VORLESUNG

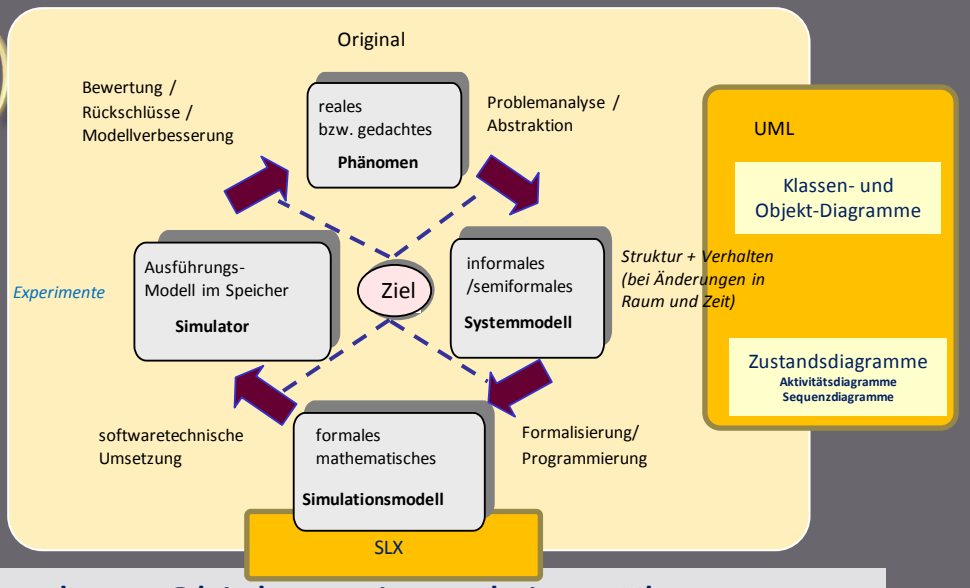
Automatisierung industrieller Workflows

Teil B: Die Spache UML

Joachim Fischer

Zusammenfassung (2)

... formuliert als Fragensammlung



- Charakterisieren Sie den Laufzeitzustand von Objekten einer aktiven Klasse, deren Lebenslauf durch UML-Zustandsautomaten beschrieben ist.
- Erläutern Sie die Präzisierung der UML-Semantik für den Pool von Eingangsnachrichten (Signale, Timeouts, Remote-Prozedurrufe) entspr. SDL
- Welche Triggerarten sind für einen UML-Automaten definiert?
- Welche Bedeutung hat die verpflichtende Angabe potentieller Eingabe-Signale?
- Welche Reihenfolgen von Aktionen (ENTRY, DO, EXIT, DEFER, Aktionen beim Zustandsübergang) ergeben sich bei einem Zustandsübergang?
- Was versteht man unter einem Finalisierungsereignis, wie wird es ausgelöst, wie wird es als Trigger eines Zustandsübergangs wirksam?
- Welcher Unterschied besteht zwischen der Behandlung eines Zustandsereignisses und einer sogenannten Wächterbedingung?

Inhalt

- **Teil A**
  Aspekte von Modellierung und Simulation dynamischer Systeme
- **Teil B**
  Die Modellierungssprache UML
- **Teil C**
  Die ausführbare Modellierungssprache SLX
- **Teil D**
  Modellierung von Lieferketten

- **B.1**
  Wozu UML im Kontext der Computersimulation?
- **B.2**
  UML-Teilsprachen, Sprachkonzepte
- **B.3**
  Klassendiagramme
- **B.4**
  Verhaltensbeschreibung mit Zustandsautomaten
- **B.4' (zweite Lesung)**
  Verhaltensbeschreibung mit Zustandsautomaten
  - Wirkung von Triggerkombinationen
  - Beispiel-3 „Fahrkartenautomat"
  - Beispiel-4 „Maschinenbelegung"
- **B.5**
  OCL (Ein kurzer Streifzug)

Object Constraint Language (OCL)

Mengen u. Prädikatenlogik 1.Stufe

- Definition von
  - Invarianten für Klassen u. Typen in Klassendiagrammen
  - Vor- und Nachbedingungen für Operationen
  - Wächterbedingungen
  - Ziele (Zielmengen) von Nachrichten
  - Ableitungsregeln von Attributen
  - **Keine Aktionen !!!**
- Navigationsbeschreibung (im Objektmodell)
  - Anfragesprache

Präzisierung von UML-Modellen

- Built-In-Nutzung zur
  - Definition von *Well-Formedness Rules* für
    - Invarianten und
    - Metaklassen

    in der Abstrakten Syntax der Sprache

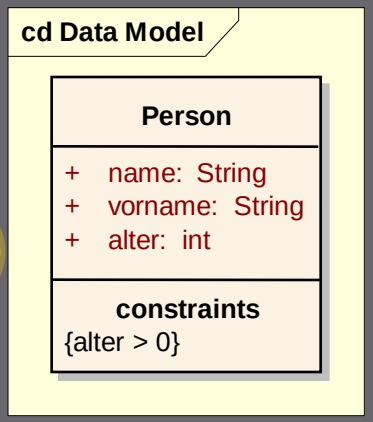
Präzisierung der UML-Sprach-definition

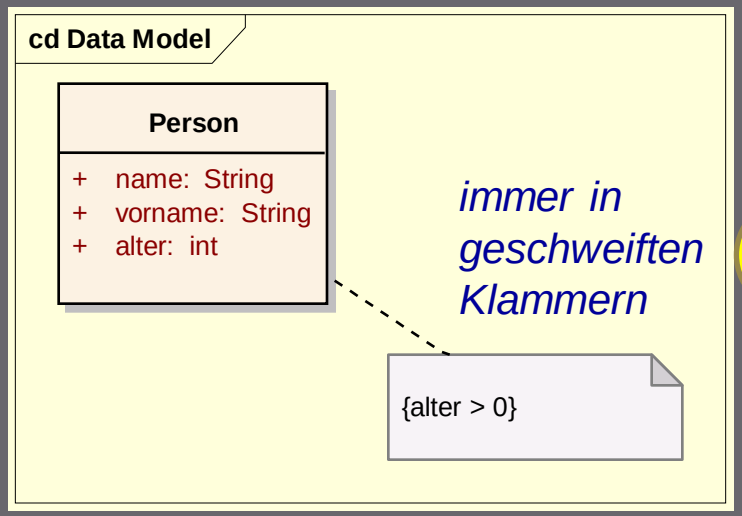
Arten einer Invarianten-Notation

Invarianten für
Klassen, Typen, Stereotypen

im UML-File

1





immer in geschweiften Klammern

2

als OCL-File
mit Bezug zum UML-File

3

```
package Datenbasis

    context Person
        inv:
        alter > 0

    context ...

endpackage
```

