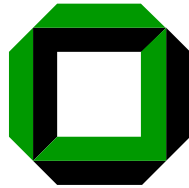


# Formale Systeme

Prof. Dr. Bernhard Beckert

Fakultät für Informatik  
Universität Karlsruhe (TH)



Winter 2008/2009



# UML

## Kurze Wiederholung

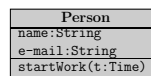


## Syntax von UML-Klassendiagrammen

Grundkonzepte:

### Klasse

Sammlung ähnlicher Objekte in einem System

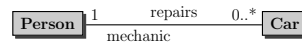


### Attribut

### Operation/Methode

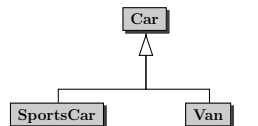
### Assoziation

Relation zwischen Klassen (setzt Paare von Instanzen von Klassen (Objekte) in Beziehung)

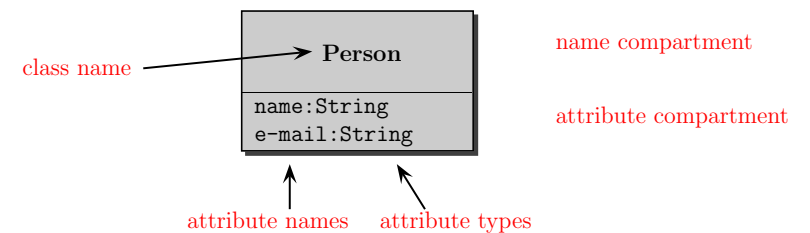


### Generalisierung

Spezialisierungs-/Generalisierungsbeziehung zwischen Klassen



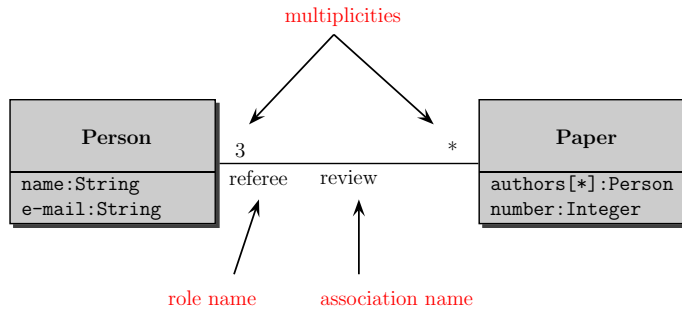
## Semantik von Klassen



$\mathcal{I}(\text{Person})$  ist eine (evtl. leere) Menge von **Objekten**  
 $\mathcal{I}(\text{name})$  ist eine partielle Funktion von  $\mathcal{I}(\text{Person})$  nach  $\mathcal{I}(\text{String})$   
 $\mathcal{I}(\text{name})(aPerson)$  gibt einen String oder ist undefiniert



## Semantik von Assoziationen



$\mathcal{I}(\text{review})$  ist eine Relation zwischen  $\mathcal{I}(\text{Person})$  und  $\mathcal{I}(\text{Paper})$

Multiplizität 3 verlangt:

für alle  $\text{pap} \in \mathcal{I}(\text{Paper})$ :

$\text{card}(\{\text{pers} \in \mathcal{I}(\text{Person}) \mid \mathcal{I}(\text{review})(\text{pers}, \text{pap})\}) \in \mathcal{I}(3) = \{3\}$

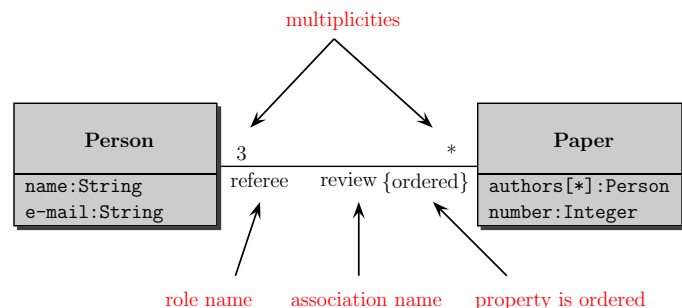


## Semantik von Multiplizitäten

$M$	$\mathcal{I}(M)$
0..1	$\{0, 1\}$
0..*	$\mathbb{N}$
*	$\mathbb{N}$
1..3	$\{1, 2, 3\}$
3	$\{3\}$



