



# Projekt Erdbebenfrühwarnung im SoSe 2011



## Entwicklung verteilter echtzeitfähiger Sensorsysteme



Joachim Fischer  
Klaus Ahrens  
Ingmar Eveslage

[fischer|ahrens|eveslage@informatik.hu-berlin.de](mailto:fischer|ahrens|eveslage@informatik.hu-berlin.de)

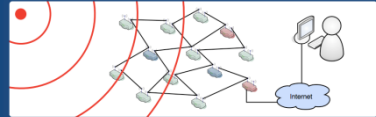


EDIM

SOSEWIN-extended



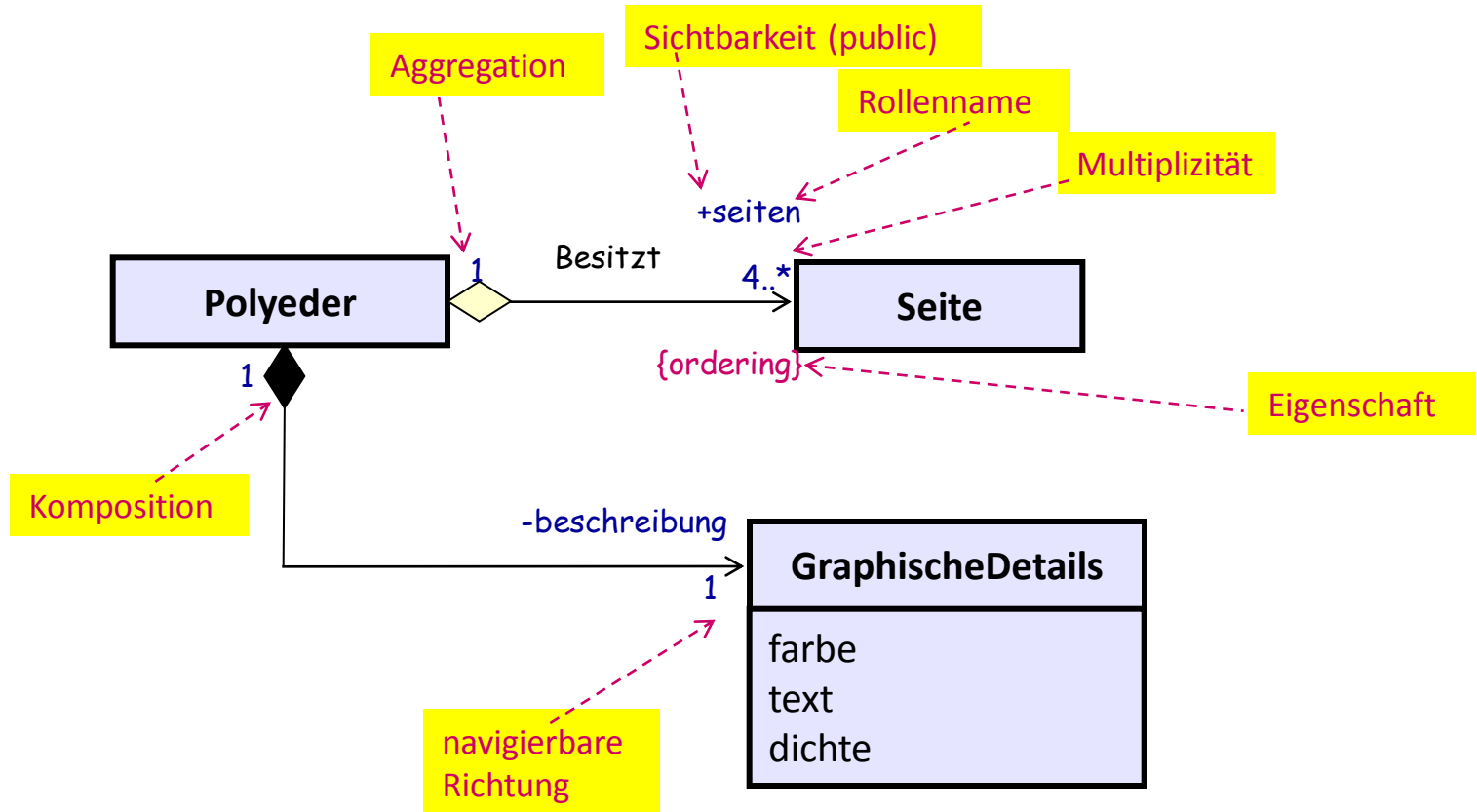
Systemanalyse



## 3. Ausgewählte UML-Konzepte

1. Klassen und Objekte als UML-Basiskonzepte
2. Assoziationen und Attribute (Forts.)
3. Operationen und Methoden
4. UML-Schlüsselworte
5. Constraints in OCL

# Assoziationsenden und Adornments (Wdh.)



# Zusammenhang von Attribut und Assoziation

- UML 2 kennt **keinen** semantischen Unterschied zwischen
  - Attributen einer Klasse und
  - navigierbaren Assoziationsenden (Rollen),

es gibt nur

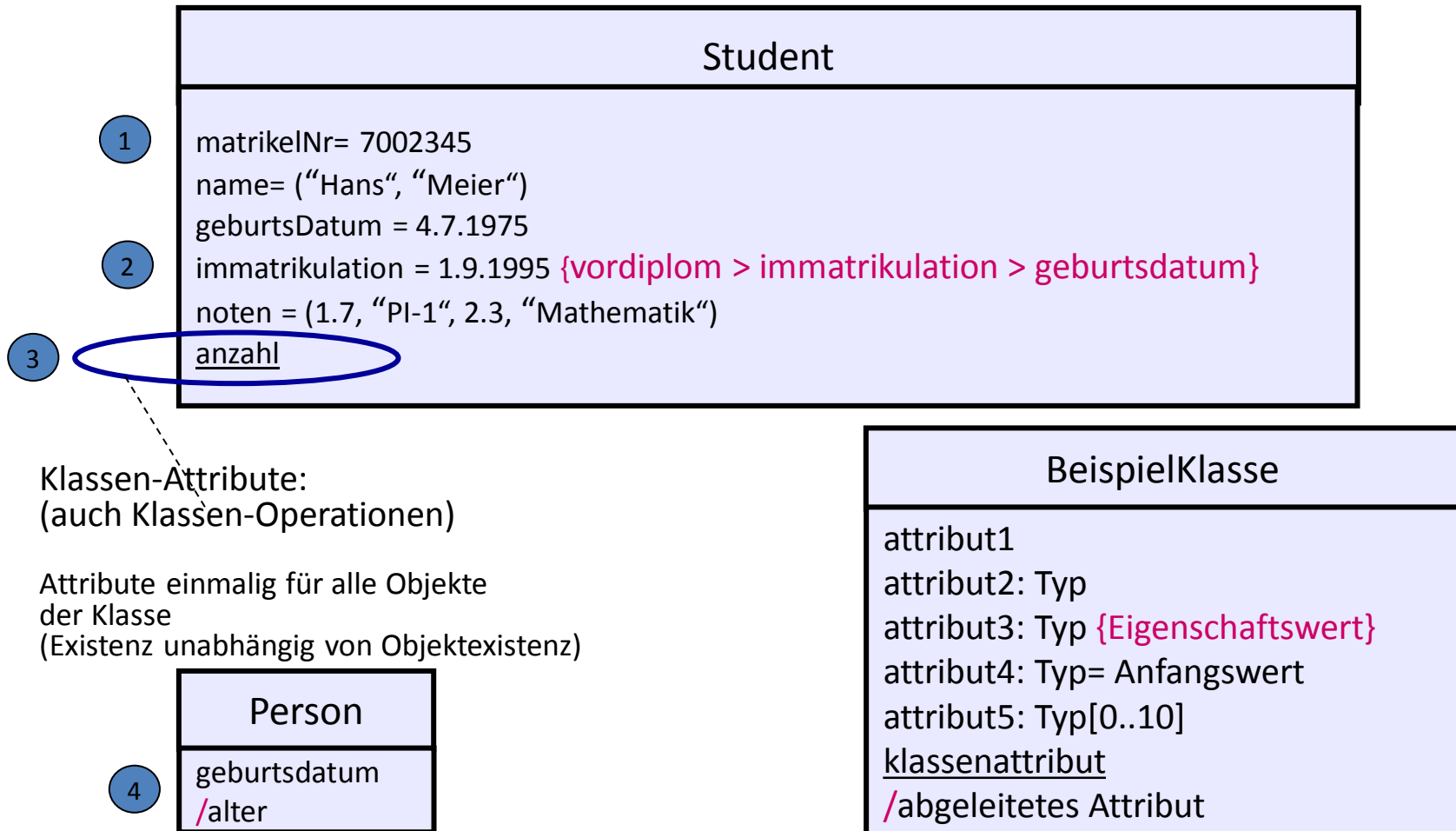
- charakteristische Ausprägungen (Properties) und
- Modellelemente (hier Klasse, später weitere), die die Ausprägungen besitzen

- Ist das Ende einer Assoziation als **navigierbar** gekennzeichnet, so besitzt die navigierende Klasse ein Attribut mit dem Namen dieses Endes (Rollename)
- Ist das Ende einer Assoziation als **nicht navigierbar** gekennzeichnet, so besitzt die Assoziation selbst eine Ausprägung mit dem Namen des Endes (z.B. als Repräsentation im Repository)

# Attribut-Einschränkungen und Klassenattribute

## Einschränkungen (Constraints)

sind wie bei Assoziationsenden Zusicherungen, die immer wahr sein müssen



Klassen-Attribute:  
(auch Klassen-Operationen)

Attribute einmalig für alle Objekte  
der Klasse  
(Existenz unabhängig von Objektexistenz)