oder …

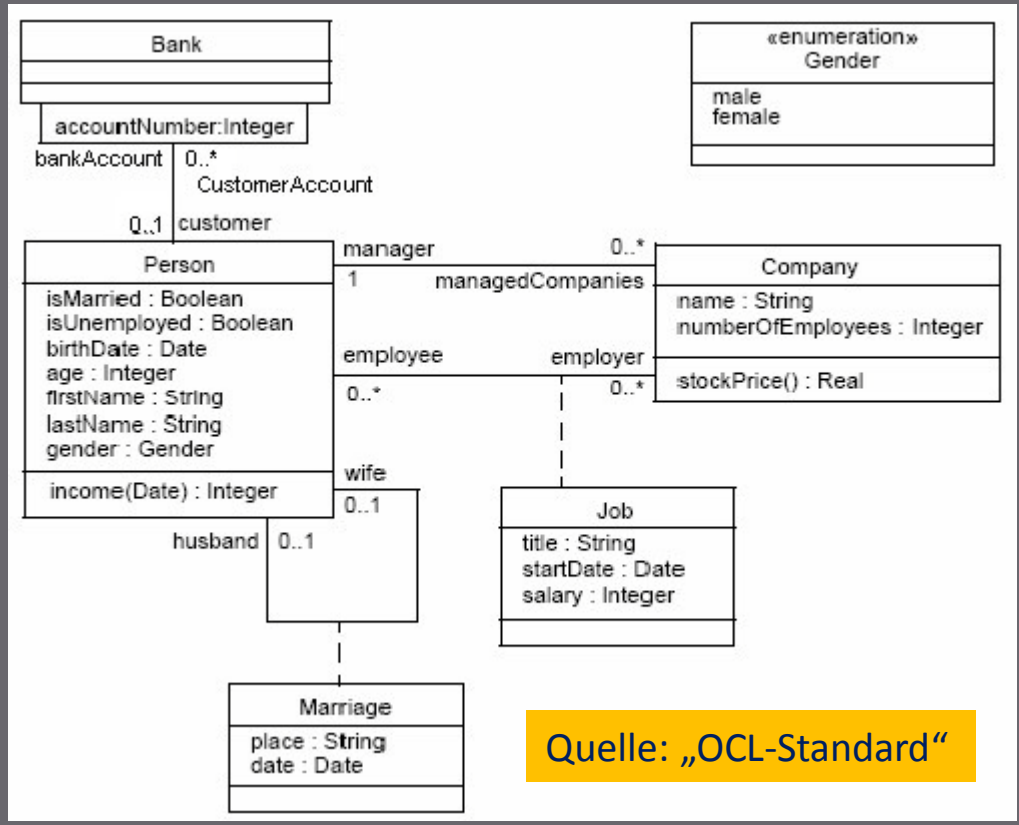
```
context
Datenbasis::Person
    inv:
    alter > 0
```

4

Bezug zum UML-Modell

Jeder OCL-Ausdruck ist im Kontext einer Instanz eines spezifischen Typs definiert

self stellt den Bezug her



context Company **inv**:
self.numberOfEmployees > 50

~ Invariante
logische Bedingung, immer True

context Person::income(d : Date) : **Integer**
post: **result** = 5000

~ Nachbedingung, immer True

Navigation entlang einer Assoziation

object . associationEndName

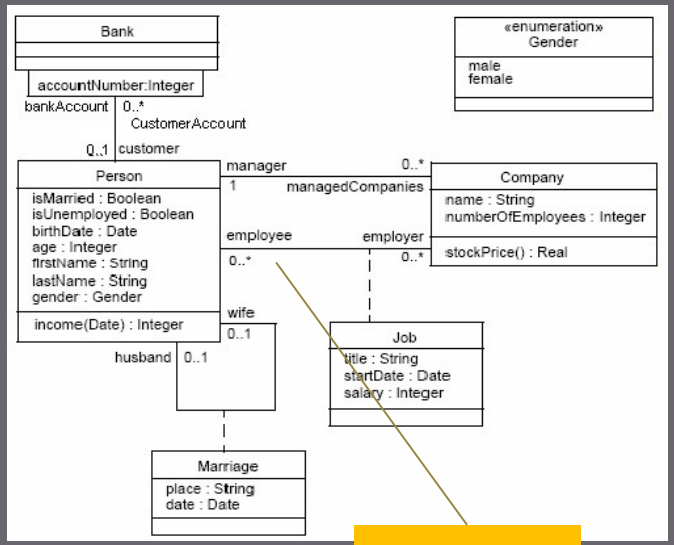
Wert ist Objekt-Kollektion

Ist die Multiplizität “0..1” oder “1”, dann ist der Wert des Ausdrucks ein Object.

Objekt

context Company **inv**:
  self . manager . isUnemployed = false
  self . employee -> notEmpty()

Kollektionsobjekt
Kardinalität >= 0



Standardannahme: employee: Set<Person>

{ordered}

dann: employee: OrderedSet<Person>

OCL-Kollektionen

Vier Ausprägungen von Kollektionen : Kollektor-Template <T>
Aber: OCL ist keine OO-Sprache

- Set(T)
- Sequence(T)
- Bag(T)
- OrderedSet(T)

<u>Operationen</u>
Für alle Kollektionen gibt es eine Menge gemeinsamer OCL-Operationen

Zusätzl. gibt es für einige Kollektionsarten spezifische Operationen

Navigationsoperatoren

- Zwei Navigationsoperatoren: "." and "->".

- "."
  Navigation, ausgehend von einem einzelnen Objekt
  objekt . Attribut
  objekt. Operation()

  z.B.: aString.indexOf(':')

- "->"
  Navigation, ausgehend von einer Kollektion
  kollektion->attribut
  kollektion->Operation()
  Kollektion->Iteration

aBag->elementType
aSet->union(anotherSet)
aSet->collect(name)

Kurznotationen

zwei Kurznotationen: "." and "->".

- "."

  aSet.name

  Kurznotation für Collect-Operation

  aSet->collect(name)

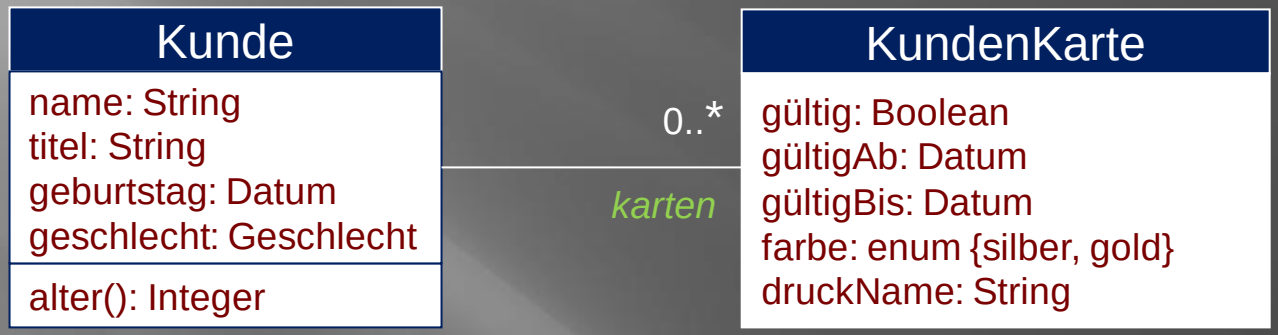
- "->"

  anObject->union(aSet)

  Kurznotation für

  anObject.oclAsSet()->union(aSet)

Einfaches Beispiel



context Kunde::alter(): **Integer**
post: result>= 18

context KundenKarte **inv**:
gültigAb.liegtVor(gültigBis)

Allgemeine Kollektionsoperationen

Standard-Operationen (1)

```
a->size()                          : Integer (-- Kardinalität)
a->count (Object)                  : Integer (-- Vorkommen eines Objektes)
a->includes (Object)               : Boolean (-- true, falls Objekt ein Element von a)
a->includesAll (Collection)        : Boolean (-- true, falls alle Collection-Objekte Element …)
a->isEmpty ()                      : Boolean
a->notEmpty ()                     : Boolean
a->sum()                           : Integer/Real  (-- a muss Elemente enthalten
                                                      für die +Operation gilt)
a->union(B)                        : Collection
a->intersection(B)                 : Collection
a->product(c2: Collection(T2))     : Set (Tuple (first: T, second: T2))
a->flatten(): Set(T2)              : einfache Menge
```

Allgemeine Kollektionsoperationen

Standard-Operationen (2)

a->forAll (expr oclExpr): Boolean

a->exists (expr oclExpr): Boolean

a->isUnique (): Boolean

a->any (expr BooleanExpr): T

a->one (expr BooleanExpr): Boolean

a->collect ...

a->select ...

a->iterate ...

