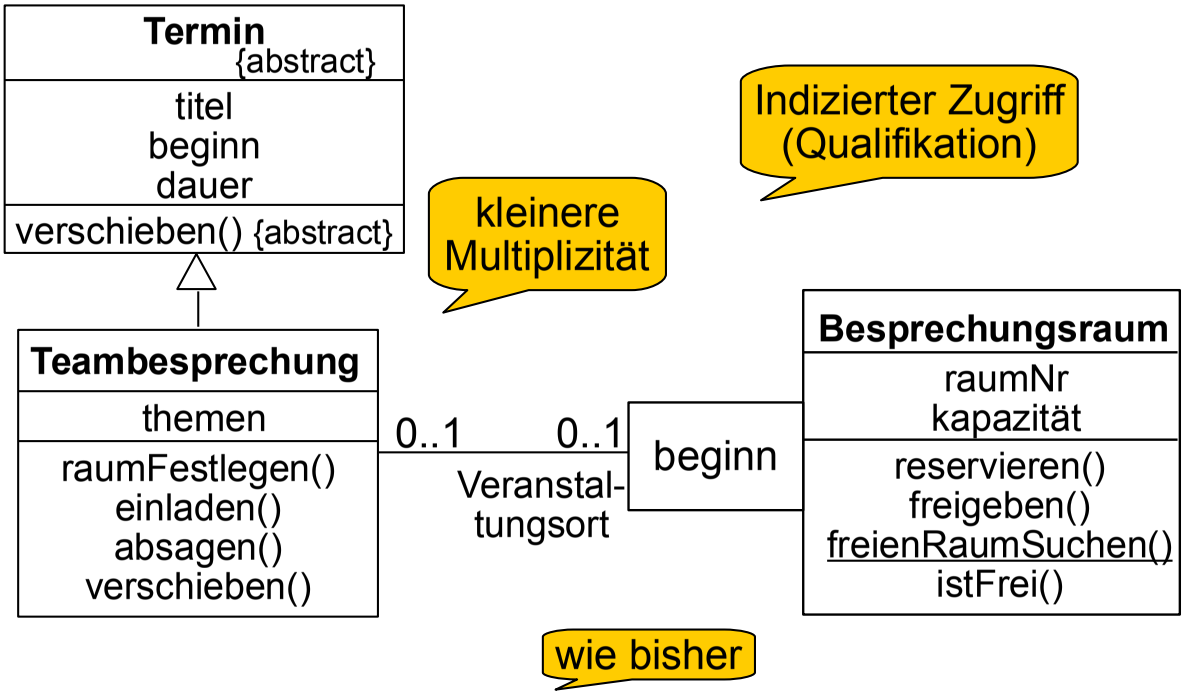
Qualifizierte Assoziation: Beispiel (1)



Raum12.istFrei(start=04.05.02 10:00, dauer=60);

führt zu einer Suche über alle assoziierten Teambesprechungen !

