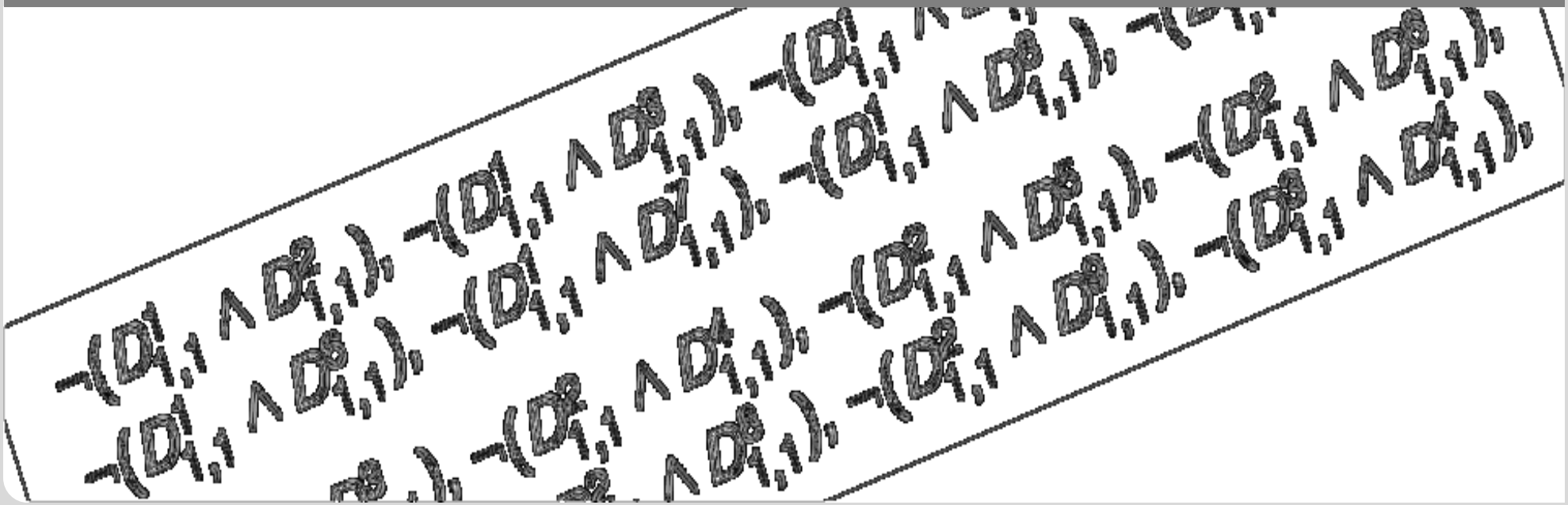
Karlsruhe Institute of Technology

Formale Systeme

Object Contraint Language

Prof. Dr. Bernhard Beckert | WS 2010/2011

KIT – Institut für Theoretische Informatik



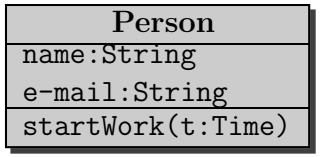
UML

Kurze Wiederholung

Syntax von UML-Klassendiagrammen

Grundkonzepte:

Klasse

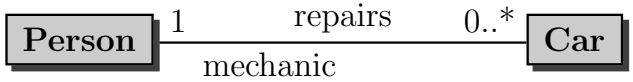


Sammlung ähnlicher Objekte in einem System

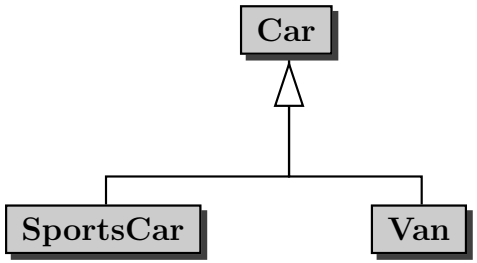
Attribut

Operation/Methode

Assoziation



Relation zwischen Klassen



Generalisierung

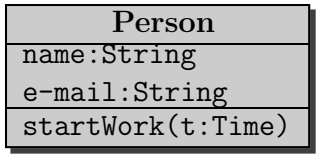
Spezialisierungs-/Generalisierungsbeziehung zwischen Klassen

Syntax von UML-Klassendiagrammen

Grundkonzepte:

Klasse

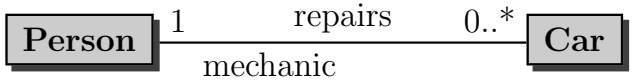
Sammlung ähnlicher Objekte in einem System



Attribut

Operation/Methode

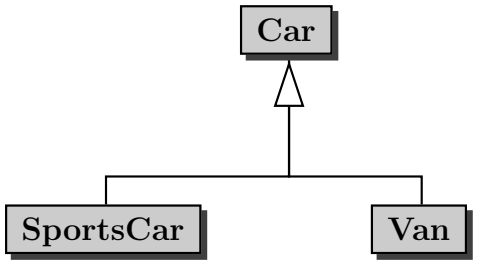
Assoziation



Relation zwischen Klassen

Generalisierung

Spezialisierungs-/Generalisierungsbeziehung zwischen Klassen

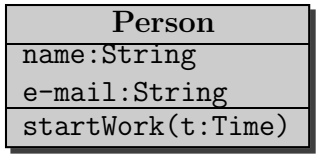


Syntax von UML-Klassendiagrammen

Grundkonzepte:

Klasse

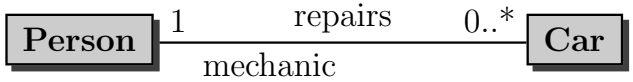
Sammlung ähnlicher Objekte in einem System



Attribut

Operation/Methode

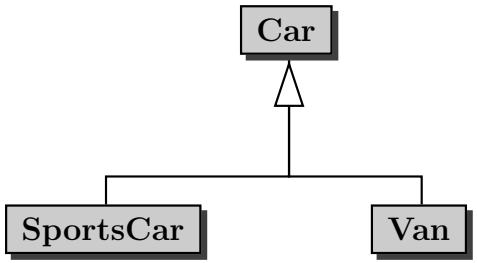
Assoziation



Relation zwischen Klassen

Generalisierung

Spezialisierungs-/Generalisierungsbeziehung zwischen Klassen

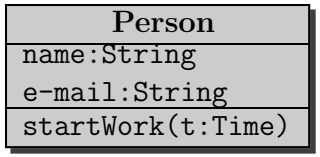


Syntax von UML-Klassendiagrammen

Grundkonzepte:

Klasse

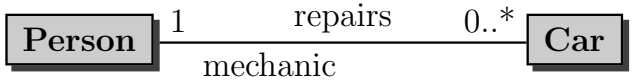
Sammlung ähnlicher Objekte in einem System



Attribut

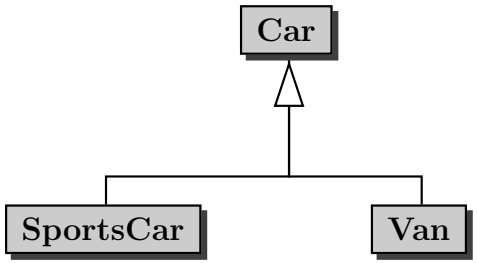
Operation/Methode

Assoziation



Relation zwischen Klassen

Generalisierung



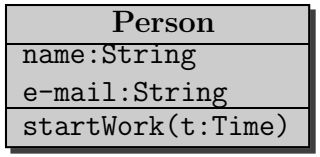
Spezialisierungs-/Generalisierungsbeziehung zwischen Klassen

Syntax von UML-Klassendiagrammen

Grundkonzepte:

Klasse

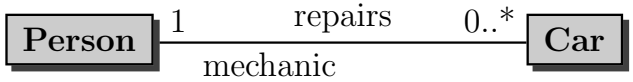
Sammlung ähnlicher Objekte in einem System



Attribut

Operation/Methode

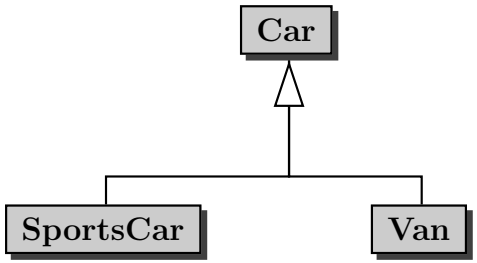
Assoziation



Relation zwischen Klassen

Generalisierung

Spezialisierungs-/Generalisierungsbeziehung zwischen Klassen

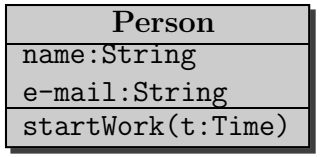


Syntax von UML-Klassendiagrammen

Grundkonzepte:

Klasse

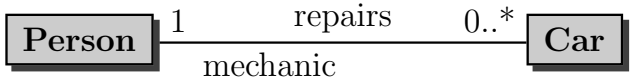
Sammlung ähnlicher Objekte in einem System



Attribut

Operation/Methode

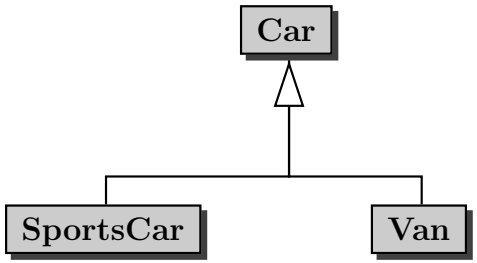
Assoziation



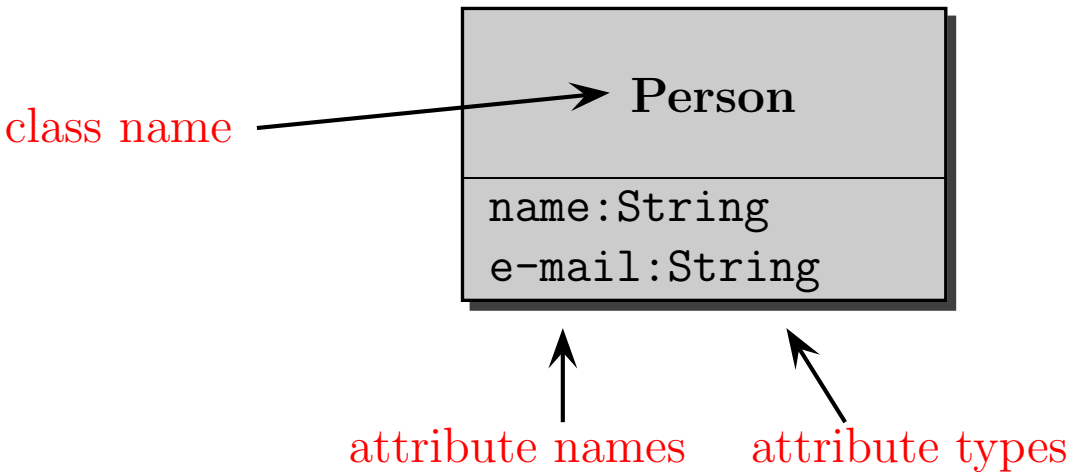
Relation zwischen Klassen

Generalisierung

Spezialisierungs-/Generalisierungsbeziehung zwischen Klassen

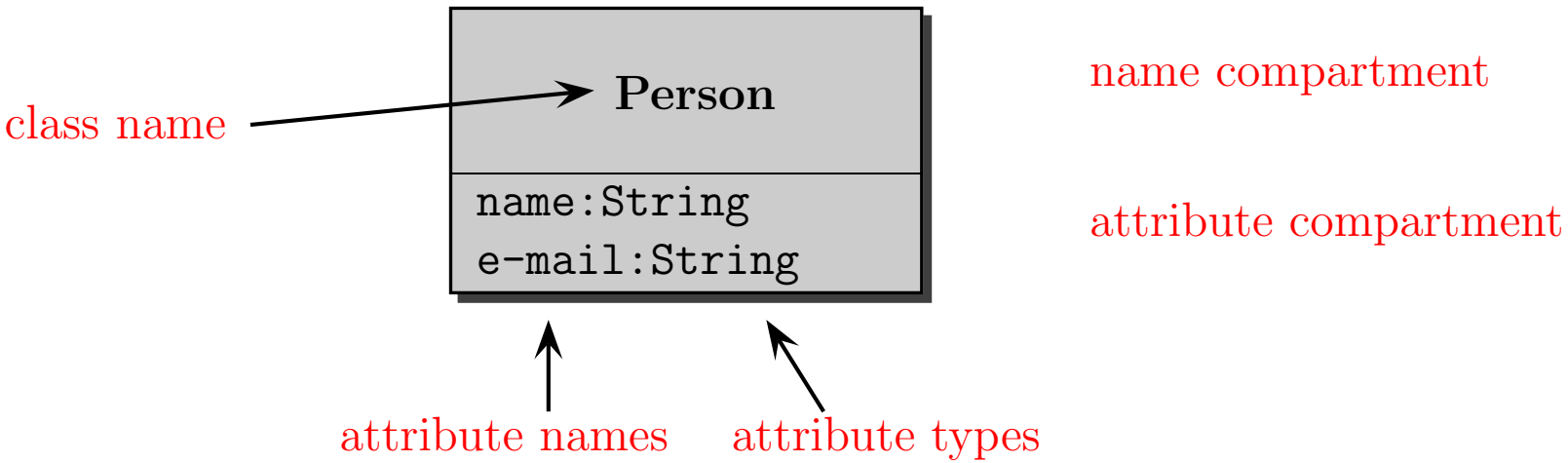


Semantik von Klassen



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

Semantik von Klassen

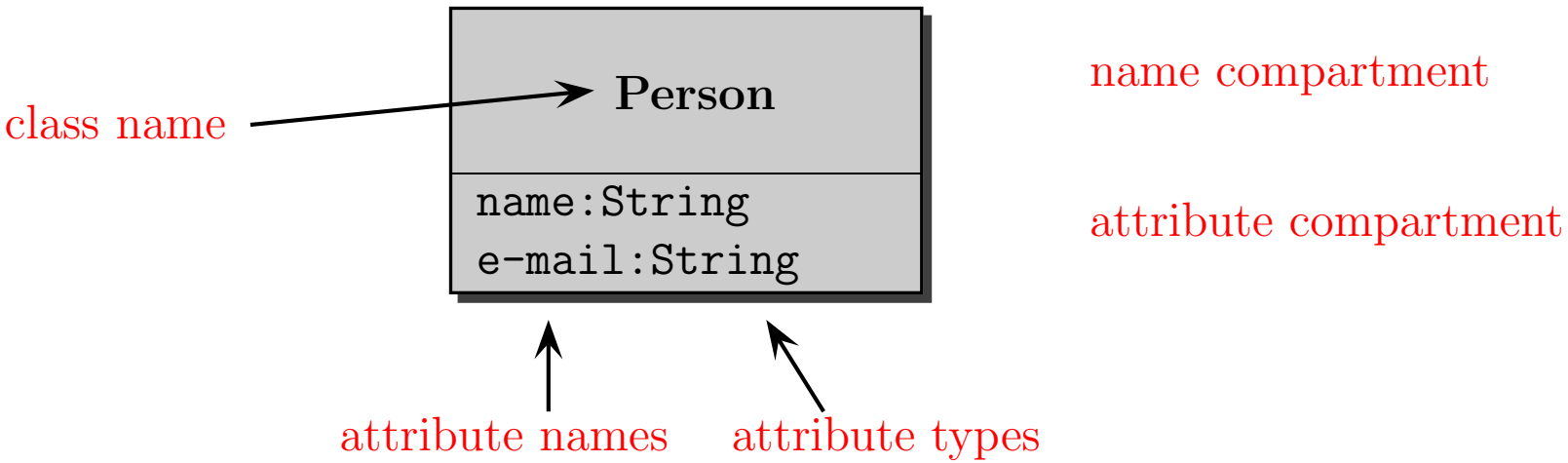


$\mathcal{I}(\text{Person})$ ist eine (evtl. leere) Menge von **Objekten**

$\mathcal{I}(\texttt{name})$ ist eine partielle Funktion von $\mathcal{I}(\text{Person})$ nach $\mathcal{I}(\text{String})$

$\mathcal{I}(\texttt{name})(\underline{\textit{aPerson}})$ gibt einen String oder ist undefiniert

Semantik von Klassen

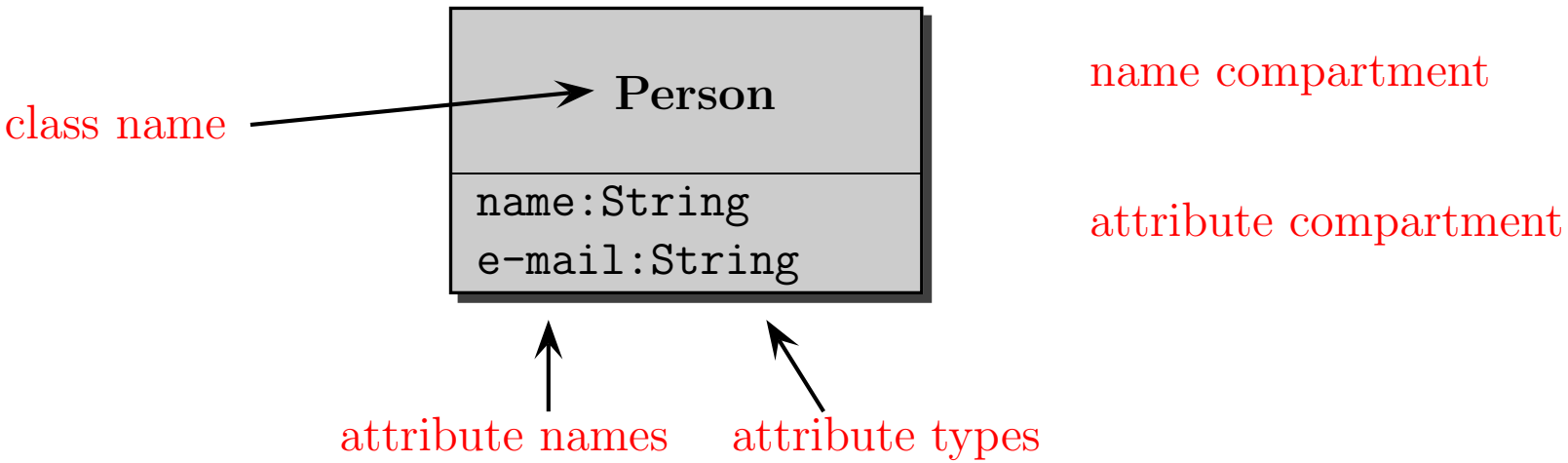


$\mathcal{I}(\text{Person})$ ist eine (evtl. leere) Menge von **Objekten**

$\mathcal{I}(\texttt{name})$ ist eine partielle Funktion von $\mathcal{I}(\text{Person})$ nach $\mathcal{I}(\text{String})$