Präzisierung im aktuellen OCL-Standard

any: liefert nichtdeterministisch ein Element der Kollektion, für das die Eigenschaft gilt

one: true, falls es genau ein Element in der Kollektion gibt, für das die Eigenschaft gilt

collect: sammelt ein bestimmtes Attribut aller Elemente der Kollektion ein und bildet daraus eine neue Kollektion

select: bildet Teilmenge der Kollektion, deren Elemente eine bestimmte Bedingung erfüllen

Mengen- und Sequenz-Operationen

- **Set**

  Set {1,2,3,4} - Set {2,4} /* liefert Set {1,3} */
  Set {1,2,3,4} ->symmetricDifference(Set {2,4,5}) /* liefert Set {1,3,5} */

- **OrderedSet**

  a->asSet() /* liefert Set */

- **Sequence**

  a->first()
  a->last()
  a->at (int)
  a->append (Object)
  a->prepend (Object)

Sequenz und OrderedSet sind zwar geordnet, aber nicht a priori sortiert!!!

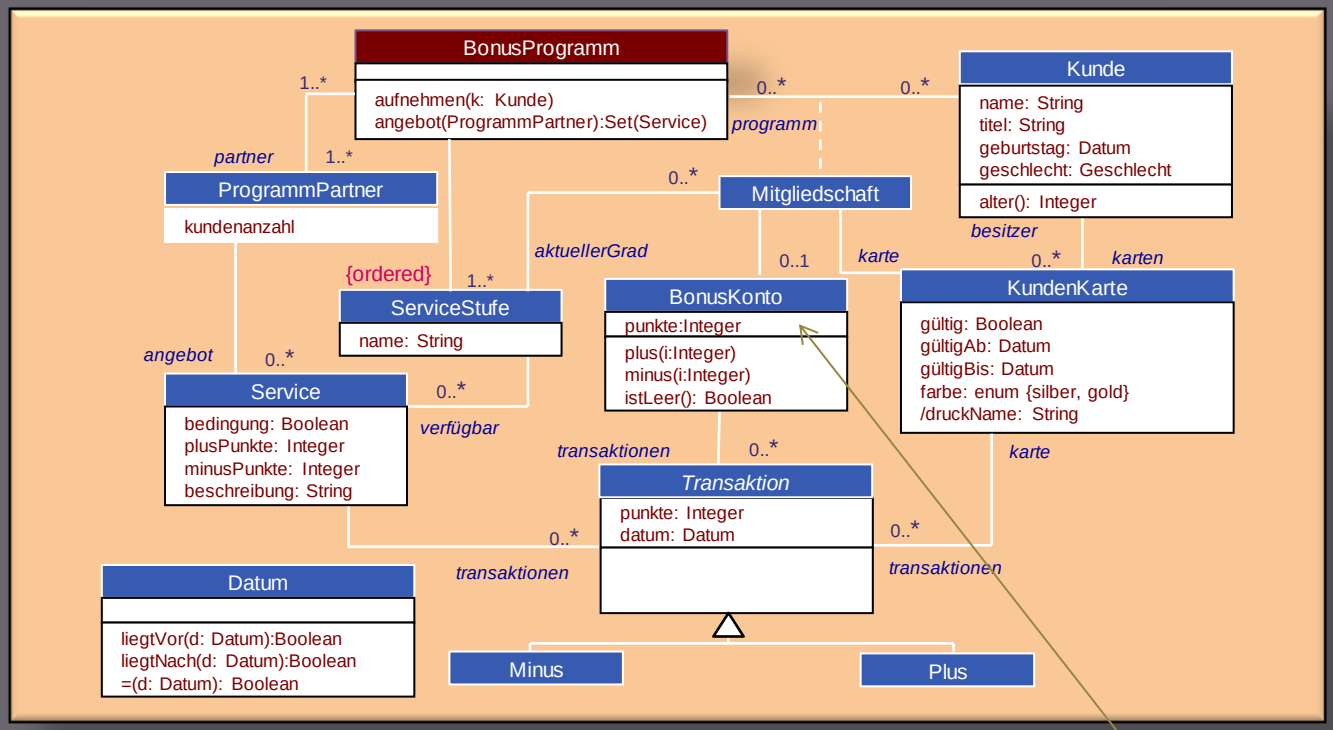
- **sortierte Kollektionen (wenn als Invariante formuliert)**

  a->sortedBy (<Property vom Typ von A>) als Sequenz bzw. OrderedSet

  // entsprechend < -Elementevergleich
  // Voraussetzung: für Property ist dieser Vergleichsoperator definiert

Sortierung

Sequence und OrderedSet sind zwar geordnet, aber nicht sortiert!



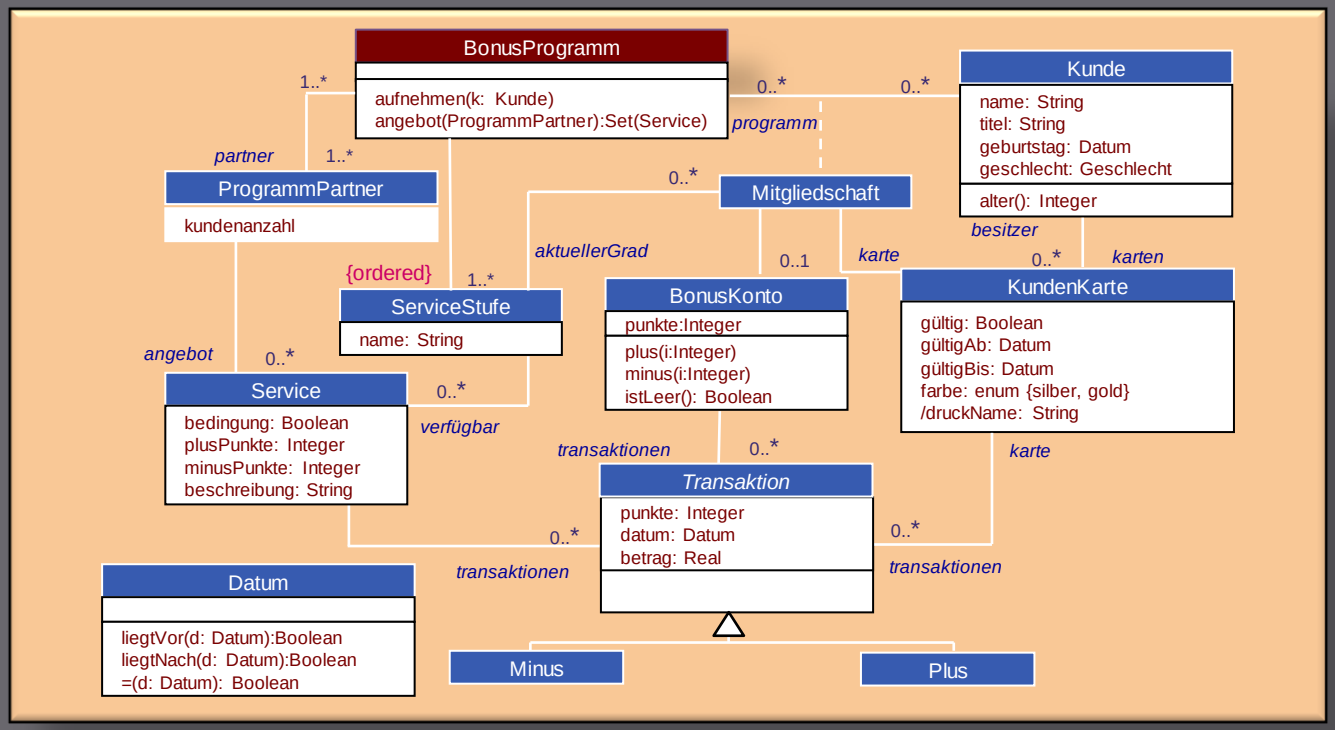
context Bonusprogramm

def: sortedAccounts : Sequence(BonusKonto) =
self.mitgliedschaft.bonusKonto->sortedBy (punkte)