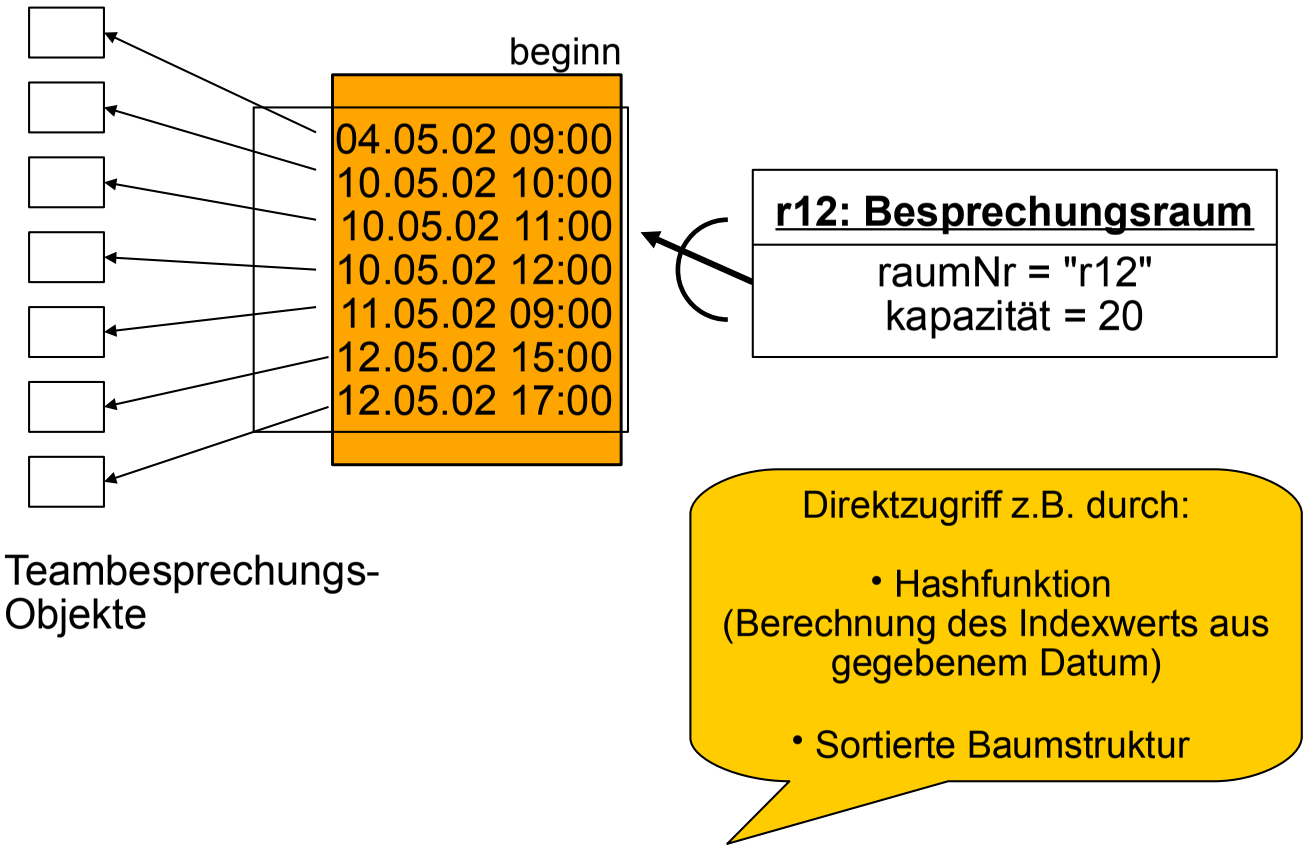
Qualifizierte Assoziation: Beispiel (2)



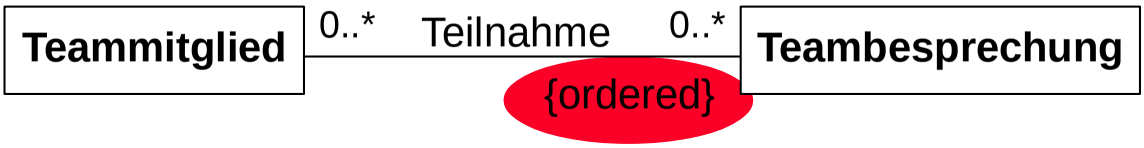
Raum12.istFrei(start=04.05.02 10:00, dauer=60);

kann direkt nach Datum abfragen, ob eine Assoziation besteht

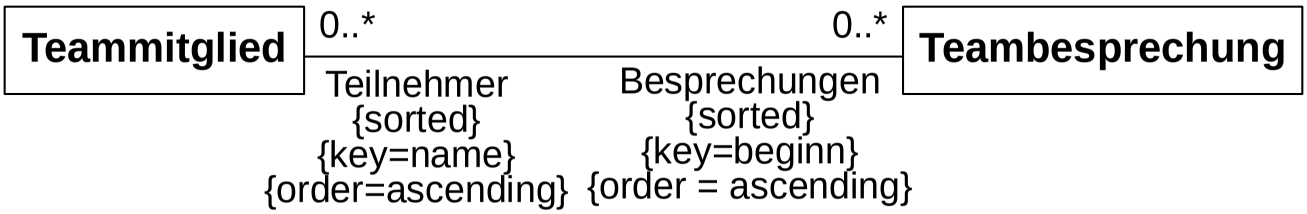
Realisierung einer qualifizierten Assoziation