## Semantik von Rollennamen



$\mathcal{I}(\text{referee}) : \mathcal{I}(\text{Paper}) \rightarrow \text{Set}(\mathcal{I}(\text{Person}))$

$\mathcal{I}(\text{paper}) : \mathcal{I}(\text{Person}) \rightarrow \text{Sequence}(\mathcal{I}(\text{Paper}))$

(Standard-Rollenname: Klassenname mit **kleinem** Anfangsbuchstaben)



## Snapshots

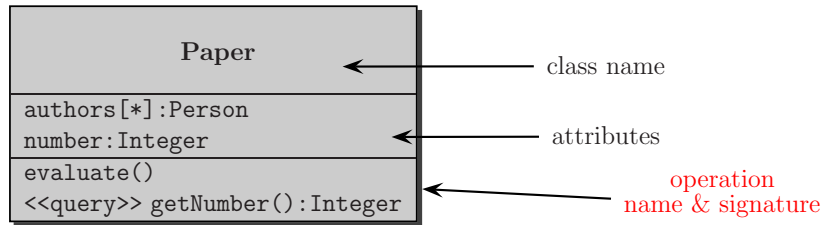
Ein **Snapshot** eines Klassendiagramms  $\mathcal{C}$  ist eine bestimmte Interpretationsfunktion  $\mathcal{I}$  von  $\mathcal{C}$

- UML-Objektdiagramm (für  $\mathcal{C}$ ) bestehend aus
  - der Menge der momentan existierenden Instanzen  $\mathcal{I}(C)$  für jede Klasse  $C$
  - Abbildungen  $\mathcal{I}(a) : C \rightarrow C'$  für alle Attribute  $a$  des Typs  $C'$  in der Klasse  $C$
  - Instanzen von Assoziationen ("links"): Elemente aus  $\mathcal{I}(C) \times \mathcal{I}(C')$
- einer Interpretation für Operationen
- (Standard-)Interpretation von vordefinierten primitiven Datentypen und ihren Operationen (Integer, String, ...)

Multiplizitäten und Constraints beschränken die Menge der zulässigen Snapshots



## Semantik von Operationen / Queries



**Operation:** Transition von Snapshot zu Snapshot

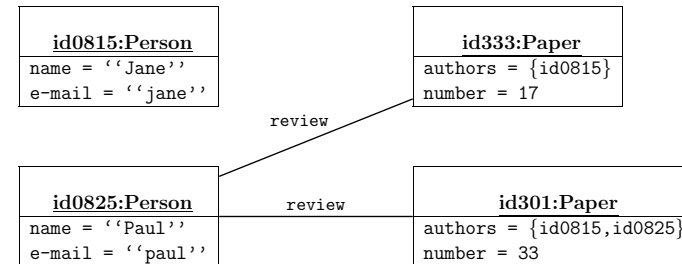
**Query:** Partielle Funktion von definierender Klasse und Argumentklassen nach der Ergebnisklasse



## Objektdiagramm



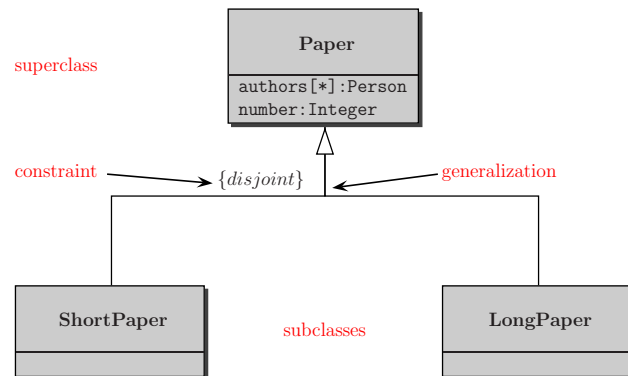
UML-Objektdiagramme repräsentieren Snapshots:



Illegale (und nicht gewollte) Instanz — Warum?