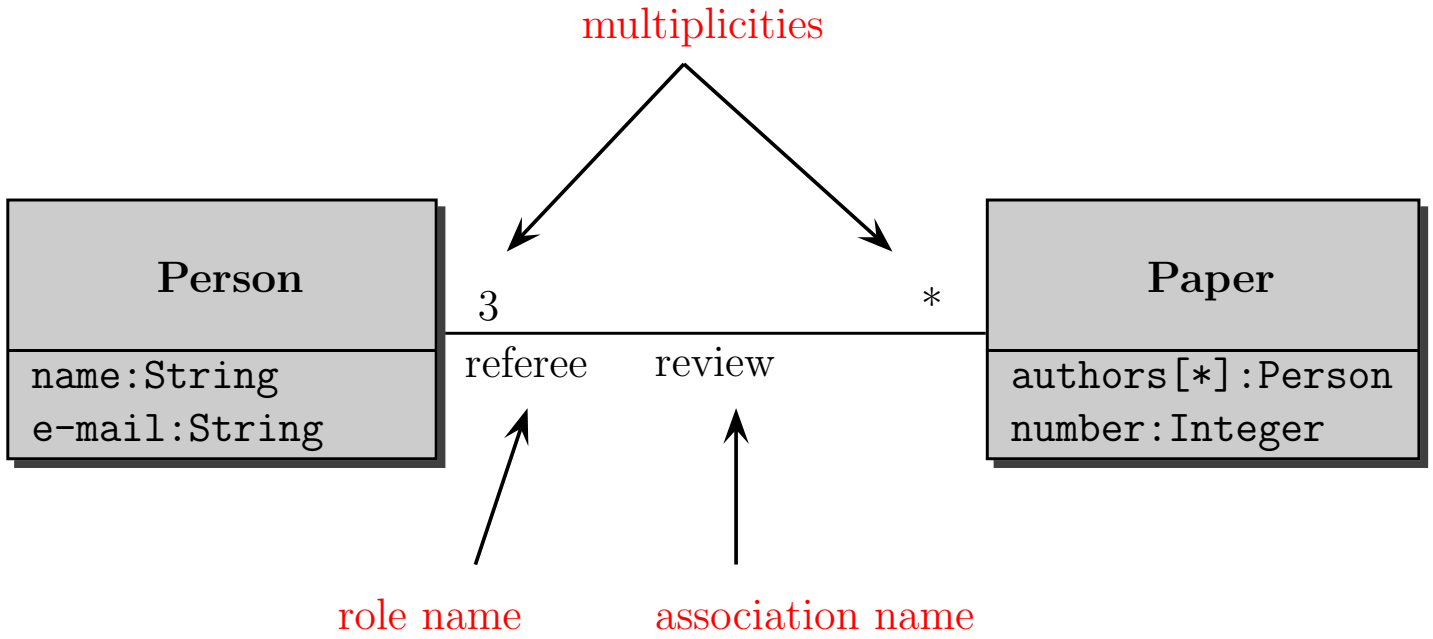
$\mathcal{I}(\texttt{name})(\underline{\textit{aPerson}})$ gibt einen String oder ist undefiniert

Semantik von Klassen



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

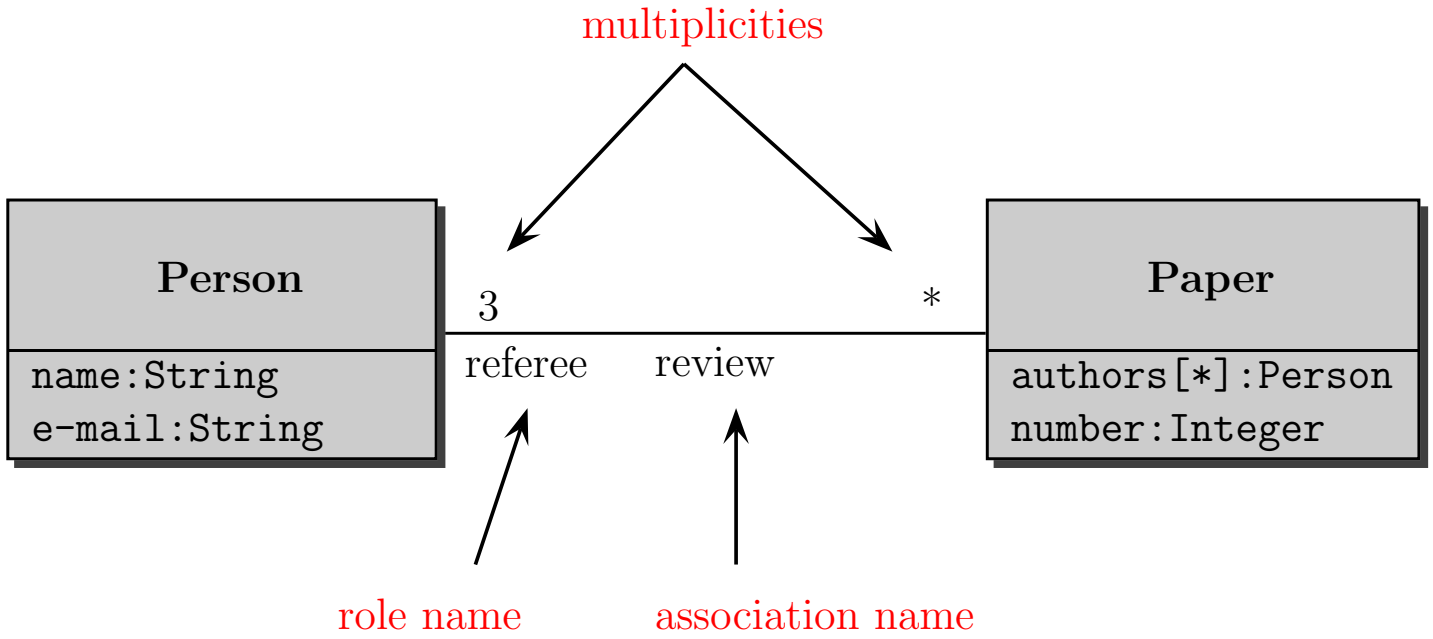
Semantik von Assoziationen



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

Multiplizität 3 verlangt:
für alle $pap \in \mathcal{I}(\text{Paper})$:
$card(\{pers \in \mathcal{I}(\text{Person}) \mid \mathcal{I}(\text{review})(pers, pap)\}) \in \mathcal{I}(3) = \{3\}$

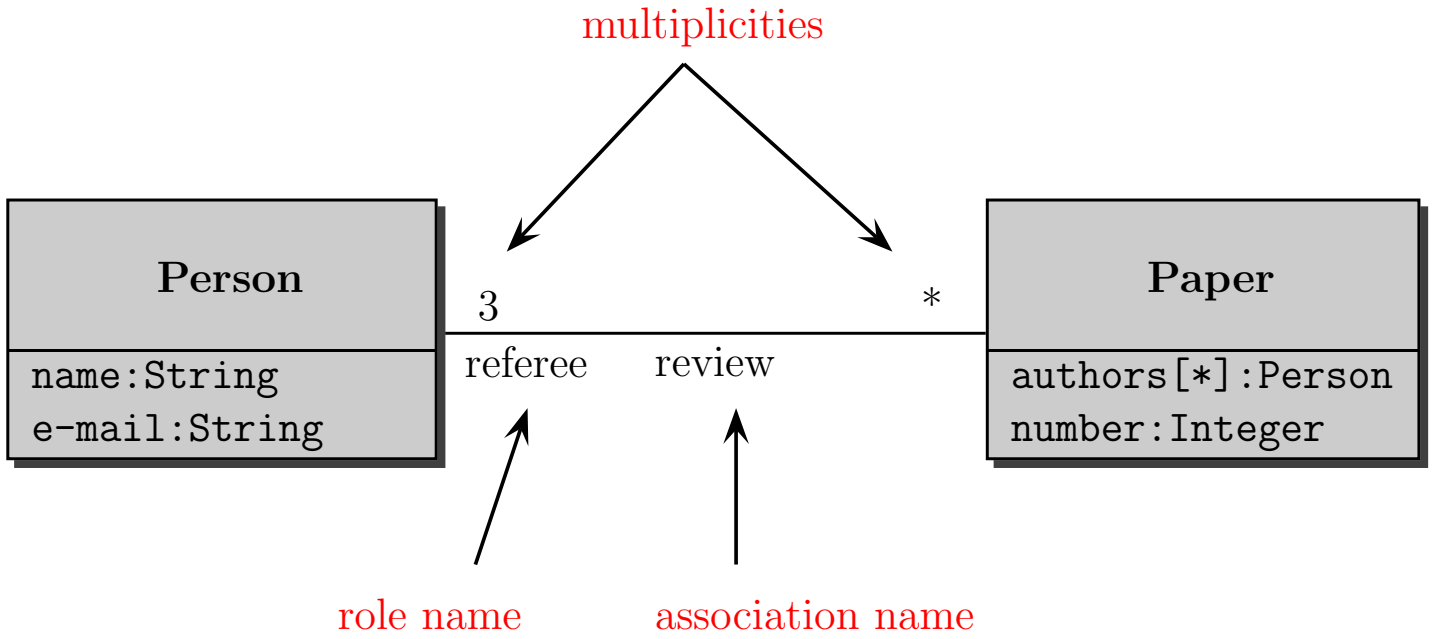
Semantik von Assoziationen



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

Multiplizität 3 verlangt:
für alle *pap* $\in \mathcal{I}(\text{Paper})$:
card$(\{\textit{pers} \in \mathcal{I}(\text{Person}) \mid \mathcal{I}(\text{review})(\textit{pers}, \textit{pap})\}) \in \mathcal{I}(3) = \{3\}$

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

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 Multiplizitäten

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

Semantik von Multiplizitäten

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

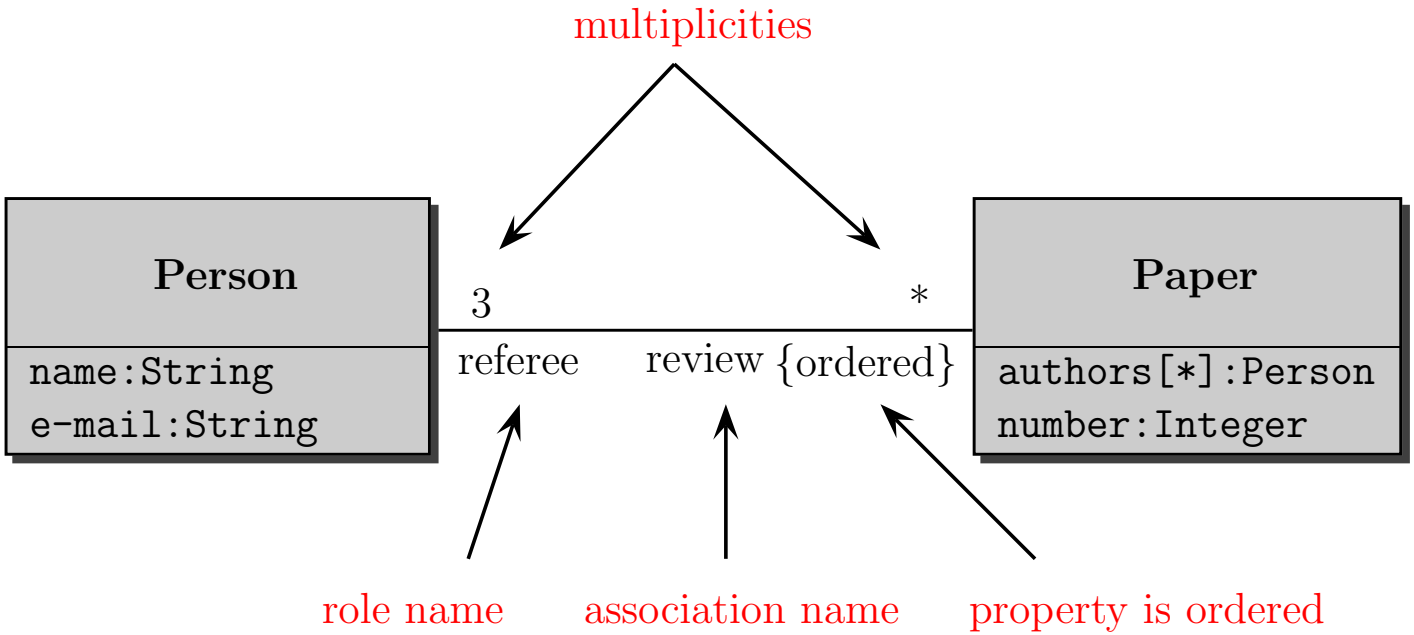
Semantik von Multiplizitäten

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

Semantik von Multiplizitäten

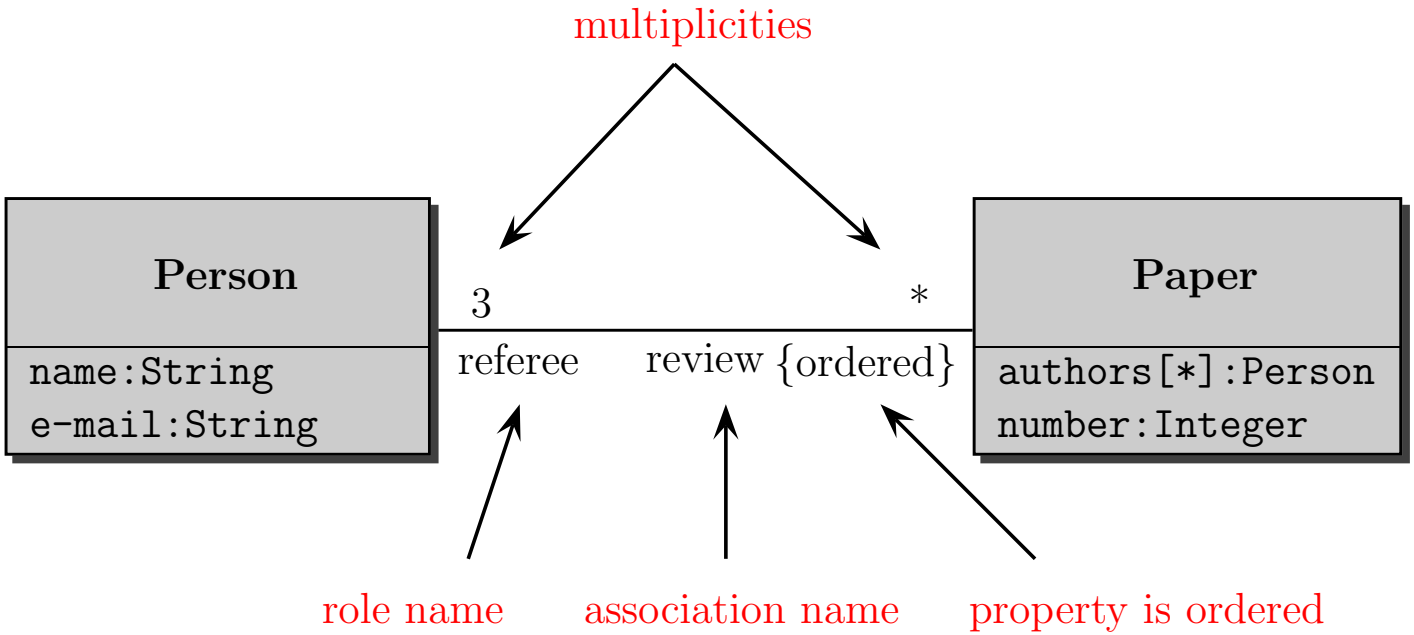
M	$\mathcal{I}(M)$
0..1	$\{0, 1\}$
0.. $*$	$\mathbb{N}$
$*$	$\mathbb{N}$
1..3	$\{1, 2, 3\}$
3	$\{3\}$

Semantik von Rollennamen



$\mathcal{I}(\text{referee}) : \mathcal{I}(\text{Paper}) \rightarrow \mathit{Set}(\mathcal{I}(\text{Person}))$
$\mathcal{I}(\text{paper}) : \mathcal{I}(\text{Person}) \rightarrow \mathit{Sequence}(\mathcal{I}(\text{Paper}))$
(Standard-Rollenname: Klassenname mit kleinem Anfangsbuchstaben)

Semantik von Rollennamen

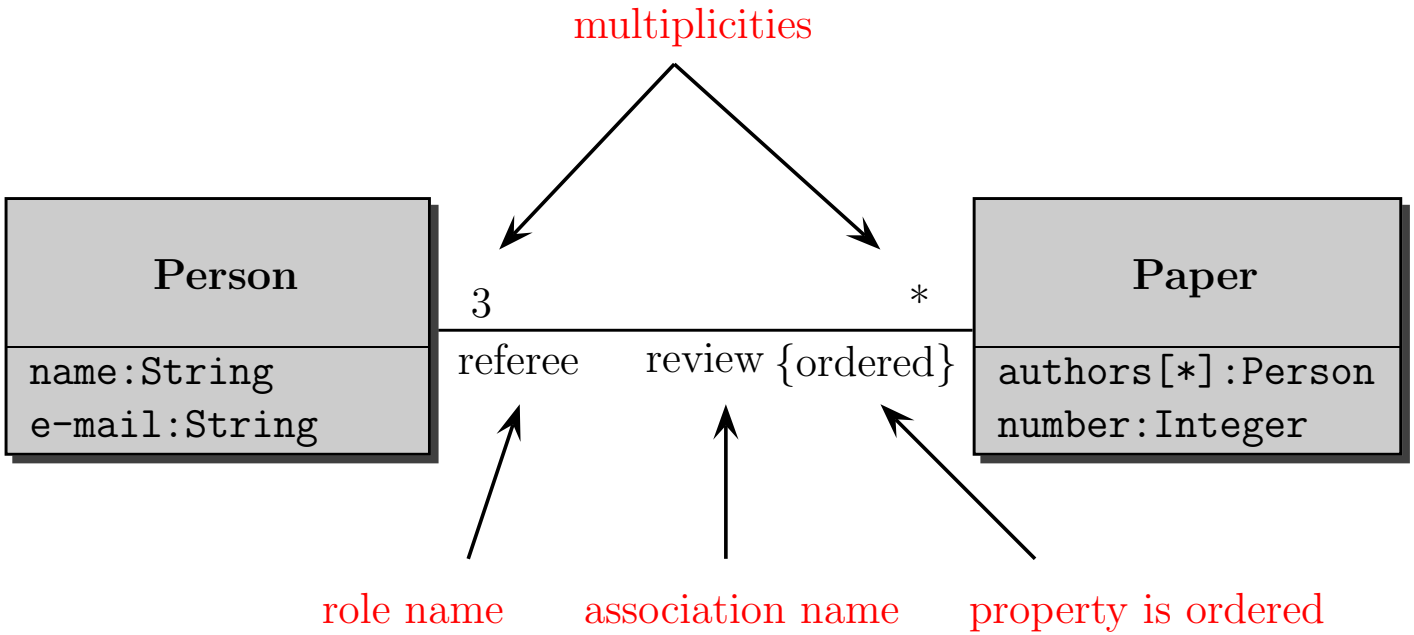


$\mathcal{I}(\text{referee}) \ : \ \mathcal{I}(\text{Paper}) \rightarrow \mathit{Set}(\mathcal{I}(\text{Person}))$