// nach kleiner-als-Elementevergleich

// Voraussetzung: für Property ist < -Operator definiert

Definition lokaler Variablen und Operationen



context BonusKonto
def: x: Real= transaktionen->collect(betrag)->sum()

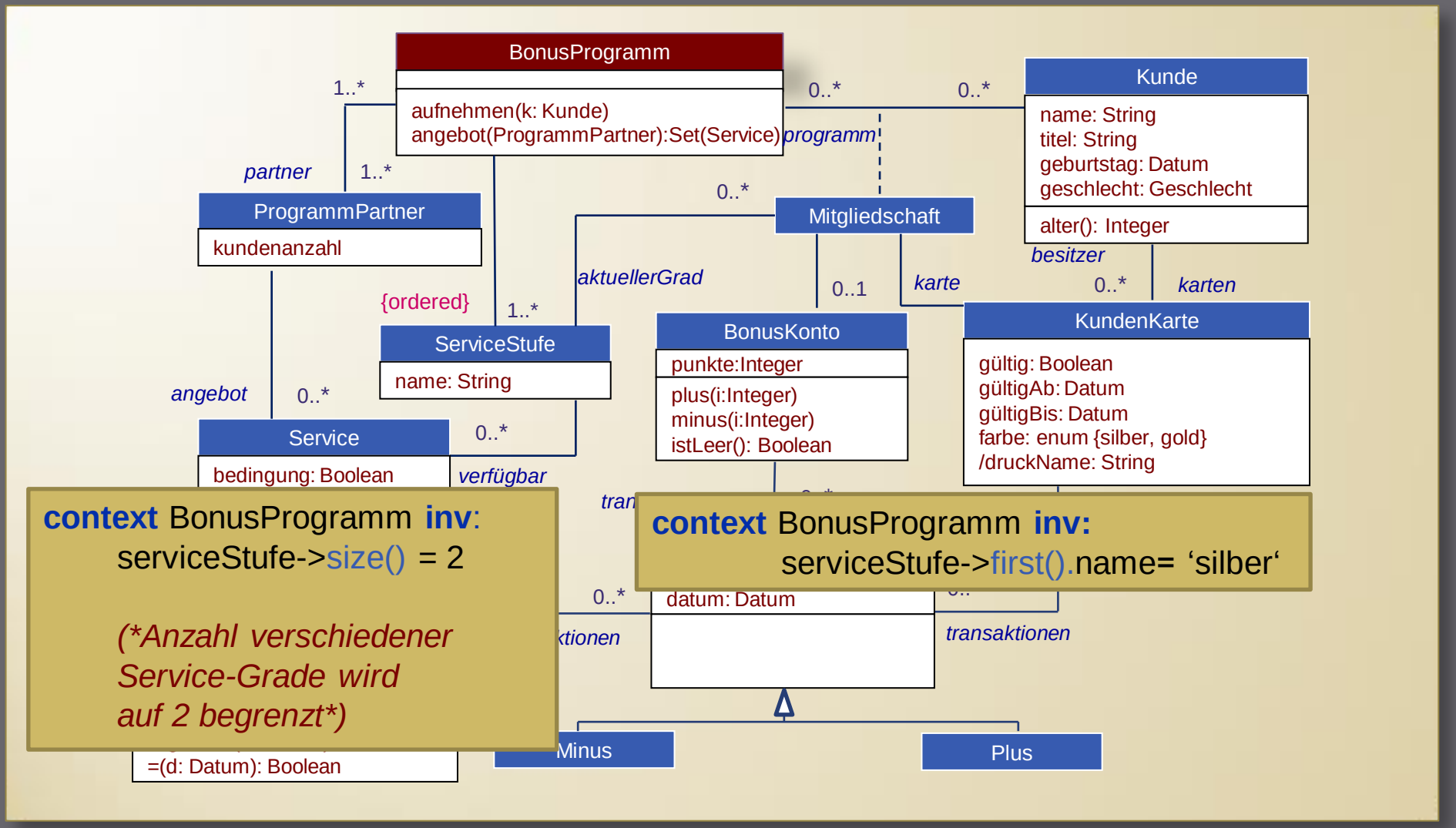
Menge von Real-Werten *sum iteriert über Ausgangsmenge*

Bedingung für Rückgabewert

context Bonusprogramm::gesamtAngebot(): Set(Service)
body: partner->collect(angebot)->asSet()

Multimenge von Service-Objekten

Invarianten-Beispiel (1)