4

# Attributspezifikation: Eigenschaftswerte

- Anfangswert (Wert bei Konstruktion des Objektes)
- Eigenschaftswerte  
besondere Charakteristik von Attributen  
Angabe in { }-Klammerung als Liste
  - z.B.:
    - **{readonly}** Werte dürfen nicht verändert werden (in C++ const)
    - **{default}** Wert wird bei Objektinstanziierung in Slot abgelegt
    - **{derived}** Wert lässt sich stets aus anderen berechnen  
(Attribut ist damit keine Zustandsgröße)
    - **{ordered}** Inhalte des Attributs sind geordnet  
z.B.: [1, 1, 3, 5, 6, 6, 9]
    - **{unique}** Inhalte des Attributs sind duplikationsfrei  
z.B. [ 3, 500, 1, 4, 2]
    - ...  
**union, subsets, redefines, composite**

**Achtung:** Einschränkungen sind Zusicherungen, die immer wahr sein müssen

# Kombination von Eigenschaftswerten: Eindeutigkeit und Ordnung

... bestimmen Datenstruktur der entspr. Attribute

Default-Einstellung

Ordnung	Eindeutigkeit	Datenstruktur
<i>{ordered}</i>	<i>{unique}</i>	
ja	ja	geordnete Menge (ordered set, unique list)
ja	Nein <i>{nonunique}</i>	Liste (list)
nein <i>{unordered}</i>	ja	Menge (set)
nein <i>{unordered}</i>	Nein <i>{nonunique}</i>	Multimenge (bag)

# Eigenschaftswert: **bag** (1)

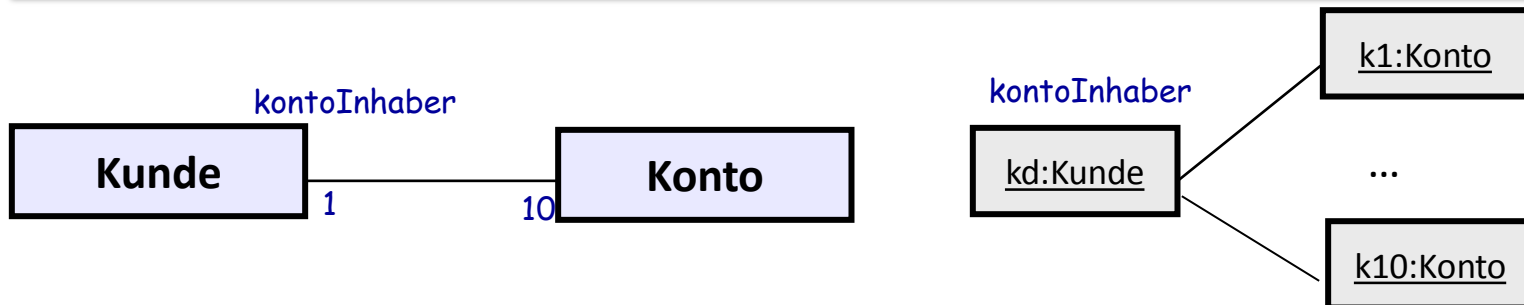
Instanzenkollektion darf mehrfache identische Kopien von Elementen enthalten



Damit sind mehrere Links zu einem Konto-Objekt möglich

Extremfall: kd hat 10 Links zu einem einzigen Konto-Objekt

Standardfall

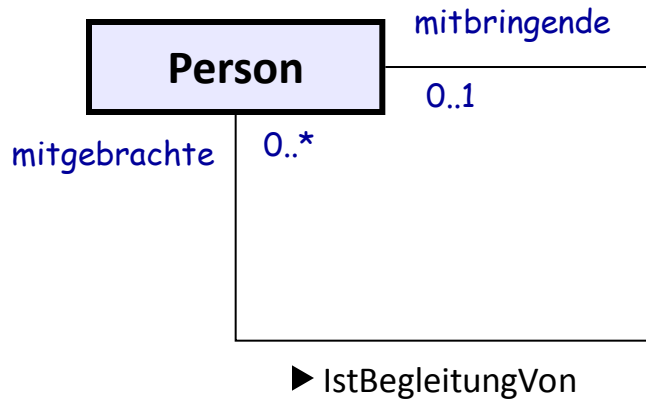


alle 10 Konten sind verschiedene Instanzen



# Eigenschaftswert: **bag** (2)

Anwendungskontext: Party



*typische Assoziation: zirkulär reflexiv*

## Standardsemantik

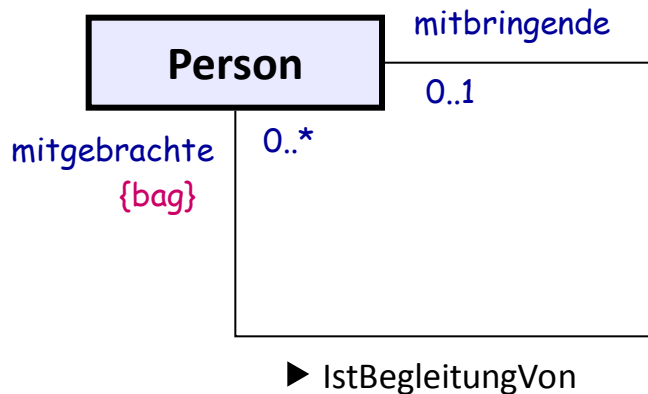
Objekte sind jeweils voneinander verschieden

## Achtung:

ein Person-Objekt könnte sich dennoch selbst begleiten,  
da kein Constraint zwischen den Kollektionen *mitbringende* und *mitgebrachte*

zusätzl. Regel:

$\{ \{ \text{mitbringende} \} \cap \text{mitgebrachte} = \emptyset \}$



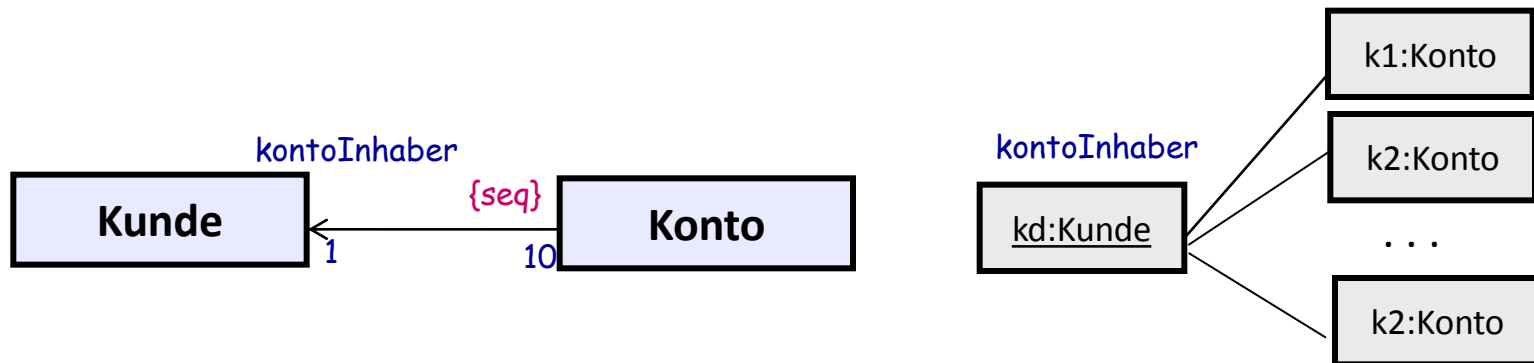
Eine Person könnte so mehrmals mitgebracht werden

→ keine vernünftige Semantik

# Eigenschaftswert: seq

Def:

Kollektor mit {seq} ist eine geordnete Multimenge

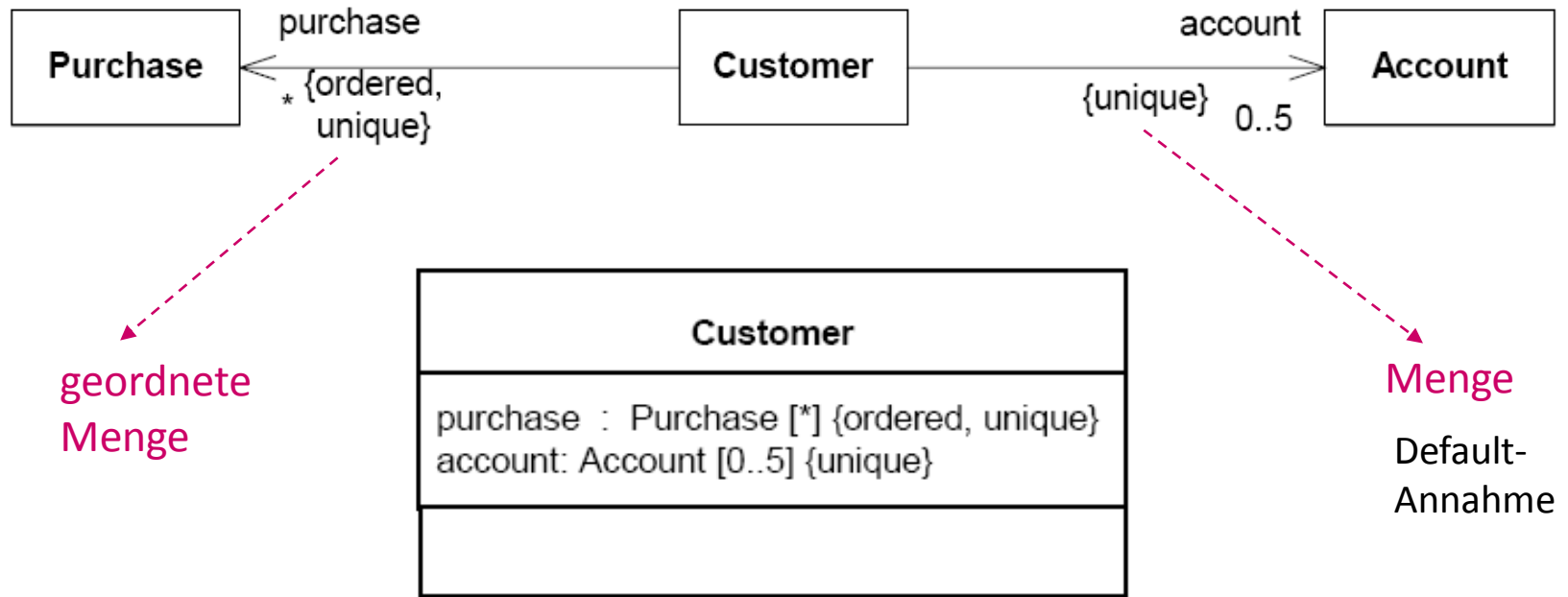


**z.B.:**

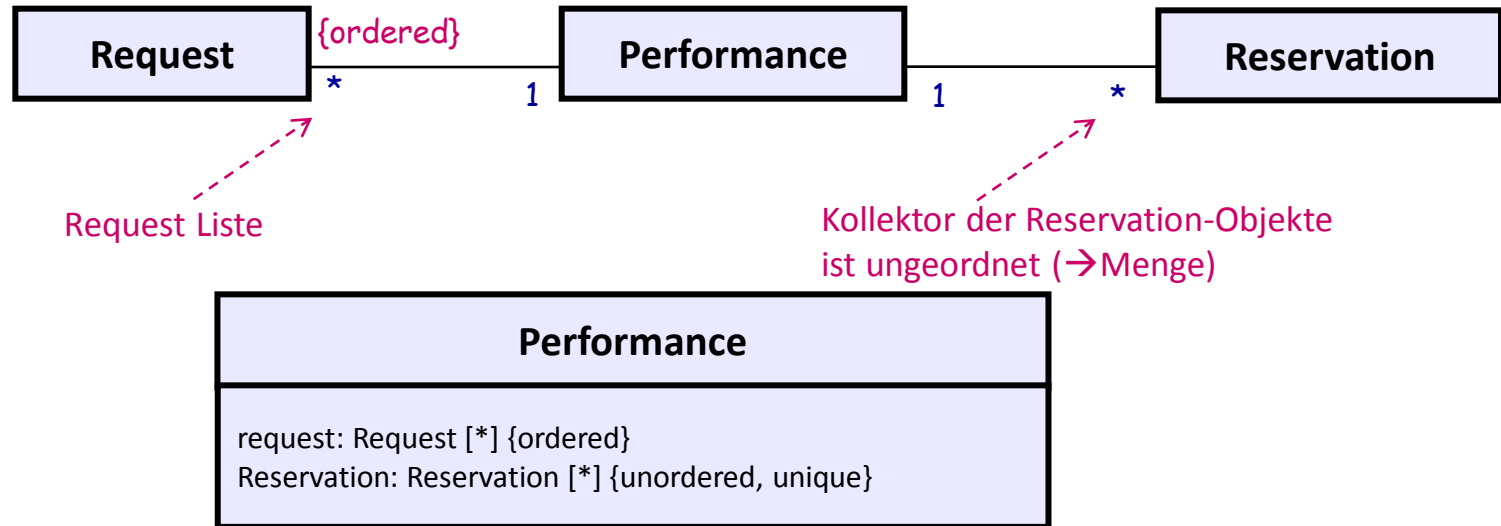
Assoziation:

Kontenbewegung (Ein- u. Auszahlungen)

# Eigenschaftswert: **unique** und **nonunique**



# Eigenschaftswert: **ordered** und **unordered**



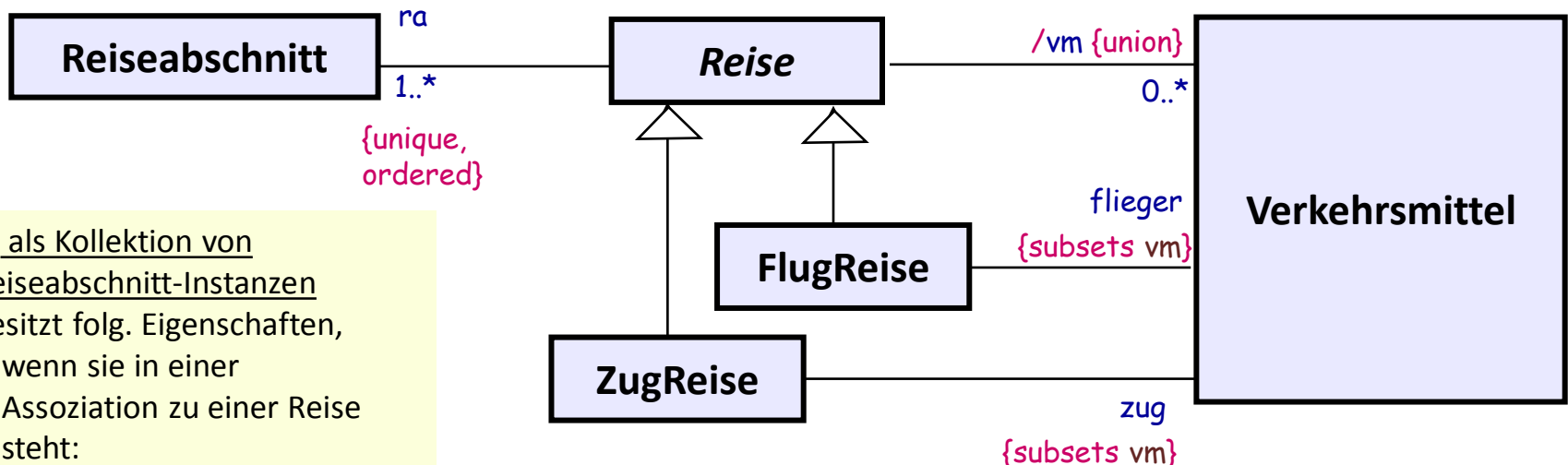
- **ordered** nur anwendbar, wenn: obere Multiplizitätsgrenze > 1
- gilt auch für Attribute
- bei Aufnahme eines neuen Request-Links muss Position angegeben werden
- Spezifikation der Sortierung für Request-Objekte bedarf einer zusätzlichen Constraint-Angabe

# Beispiel: mehrere Eigenschaften von Assoziationsenden

Kollektion von Verkehrsmitteln *vm*

- kann leer sein
- unter *vm* ist diese Menge einem Reise-Objekt bekannt

- *vm* ist abgeleitet u. bestimmt sich vollständig aus seinen Teilmengen



*ra* als Kollektion von Reiseabschnitt-Instanzen besitzt folg. Eigenschaften, wenn sie in einer Assoziation zu einer Reise steht:

- nicht leer
- echte Menge
- es gibt eine Ordnung

zwei Teilmengen von *Verkehrsmittel*-Instanzen sind ausgezeichnet

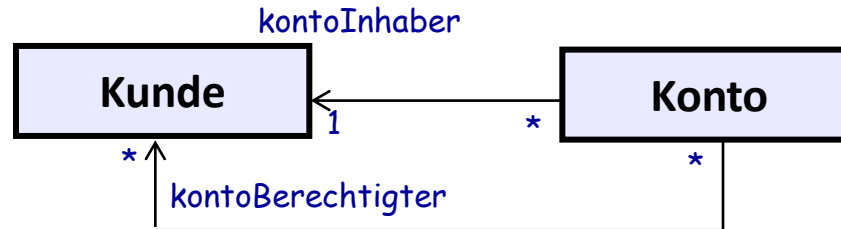
- können leer sein
- unter *flieger* und *zug* sind diese Mengen einem *Reise*-Objekt bekannt, falls es sich dabei um ein *Flugreise*- oder *Zugreise*-Objekt handelt
- ist eine Reise eine Flugreise, so sind alle *Verkehrsmittel* über *flieger* erreichbar

Zusatzforderung: Konformität zwischen Union-Kollektor und seinen Subset-Kollektoren

- dies muss konkret durch die Regelsprache (OCL) definiert sein

# Assoziationen: Einschränkungen

Assoziationsenden können um Einschränkungen (Constraints) ergänzt werden

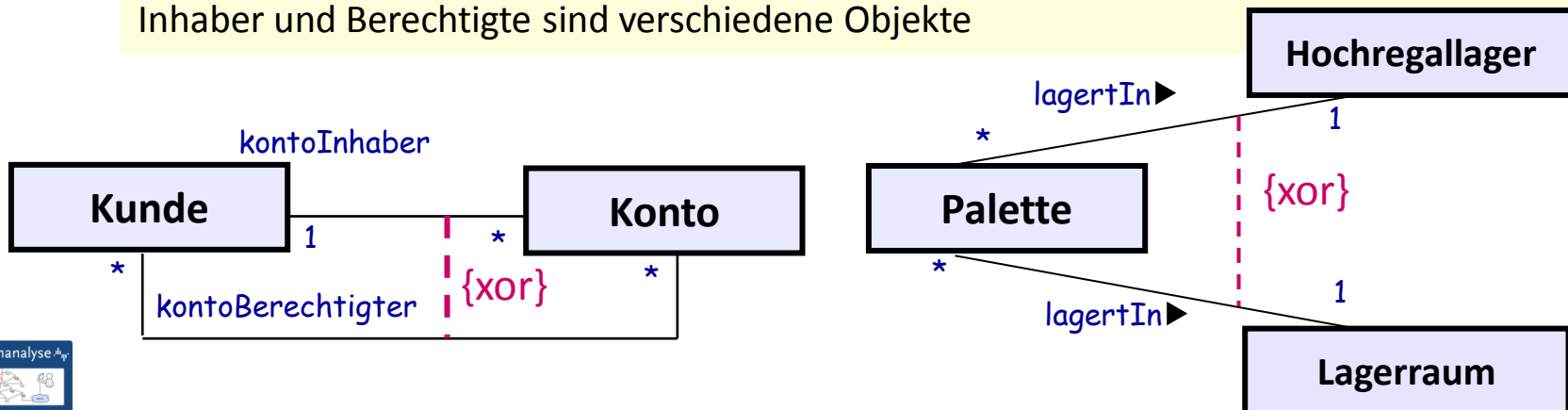


```
{ context Konto inv:  
    kontoBerechtigter->notEmpty() implies  
        not kontoberechtigter->includes(kontoInhaber) }
```