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

(Standard-Rollenname: Klassenname mit kleinem Anfangsbuchstaben)

Semantik von Rollennamen



$\mathcal{I}(\text{referee}) : \mathcal{I}(\text{Paper}) \rightarrow \mathit{Set}(\mathcal{I}(\text{Person}))$
$\mathcal{I}(\text{paper}) : \mathcal{I}(\text{Person}) \rightarrow \mathit{Sequence}(\mathcal{I}(\text{Paper}))$
(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 $\mathcal{I}(a) : C \to C'$ 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

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 $\mathcal{I}(a) : C \to C'$ 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

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 $\mathcal{I}(a) : C \to C'$ 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

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 $\mathcal{I}(a) : C \to C'$ 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

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 $\mathcal{I}(a) : C \to C'$ 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

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 $\mathcal{I}(a) : C \to C'$ 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

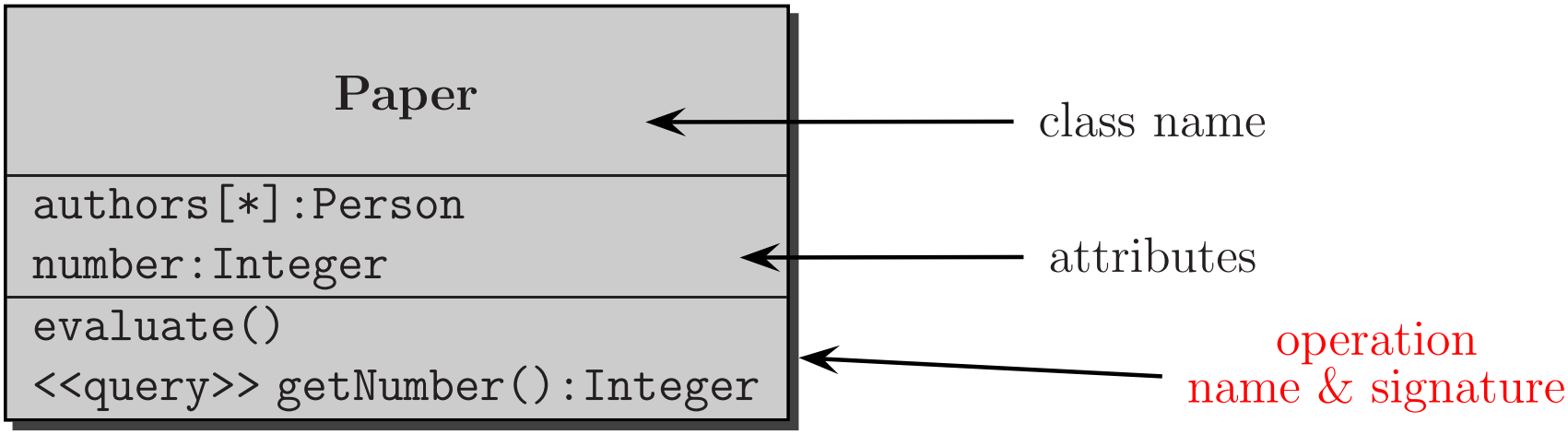
Snapshots

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 $\mathcal{I}(a) : C \to C'$ 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

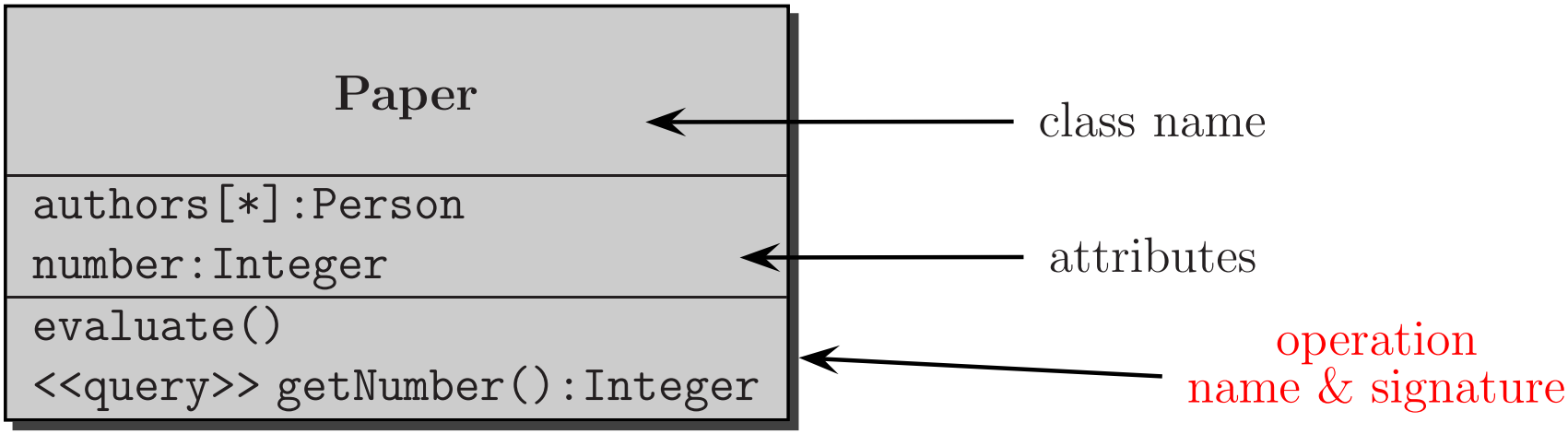
Semantik von Operationen / Queries



Operation: Transition von Snapshot zu Snapshot

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

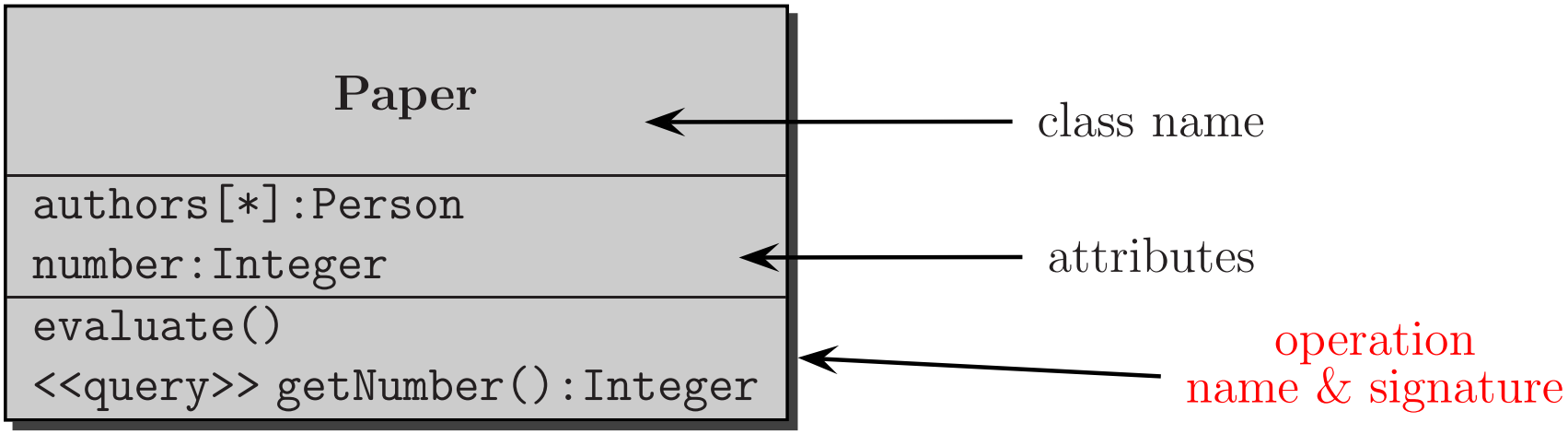
Semantik von Operationen / Queries



Operation: Transition von Snapshot zu Snapshot

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

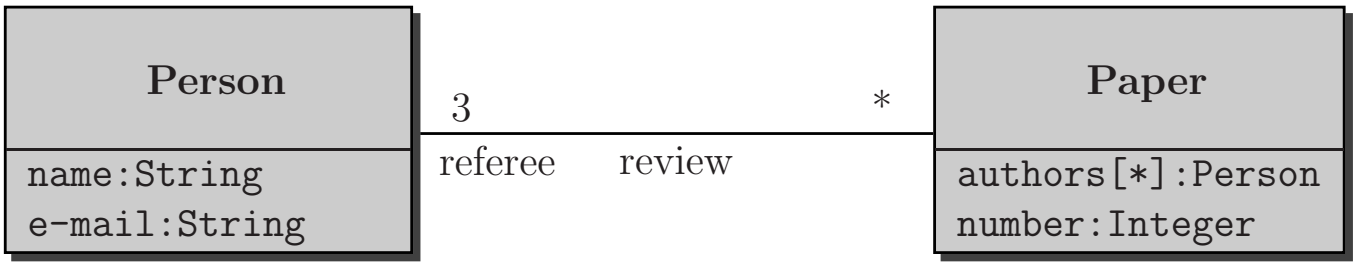
Semantik von Operationen / Queries



Operation: Transition von Snapshot zu Snapshot

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

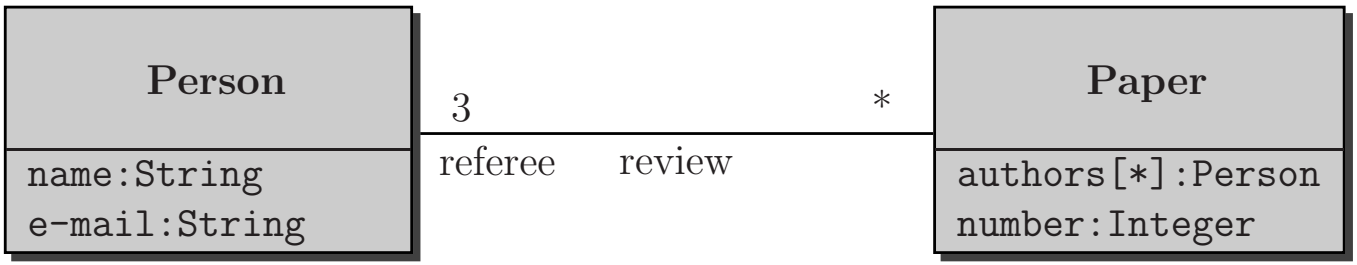
Objektdiagramm



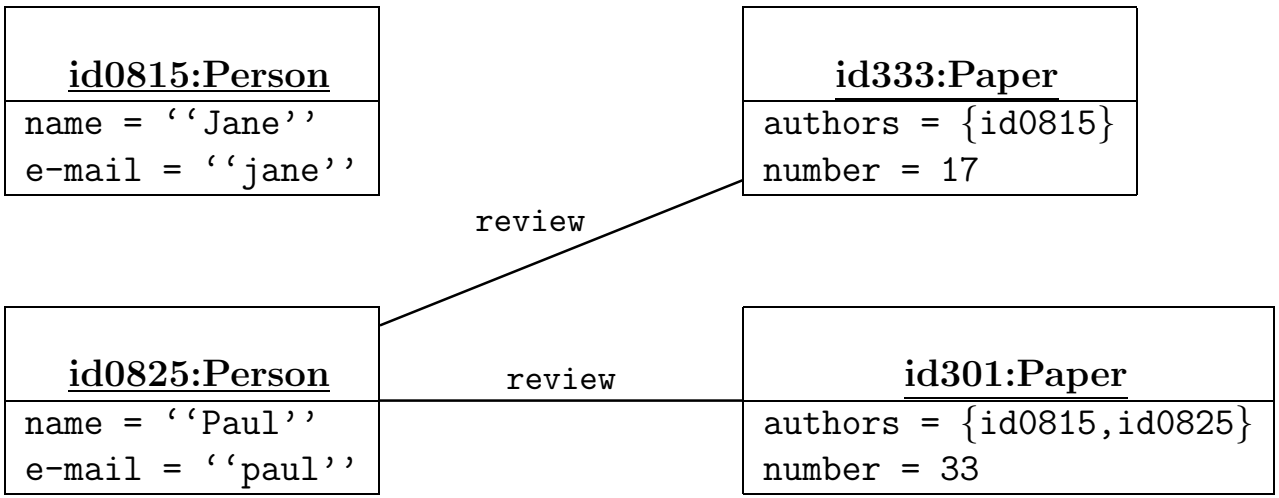
UML-Objektdiagramme repräsentieren Snapshots:

Illegale (und nicht gewollte) Instanz — Warum?

Objektdiagramm

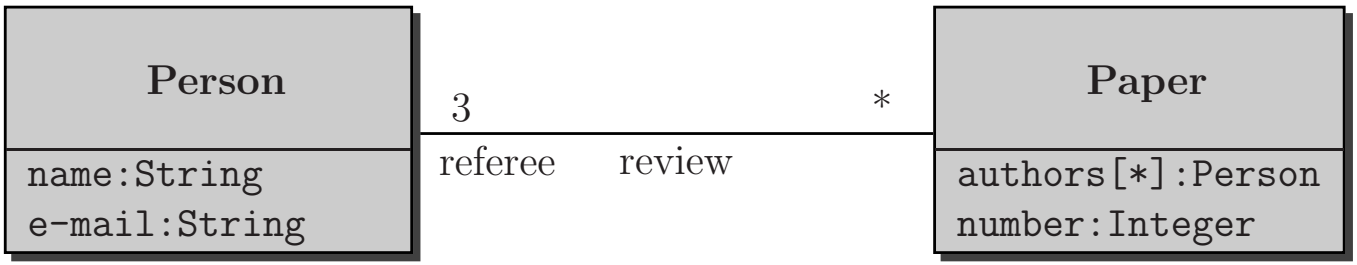


UML-Objektdiagramme repräsentieren Snapshots:

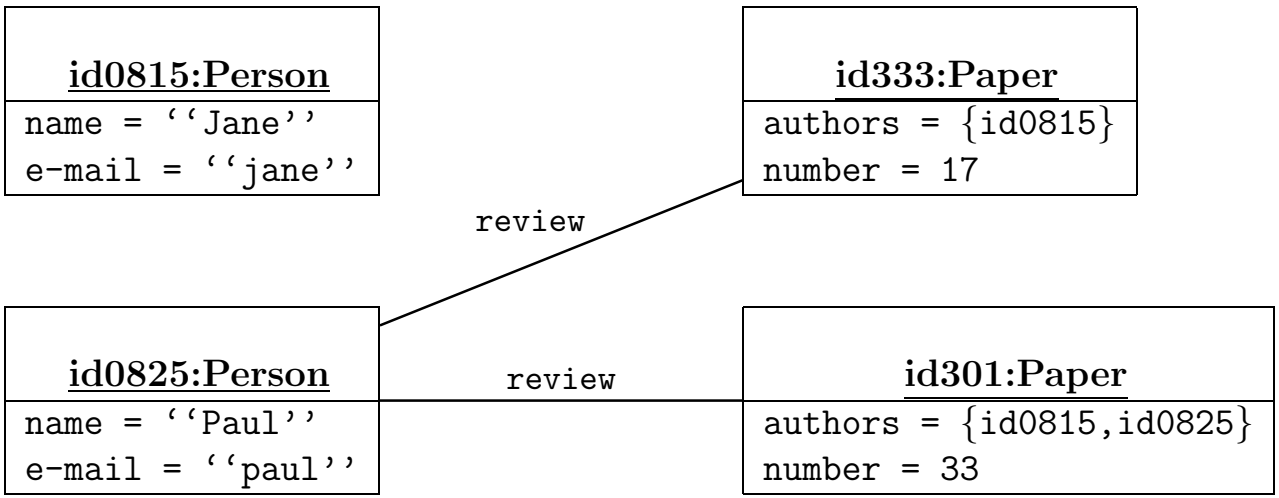


Illegale (und nicht gewollte) Instanz — Warum?