Geordnete und sortierte Assoziation

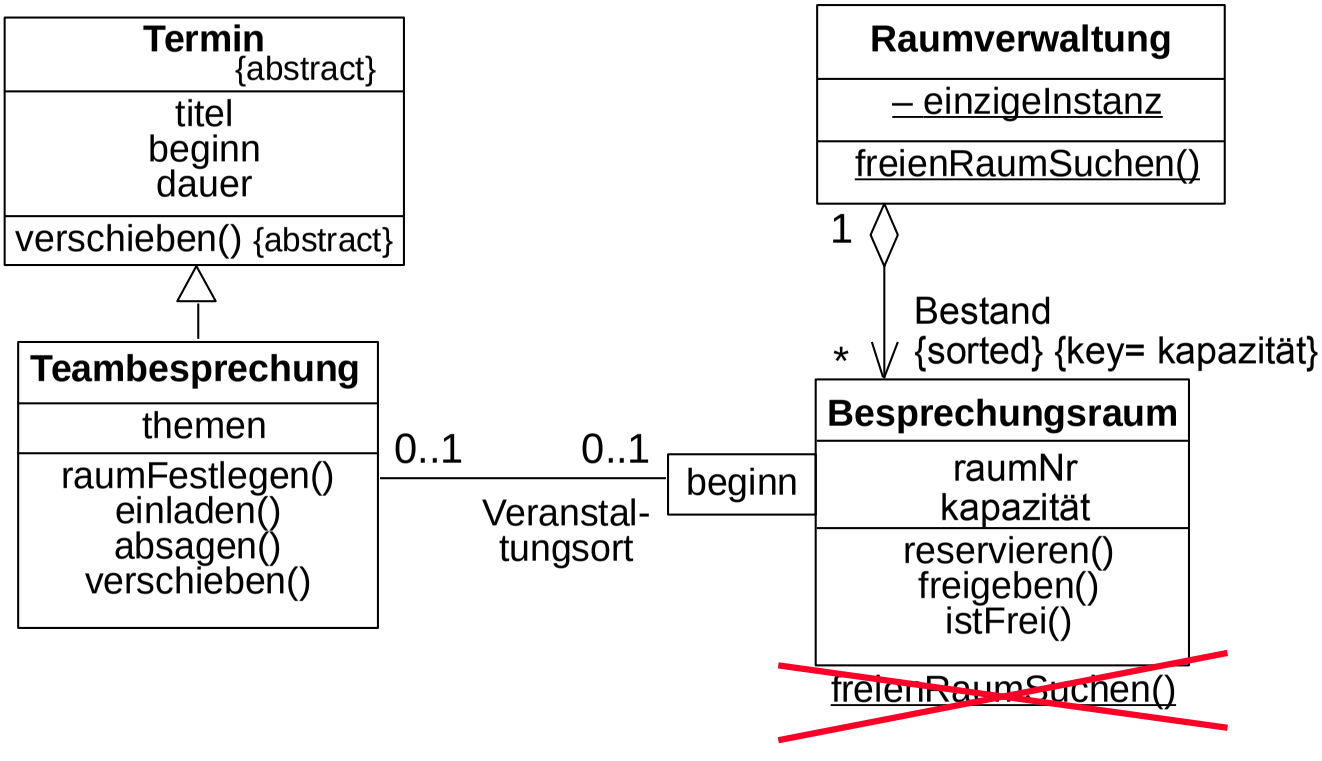


- {ordered} an einem Assoziationsende:
  - Es besteht eine feste Reihenfolge, in der die assoziierten Objekte durchlaufen werden können (*Sequenz*).
  - Mehrfachvorkommen eines Objekts sind verboten
- Keine Angabe an einem Assoziationsende:
  - Die assoziierten Objekte sind als *Menge* strukturiert.
- Spezieller Einschränkungen als Annotationen möglich, z.B. die Forderung nach Sortierung gemäß bestimmter Attribute:



Verwaltungsklassen (Materialbehälterklassen)

- Hat man eine Menge von Objekten, die verwaltet werden müssen, kann eine *Verwaltungs-* oder *Materialbehälterklasse* identifiziert werden

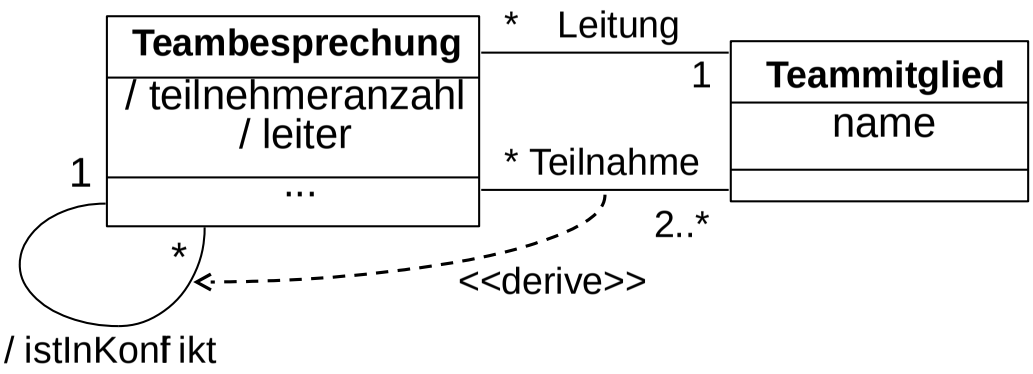


Identifikation von abgeleiteten (redundanten) Elemente

- **Definition** Ein ***abgeleitetes*** Modellelement (z.B. Attribut, Assoziation) ist ein Modell-Element, das jederzeit aus anderen (nicht abgeleiteten) Elementen rekonstruiert werden kann.
- **Notation**