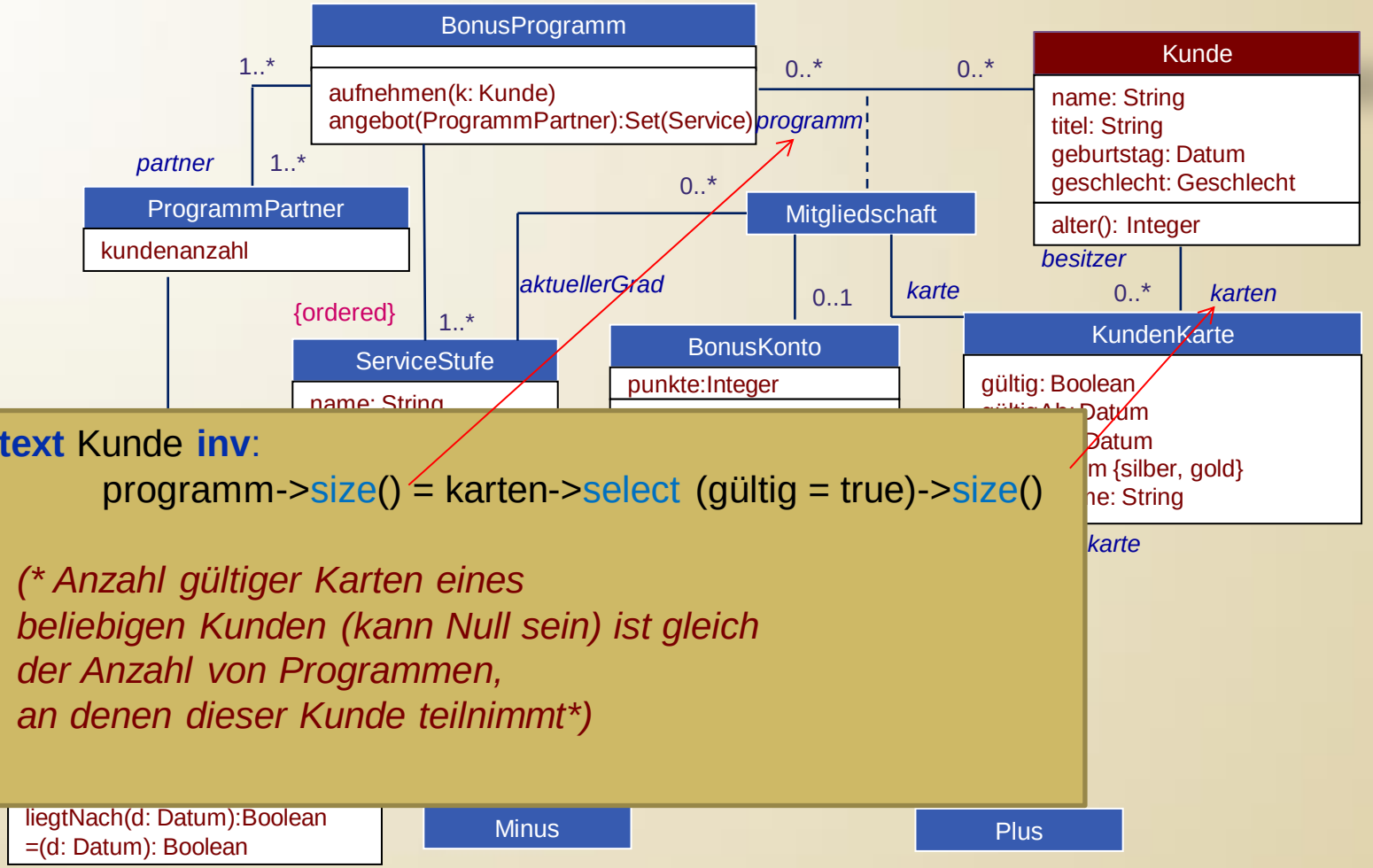


context BonusProgramm **inv**:
serviceStufe->size() = 2

*(*Anzahl verschiedener Service-Grade wird auf 2 begrenzt*)*

context BonusProgramm **inv:**
serviceStufe->first().name= ‘silber‘

Invarianten-Beispiel (2)

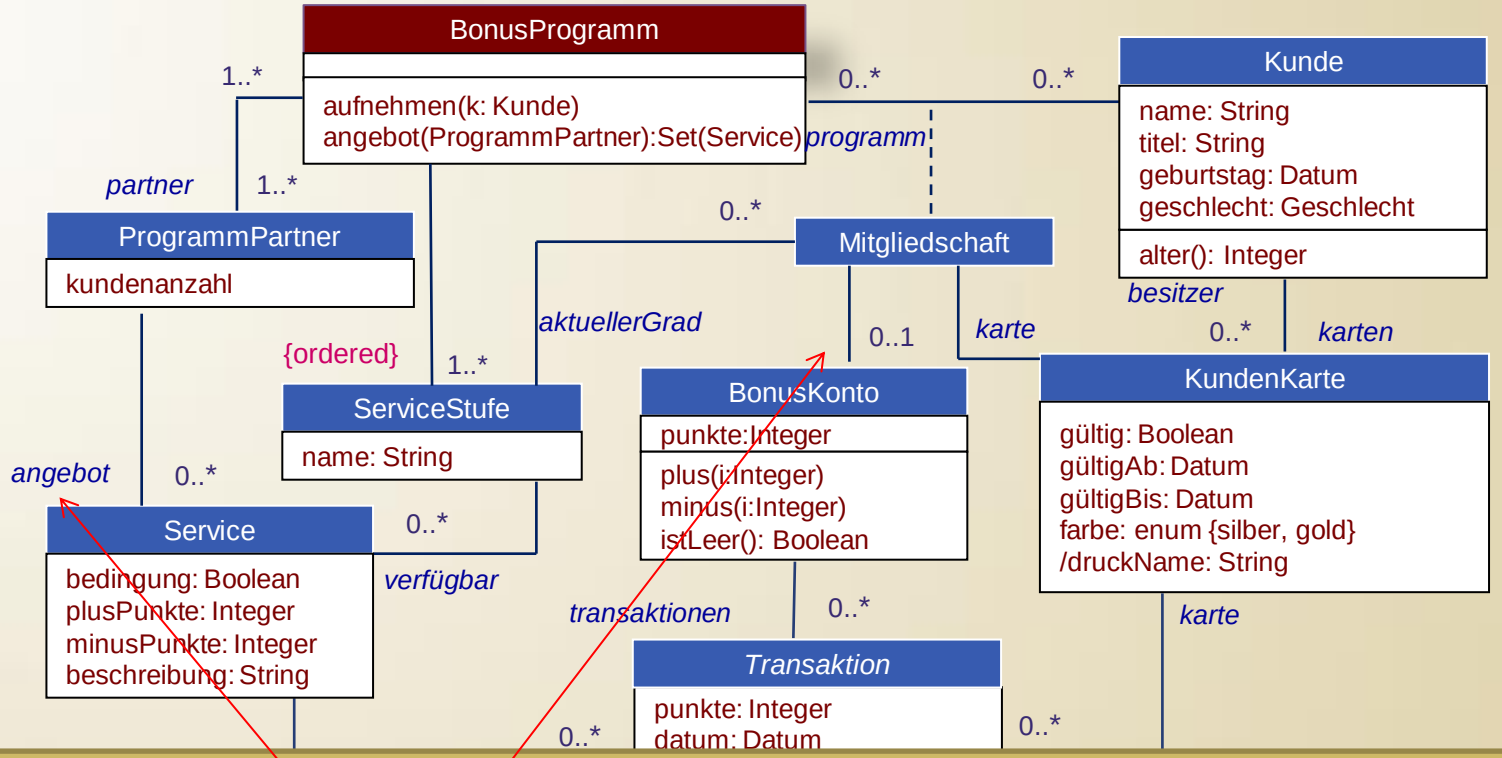


context Kunde **inv**:

programm->size() = karten->select (gültig = true)->size()