## Semantik von Unterklassen



Unterklassenrelation:  $\mathcal{I}(\text{ShortPaper}) \subseteq \mathcal{I}(\text{Paper})$

hier zusätzlich Forderung nach Disjunktheit:

$\mathcal{I}(\text{ShortPaper}) \cap \mathcal{I}(\text{LongPaper}) = \emptyset$



## OCL

Object Constraint Language



- 1996: Object Management Group (OMG) task force für objekt-orientierte Analyse und Entwurf.
- Jos Warmer entwickelt einen gemeinsamen Vorschlag von IBM und ObjecTime Limited
- Vorgänger: Syntropy Methode von Steve Cook und John Daniels
- OCL wird Bestandteil von UML 1.1.
- 2003: OCL 2.0 als OMG Standard abgesegnet.



aus dem Buch von Warmer und Kleppe:

*The second option for a constraint notation is some kind of mathematical notation. All experience with formal or mathematical notations leads to the same conclusions: the people who can use the notation can express things precisely and unambiguously, but very few people can really use such a notation...*

*The aim of a standard is that it be widely used and not that it be exact but rarely used.*



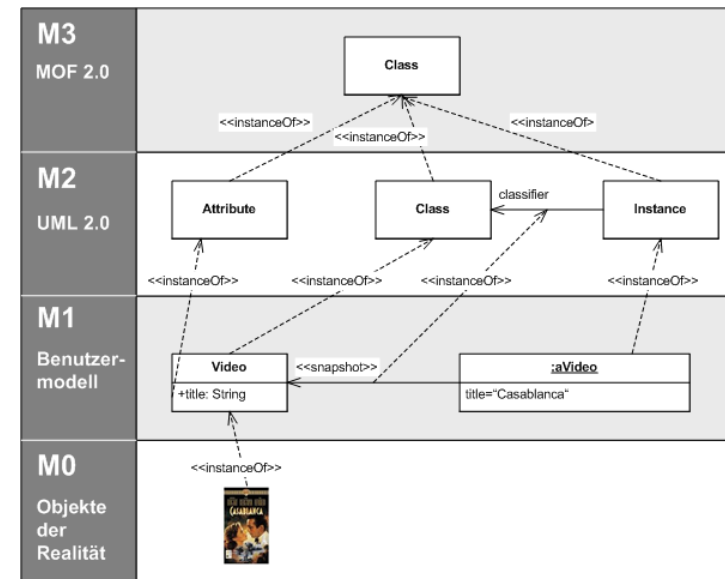
Metadatenarchitektur

- Definiert von der OMG
- Wichtig für das Verständnis der UML/OCL
- Schichtenmodell
- 4 Schichten: M0, M1, M2, M3

Schicht	Beispiel
M3 Meta-Meta-Modell	Meta Object Faciliy (MOF)
M2 Meta-Modell	UML-Meta-Modell
M1 Modell	UML-Klassendiagramm
M0 Objekte der Realität	Auto



4-Schichtenmodell

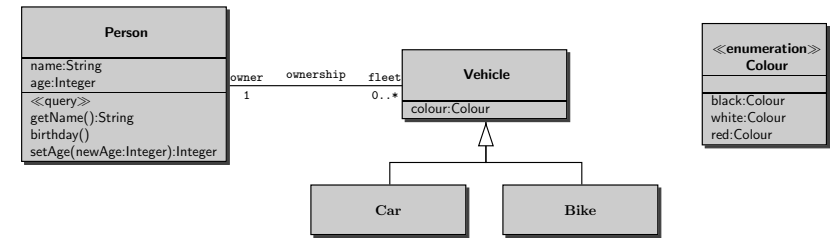


## Wo OCL?

- Im UML-Standard: Einschränkung der gültigen M2-Instanzen (= M1-Modellen)
- Im Benutzermodell: Einschränkung der gültigen Snapshots
- Trend heute: OCL für MDA (Model Driven Architecture)



## UML alleine reicht nicht ...

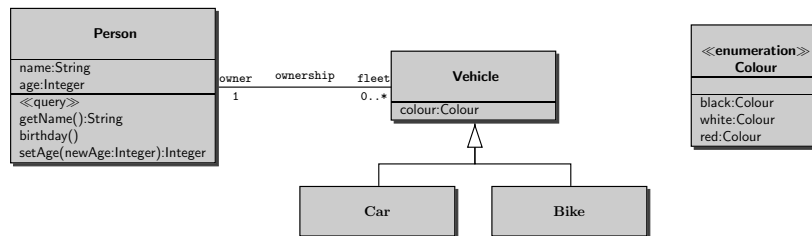


- Wieviele Personen können ein Auto besitzen?
- Wie alt muß der Besitzer eines Autos sein?
- Wie kann man ausdrücken, daß eine Person höchstens ein schwarzes Auto besitzen darf?
- Wie kann man ausdrücken, daß age nach Aufruf von setAge(i) den Wert i haben soll?