## Aussage:

Für **alle** Konto-Objekte, die in Beziehung zu einem Kunde-Objekt stehen, die jeweils einen *kontoInhaber* und beliebige *kontoBerechtigte* aufweisen, **gilt**:

Inhaber und Berechtigte sind verschiedene Objekte



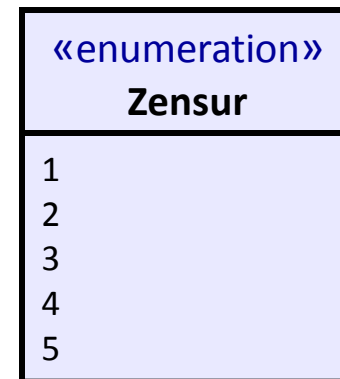
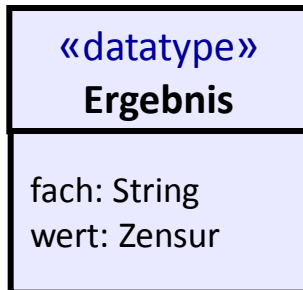
# Grundsätzliches zu Einschränkungen

- Einschränkungen sind
  - vordefinierte Standardeinschränkungen für UML-Modellelemente (aus dem Metamodell)
  - nutzerdefinierte Einschränkungen
- Syntax
  - {...} oder
  - {<name der Einschränkung>}
- jedem Modellelement dürfen beliebig viele Einschränkungen zugeordnet werden
- Beschreibung:
  - OCL, natürliche Sprache, beliebige Notation

# Datentypen

- ... sind Typen, deren Werte keine Identität besitzen

Boolean  
Integer  
UnlimitedNatural

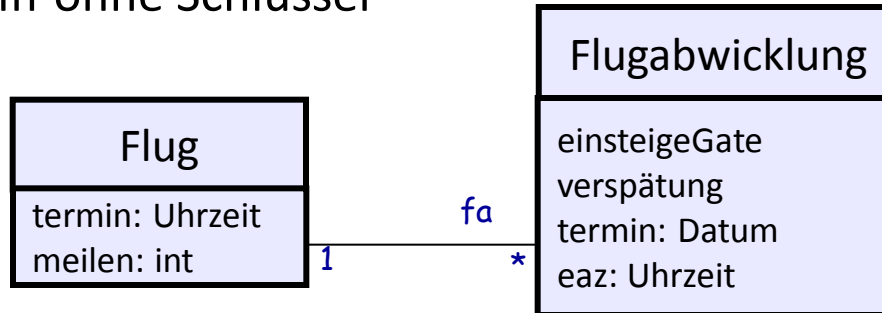


- Definition der Datentypen erfolgt als Klasse, aber mit Schlüsselwort (sehen aus wie Stereotype, sind aber keine).
- Stereotyp nimmt Bezug auf existierendes UML-Modellelement, schränkt aber Semantik ein (**hier**: Klassensymbol wird mit anderer Semantik eingesetzt)
- es gibt vordefinierte und nutzerdefinierte Stereotypen



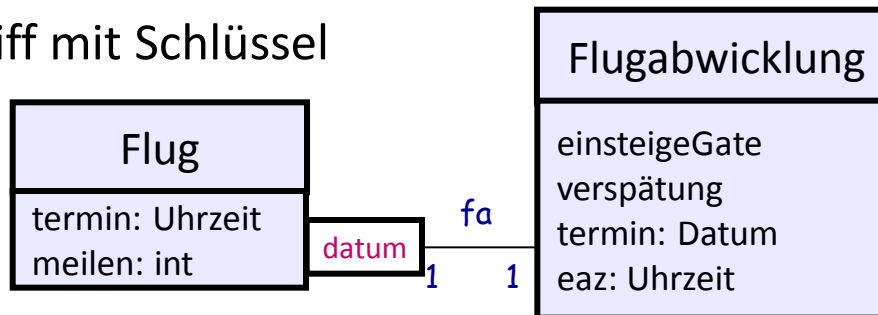
# Qualifizierte Assoziationen (1)

- Zugriff ohne Schlüssel



erlaubt für binäre Assoziationen (Schlüssel = Index)

- Zugriff mit Schlüssel

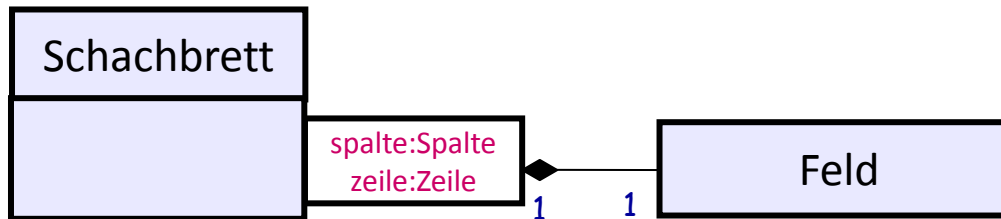
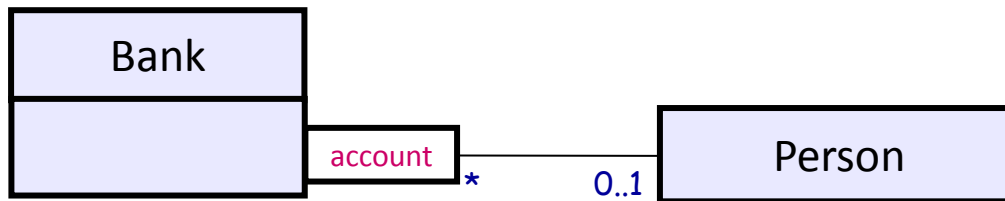


...wird eingesetzt, um die Vielfachheit einer Assoziation zu reduzieren

# Qualifizierte Assoziationen (2)

(bank,account)-Tupel  $\rightarrow$  0 oder 1 person-Objekt

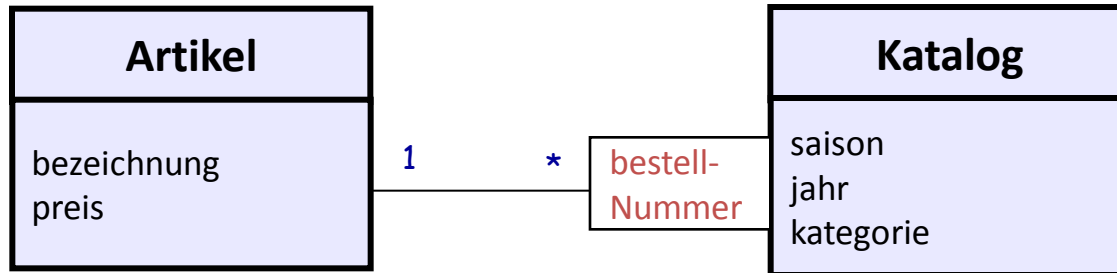
person -Objekt  $\rightarrow$  mehrere (bank,account)-Tupel



(brett,spalte,zeile)-Tupel  $\rightarrow$  1 feld-Objekt

feld-Objekt  $\rightarrow$  1 (brett,spalte,zeile)-Tupel

# Qualifizierte Assoziationen (3)



- innerhalb eines Kataloges bezeichnet eine Bestellnummer genau einen Artikel, wobei ein Artikel in mehreren Katalogen auftauchen kann,
- die Bestellnummern in verschiedenen Katalogen könnten dabei auch gleich sein



- ein Katalog enthält verschiedene Artikel
- ein Artikel kann in verschiedenen Katalogen enthalten sein
- es ist nicht ausgeschlossen, dass verschiedene Artikel eines Kataloges die gleiche Bestellnummer haben

semantischer Unterschied

# Liste möglicher Eigenschaften

für Assoziationsenden (auch für Attribute)

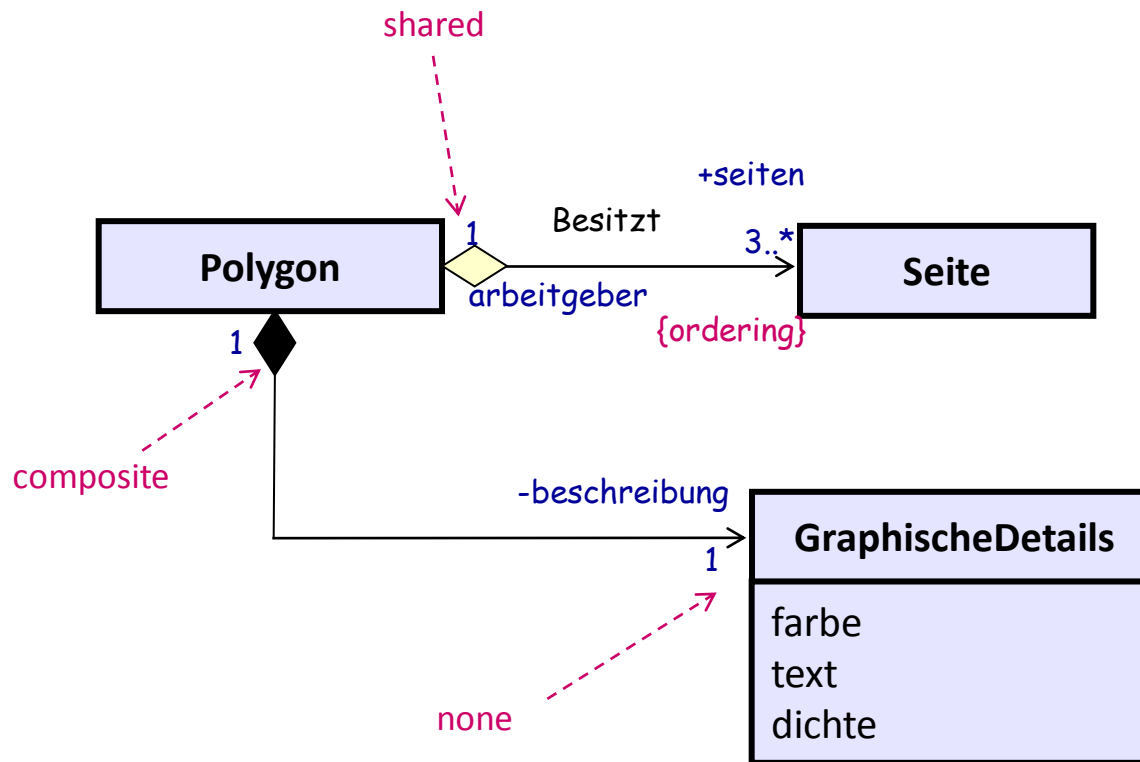
- / (abgeleitete Assoziation)
- readonly  $\leftarrow \rightarrow$  unrestricted
- composite
- redefines *Attr*
- subsets *Attr*
- union
- unique  $\leftarrow \rightarrow$  nonunique
- ordered  $\leftarrow \rightarrow$  unordered
- seq (Vor.: Multiplizität > 1)
- bag (Vor.: Multiplizität > 1)