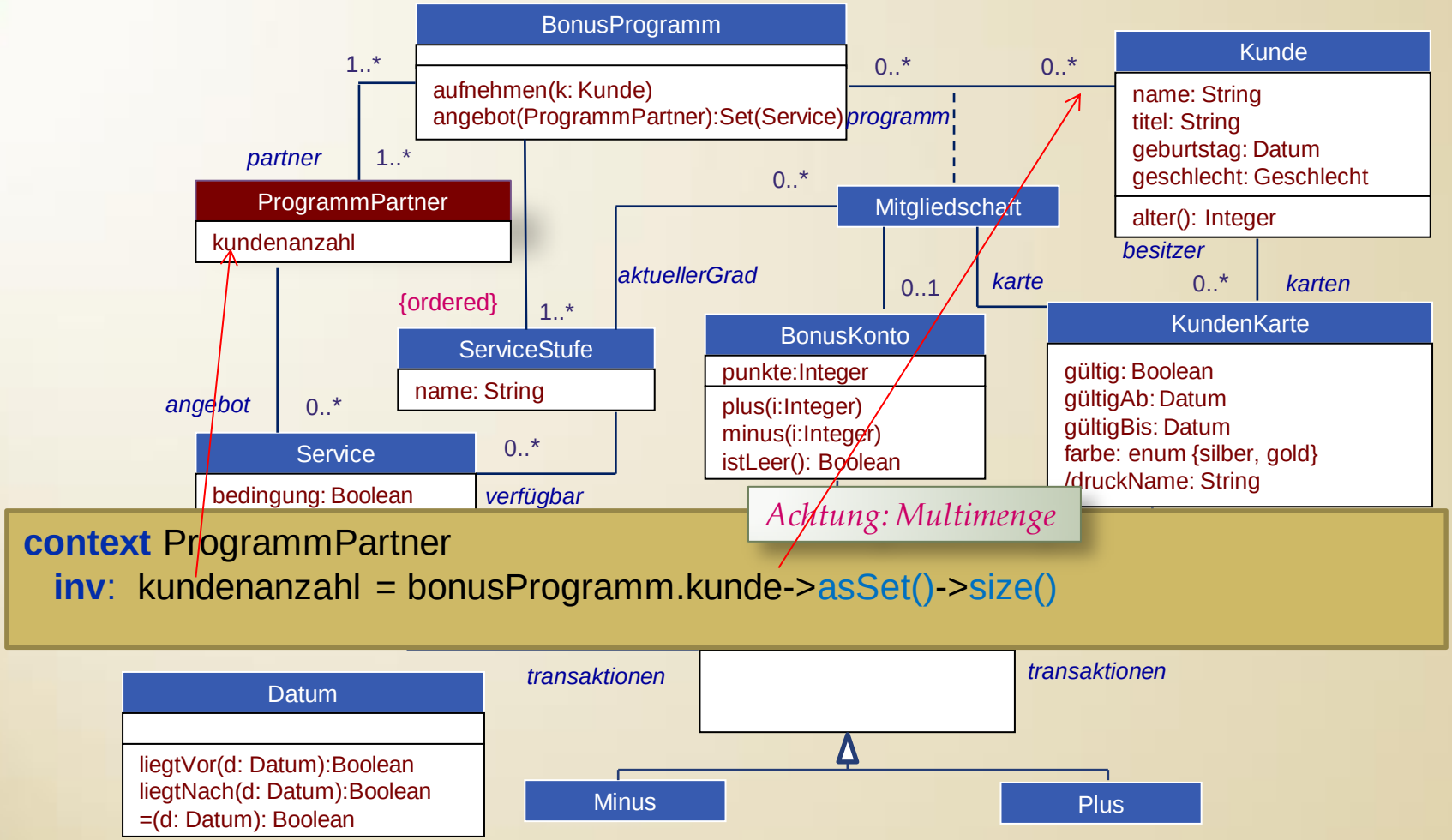
(Anzahl gültiger Karten eines beliebigen Kunden (kann Null sein) ist gleich der Anzahl von Programmen, an denen dieser Kunde teilnimmt*)*

Invarianten-Beispiel (3)



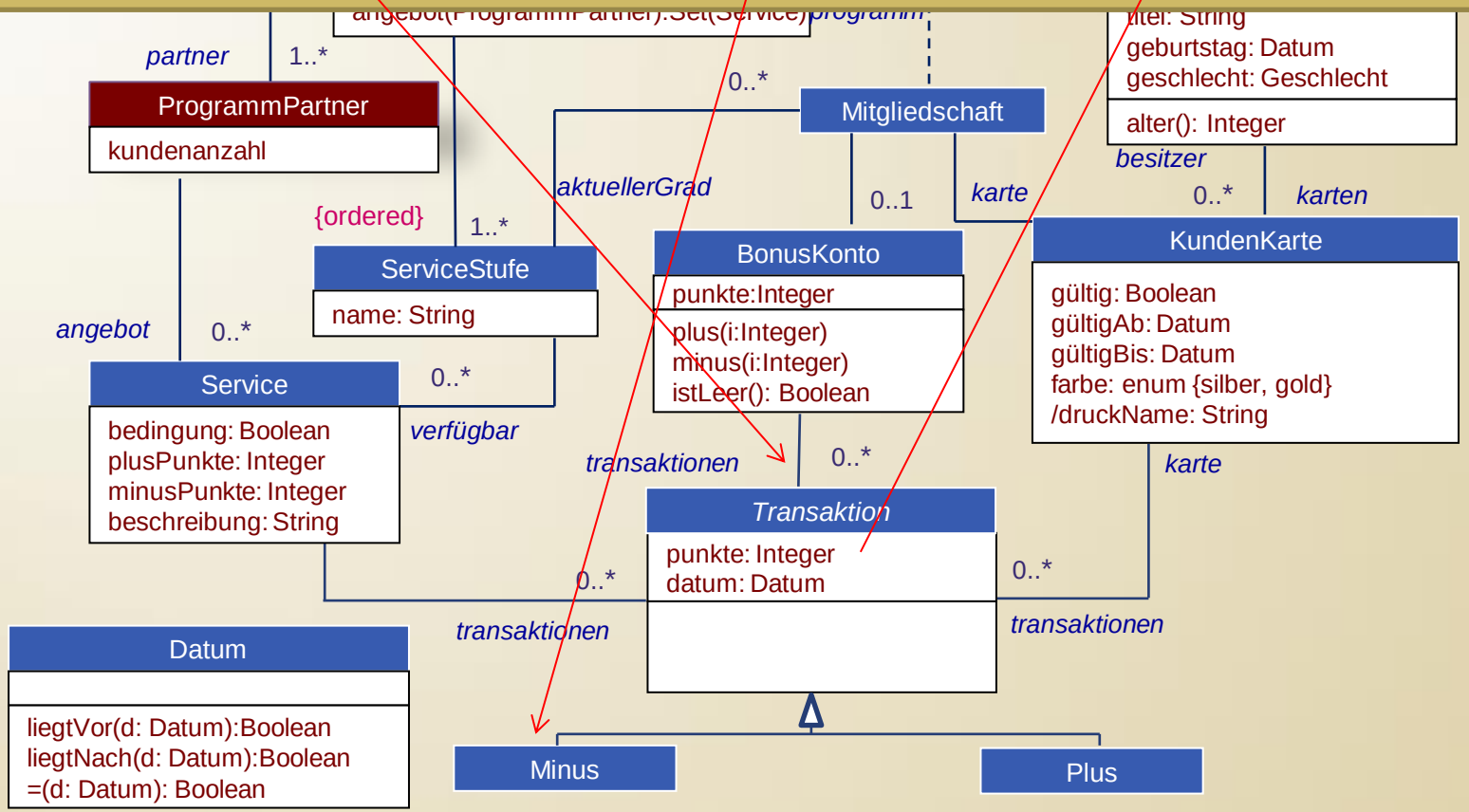

```
context BonusProgramm inv:
        partner-> angebot-> forAll ( plusPunkte=0 and minusPunkte=0 )
      implies
        mitgliedschaft.bonusKonto->isEmpty()
```

Invarianten-Beispiel (4)