UML-Klassendiagramme können gewisse **semantische Details** eines Designs nicht ausdrücken



## OCL-Beispiele

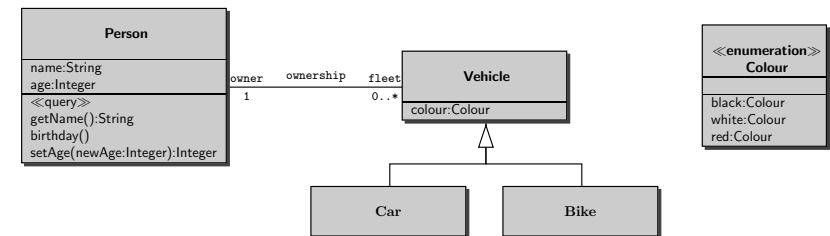


“Keine Person besitzt mehr als 3 Fahrzeuge.”

**context** p:Person oder Änderung der Multiplizität  
**inv:** p.fleet->size() <= 3



## OCL-Beispiele



“Keine Person besitzt mehr als 3 **schwarze** Fahrzeuge.”

**context** p:Person  
**inv:** p.fleet->select(v | v.colour = Colour.black)->size() <= 3



Es gibt **zwei** Arten von OCL-Constraints:

- Invarianten für eine Klasse  $C$
- Paare von Vor- und Nachbedingungen ( $pre, post$ ) für eine Operation  $p$



## Semantik von Invarianten

Object Constraint Language, OMG Available Specification, V2.0, May 2006

Section 12.6 (p. 176)

*An invariant constraint is a constraint that is linked to a Classifier. The purpose of an invariant constraint is to specify invariants for the Classifier. An invariant constraint consists of an OCL expression of type Boolean. The expression must be true for each instance of the classifier at any moment in time. Only when an instance is executing an operation, this does not need to evaluate to true.*



Der **Context** legt fest, auf welches Element sich ein Constraint bezieht.

```

context  c: typeName
inv:    OclExpression1
          ⋮
inv:    OclExpressionn
    
```

## Beispiel

```

context  p:Person
inv:    p.fleet->select(v | v.colour = Colour.black)->size() <= 3
    
```

```

context  c:typeName::opName(p1:type1,... ,pk:typek):rtype
pre:    OclExpression1pre
post:   OclExpression1post
          ⋮
pre:    OclExpressionnpre
post:   OclExpressionnpost
    
```



## Beispiel

## Semantik von Vor- und Nachbedingungen

Object Constraint Language, OMG Available Specification, V2.0, May 2006

Section 12.7. (p.162)

*The purpose of a precondition is to specify the conditions that must hold before the operation executes. A precondition consists of an OCL expression of type Boolean. The expression must evaluate to true whenever the operation starts executing, but only for the instance that will execute the operation.*

Section A.3.2 (p.222)

*If the precondition holds, the contract of the operation guarantees that the postcondition is satisfied after completion of op.*



## Noch ein Zitat

aus: *The Unified Modeling Language Reference Manual*,  
J.Rumbaugh, I. Jacobson, G. Brooch, p. 392

A precondition is a constraint that must be true when an operation is invoked.

It is the responsibility of the caller to satisfy the condition. It is not a condition that the receiver should have to check.

A precondition is not a guard condition; it is a condition that must be true, not a way to optionally execute an operation.

It can be useful to test the precondition at the beginning of the operation for reliability, but this is in the nature of debugging a program.

The condition is supposed to be true, and anything else is a programming error. If the condition is not satisfied, no statement can be made about the integrity of the operation or the system. It is liable to utter failure.



## Design by Contract

Vor- und Nachbedingung für eine Operation werden angesehen als ein Vertrag zwischen dem **Anbieter** (receiver, supplier) und dem **Benutzer** (caller, user) einer Operation.

- der Benutzer ist verantwortlich dafür, daß die Vorbedingung beim Aufruf der Operation gilt.
- der Anbieter garantiert die Nachbedingung unter der Voraussetzung, daß die Vorbedingung erfüllt ist.



## Gesamtheiten

In der Objektbeschreibungssprache OCL benutzen wir den Begriff

### Gesamtheit (collection)

als gemeinsamen Oberbegriff für

- Menge (set)  
Jedes Element kommt nur einmal vor.
- Multimenge (bag)  
Elemente können mehrfach auftreten.
- Folge (sequence)  
Elemente können mehrfach auftreten und sind in einer Reihenfolge angeordnet.