default

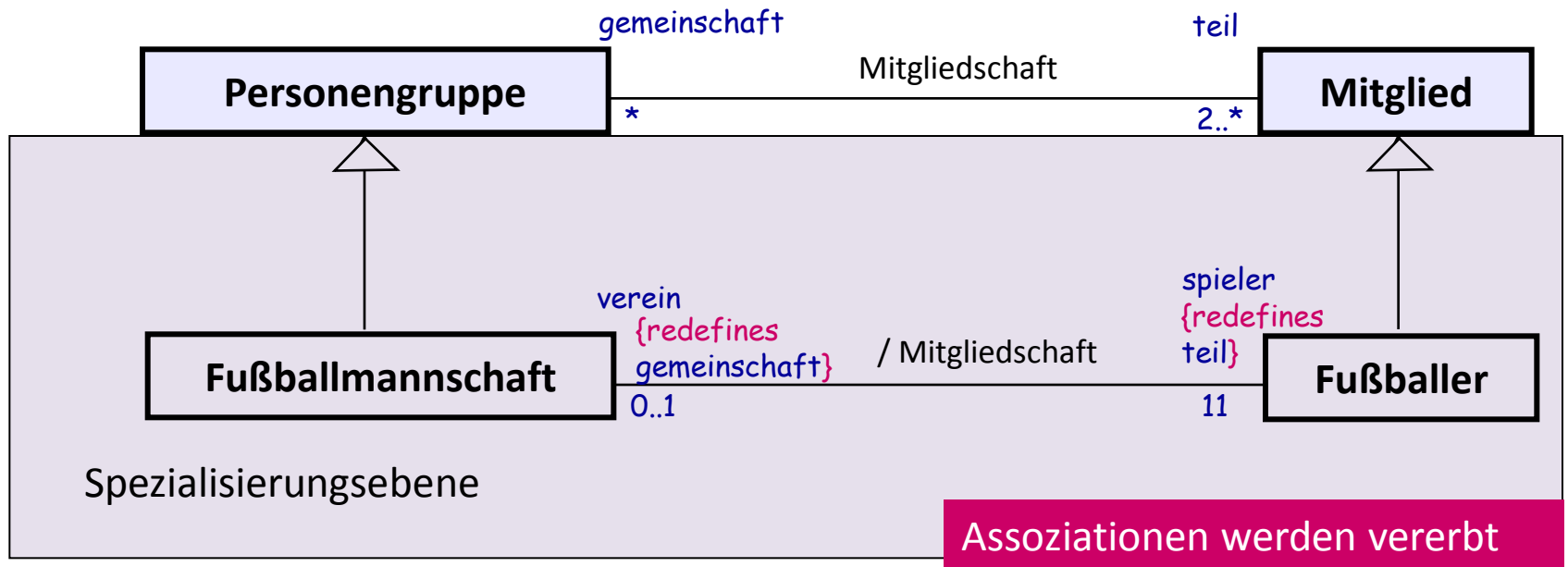
bedingt kombinierbar

# Eigenschaftswert: composite

Werte für composite: none, shared, composite



# Eigenschaftswert: **redefines**



Überschreibung von Rollennamen und Multiplizität  
→ Assoziationsenden werden **ersetzt** (nicht ergänzt)

Mitglied hat eine Eigenschaft

- **gemeinschaft** (Menge (evtl. leer) von Personengruppen)

Fußballer hat auch nur eine Eigenschaft

- **verein** (eine oder keine Fußballmannschaft)

Achtung:  
redefines nur im  
Spezialisierungsfall  
anwendbar

Properties von  
Classifiern  
sind potentielle  
Kandidaten

# Redefinition (allgemein)

- immer im Kontext von Classifier-Spezialisierungen
- Redefinition erfolgt in der Spezialisierung bei Angabe der originalen Definition und des dazugehörigen Classifiers
- Redefinition kann prinzipiell von folgendem Gehalt sein
  - Erweiterung
  - Einschränkung
- Notation hängt von der jeweiligen Elementtyp ab
  - *Behaviour*
  - *Classifier*
  - *Operation*
  - *Property*
  - *State Machine*
  - *Template*

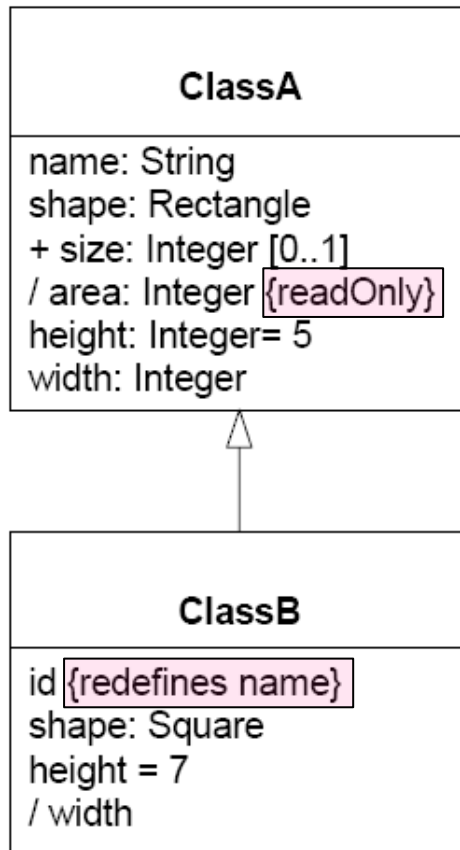
bislang vorgestellt

Was darf redefiniert werden?

Property-Redefinition des/der

- Namens
- Typs (als Spezialisierung wegen Polymorphie)
- Anfangswertes
- Ableitungstatus (/)
- Sichtbarkeit
- Multiplizität
- Werte-Constraints

# Eigenschaftswert: **readOnly** und **unRestricted**



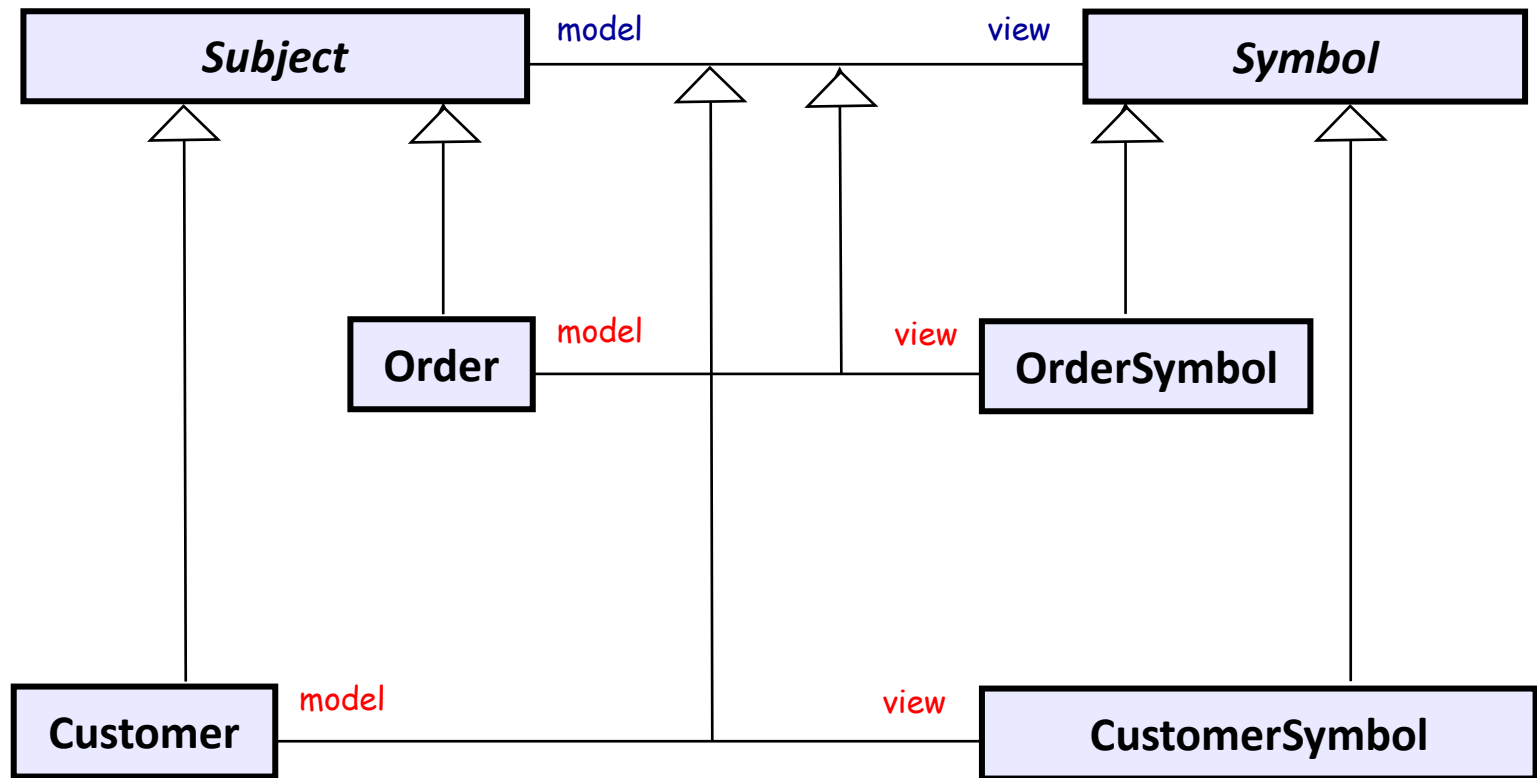
Default-Einstellung  
*{unRestricted}*

{derived} und {readOnly} sind kombinierbar

Redefinition von Properties  
(Attribute oder Assoziationsenden)