Objektdiagramm

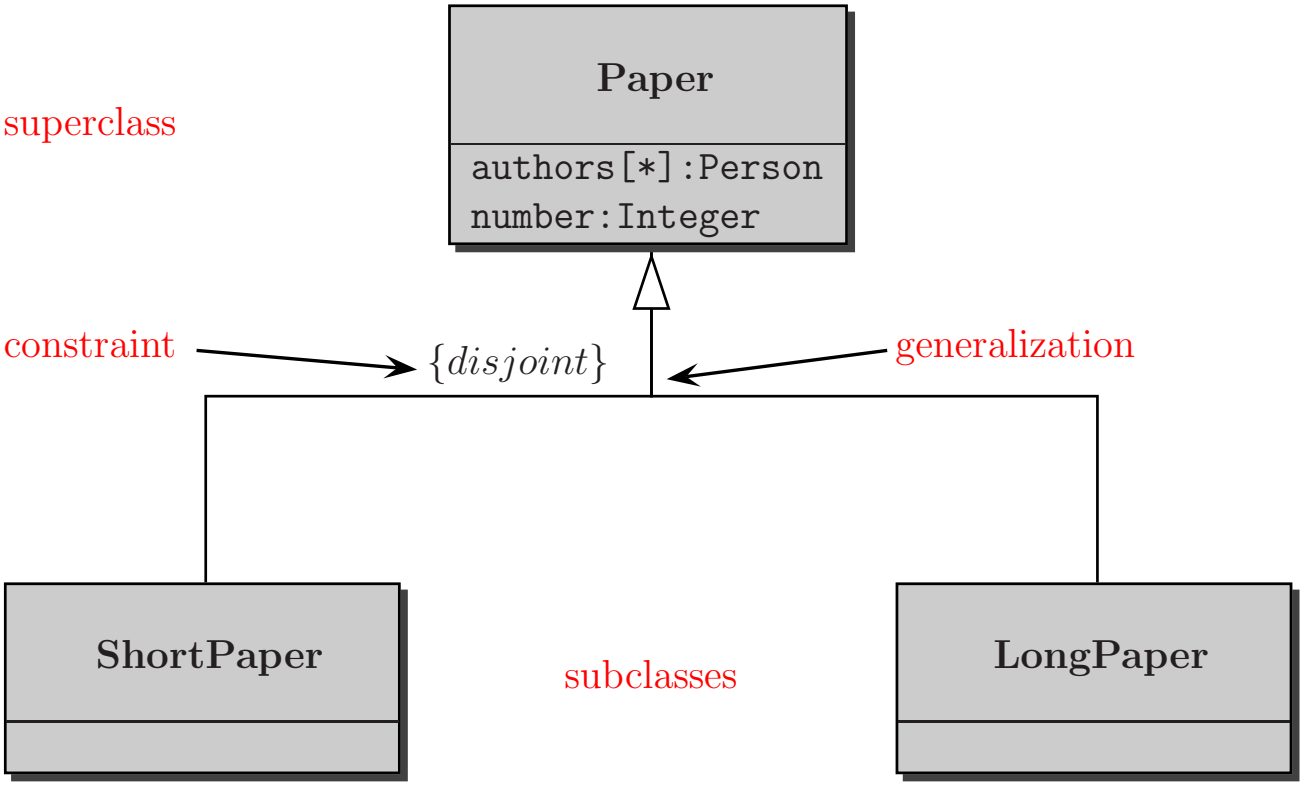


UML-Objektdiagramme repräsentieren Snapshots:



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

Historie

- 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.

Historie

- 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.

Historie

- 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.

Historie

- 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.

Historie

- 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.

Motivation

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

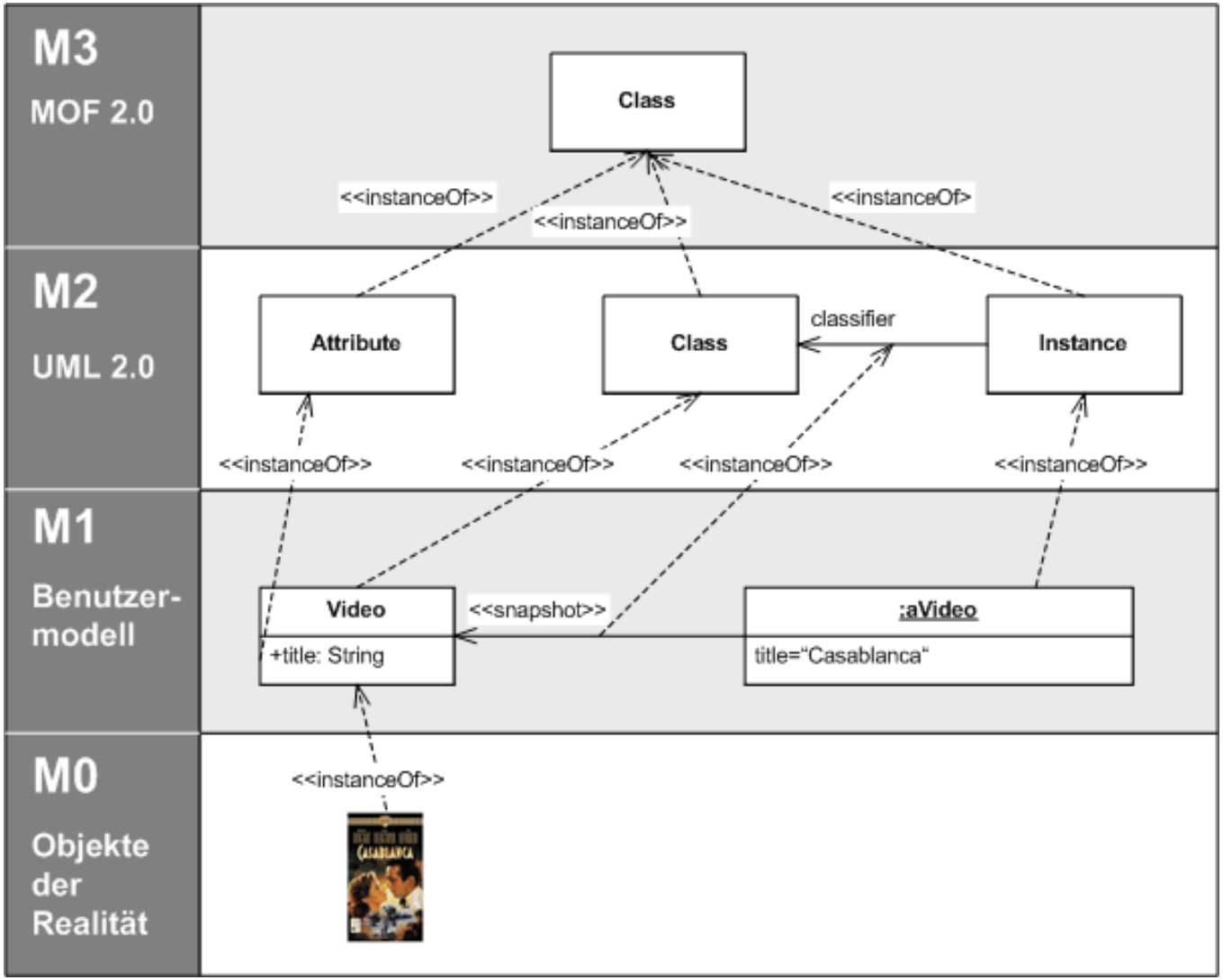
	Schicht	Beispiel
M3	Meta-Meta-Modell	Meta Object Faciliy (MOF)
M2	Meta-Modell	UML-Meta-Modell
M1	Modell	UML-Klassendiagramm
M0	Objekte der Realität	Auto

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)

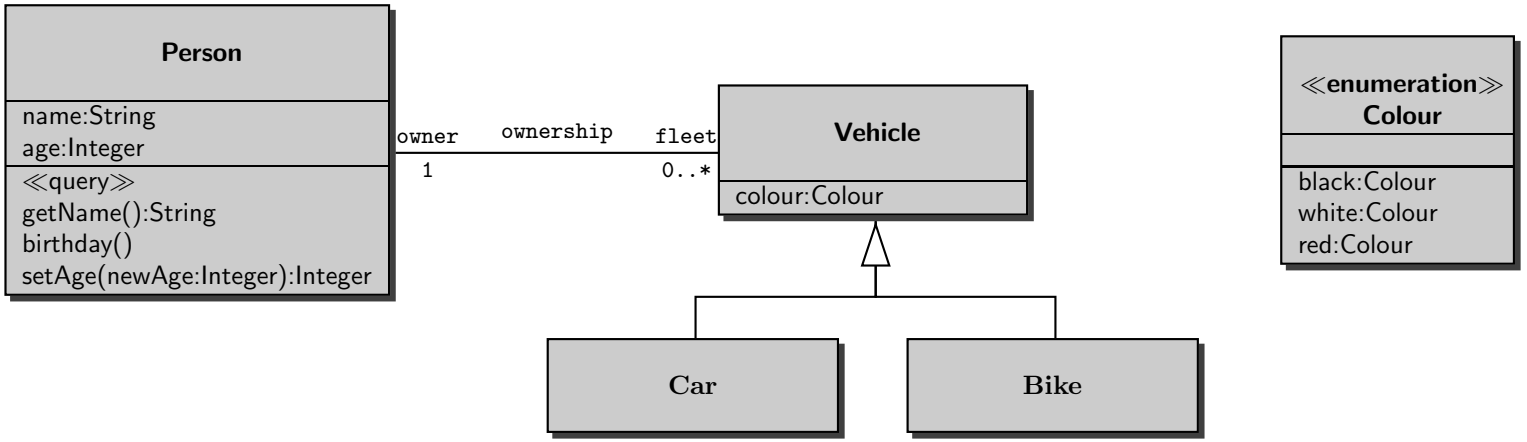
Wo OCL?

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

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)

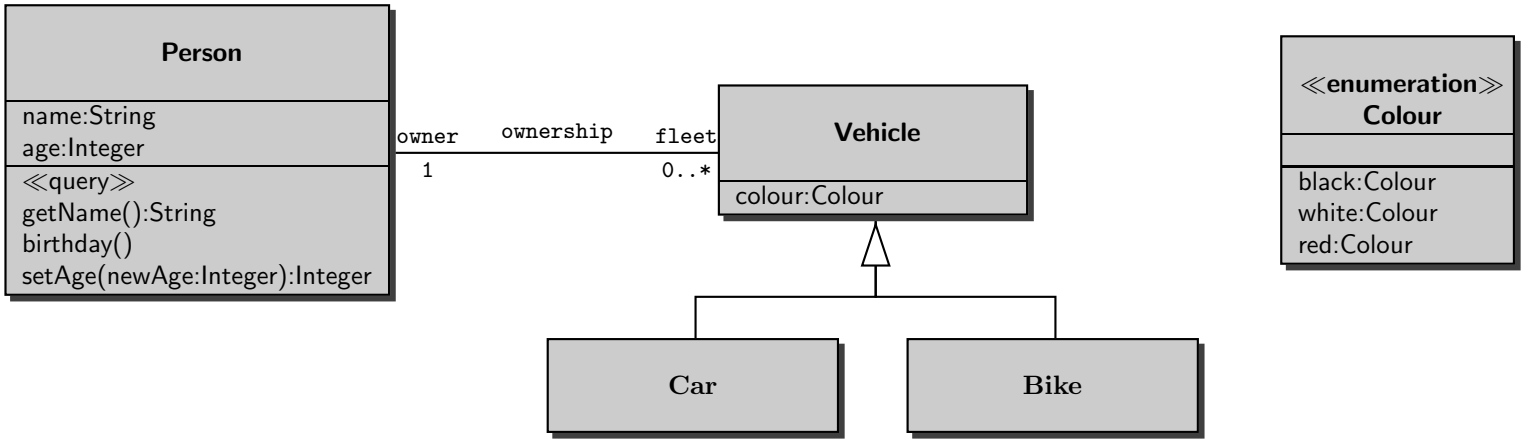
UML alleine reicht nicht . . .



- 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

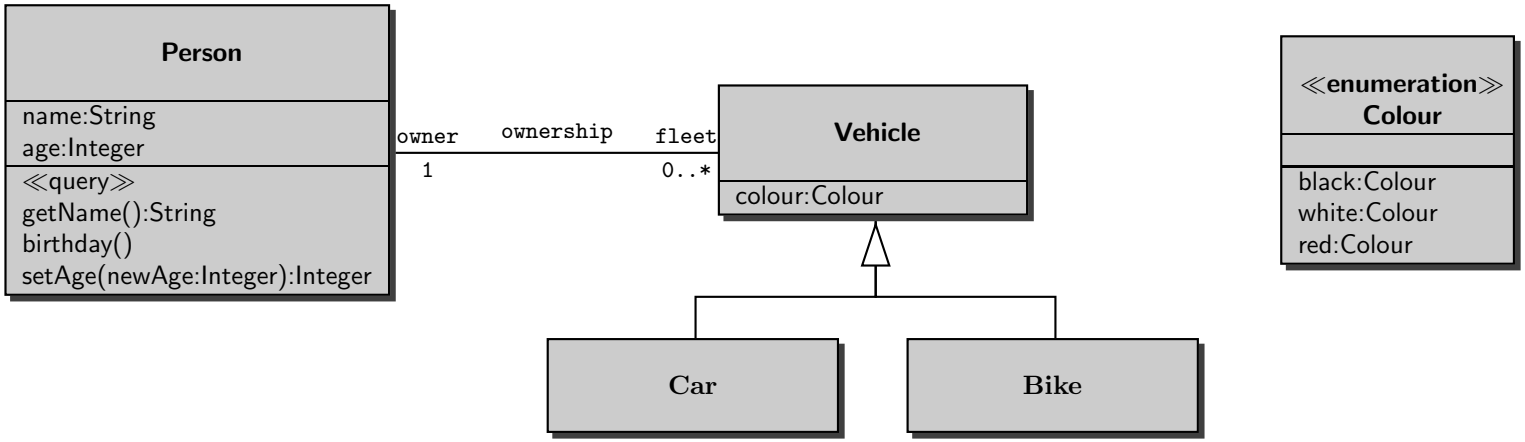
UML alleine reicht nicht . . .



- 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

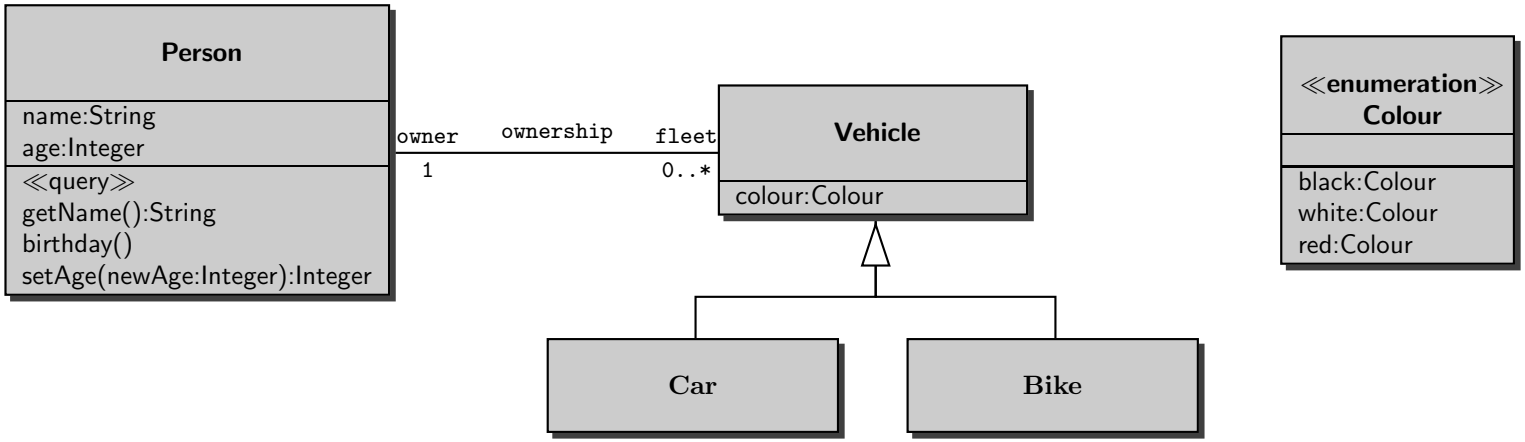
UML alleine reicht nicht . . .



- 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

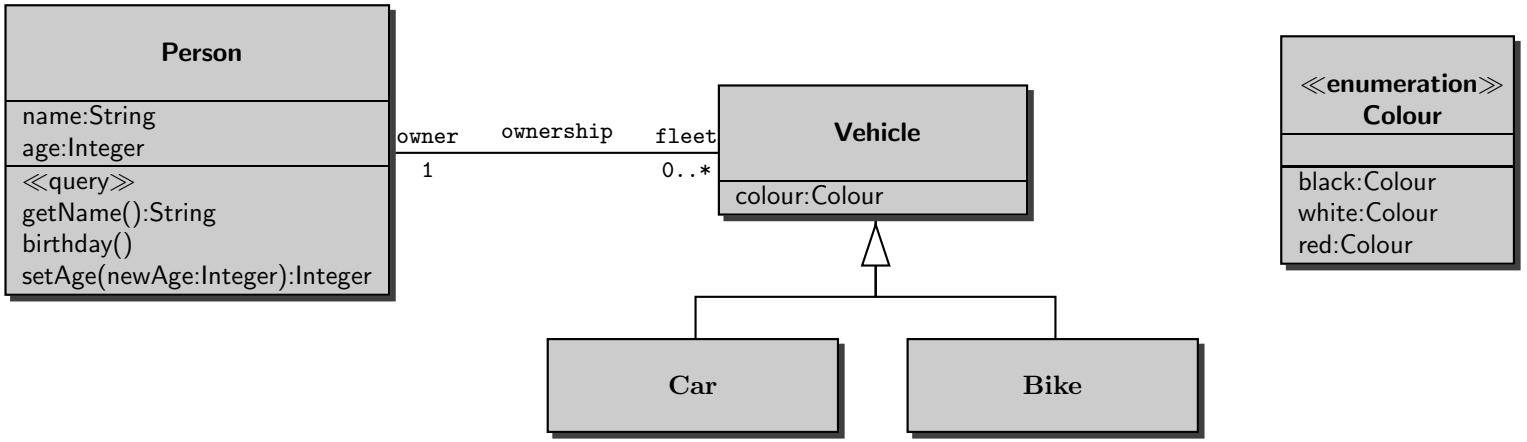
UML alleine reicht nicht . . .