  / Modellelement oder

  Modellelement {derived}
- **Beispiele:**

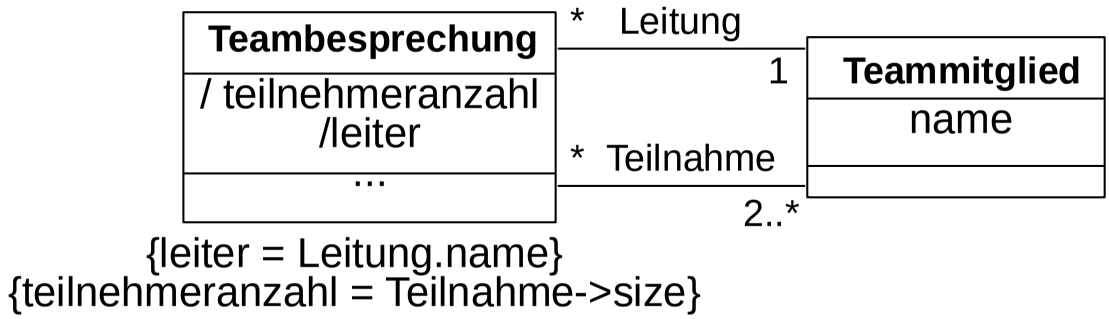


- Abhängigkeitspfeil mit <<derive>>: Optionale Angabe des Ursprungselements einer Abhängigkeit anzugeben.

Detailinformation zu abgeleiteten Elementen

- Zweck: Durch Ableitung kann Redundanz ausdem Model eliminiert werden, und das führt zu einer besseren Konsistenz
- Man kann die Ableitungsregel für abgeleitete Elemente explizit angeben.

  (Notation: *Object Constraint Language OCL* von UML)



42.2 Verfeinerung von Vererbung

Elimination von nicht-konformer Vererbung

- In Analysemodellen hat die Vererbungsrelation oft mehrere Bedeutungen
- Während das Analysemodell diese Feinheiten nicht unterscheiden muss, sollte das Entwurfsmodell dies tun
  - Ansonsten kann es zu Laufzeitfehlern kommen

Wiederh.: Ähnlichkeitsrelationen (Similarity Relationships)

- *is-a:* zeigt Ähnlichkeit an
  - is-a ist azyklische Relation, bei einfacher Vererbung baumförmig
- *is-structured-like:* zeigt ähnliche Struktur an (structurelle 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 B gemacht worden

Konforme Vererbung (Konformität, behaves-like)

Liskov'sches Substitutionsprinzip (Liskov substitution principle):

Eine Unterklasse U heisst *verhaltenskonform* zu einer Oberklasse O, wenn jedes Objekt aus U jedes Objekt aus O ersetzen kann, ohne eine Anwendungsklasse, die O verwendet, in einen fehlerhaften Zustand zu versetzen

- Konforme Vererbung stellt sicher,
  - dass Ableiten von Unterklassen niemals Fehler in eine Anwendung einbringt (*Robustheit*)
  - dass bei der Unterklassenbildung die Semantik von Oberklassen erhalten bleibt

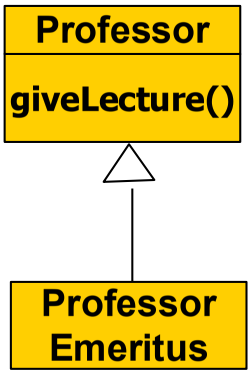
Extravaganz (restrictedly-behaves-like)

- Eine Unterklasse U heisst *extravagant (eingeschränkt)* zu einer Oberklasse O, wenn nicht alle Objekte aus U alle Objekte aus O ersetzen können, ohne dass in einer Anwendung, die O verwendet, Fehler auftreten
- Aka: Eingeschränkte Vererbung (restriction inheritance)

Frage: Was passiert, wenn ein emeritierter Professor keine Vorlesungen mehr anbietet?

Antwort: Eine Anwendung der Klasse Professor, die dies erwartet, endet in einem fehlerhaften Zustand.