# Generalisierung von Assoziationen



## 3. Ausgewählte UML-Konzepte

1. Klassen und Objekte als UML-Basiskonzepte
2. Assoziationen und Attribute (Forts.)
3. Operationen und Methoden
4. UML-Schlüsselworte
5. Constraints in OCL

# Zusammenhang von Operationen u. Methoden

- Operationen werden eingesetzt, um das **Verhalten** von Objekten zu beschreiben
- alle Operationen einer Klasse (zusammengenommen) definieren die Möglichkeit der **Interaktion** mit Objekten dieser Klasse
- Operationen stellen die einzige Möglichkeit dar, **Zustandsänderungen** eines Objektes herbeiführen zu können
- Operationen definieren **Signaturen** (Operationsname, Übergabe-, Rückgabeparametertypnamen) von den sie implementierenden Methoden
- Operationen kennen **Ausnahmen** (*exception*), die aber nicht signaturbestimmend sind
- Methoden werden auf **Objekten** einer Klasse oder auf der **Klasse** selbst ausgeführt
- zu einer Operation kann es mehrere Methoden (d.h. Implementierungen) geben (**Polymorphie** von Operationen)

# Beispiele, Syntax

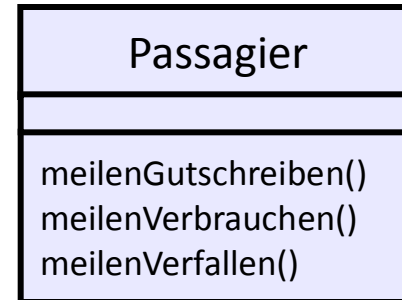
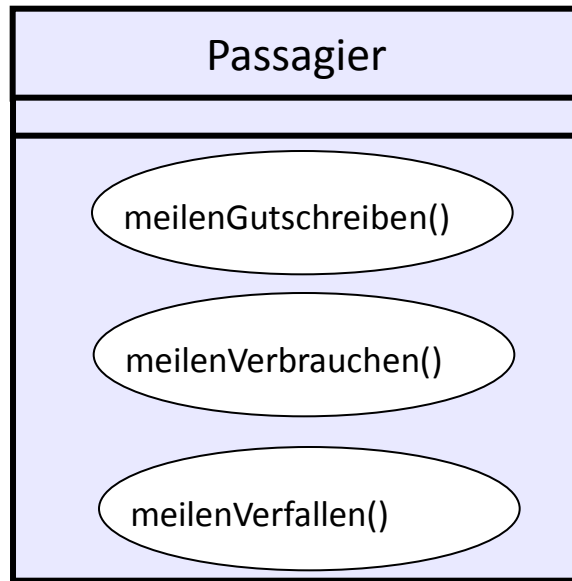
```
+ display(): Location  
+ hide()  
«constructor» +new(p: Policy)  
-attachXWindow(xwin: XWindow)
```

## Operationssyntax:

```
[«stereotype»]  
[visibility] name (paramList) [: returnType]  
[propertyStringList]
```

# Operationen (Syntaxformen)

- üblich: 2. separates Abteil



# Operationscharakteristika

«stereotype»  
[visibility] name (paramList) [:  
returnType]  
[propertyStringList]

- Abstraktheit (*abstract*)
- Ausführung (*concurrency*)
- Laufzeit-Bedingungen (*constraints*)
- Finalisierung (*leaf*)
- Seiteneffekt (*query*)
- Ausnahme (*exceptions*)
- **Name**
- Parameterliste
- Rückgabewerte
- Stereotype/Schlüsselwort
- Static (*static*)
- Sichtbarkeit
- Method (*method*)
- Signal (*signal*)
- Redefinition (*redefined*)

Ausprägung: sequential, guarded, concurrent

Ausprägung: pre-, post-, body-condition

Liste möglicher Ausnahmen

Name + Liste der Parametertypen (Return) → Signatur

Komma-separierte Liste von Tripeln: Richtung, Name, Typ

Komma-separierte Liste von: Klassen, Datentypen

«constructor», «destructor»

+, -, #, ~

wird gesetzt, sobald Implementierung abstrakt oder konkret vorliegt

«signal»

# Operation: return type, exception

- **Rückgabotyp**

- Liste von Classifier-Namen:  
Klassen, Datentypen, Interfaces (Rückgabe eines Objektes einer Klasse, die das Interface implementiert)
- es gibt Sprachen mit Operationen, wo mehrere Rückgabewerte erlaubt sind

- **Ausnahmen**

- als `propertyString`
- Ausnahmen sind Typen, Werte dieser Typen werden im Ausnahmefall an den Rufer zurückgegeben

boardPassenger (who passenger) **exception** AlreadyBoarded

`propertyString`

# Operation: parameter list

- **Syntax**
  - [direction] name : type [= default-value]
  - direction-Möglichkeiten: in, out, inout

**Achtung:**

out- und inout-Parameter sind äquivalent zu

- einem return-Parameter und
- einem in-Parameter



# Operation: **Signatur**

... wird gebildet aus

- Name der Operation (Schreibweise-Konvention: Verb, klein)
- Parametertypnamen (keine Parameternamen)
- Rückgabetyt (~ impl.abhängig)

d.h. ohne Ausnahmen

C++, Java, Python: ohne  
SDL: mit

## Achtung:

kein klarer Hinweis darauf, ob dies über einen semantischen Variationspunkt geregelt werden kann/muss, Vermutung: das ist ein Variationspunkt

- zwei Operationen mit **gleicher Signatur** sind über Spezialisierungsgrenzen hinweg nur erlaubt, wenn die Spezialisierung dabei die Operation als *redefined* kennzeichnet

# Operation: Exception und Signal

- Exception-Typen wird durch nutzereigene passive Klassen dargestellt.
- Vordefinierte Exception gibt es nicht in UML
- Das Werfen einer Exception entspricht dann der Lieferung einer Referenz zu einem entsprechendem Objekt

- Signale (-Typen) sind spezielle Classifier
- Ereignisse (Signalsenden- Signalempfang) sind mit Instanzen der Signale verbunden

# Operation: **abstract** und **leaf**

- Kennzeichnung abstrakter Operationen:
  - Kursive Namensschreibweise
  - **abstract** (propertyString)
- jeder nicht-abstrakten Operation muss mindestens) eine Methode zugeordnet sein (Implementation)

- Kennzeichnung nicht redefinierbarer Operationen:
  - **leaf** (propertyString)
- kann (im weiteren Spezialisierungskontext) **nicht** redefiniert werden

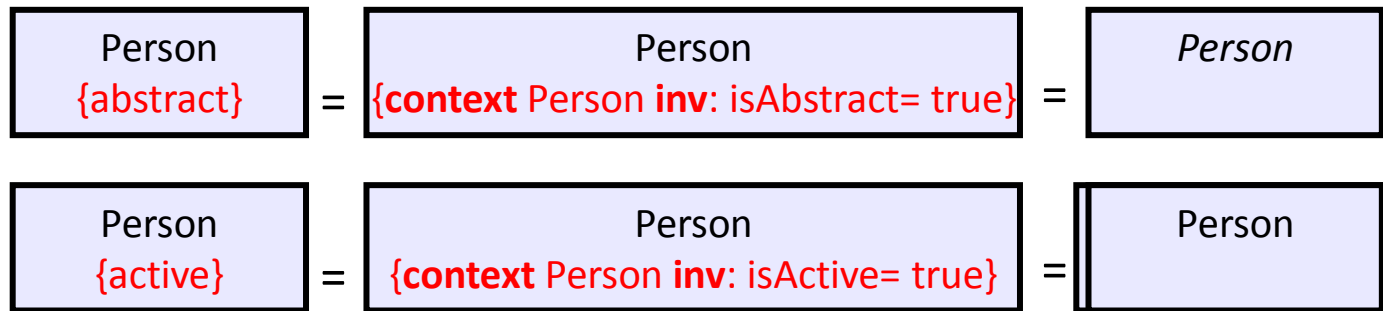
# Operation: *concurrency*

## Werte:

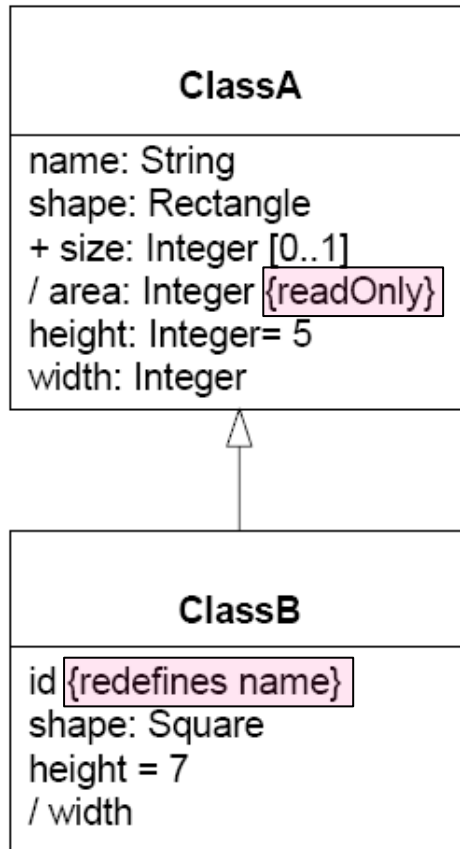
- *sequential*  
gleichzeitiger mehrfacher Aufruf wird **nicht** unterstützt,  
Integrität wäre verletzt
- *guarded*  
Blockierung mit Zwangssequentialisierung der Aufrufe
- *concurrent*  
gleichzeitige mehrfache Ausführung wird unterstützt  
(Speicherkonflikte werden irgendwie gelöst)