- 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

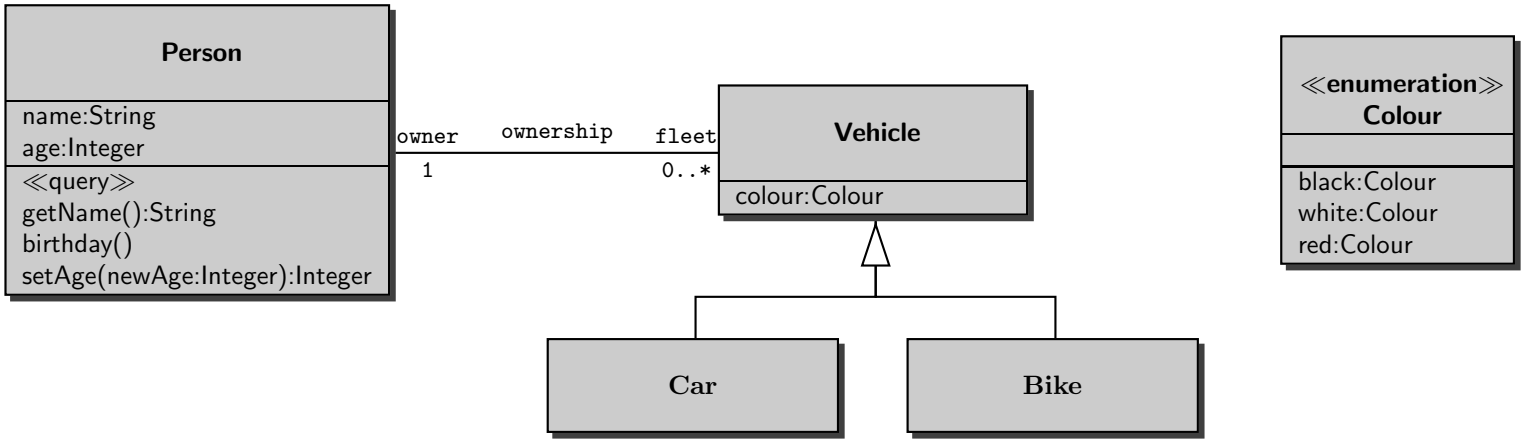
UML alleine reicht nicht . . .



- 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

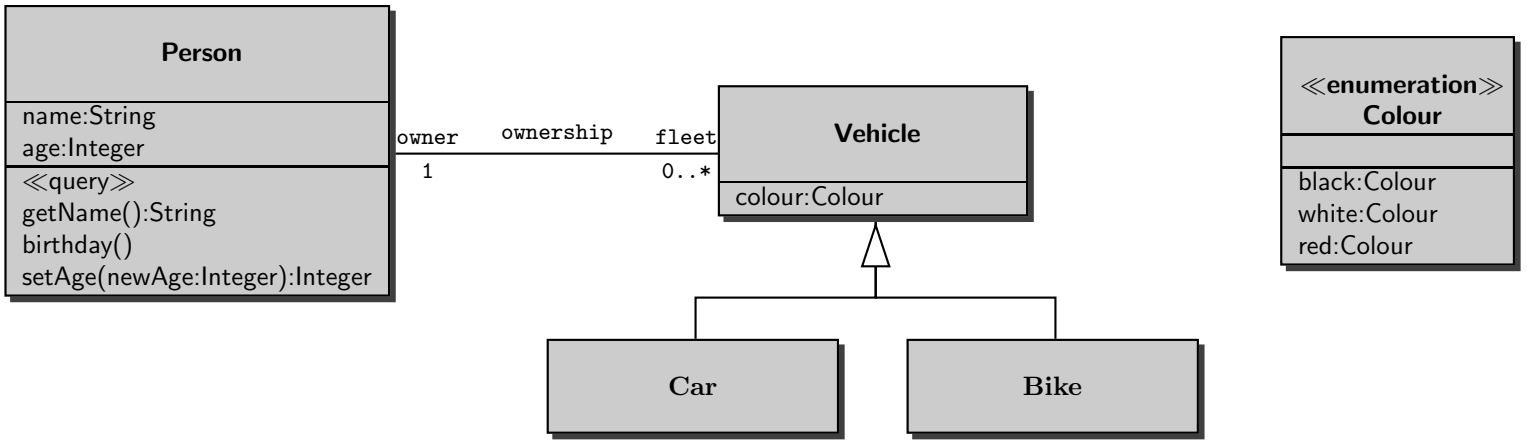
UML alleine reicht nicht . . .



- 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

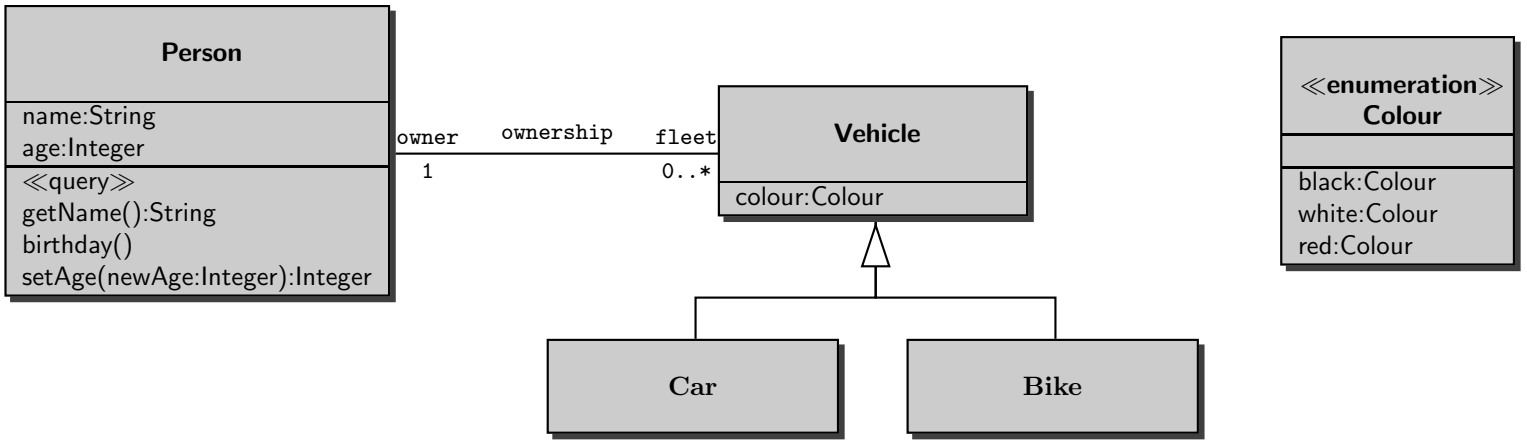


"Keine Person besitzt mehr als 3 Fahrzeuge."

context p:Person
inv: p.fleet->size() <= 3

oder Änderung der Multiplizität

OCL-Beispiele



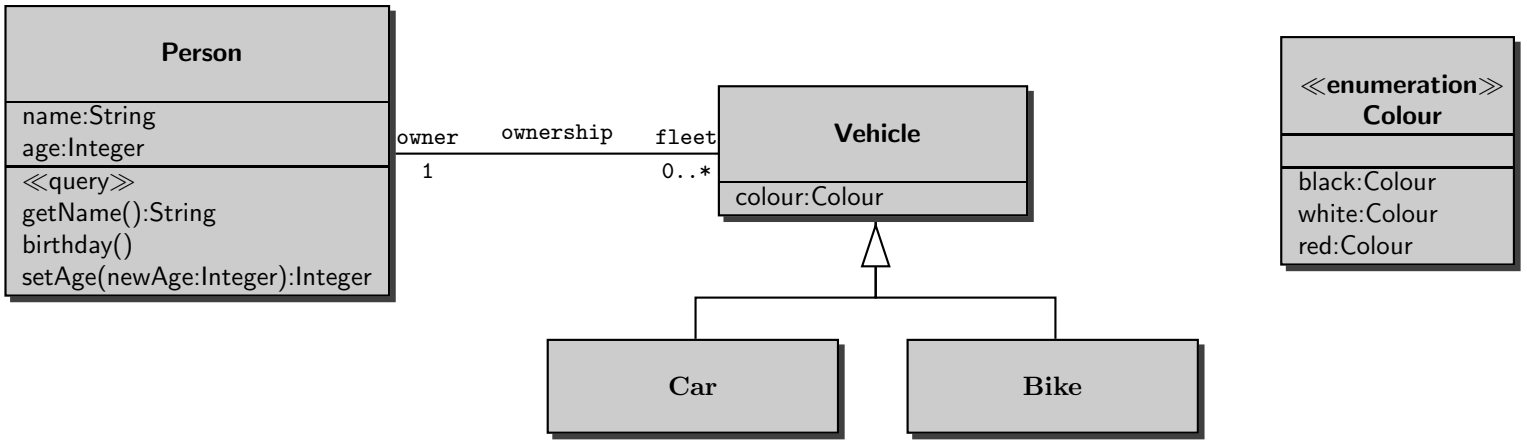
"Keine Person besitzt mehr als 3 Fahrzeuge."

context p:Person

inv: p.fleet->size() <= 3

oder Änderung der Multiplizität

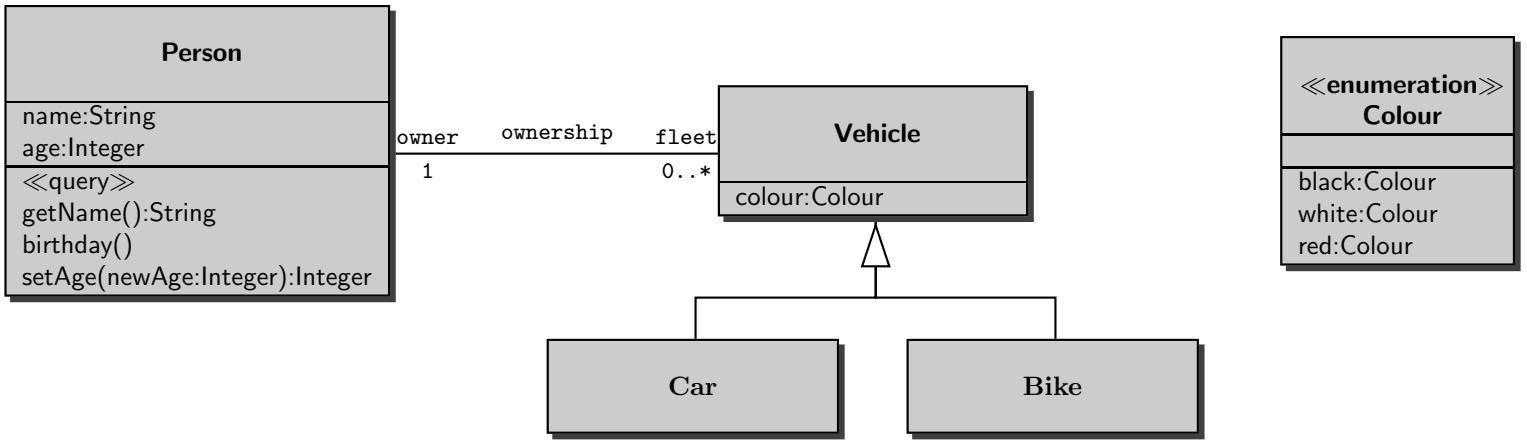
OCL-Beispiele



"Keine Person besitzt mehr als 3 Fahrzeuge."

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

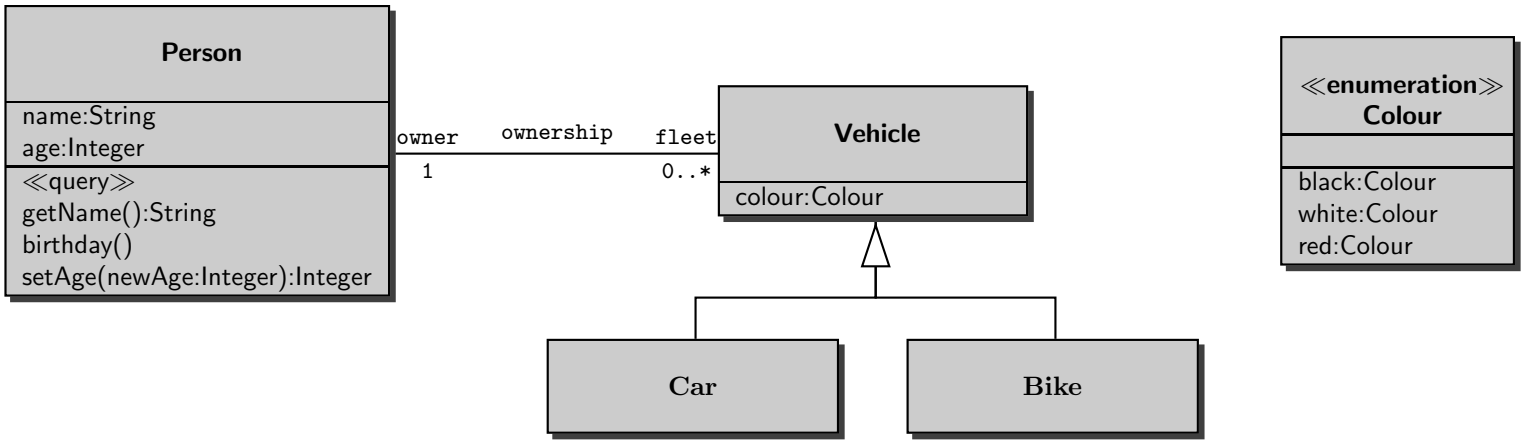
OCL-Beispiele



"Keine Person besitzt mehr als 3 **schwarze** Fahrzeuge."

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

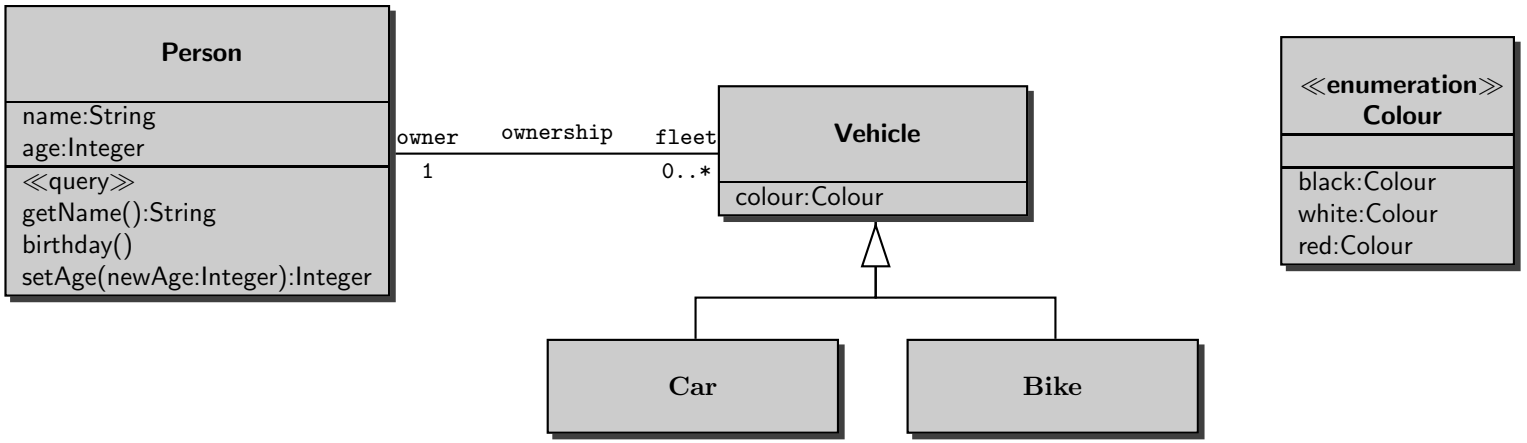
OCL-Beispiele



"Keine Person besitzt mehr als 3 **schwarze** Fahrzeuge."

context p:Person
inv: p.fleet–>select(v | v.colour = Colour.black)–>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

OCL-Constraints

Es gibt zwei Arten von OCL-Constraints:

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

OCL-Constraints

Es gibt zwei Arten von OCL-Constraints:

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

OCL-Constraints

Es gibt zwei Arten von OCL-Constraints:

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

Der Context

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

Der Context

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

context c: typeName
inv: OclExpression_1
$\vdots$
inv: OclExpression_n

Beispiel

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

Der Context

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

context c: typeName
inv: OclExpression_1
$\vdots$
inv: OclExpression_n

Beispiel

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

Der Context

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

context c:typeName::opName(p_1:type$_1$,. . . ,p_k:type$_k$):rtype
pre: OclExpression$_1^{pre}$
post: OclExpression$_1^{post}$
⋮
pre: OclExpression$_n^{pre}$
post: OclExpression$_n^{post}$

Beispiel

context c:Person::getName():String
post: result = c.name

"Ein Aufruf von getName() liefert den Wert des Attributs name"

Der Context

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

context c:typeName::opName(p_1:type$_1$,. . . ,p_k:type$_k$):rtype
pre: OclExpression$_1^{pre}$
post: OclExpression$_1^{post}$
⋮
pre: OclExpression$_n^{pre}$
post: OclExpression$_n^{post}$

Beispiel

context c:Person::getName():String
post: result = c.name

“Ein Aufruf von getName() liefert den Wert des Attributs name”

Der Context

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

context c:typeName::opName(p_1:type$_1$,. . . ,p_k:type$_k$):rtype
pre: OclExpression$_1^{pre}$
post: OclExpression$_1^{post}$
⋮
pre: OclExpression$_n^{pre}$
post: OclExpression$_n^{post}$

Beispiel

context c:Person::getName():String
post: result = c.name

“Ein Aufruf von getName() liefert den Wert des Attributs name”

Semantik von Invarianten

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.*

Semantik von Vor- und Nachbedingungen

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

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

It is the responsability 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.

Noch ein Zitat

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

Noch ein Zitat

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

Noch ein Zitat

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

Noch ein Zitat

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

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.

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.

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.