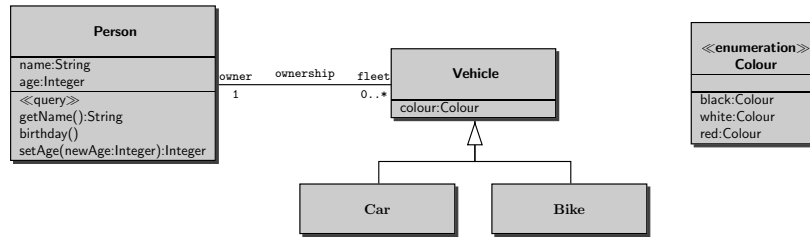
## Operationen, die für alle Gesamtheiten anwendbar sind

Sei  $M$  ein Objekt vom Typ *Gesamtheit*.

Operation	Erklärung
size	Anzahl der Elemente von $M$
count( $o$ )	Anzahl der Vorkommen von $o$ in $M$
includes( $o$ )	Wahr, wenn $o$ ein Element von $M$ ist
includesAll( $c$ )	Wahr, wenn $c$ eine Teilmenge von $M$ ist
isEmpty	Wahr, wenn $M$ kein Element enthält
notEmpty	Wahr, wenn $M$ ein Element enthält
iterate( $exp$ )	$exp$ wird für jedes Element aus $M$ ausgewertet. Der Ergebnistyp hängt von $exp$ ab
exists( $exp$ )	Wahr, wenn $exp$ auf mindestens ein Element aus $M$ zutrifft
forAll( $exp$ )	Wahr, wenn $exp$ auf alle Elemente von $M$ zutrifft
select( $e \mid exp$ )	$\{m \in M \mid exp(m/e) \text{ ist wahr} \}$
collect( $e \mid exp$ )	$\{exp(m/e) \mid m \in M\}$



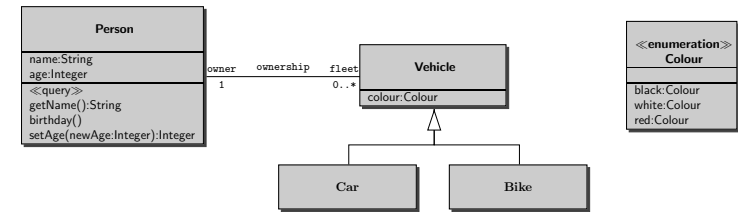
## OCL-Ausdrücke: Navigation



OCL-Ausdrücke:	Typen:
p	(vordefinierte) Variablen Person
p.age	Attribute Integer
p.getName()	Queries, <b>nicht alle Operationen</b> String
p.fleet	Rollen/Assoz. Set(Vehicle)
p.fleet->size()	vordefinierte Operationen Integer

OCL ist eine getypte Sprache.

## OCL-Ausdrücke: Vordefinierte Variablen



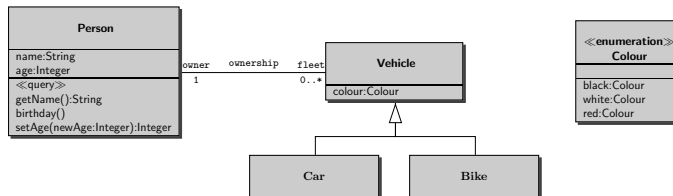
Context	Vordefinierte Variablen
<b>context</b> p:Person	p
<b>context</b> Person	self
<b>context</b> Person::getName():String	self, result
<b>context</b> Person::setAge(newAge: Integer)	self, newAge

### Beispiel

<b>context</b> p:Person	<b>context</b> Person
<b>inv:</b> p.fleet->size() <= 3	<b>inv:</b> self.fleet->size() <= 3
<b>context</b> Person::getName():String	

<b>context</b> Person::setAge(newAge: Integer)
<b>post:</b> self.age=newAge

## OCL-Ausdrücke: Vordefinierte Operatoren auf collections



Operator	Ergebnistyp	Bedeutung
size	Integer	liefert die Größe einer collection
select	Collection	filtert die Elemente nach einer Bedingung
collect	Collection	Wendet eine Funktion auf alle Elemente an
forAll/exist	Boolean	Erfüllt jedes/ein Element die Bedingung?
includes	Boolean	Ist das gesuchte Element in der collection?