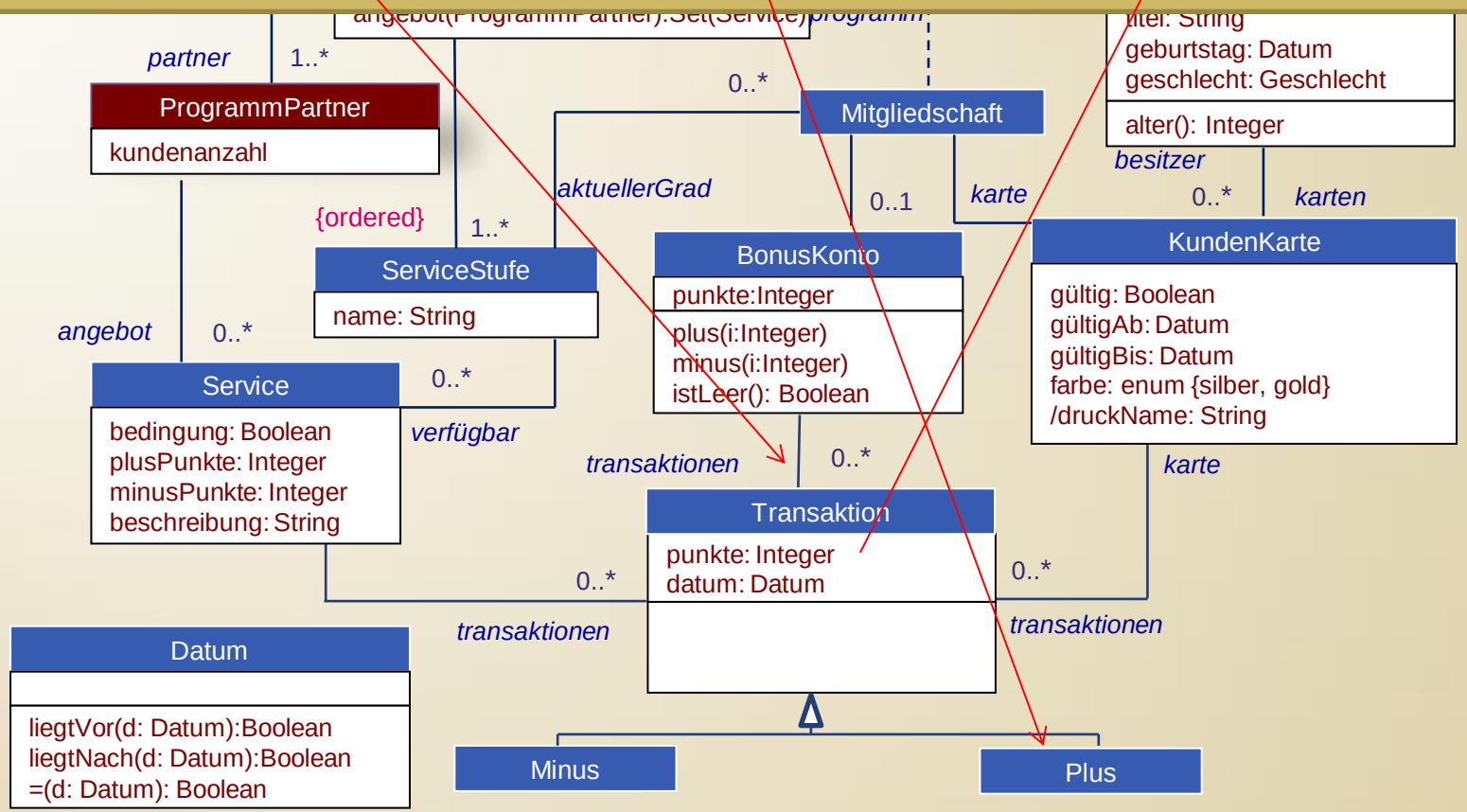
Invarianten-Beispiel (5)

```
context ProgrammPartner inv:
   self.angebot->transaktionen->select (oclIsTypeOf(Minus))->collect(punkte)->sum()
 <=
   self.angebot->transaktionen->select (oclIsTypeOf(Plus))->collect(punkte)->sum()
```

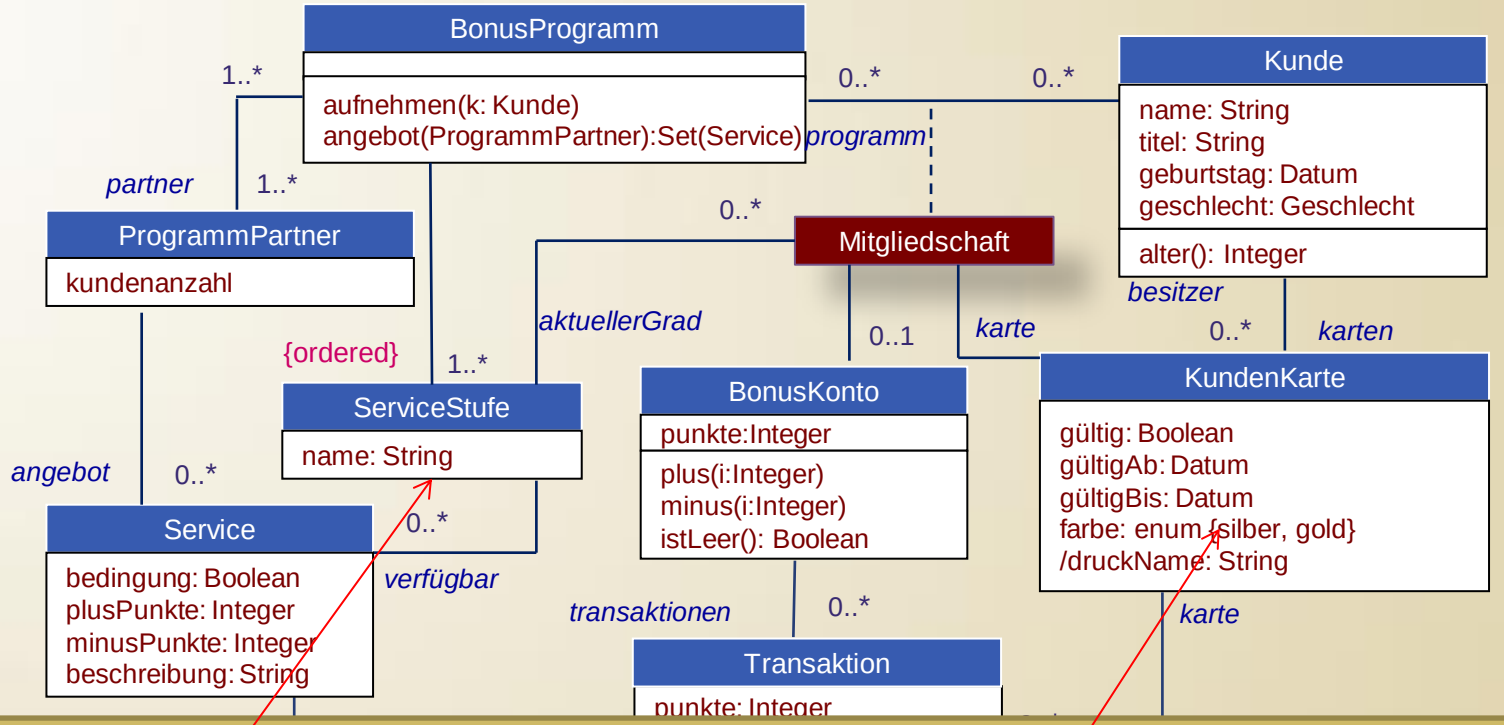


Invarianten-Beispiel (5)

```
context ProgrammPartner inv:
   self.angebot->transaktionen->select (oclIsTypeOf(Minus))->collect(punkte)->sum()
 <=
   self.angebot->transaktionen->select (oclIsTypeOf(Plus))->collect(punkte)->sum()
```



Invarianten-Beispiel (6)