Gesamtheiten

In der Objektbeschreibungssprache OCL benutzen wir den Begriff

Gesamtheit (collection)

als gemeinsamen Oberbegriff für

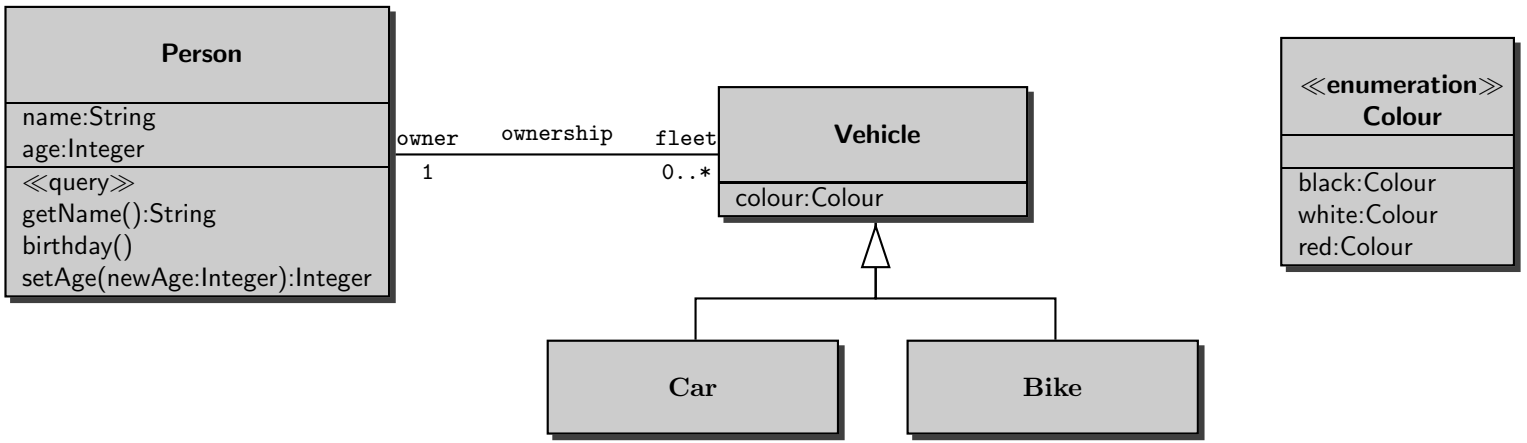
- Menge (set)
  Jedes Element kommt nur einmal vor.
- Multimenge (bag)
  Elemente können mehrfach auftreten.
- Folge (sequence)
  Elemente können mehrfach auftreten und sind in einer Reihenfolge angeordnet.

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($e \mid exp$)	$\{m \in M \mid exp(m/e)$ ist wahr $\}$
collect($e \mid exp$)	$\{exp(m/e) \mid m \in M\}$

OCL-Ausdrücke: Navigation

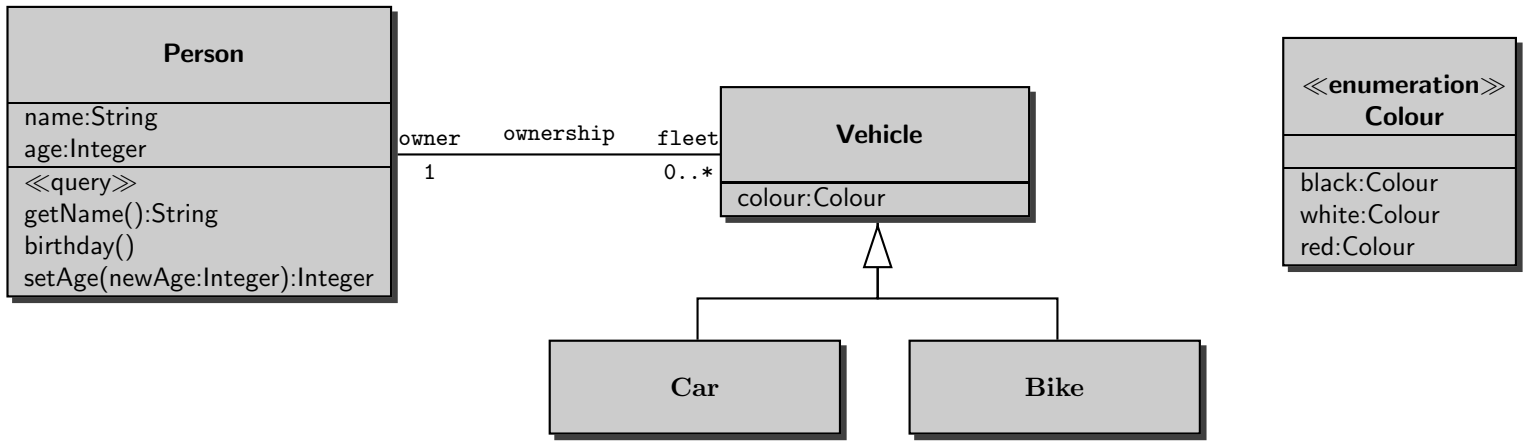


context p:Person

OCL ist eine getypte Sprache.

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

OCL-Ausdrücke: Navigation

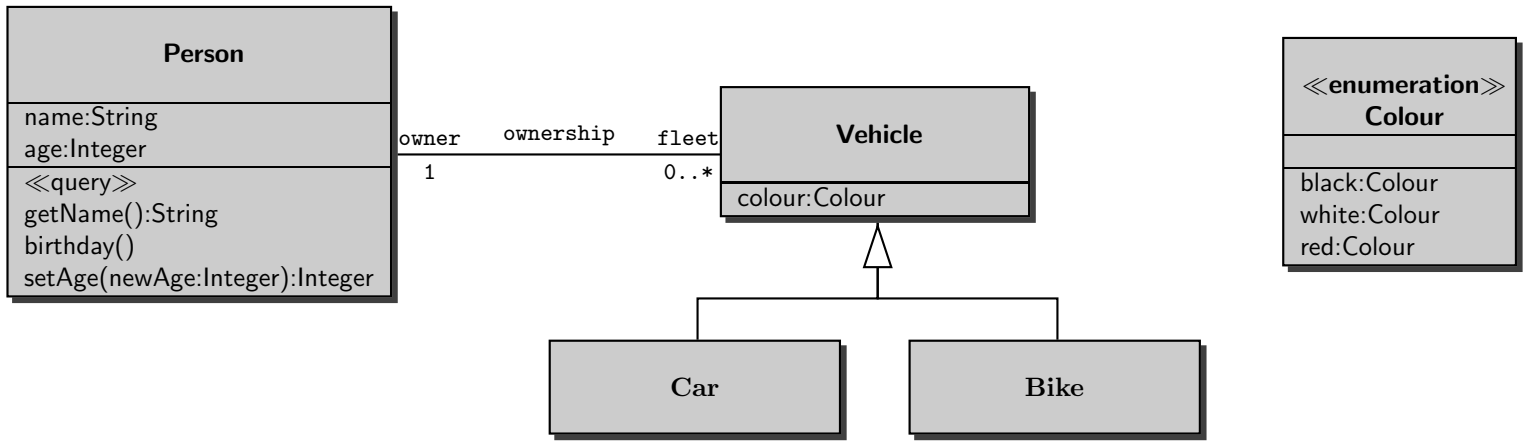


context p:Person

OCL ist eine getypte Sprache.

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

OCL-Ausdrücke: Navigation

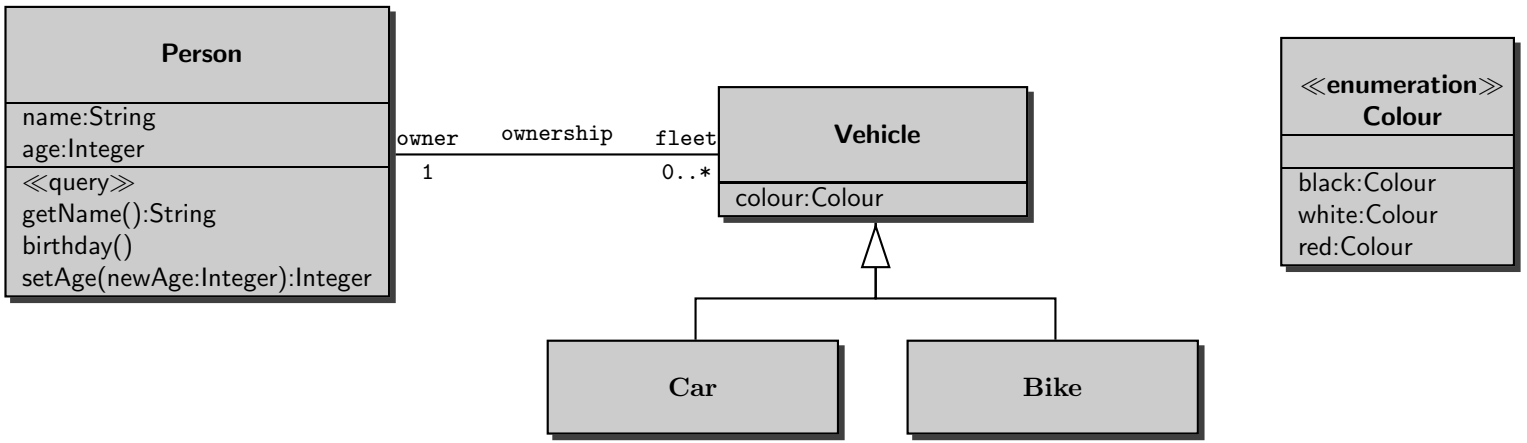


context p:Person

OCL ist eine getypte Sprache.

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

OCL-Ausdrücke: Navigation

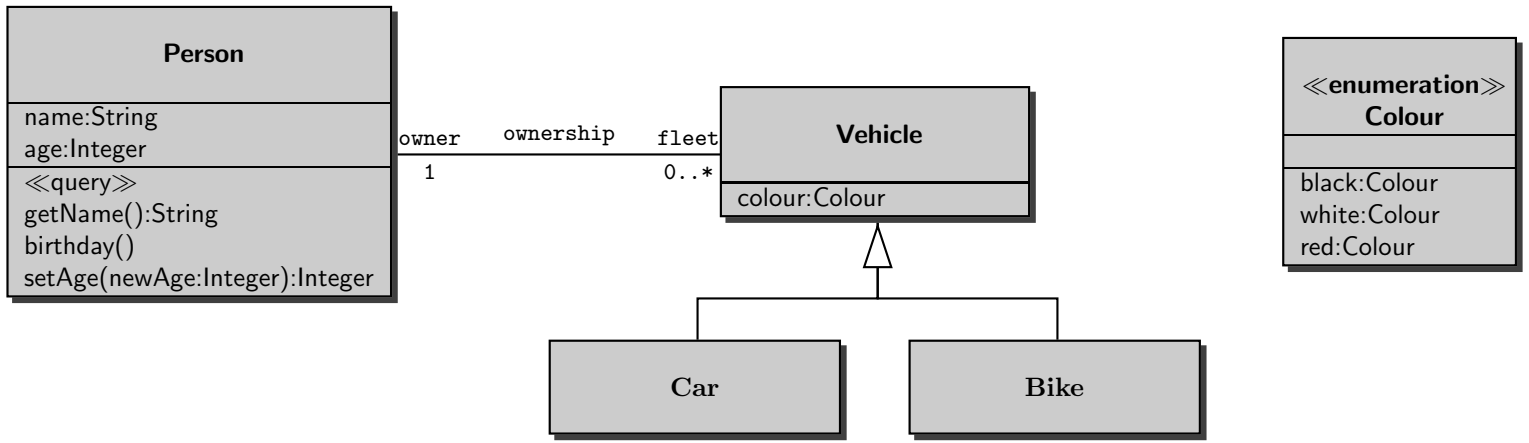


context p:Person

OCL ist eine getypte Sprache.

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

OCL-Ausdrücke: Navigation

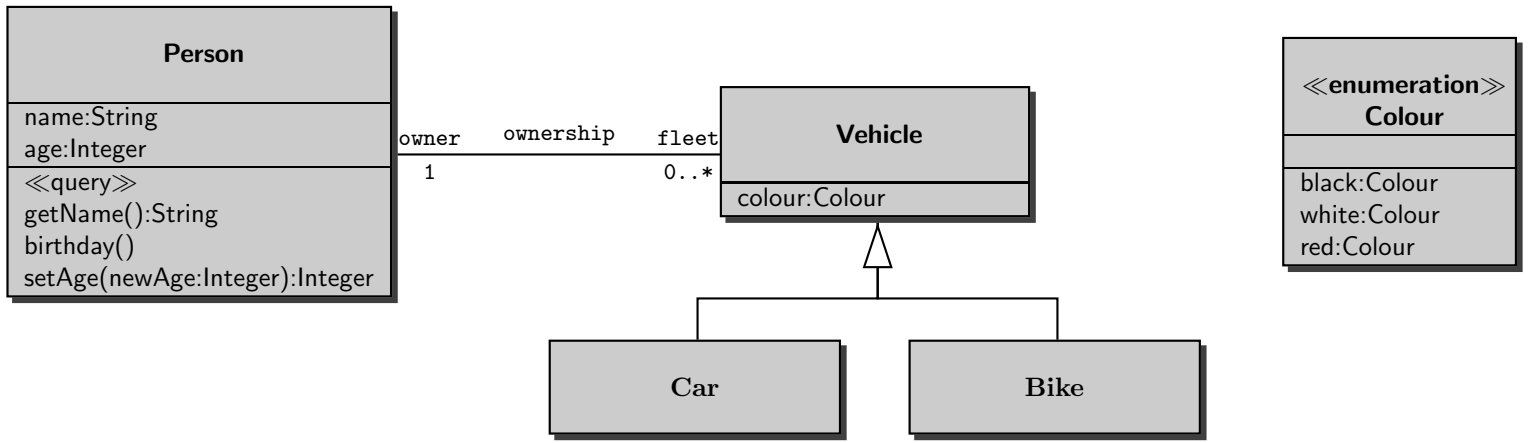


context p:Person

OCL ist eine getypte Sprache.

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

OCL-Ausdrücke: Navigation

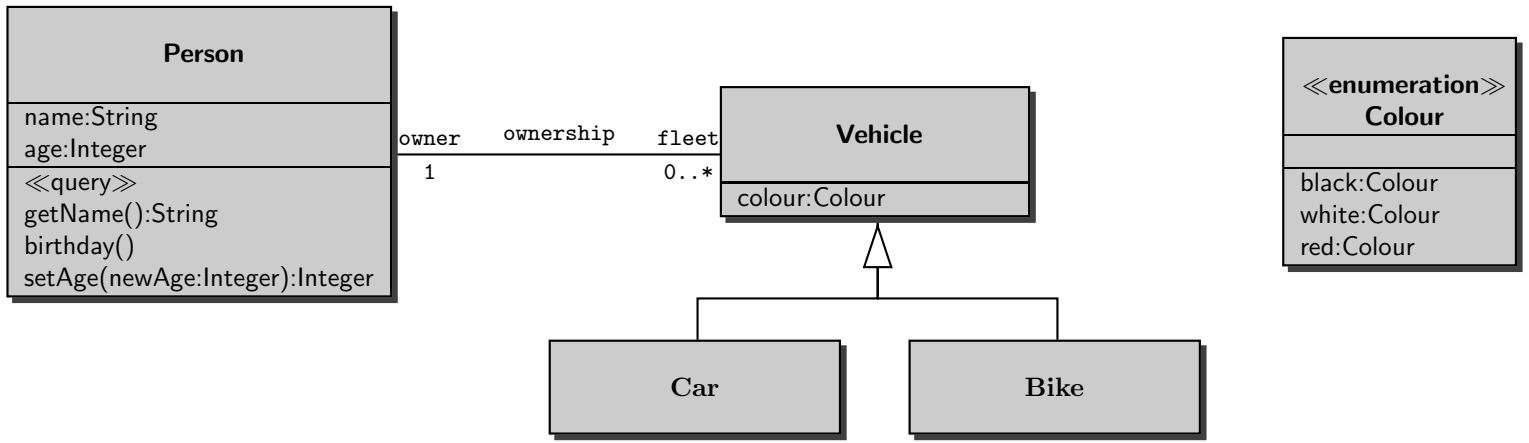


context p:Person

OCL ist eine getypte Sprache.

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

OCL-Ausdrücke: Navigation

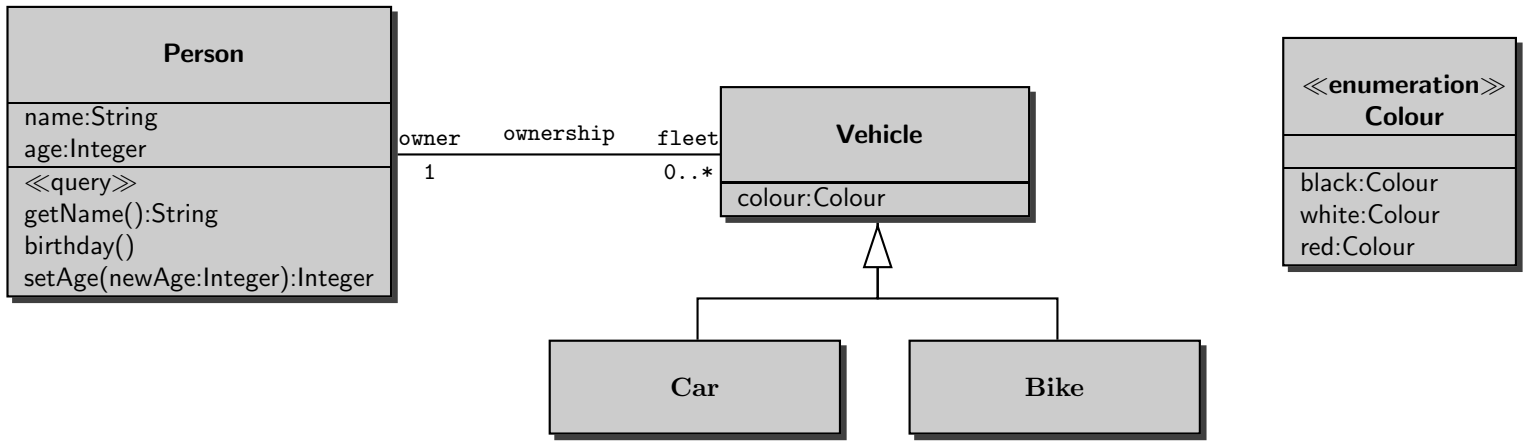


context p:Person

OCL ist eine getypte Sprache.