### Beispiel

<b>context</b> Person
<b>inv:</b> self.fleet->size() <= 3
<b>context</b> Person

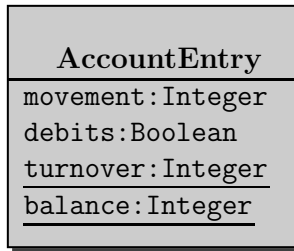
## Vereinfachende Schreibweisen

<b>context</b> Person	<b>context</b> c:Person	<b>context</b> Person
<b>inv:</b> self.age >= 18	<b>inv:</b> c.age >= 18	<b>inv:</b> age >= 18

<b>context</b> Person	- - alle constraints sind äquivalent
<b>inv:</b> fleet ->collect(v:Vehicle   v.colour) ->asSet() ->size() = 1	
<b>inv:</b> fleet ->collect(v   v.colour) ->asSet() ->size() = 1	
<b>inv:</b> fleet ->collect(colour) ->asSet() ->size() = 1	
<b>inv:</b> fleet.colour ->asSet() ->size() = 1	

Die letzte Abkürzung ist nur erlaubt für 'collect'





**context** AccountEntry  
**inv:** AccountEntry.turnover=  
 AccountEntry.allInstances ->  
 iterate(a:AccountEntry ; m:Integer=0 | m+a.movement)



Falls C ein Typ-Symbol ist, dann ist C.allInstances ein gültiger OCL-Ausdruck

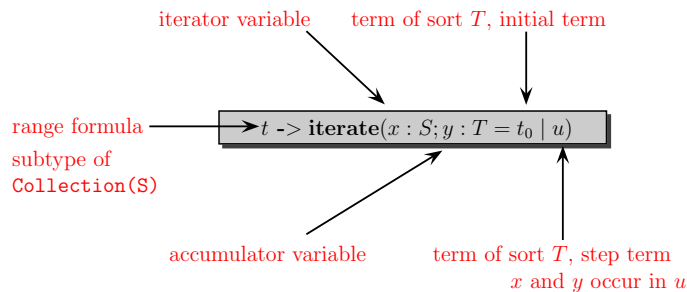
$\mathcal{I}(C.allInstances)$  ist die Menge aller Instanzen von C in dem betrachteten Snapshot.

In OCL wird angenommen, daß alle Menge endlich sind (weils sich dann Werkzeuge leichter implementieren lassen).  
 Was ist mit Integer.allInstances?

$\mathcal{I}(Integer.allInstances)$  ist undefiniert!



iterate Syntax



iterate Semantics

$t \rightarrow \text{iterate}(x:S; y:T=t_0 | u)$

Java-Pseudocode:

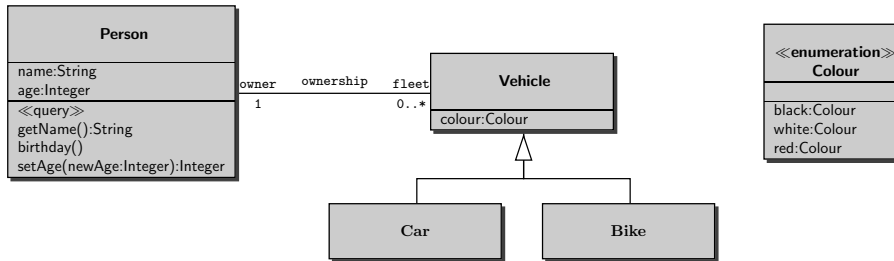
```

S x;
T y = t0;
for (Iterator it = t.iterator(); it.hasNext();) {
    x = it.next();
    y = u(x,y);
}
    
```

Typ von `x` und `y` kann von `t` und `u` abgeleitet werden  
 iterate kann alle anderen Operationen auf collections simulieren



## Bezug auf frühere Werte



**context** Person::birthday()  
**pre:** age  $\geq$  0  
**post:** age = age@pre + 1



## Junktoren

Selbstverständlich gibt es logische Junktoren in OCL. Die folgenden Ausdrücke sind Boolesche OCL-Ausdrücke, falls  $e_1$  und  $e_2$  Boolesche OCL-Ausdrücke sind:

- not  $e_1$
- $e_1$  and  $e_2$
- $e_1$  or  $e_2$
- $e_1$  implies  $e_2$
- $e_1 = e_2$
- if  $e_1$  then  $e_2$  else  $e_3$  endif  
 äquivalent zu  
 ( $e_1$  implies  $e_2$ ) and (not  $e_1$  implies  $e_3$ )

Klammern dürfen wie üblich gesetzt werden



## Snapshots und OCL-Constraints

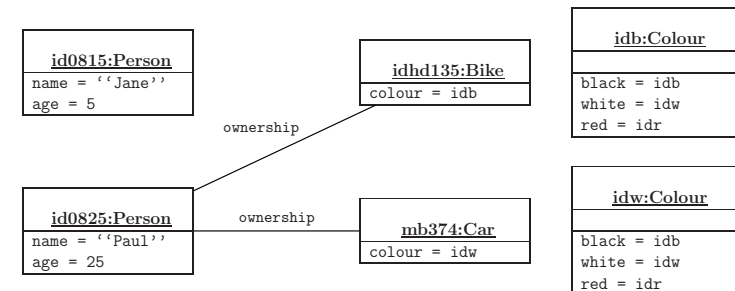
- OCL constraints werden relativ zu einem Snapshot  $\mathcal{I}$  ausgewertet
- OCL constraints sind vom Typ Boolean  $\Rightarrow$  sie sind wahr oder falsch bezüglich  $\mathcal{I}$
- OCL constraints beschränken die erlaubten Snapshots eines UML-Klassendiagramms

Erlauben die beabsichtigte Semantik eines UML-Diagramms auszudrücken

- Die Semantik von OCL kann basierend auf Snapshots angegeben werden  
 Vorlesung: Definition der Semantik durch übersetzung in PL1



## Snapshots und OCL-Constraints



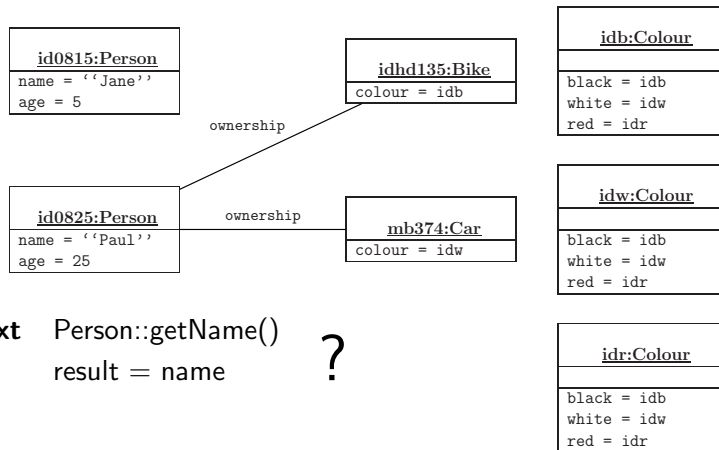
**context** Vehicle  
**inv:** self.owner.age  $\geq$  18 ✓

**context** Person  
**inv:** fleet  $\rightarrow$  forAll(colour = Colour.black) ✗

**context** Person  
**inv:** fleet  $\rightarrow$  select(colour = Colour.black)  $\rightarrow$  size()  $\leq$  3 ✓

**inv:** Car.allInstances  $\rightarrow$  exists(colour = Colour.red) ✗





OCL	PK1
semantische Unklarheiten	klare Semantik
Notation muß gelernt werden (leichter?!)	Notation muß gelernt werden
Ausdrücke eher länger	Ausdrücke eher kürzer
enthält Mengen und natürliche Zahlen	Mengen und natürliche Zahlen nicht direkt enthalten
keine eigene Beweistheorie	etablierte Beweistheorie und Beweiser
Festlegung der Signatur durch UML-Diagramm	keine Angaben über Herkunft der Signatur



### Zusammenfassung

- OCL-Constraints werden immer relativ zu einem Snapshot eines UML-(Klassen-)Diagramms ausgewertet
- Zwei Arten von OCL-Constraints: Invarianten und Vor-/Nachbedingungen
- Context legt fest, worauf sich Constraints beziehen
- OCL-Ausdrücke können Aussagen über collections machen. Die wichtigsten Operationen auf collections sind: size, select, collect, forAll, exist, includes, iterate
- In Nachbedingungen kann man mithilfe von @pre auf Werte (von Attributen) vor Ausführung der Methode zugreifen