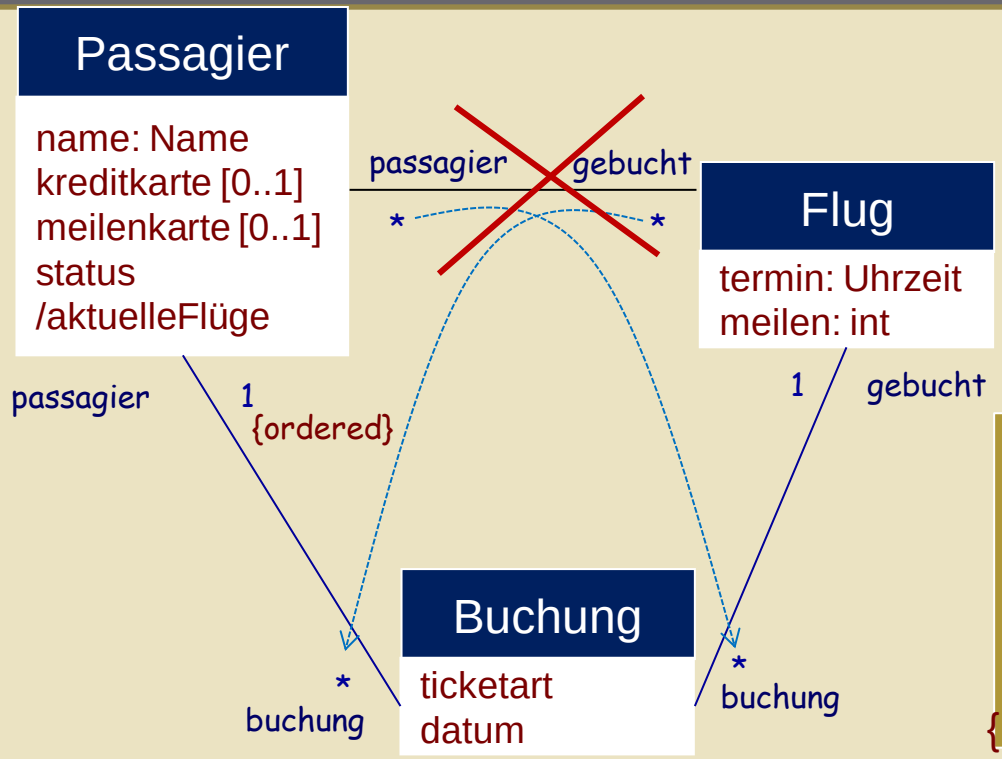
context Mitgliedschaft **inv**:

aktuellerGrad.name= ‘silber‘ **implies** karte.farbe = #silber

and

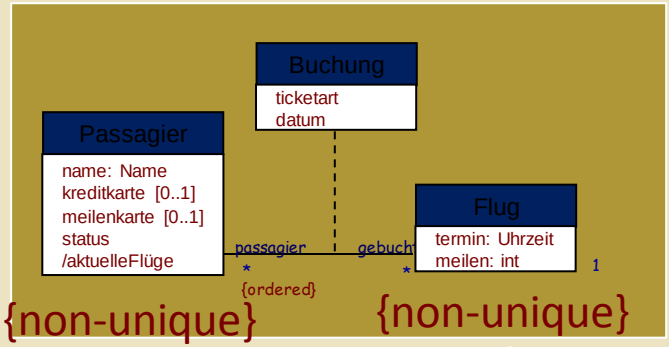
aktuellerGrad.name= ‘gold‘ **implies** karte.farbe = #gold

Ersetzung: Assoziationsklasse → Normale Klasse



Frage: semantsich äquivalent?

jetzt: semantsich äquivalent,
aber die Semantik wollte man nicht

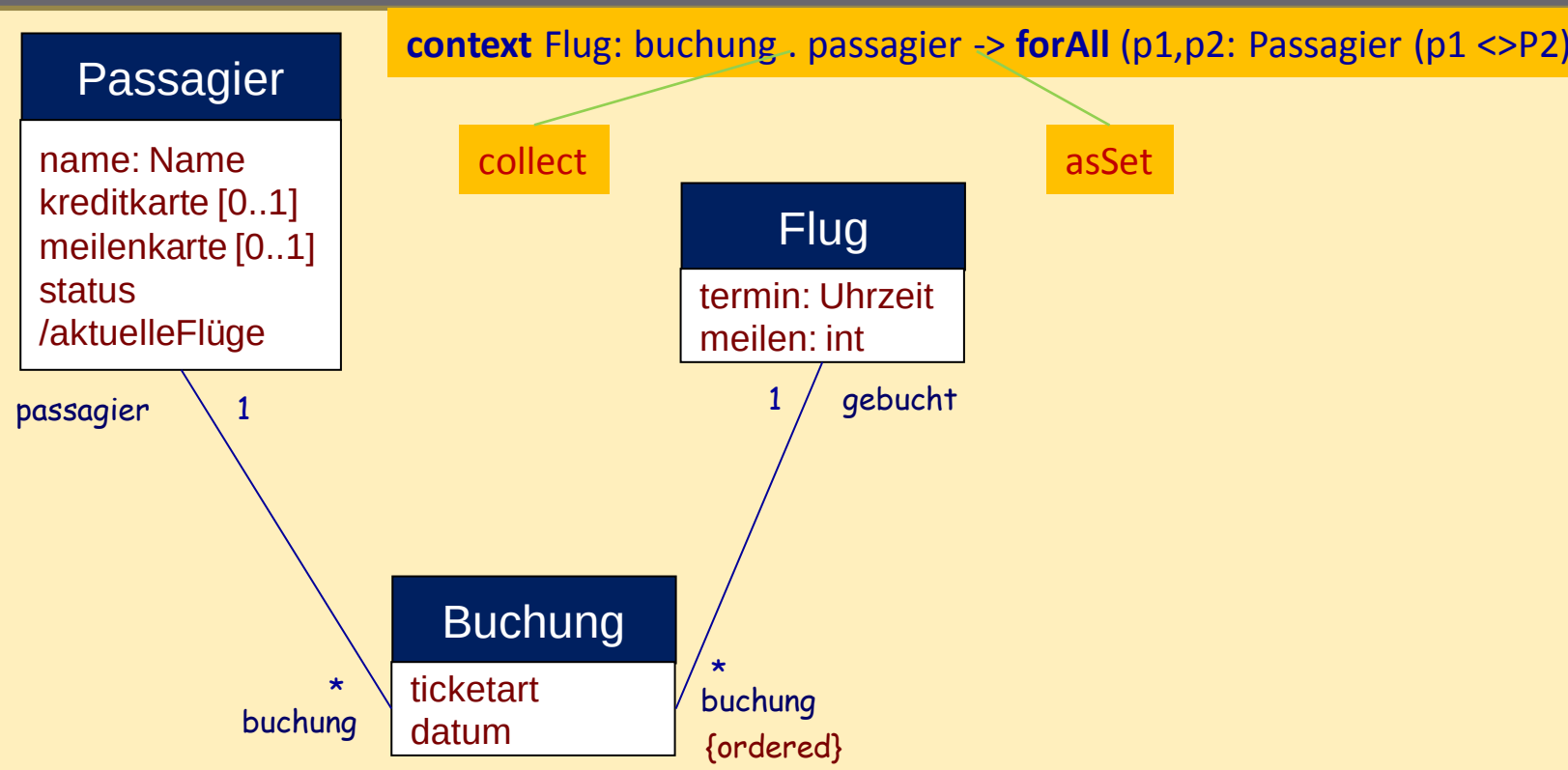


- ein Flug kann beliebig oft gebucht werden
- jede Buchung gehört zu genau einem Passagier
- damit kann für einen Flug mehrere Buchungen desselben Passagiers ausgelöst werden

➔ Die Assoziationsklasse verhinderte das

Man könnte die attributierte Assoziation aber anpassen

Ersetzung: Assoziationsklasse → Normale Klasse



- ein Flug kann von einem Passagier nicht mehr als einmal gebucht werden