Default-Einstellung: *sequential*

# Aktive und abstrakte Klassen



# Eigenschaftswert: **readOnly** und **unRestricted**



Default-Einstellung  
*{unRestricted}*

{derived} und {readOnly} sind kombinierbar

Redefinition von Properties  
(Attribute oder Assoziationsenden)

## *3. Ausgewählte UML-Konzepte*

1. Klassen und Objekte als UML-Basiskonzepte
2. Assoziationen und Attribute (Forts.)
3. Operationen und Methoden
4. UML-Schlüsselworte
5. Constraints in OCL

# Anwendungsklassen von Schlüsselworten

(aus Anhang B der Superstructure...)

- graphische Unterscheidung von Symbolen für verschiedene **UML-Konzepte**, z.B.: «interface», «signal», ...
- graphische Unterscheidung von verschiedenen **Abhängigkeitsbeziehungen**, z.B.: «create»,
- Spezifikation von sogenannten **Modifier** (Metaklassenattribute), z.B.: «active»,
- Anzeige von Standard-**Stereotype** und Stereotype nutzer-definierter UML-Profile, z.B.: «modelLibrary» für ein Package: zeigt ein Paket an, dessen Elemente von vielen Modellen geteilt werden

1. Nicht alle Worte mit «...» markierten Worte sind Schlüsselworte (z.B. Zeichenketten)
2. Nicht alle Worte mit «...» markierten Worte identifizieren Stereo-Typen
3. Falls mehrere Schlüsselworte bzw. Stereotype ein Modellelement charakterisieren: «label1, label2, ...»



# Liste von Schlüsselworten (1)

Schlüsselwort	Konzept	S T	Schlüsselwort	Konzept	S T
«abstraction»	Abstraction		«create»	Usage	
«access»	Package Import		«create»	Behavioural Feature	
«activity»	Activity		«create»	Dependency	
«actor»	Actor		«datastore»	DataStoreNode	
«after»	TimeEvent		«datatype»	DataType	
«all»	Any-Receive-Event		«delegate»	Konconnector	
«apply»	Profile-Application		«deploy»	Konconnector	
«artifact»	Artifact		«deployment spec»	Deployment Specification	
«artifacts»	Component		«derive»	Abstraction	
«at»	TimeEvent		«destroy»	Behavioral Feature	
«attribute»	ActivityPartition ::represents		«device»	Device	
«auxiliary»	Classifier		«document»	Artifact	
«buildComponent»	Component		«element access»	Element Import	
«call»	Usage		«element import»	Element Import	
«centralBuffer»	CentralBufferNode		«entity»	Component	
«class»	ActivityPartition				
«component»	Component				

*Schlüsselwort zur Identifikation von Sterotype*

= constructor

*bereits eingeführt*

= destructor

# Liste von Schlüsselworten (2)

Schlüsselwort	Konzept	S T
«enumeration»	Enumeration	
«executable»	Artefact	
«execution environment»	Execution Environment	
«extend»	Extend	
«extended»	Region	
«extended»	State Machine	
«external»	ActivityPartition	
«file»	Artifact	
«focus»	Class	
«framework»	Package	
«from»	Trigger	
«implement»	Component	
«implementation Class»	Class	
«import»	Packageimport	
«include»	Include	
«information»	InformationItem	
«instantiate»	Dependency	

Schlüsselwort	Konzept	S T
«instantiate»	Usage	
«interface»	Interface	
«library»	Artefact	
«local Postcondition»	Constraint	
«local Precondition»	Constraint	
«manifest»	Manifestation	
«merge»	PackageMerge	
«metaclass»	Classifier	
«metamodel»	Model	
«model»	Model	
«modelLibrary»	Package	
«multicast»	ObjectFlow	
«multireceive»	ObjectFlow	
«occurence»	Collaboration	
«postcondition»	Constraint	

# Liste von Schlüsselworten (3)

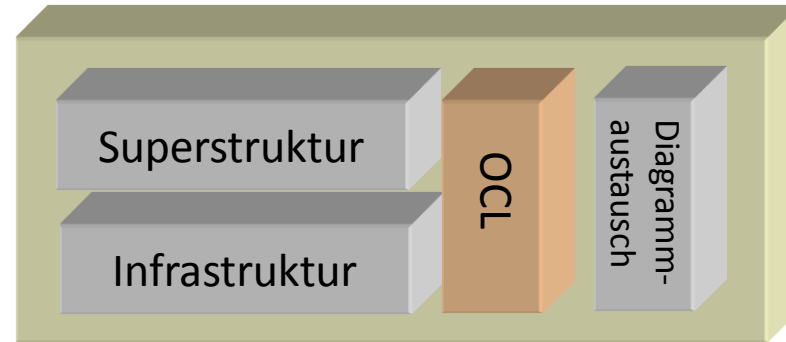
Schlüsselwort	Konzept	S T
«precondition»	Constraint	
«primitive»	Primitive Type	
«process»	Component	
«profile»	Profile	
«provided interfaces»	Component	
«realization»	Classifier	
«realizations»	Component	
«refine»	Abstraction	
«representation»	Classifier	
«represents»	Collaboration	
«required interfaces»	Component	
«responsibility»	Usage	
«script»	Artefact	
«selection»	Behaviour	
«selection»	Behaviour	
«send»	Usage	

Schlüsselwort	Konzept	S T
«service»	Component	
«signal»	Signal	
«singleExecution»	Activity	
«source»	Artefact	
«specification»	Classifier	
«statemachine»	Behaviour	
«stereotype»	Stereotype	
«structured»	Structured ActivityNode	
«substitute»	Substitution	
«subsystem»	Component	
«systemModel»	Model	
«trace»	Abstraction	
«transformation»	Behaviour	
«type»	Class	
«use»	Usage	
«utility»	Class	
«when»	ChangeEvent	

## 3. Ausgewählte UML-Konzepte

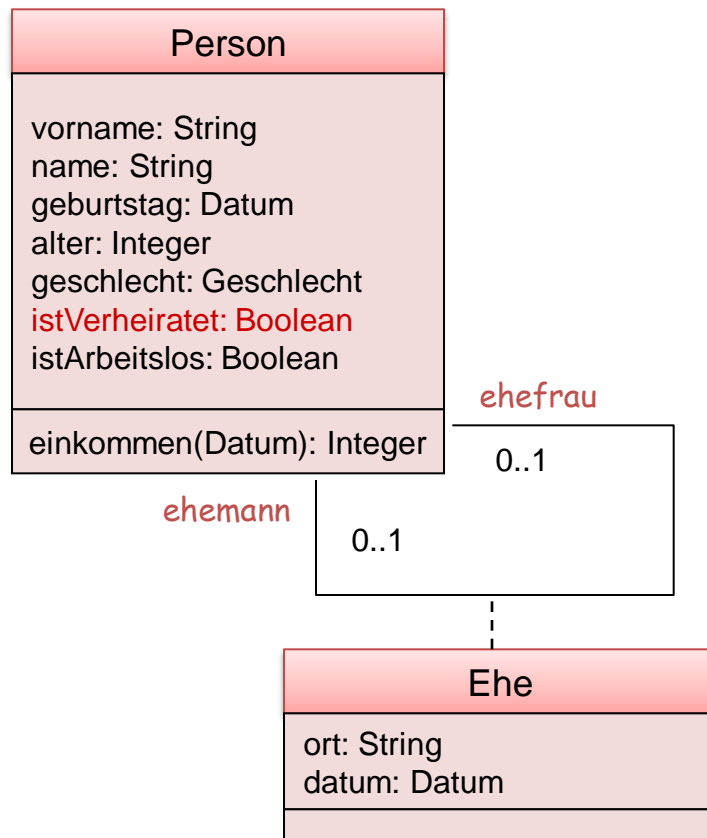
1. Klassen und Objekte als UML-Basiskonzepte
2. Assoziationen und Attribute (Forts.)
3. Operationen und Methoden
4. UML-Schlüsselworte
5. Constraints in OCL
  - Allgemeine Charakterisierung
  - OCL-Typen und Standardoperationen (Überblick)
  - Beispiel-Constraints

# Einordnung von OCL



- OCL ist eine textuelle Sprache
  - Empfehlung der OMG zur Formulierung von
    1. Randbedingungen/Einschränkungen
    2. Anfragen
    3. Aktionen
    4. Navigationen
- } in Form von Annotationen
- OCL basiert auf
    - Prädikatenlogik (1.Stufe) und
    - Mengentheorie in einer (weitgehend) intuitiven Syntax
  - prinzipiell ist aber jede andere Sprache als Alternative in UML zulässig (inkl. natürliche Sprachen)