Erklärung: ProfessorEmeritus ist eine *extravagante* Unterklasse von Professor

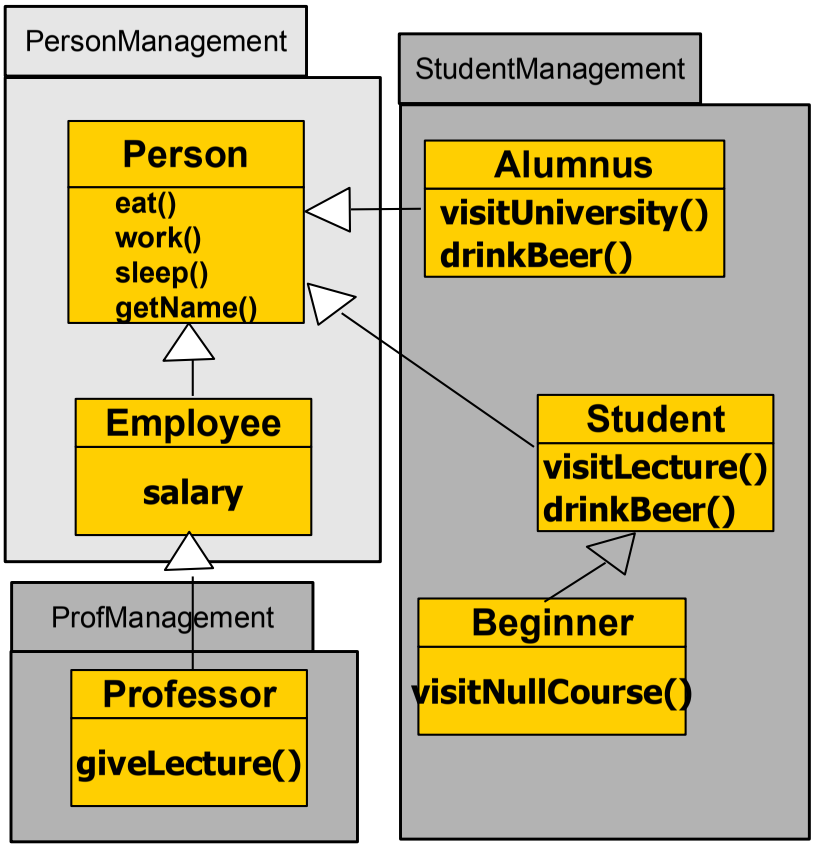


Konformitätsproblem in Analysemodellen

- Leider sind nicht alle Vererbungshierarchien konform, insbesondere nicht in Analysemodellen
  - Sowohl in UML als auch in Java drückt die Vererbungsrelation nicht unbedingt Konformität aus
  - Man muss jede einzelne Vererbung daraufhin untersuchen :-(
  - Und im Entwurfsmodell Konformität herstellen

Beispiel: Bibliotheken, Frameworks und Anwendungen

- Bibliotheken, Frameworks sind vorgefertigte Pakete mit Vererbungshierarchien von einem anderen Hersteller
- Anwendungen *leiten speziellere* Unterklassen davon ab
- Beispiele:
  - Java Development Kit
  - C++ Standard Template Library (STL)