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

OCL-Ausdrücke: Navigation

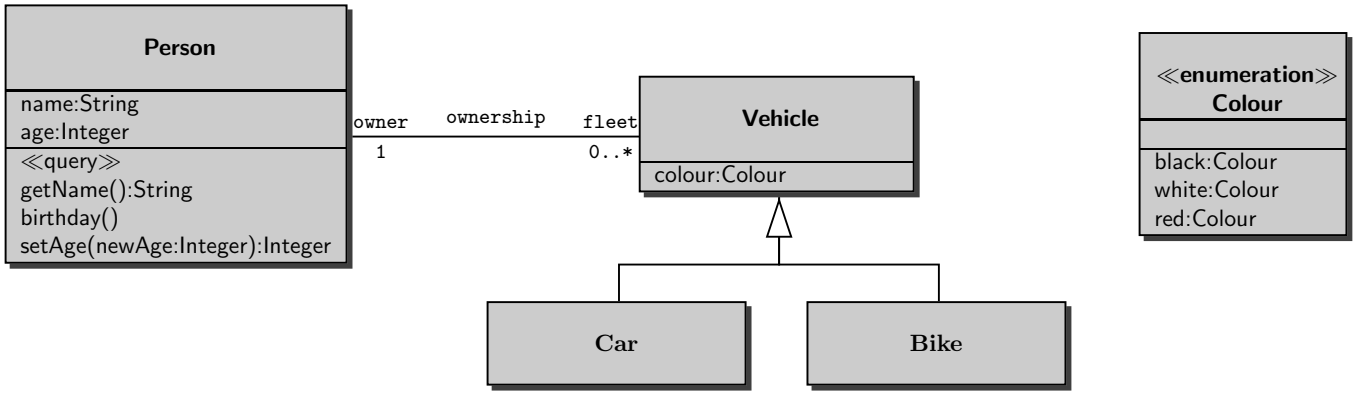


context p:Person

OCL ist eine getypte Sprache.

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

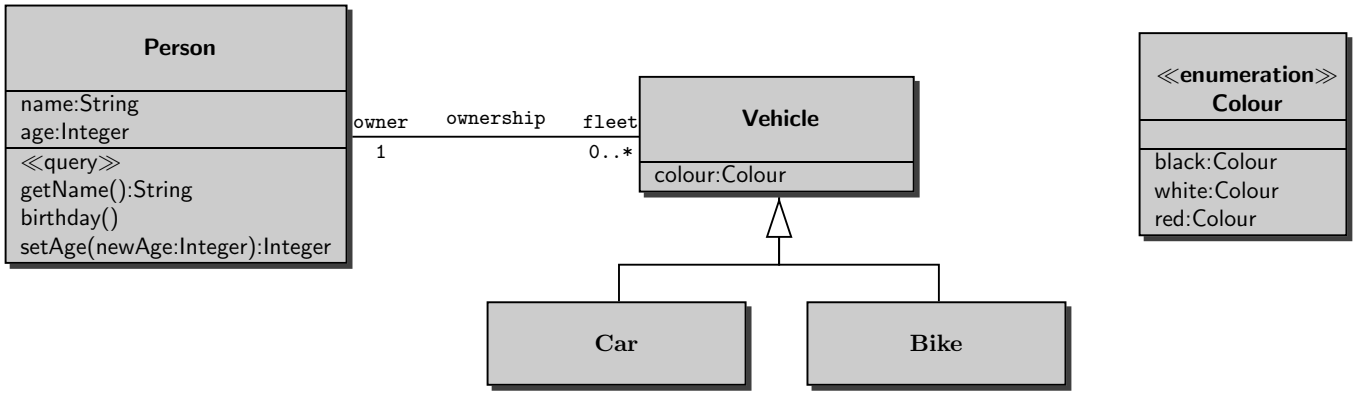
OCL-Ausdrücke: Vordefinierte Variablen



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

Beispiel

OCL-Ausdrücke: Vordefinierte Variablen

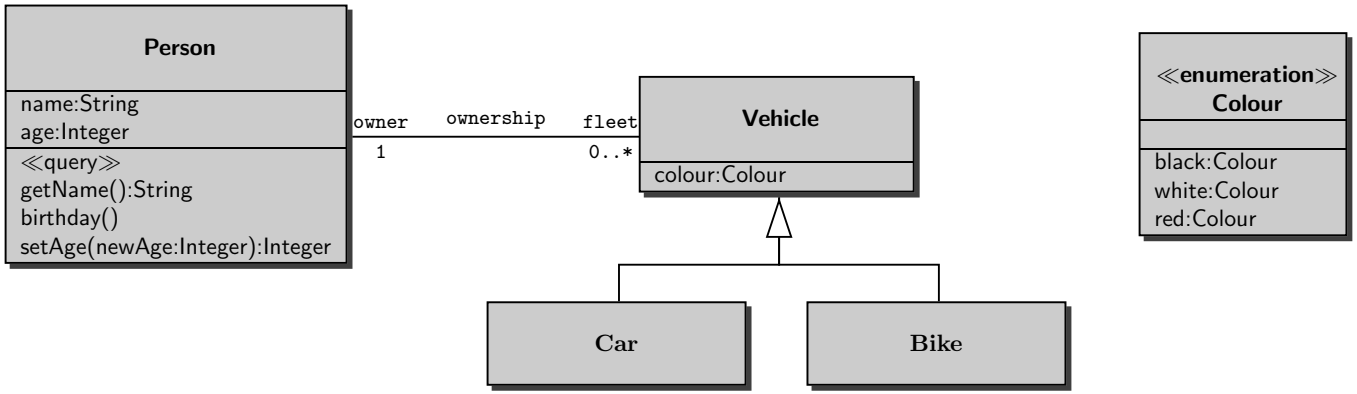


Context	Vordefinierte Variablen
context p:Person	p
context Person	self
context Person::getName():String	self, result
context Person::setAge(newAge: Integer)	self, newAge

Beispiel

context p:Person
inv: p.fleet->size() <= 3

OCL-Ausdrücke: Vordefinierte Variablen

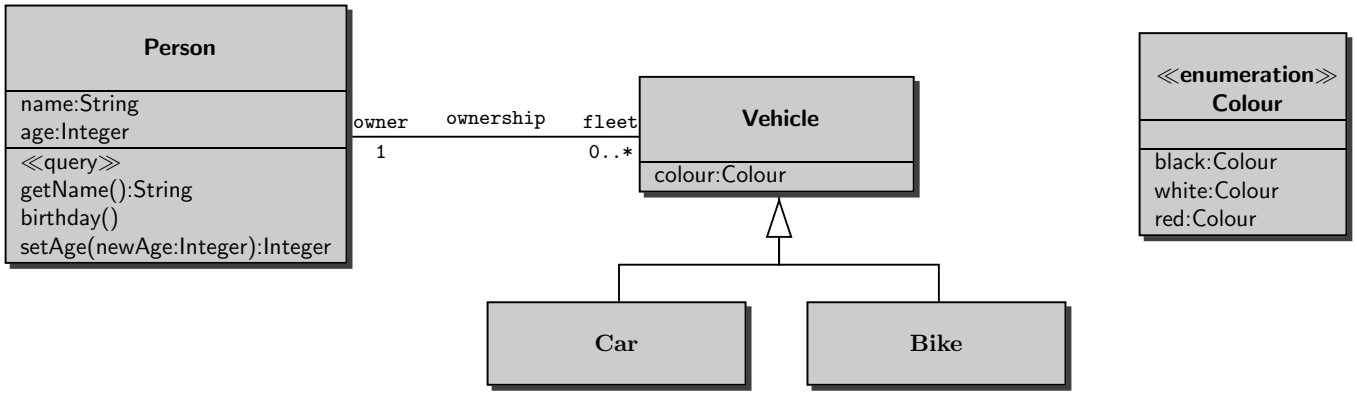


Context	Vordefinierte Variablen
context p:Person	p
context Person	self
context Person::getName():String	self, result
context Person::setAge(newAge: Integer)	self, newAge

Beispiel

context Person
inv: self.fleet–>size() <= 3

OCL-Ausdrücke: Vordefinierte Variablen

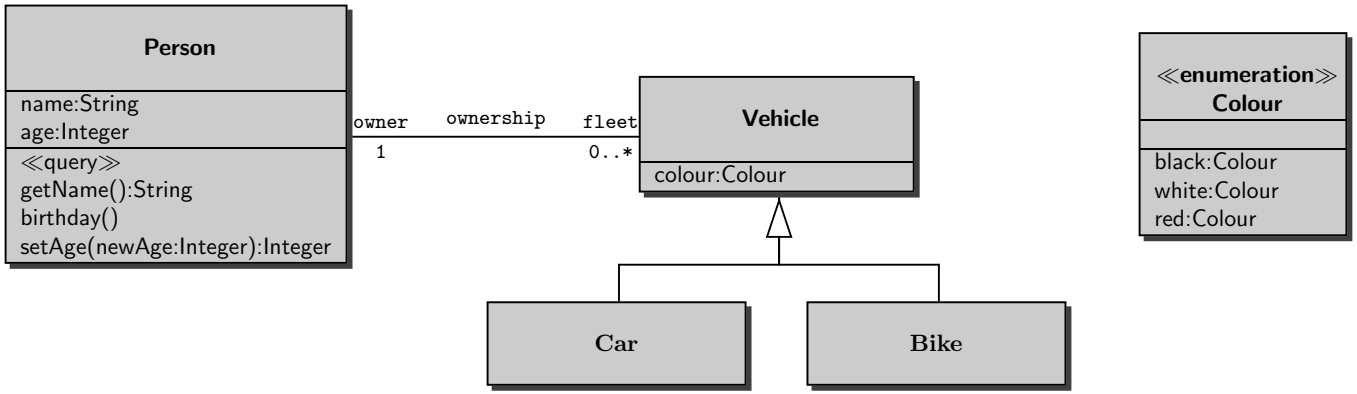


Context	Vordefinierte Variablen
context p:Person	p
context Person	self
context Person::getName():String	self, result
context Person::setAge(newAge: Integer)	self, newAge

Beispiel

context Person::getName():String
post: result = self.name

OCL-Ausdrücke: Vordefinierte Variablen



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

Beispiel

context Person::setAge(newAge: Integer)
post: self.age=newAge

OCL-Ausdrücke: Vordefinierte Operatoren

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
	Boolean	Erfüllt ein Element die Bedingung?
includes	Boolean	Ist das gesuchte Element in der collection?

Beispiel

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

OCL-Ausdrücke: Vordefinierte Operatoren

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
	Boolean	Erfüllt ein Element die Bedingung?
includes	Boolean	Ist das gesuchte Element in der collection?

Beispiel

context Person
inv: self.fleet–>select(v | v.colour = Colour.black)–>size() <= 3

OCL-Ausdrücke: Vordefinierte Operatoren

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
	Boolean	Erfüllt ein Element die Bedingung?
includes	Boolean	Ist das gesuchte Element in der collection?

Beispiel

context Person
inv: self.fleet–>collect(v | v.colour) –>asSet–>size() = 1
(asSet entfernt Duplikate)

OCL-Ausdrücke: Vordefinierte Operatoren

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
exists	Boolean	Erfüllt ein Element die Bedingung?
includes	Boolean	Ist das gesuchte Element in der collection?

Beispiel

context Person
inv: self.fleet–>exists(v | v.colour = Colour.black)

OCL-Ausdrücke: Vordefinierte Operatoren

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 a
forAll/exists	Boolean	Erfüllt jedes/ein Element die Bedingung?
includes	Boolean	Ist das gesuchte Element in der collectio

Beispiel

```
context Person
inv:    self.fleet->select(v | v.colour = Colour.black)
        ->forAll(b | b.oclIsKindOf(Bike))
```

OCL-Ausdrücke: Vordefinierte Operatoren

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 a
forAll/exists	Boolean	Erfüllt jedes/ein Element die Bedingung?
includes	Boolean	Ist das gesuchte Element in der collection

Beispiel

context Person
inv: self.fleet->includes(self.favourite)

Vereinfachende Schreibweisen

context Person **inv:** self.age $\geq$ 18	**context** c:Person **inv:** c.age $\geq$ 18	**context** Person **inv:** age $\geq$ 18

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

Die letzte Abkürzung ist nur erlaubt für 'collect'

Vereinfachende Schreibweisen

context Person	**context** c:Person	**context** Person
inv: self.age $\geq$ 18	**inv:** c.age $\geq$ 18	**inv:** age $\geq$ 18

context Person - - alle constraints sind äquivalent
inv: fleet –>collect(v:Vehicle | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(v | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(colour) –>asSet() –>size() = 1
inv: fleet.colour –>asSet() –>size() = 1
Die letzte Abkürzung ist nur erlaubt für 'collect'

Vereinfachende Schreibweisen

context	Person	**context**	c:Person	**context**	Person
inv:	self.age $\geq$ 18	**inv:**	c.age $\geq$ 18	**inv:**	age $\geq$ 18

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

Die letzte Abkürzung ist nur erlaubt für 'collect'

Vereinfachende Schreibweisen

context	Person	**context**	c:Person	**context**	Person
inv:	self.age $\geq$ 18	**inv:**	c.age $\geq$ 18	**inv:**	age $\geq$ 18

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

Die letzte Abkürzung ist nur erlaubt für 'collect'

Vereinfachende Schreibweisen

context	Person	**context**	c:Person	**context**	Person
inv:	self.age $\geq$ 18	**inv:**	c.age $\geq$ 18	**inv:**	age $\geq$ 18

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

Die letzte Abkürzung ist nur erlaubt für 'collect'

Vereinfachende Schreibweisen

context	Person	**context**	c:Person	**context**	Person
inv:	self.age $\geq$ 18	**inv:**	c.age $\geq$ 18	**inv:**	age $\geq$ 18

context Person - - alle constraints sind äquivalent
inv: fleet –>collect(v:Vehicle | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(v | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(colour) –>asSet() –>size() = 1
inv: fleet.colour –>asSet() –>size() = 1
Die letzte Abkürzung ist nur erlaubt für 'collect'

Vereinfachende Schreibweisen

context	Person	**context**	c:Person	**context**	Person
inv:	self.age $\geq$ 18	**inv:**	c.age $\geq$ 18	**inv:**	age $\geq$ 18

context Person - - alle constraints sind äquivalent
inv: fleet –>collect(v:Vehicle | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(v | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(colour) –>asSet() –>size() = 1
inv: fleet.colour –>asSet() –>size() = 1
Die letzte Abkürzung ist nur erlaubt für 'collect'

Vereinfachende Schreibweisen

context	Person	**context**	c:Person	**context**	Person
inv:	self.age $\geq$ 18	**inv:**	c.age $\geq$ 18	**inv:**	age $\geq$ 18

context Person - - alle constraints sind äquivalent
inv: fleet –>collect(v:Vehicle | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(v | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(colour) –>asSet() –>size() = 1
inv: fleet.colour –>asSet() –>size() = 1
Die letzte Abkürzung ist nur erlaubt für 'collect'

Vereinfachende Schreibweisen

context	Person	**context**	c:Person	**context**	Person
inv:	self.age $\geq$ 18	**inv:**	c.age $\geq$ 18	**inv:**	age $\geq$ 18

context Person - - alle constraints sind äquivalent
inv: fleet –>collect(v:Vehicle | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(v | v.colour) –>asSet() –>size() = 1
inv: fleet –>collect(colour) –>asSet() –>size() = 1
inv: fleet.colour –>asSet() –>size() = 1
Die letzte Abkürzung ist nur erlaubt für 'collect'

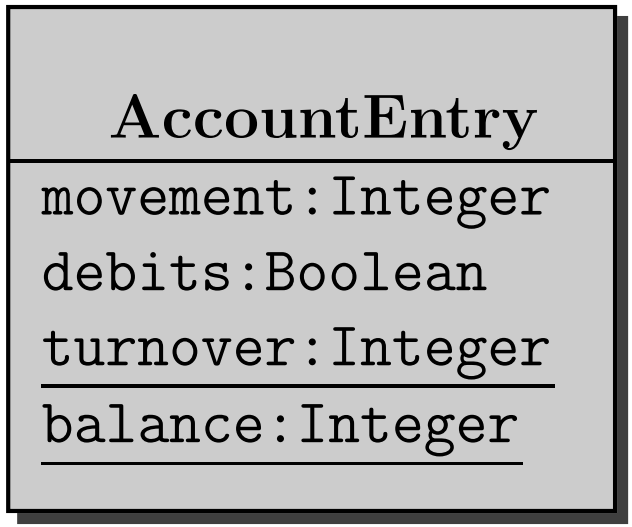
Vereinfachende Schreibweisen

context	Person	**context**	c:Person	**context**	Person
inv:	self.age $\geq$ 18	**inv:**	c.age $\geq$ 18	**inv:**	age $\geq$ 18

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

Die letzte Abkürzung ist nur erlaubt für 'collect'

allInstances und iterate



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

allInstances Syntax und Semantik

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

$\mathcal{I}(\text{C.allInstances})$ 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?

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

allInstances Syntax und Semantik

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

$\mathcal{I}(\text{C.allInstances})$ 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?

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

allInstances Syntax und Semantik

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

$\mathcal{I}(\text{C.allInstances})$ 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?

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

allInstances Syntax und Semantik

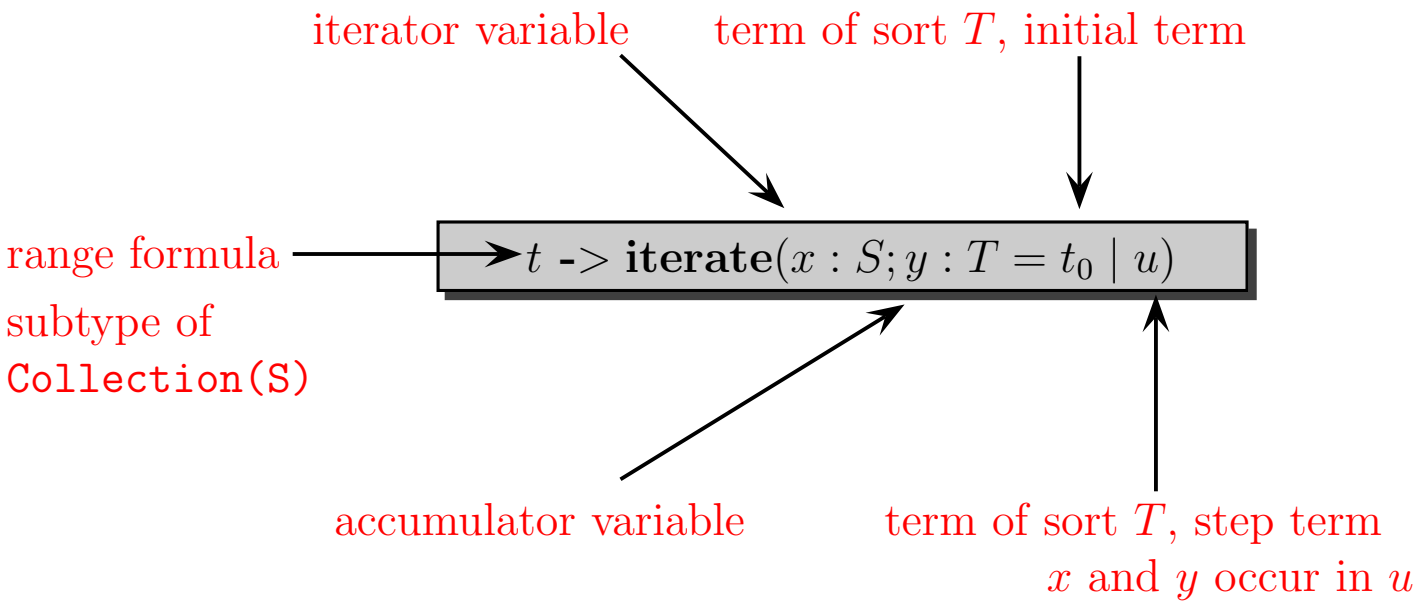
Falls C ein Typ-Symbol ist, dann ist C.allInstances ein gültiger OCL-Ausdruck

$\mathcal{I}(\text{C.allInstances})$ 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?