# Beispiel: Präzisierung durch Invarianten



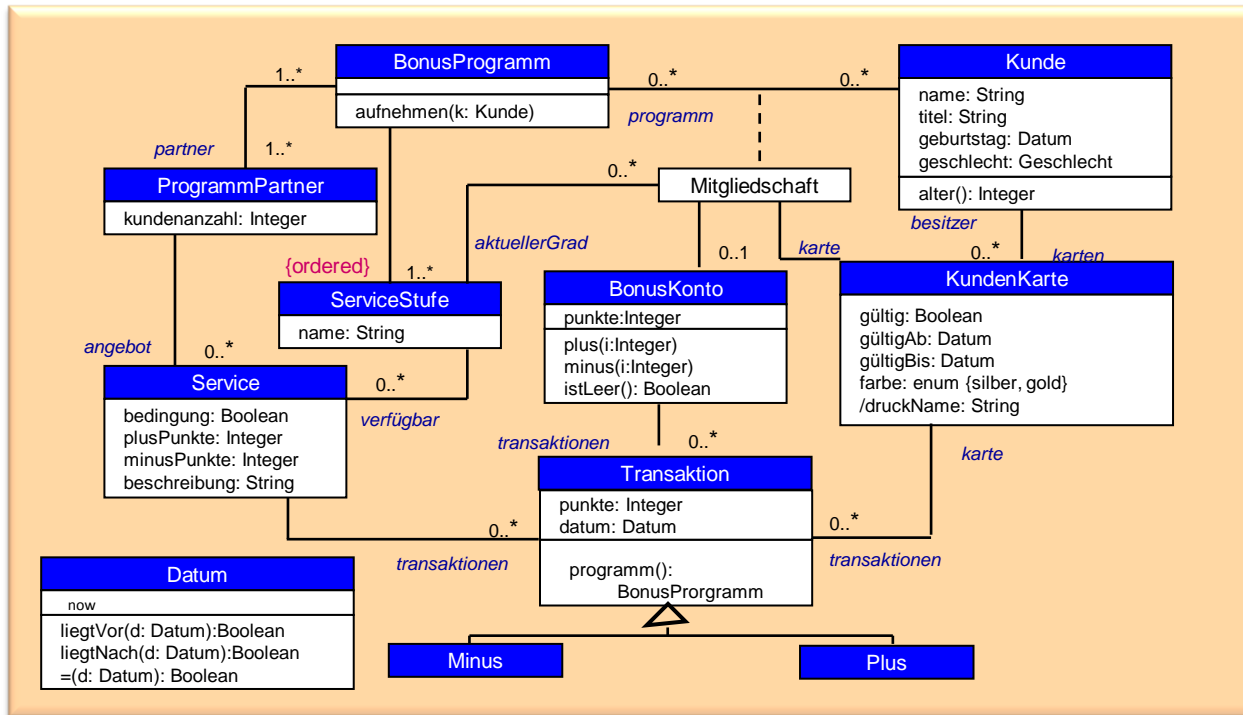
**context** Person

```
inv: self. istVerheiratet implies
    self.eheFrau->notEmpty()
    xor
    self.eheMann->notEmpty()
```

```
inv: self. eheFrau->notEmpty() implies
    self.eheMann->isEmpty()
```

```
inv: self. eheMann->notEmpty() implies
    self.eheFrau->isEmpty()
```

# Festlegung von Anfangswerten



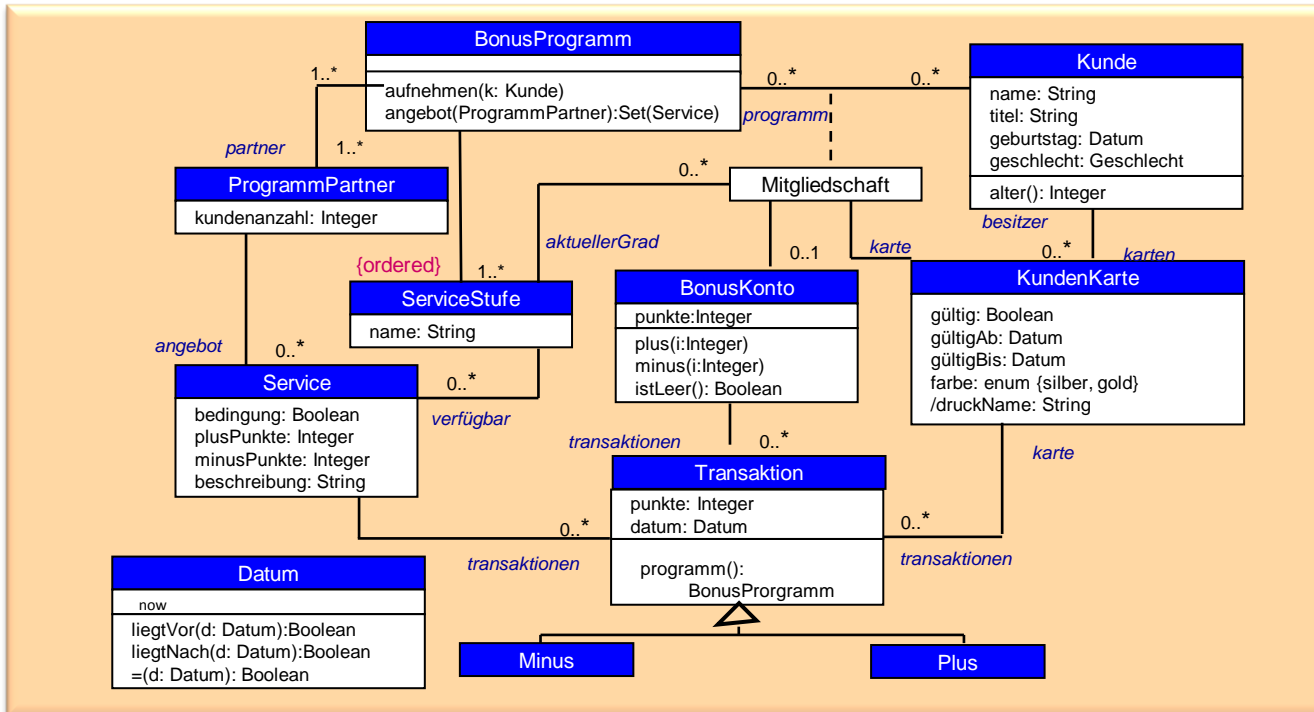
**context** BonusKonto::punkte  
init: 0

**context** Kundenkarte::gültig  
init: true

**context** Kundenkarte  
**inv:** druckName = besitzer.titel.concat (besitzer.name)

abgeleitetes Attribut

# Anfrageoperationen

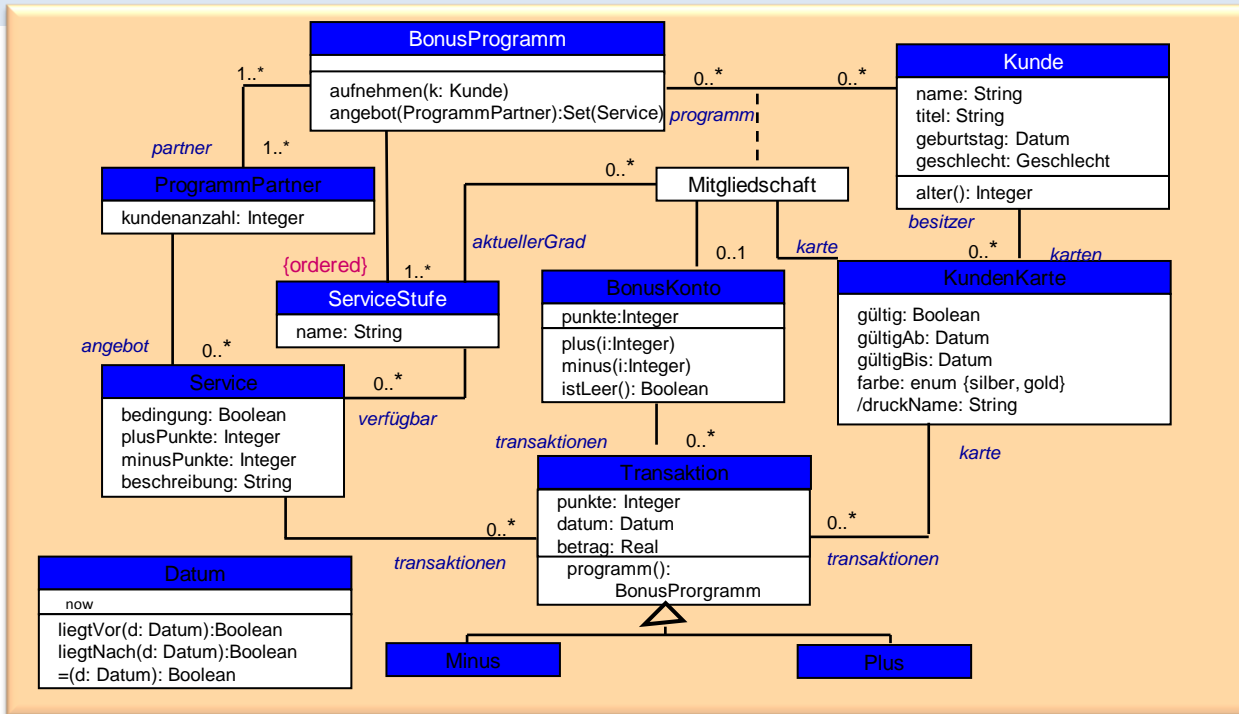


**context** Bonusprogramm::angebot(pp: ProgrammPartner): Set(Service)  
**body:** if partner->includes(pp) then pp.angebot else Set{}

Bedingung für Rückgabewert



# Assoziationsklassen



**context** BonusProgramm

**inv** bekannteServiceStufe: serviceStufe->**includesAll**(Mitgliedschaft.aktuellerGrad)

**context** Mitgliedschaft::gesamtAngebot(): Set(Service)

**inv** kartenKorrekttheit: kunde.karten->**includes**(self.karte)