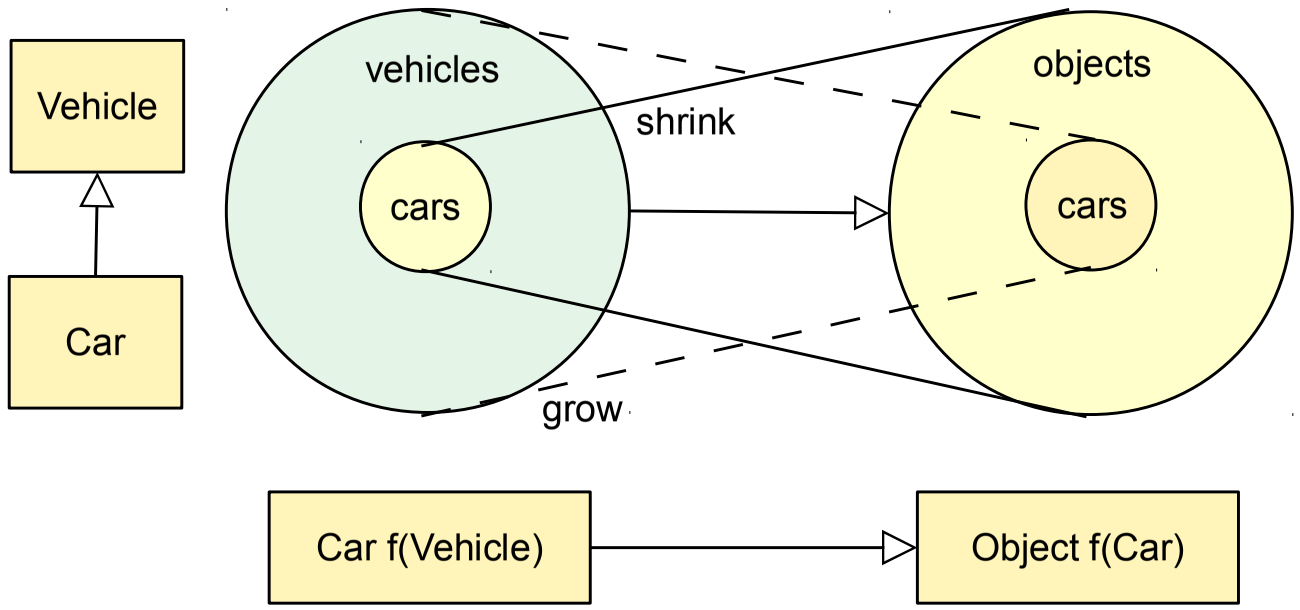


Beispiel: Konforme Vererbung von Bibliotheks- und Frameworkklassen

- Verwendet man beim Entwurf eine zugekaufte Klassenbibliothek, stelle man sicher, dass man Anwendungsklassen nur mit **konformer Vererbung** ableitet
- Ansonsten treten Laufzeitfehler in Klassen der Klassenbibliothek auf
  - Deren Fehlermeldungen sind völlig unverständlich, da sie nicht die eigentlichen Fehlerursache vermitteln *können*

Contravariance of Argument and Return Types

- "I can give equal or more to a function of a replaced object, but it will return less or equal"
- If argument class of subclass grows, return class of subclass must shrink

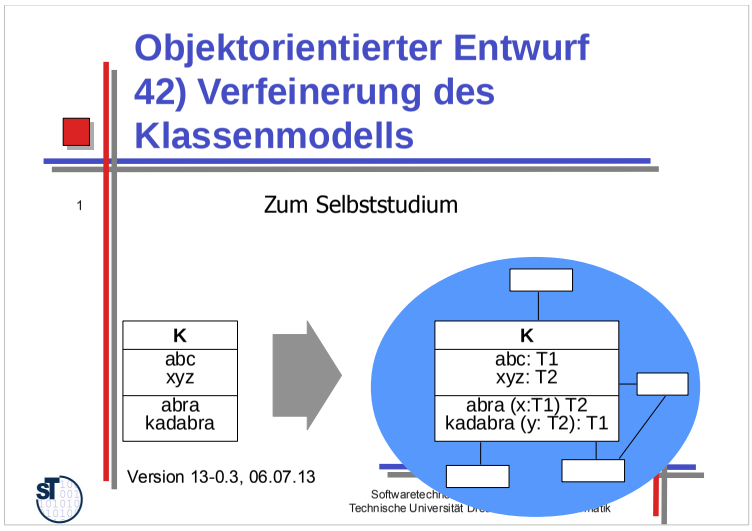


Zusammenfassung: UML-Klassenmodelle in Analyse und Entwurf

Analyse-Modell	Entwurfs-Modell
Skizze: Teilweise unvollständig in Attributen und Operationen	Vollständige Angabe aller Attribute und Operationen
Datentypen und Parameter können noch fehlen	Vollständige Angabe von Datentypen und Parametern
Noch kaum Bezug zur Realisierungssprache	Auf Umsetzung in gewählter Programmiersprache bezogen
Keine Überlegungen zur Realisierung von Assoziationen	Navigationsangaben, Qualif kation, Ordnung, Verwaltungsklassen
Nicht-konforme Vererbung	Entscheidung über Datenstrukturen
	Vorbereitung zur Anbindung von Benutzungsoberf äche und Datenhaltung an fachlichen Kern
	Konforme Vererbung

The End

- Diese Folien sind eine überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann, 2002. used by permission.



42.1 Verfeinerung von Assoziationen

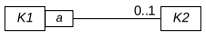
Zum Selbststudium

Qualifizierte Assoziation

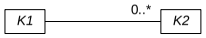
- **Definition:** Eine *Qualifikation (Qualifier)* ist ein Attribut für eine Assoziation zwischen Klassen K1 und K2, durch das die Menge der zu einem K1-Objekt assoziierten K2-Objekte *partitioniert* wird.

  Zweck der Qualifikation ist direkter Zugriff unter Vermeidung von Suche (Beschleunigung)

Notation:

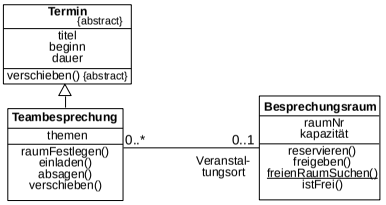


statt:



Hinweis: Qualifizierte Assoziationen werden von vielen UML-Werkzeugen nicht oder nur schlecht unterstützt.
Bedeutung vor allem im Zusammenhang mit Datenbanken (Indizes), aber auch mit geeigneten Datenstrukturen nach Java abbildbar.

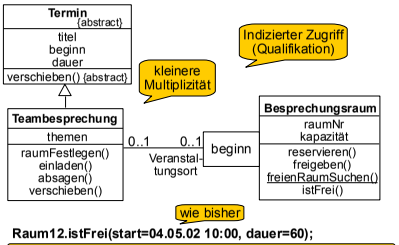
Qualifizierte Assoziation: Beispiel (1)



Raum12.istFrei(start=04.05.02 10:00, dauer=60);

führt zu einer Suche über alle assoziierten Teambesprechungen !

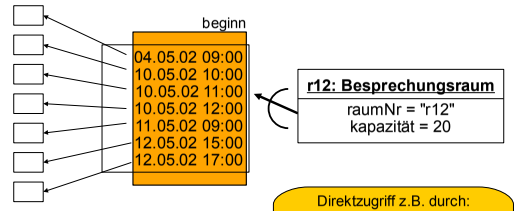
Qualifizierte Assoziation: Beispiel (2)



Raum12.istFrei(start=04.05.02 10:00, dauer=60);

kann direkt nach Datum abfragen, ob eine Assoziation besteht

Realisierung einer qualifizierten Assoziation



r12: Besprechungsraum

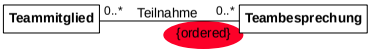
raumNr = "r12"
kapazität = 20

Teambesprechungs-Objekte

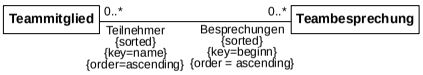
Direktzugriff z.B. durch:

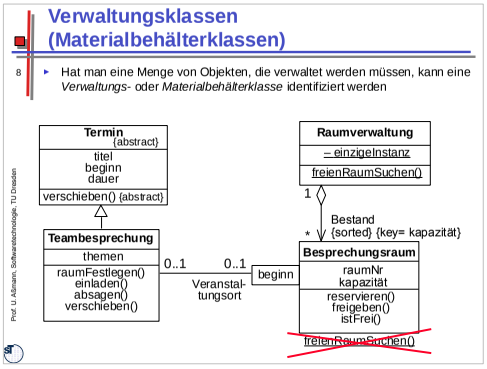
- Hashfunktion (Berechnung des Indexwerts aus gegebenem Datum)
- Sortierte Baumstruktur

Geordnete und sortierte Assoziation

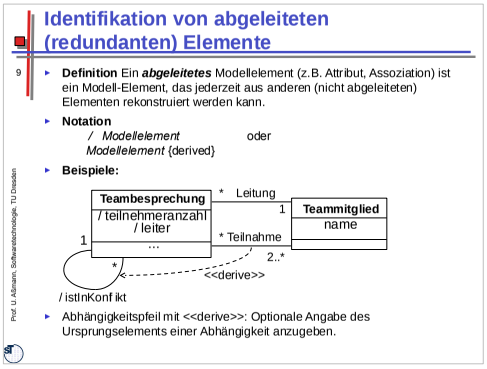


- {ordered} an einem Assoziationsende:
  - Es besteht eine feste Reihenfolge, in der die assoziierten Objekte durchlaufen werden können (*Sequenz*).
  - Mehrfachvorkommen eines Objekts sind verboten
- Keine Angabe an einem Assoziationsende:
  - Die assoziierten Objekte sind als *Menge* strukturiert.
- Spezieller Einschränkungen als Annotationen möglich, z.B. die Forderung nach Sortierung gemäß bestimmter Attribute:



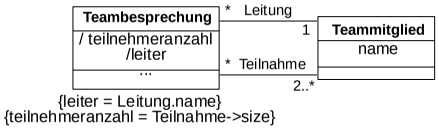


Die Klasse Raumverwaltung dient nur dazu, die Klassenmethode freienRaumSuchen effizienter zu realisieren. Von dieser Klasse soll immer nur eine Instanz existieren, was hier durch ein Klassenattribut angedeutet ist (sogenannte *singleton-*



Detailinformation zu abgeleiteten Elementen

- Zweck: Durch Ableitung kann Redundanz ausdem Model eliminiert werden, und das führt zu einer besseren Konsistenz
- Man kann die Ableitungsregel für abgeleitete Elemente explizit angeben.
  (Notation: *Object Constraint Language OCL* von UML)



{leiter = Leitung.name}
{teilnehmeranzahl = Teilnahme->size}

42.2 Verfeinerung von Vererbung

Elimination von nicht-konformer Vererbung

- In Analysemodellen hat die Vererbungsrelation oft mehrere Bedeutungen
- Während das Analysemodell diese Feinheiten nicht unterscheiden muss, sollte das Entwurfsmodell dies tun
  - Ansonsten kann es zu Laufzeitfehlern kommen

Wiederh.: Ähnlichkeitsrelationen (Similarity Relationships)

- *is-a:* zeigt Ähnlichkeit an
  - is-a ist azyklische Relation, bei einfacher Vererbung baumförmig
- *is-structured-like:* zeigt ähnliche Struktur an (structurelle 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 B gemacht worden

Konforme Vererbung (Konformität, behaves-like)

Liskov'sches Substitutionsprinzip (Liskov substitution principle):

Eine Unterklasse U heisst *verhaltenskonform* zu einer Oberklasse O, wenn jedes Objekt aus U jedes Objekt aus O ersetzen kann, ohne eine Anwendungsklasse, die O verwendet, in einen fehlerhaften Zustand zu versetzen

- Konforme Vererbung stellt sicher,
  - dass Ableiten von Unterklassen niemals Fehler in eine Anwendung einbringt (*Robustheit*)
  - dass bei der Unterklassenbildung die Semantik von Oberklassen erhalten bleibt

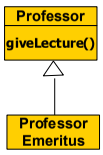
Extravaganz (restrictedly-behaves-like)

- Eine Unterklasse U heisst *extravagant (eingeschränkt)* zu einer Oberklasse O, wenn nicht alle Objekte aus U alle Objekte aus O ersetzen können, ohne dass in einer Anwendung, die O verwendet, Fehler auftreten
- Aka: Eingeschränkte Vererbung (restriction inheritance)

Frage: Was passiert, wenn ein emeritierter Professor keine Vorlesungen mehr anbietet?

Antwort: Eine Anwendung der Klasse Professor, die dies erwartet, endet in einem fehlerhaften Zustand.

Erklärung: ProfessorEmeritus ist eine *extravagante* Unterklasse von Professor

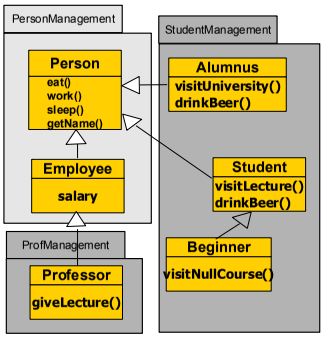


Konformitätsproblem in Analysemodellen

- Leider sind nicht alle Vererbungshierarchien konform, insbesondere nicht in Analysemodellen
  - Sowohl in UML als auch in Java drückt die Vererbungsrelation nicht unbedingt Konformität aus
  - Man muss jede einzelne Vererbung daraufhin untersuchen :-(
  - Und im Entwurfsmodell Konformität herstellen

Beispiel: Bibliotheken, Frameworks und Anwendungen

- Bibliotheken, Frameworks sind vorgefertigte Pakete mit Vererbungshierarchien von einem anderen Hersteller
- Anwendungen *leiten speziellere* Unterklassen davon ab
- Beispiele:
  - Java Development Kit
  - C++ Standard Template Library (STL)

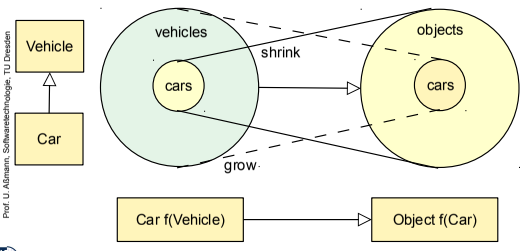


Beispiel: Konforme Vererbung von Bibliotheks- und Frameworkklassen

- Verwendet man beim Entwurf eine zugekaufte Klassenbibliothek, stelle man sicher, dass man Anwendungsklassen nur mit **konformer Vererbung** ableitet
- Ansonsten treten Laufzeitfehler in Klassen der Klassenbibliothek auf
  - Deren Fehlermeldungen sind völlig unverständlich, da sie nicht die eigentlichen Fehlerursache vermitteln *können*

Contravariance of Argument and Return Types

- "I can give equal or more to a function of a replaced object, but it will return less or equal"
- If argument class of subclass grows, return class of subclass must shrink



Zusammenfassung: UML-Klassenmodelle in Analyse und Entwurf

Analyse-Modell	Entwurfs-Modell
Skizze: Teilweise unvollständig in Attributen und Operationen	Vollständige Angabe aller Attribute und Operationen
Datentypen und Parameter können noch fehlen	Vollständige Angabe von Datentypen und Parametern
Noch kaum Bezug zur Realisierungssprache	Auf Umsetzung in gewählter Programmiersprache bezogen
Keine Überlegungen zur Realisierung von Assoziationen	Navigationsangaben, Qualif kation, Ordnung, Verwaltungsklassen
Nicht-konforme Vererbung	Entscheidung über Datenstrukturen
	Vorbereitung zur Anbindung von Benutzungsoberf äche und Datenhaltung an fachlichen Kern
	Konforme Vererbung

The End

- Diese Folien sind eine überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann, 2002. used by permission.