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

iterate Syntax



iterate Semantics

t –>iterate(x:S; y:T=t0 | 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

iterate Semantics

t –>iterate(x:S; y:T=t0 | 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

iterate Semantics

t –>iterate(x:S; y:T=t0 | 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

iterate Semantics

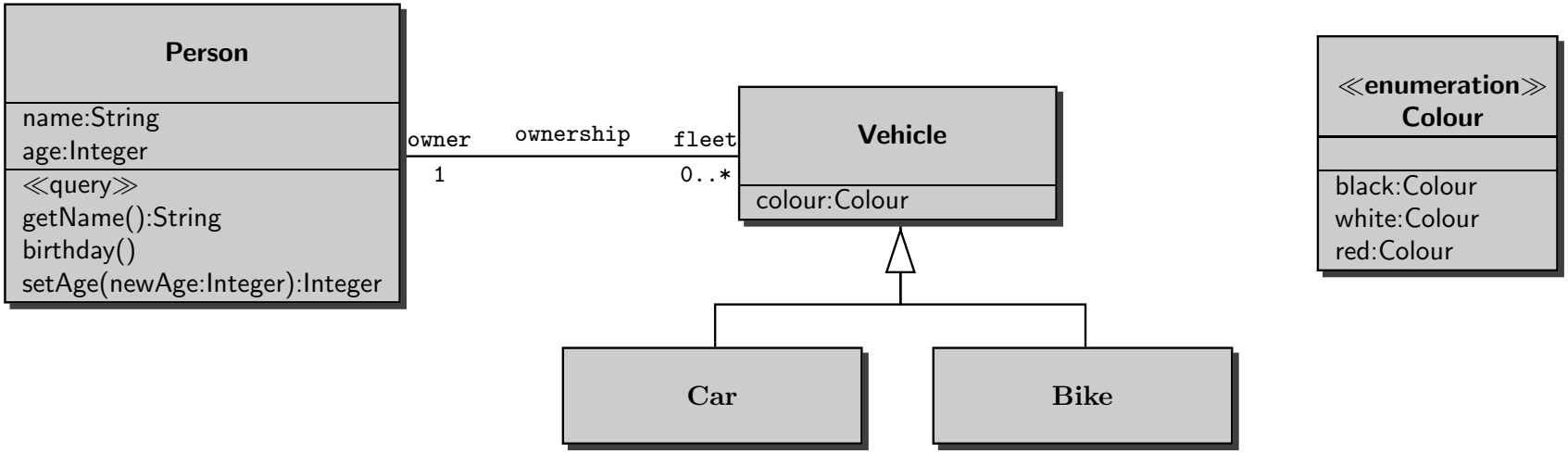
t –>iterate(x:S; y:T=t0 | u)

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 ≥ 0
post: age = age@pre + 1

Junktoren

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

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

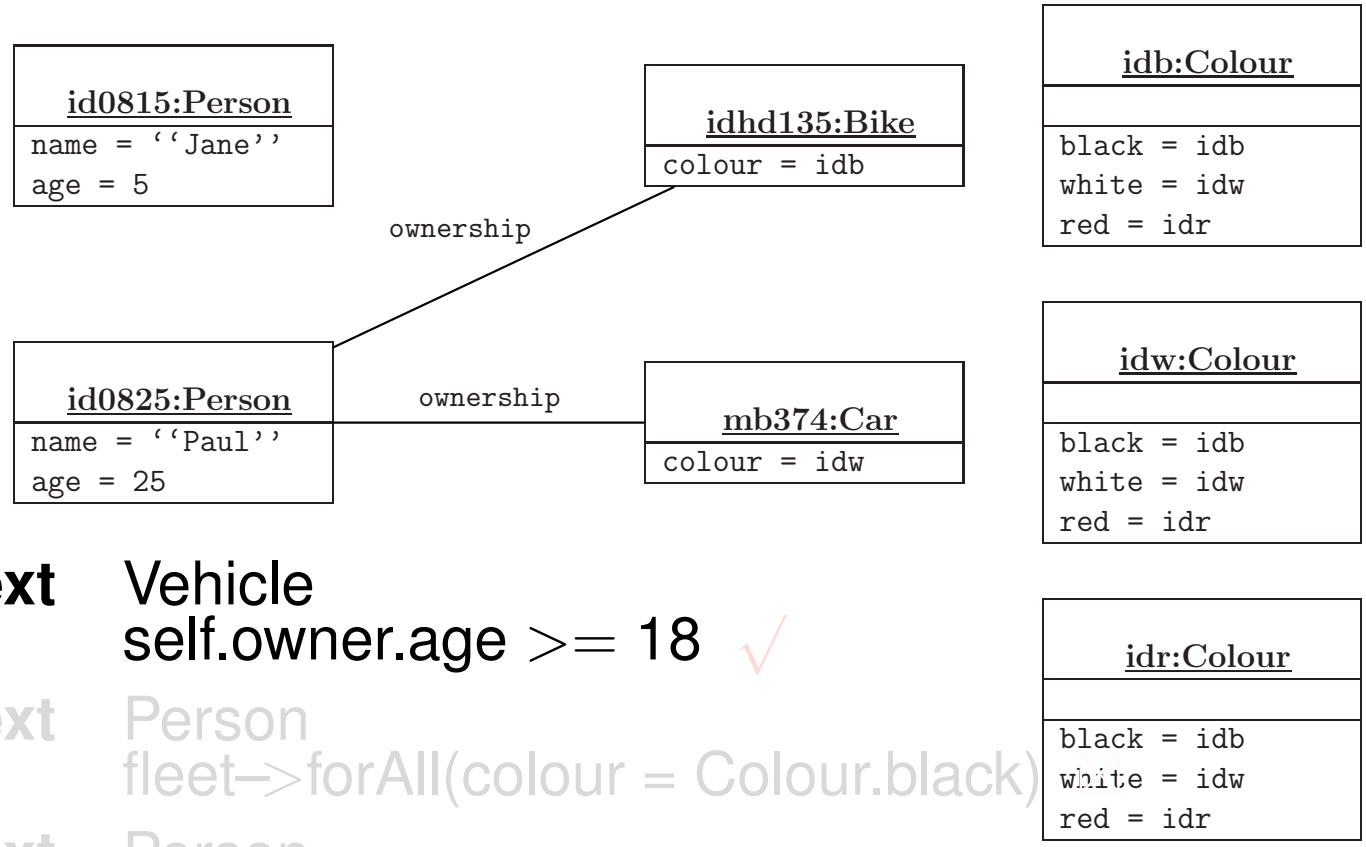
Erlauben die beabsichtigte Semantik eines UML-Diagramms auszudrücken

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

Snapshots und OCL-Constraints

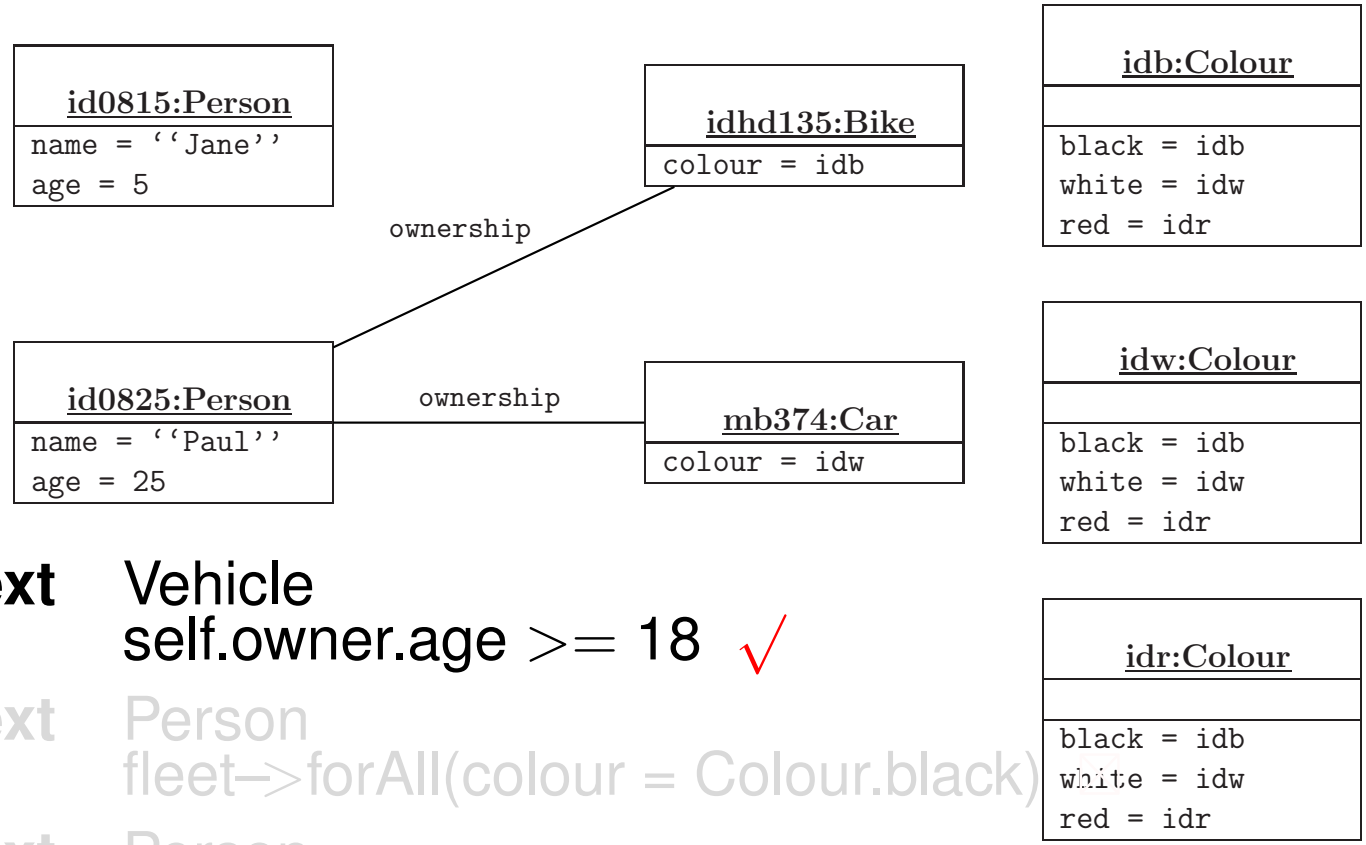


context Vehicle
inv: self.owner.age >= 18 √

context Person
inv: fleet->forAll(colour = Colour.black) ☒

context Person
inv: fleet->select(colour = Colour.black) ->size() <= 3 √
inv: Car.allInstances ->exists(colour = Colour.red) ☒

Snapshots und OCL-Constraints

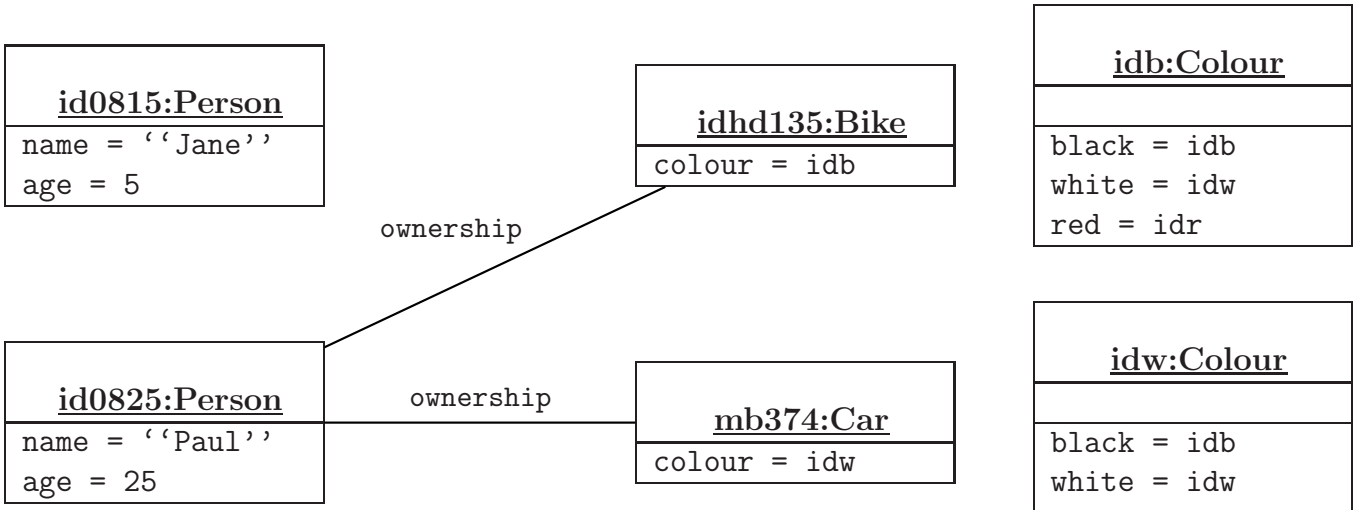


context Vehicle
inv: self.owner.age >= 18 √

context Person
inv: fleet->forAll(colour = Colour.black) ☒

context Person
inv: fleet->select(colour = Colour.black) ->size() <= 3 √
inv: Car.allInstances ->exists(colour = Colour.red) ☒

Snapshots und OCL-Constraints

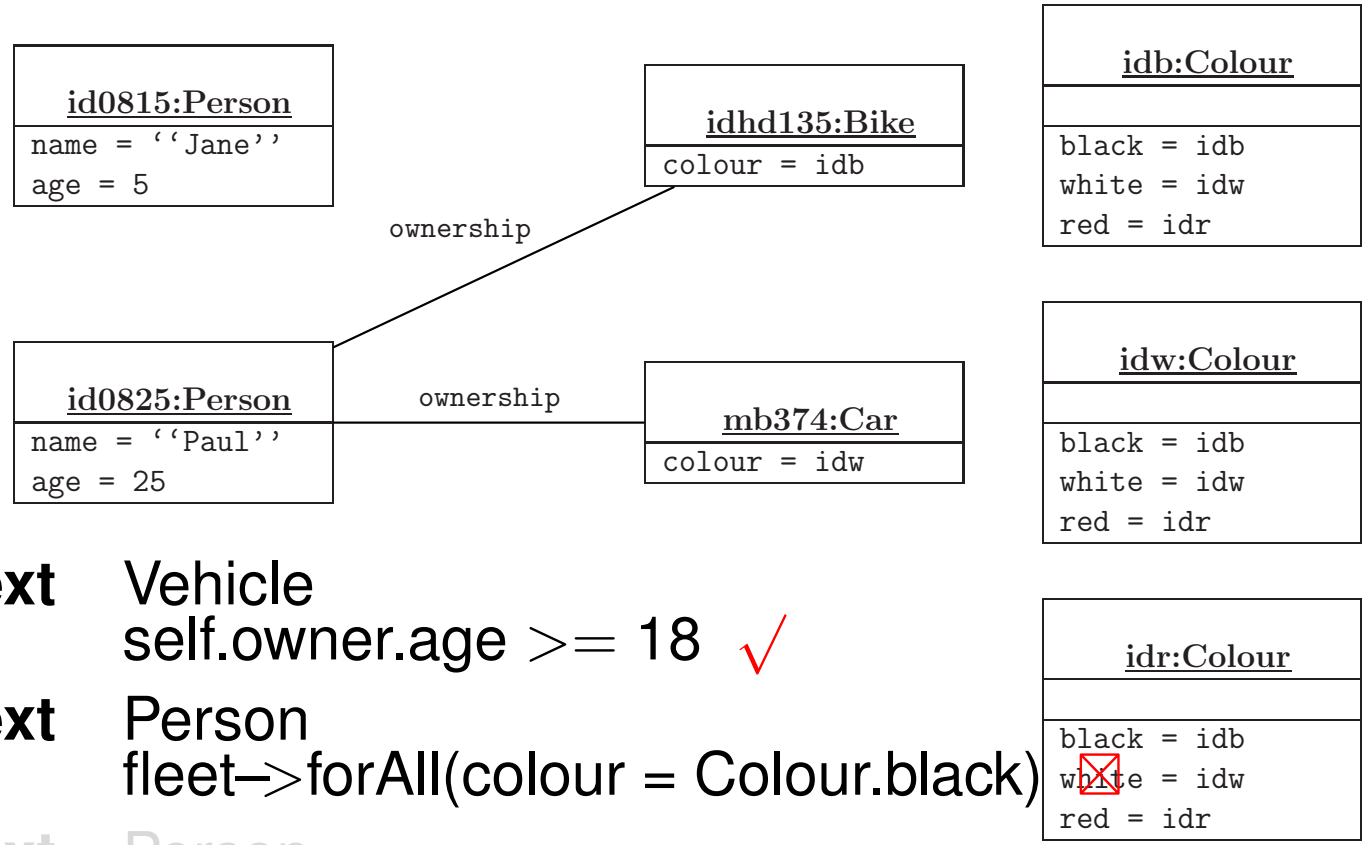


context Vehicle
inv: self.owner.age >= 18 √

context Person
inv: fleet->forAll(colour = Colour.black) ☒

context Person
inv: fleet->select(colour = Colour.black) ->size() <= 3 √
inv: Car.allInstances ->exists(colour = Colour.red) ☒

Snapshots und OCL-Constraints

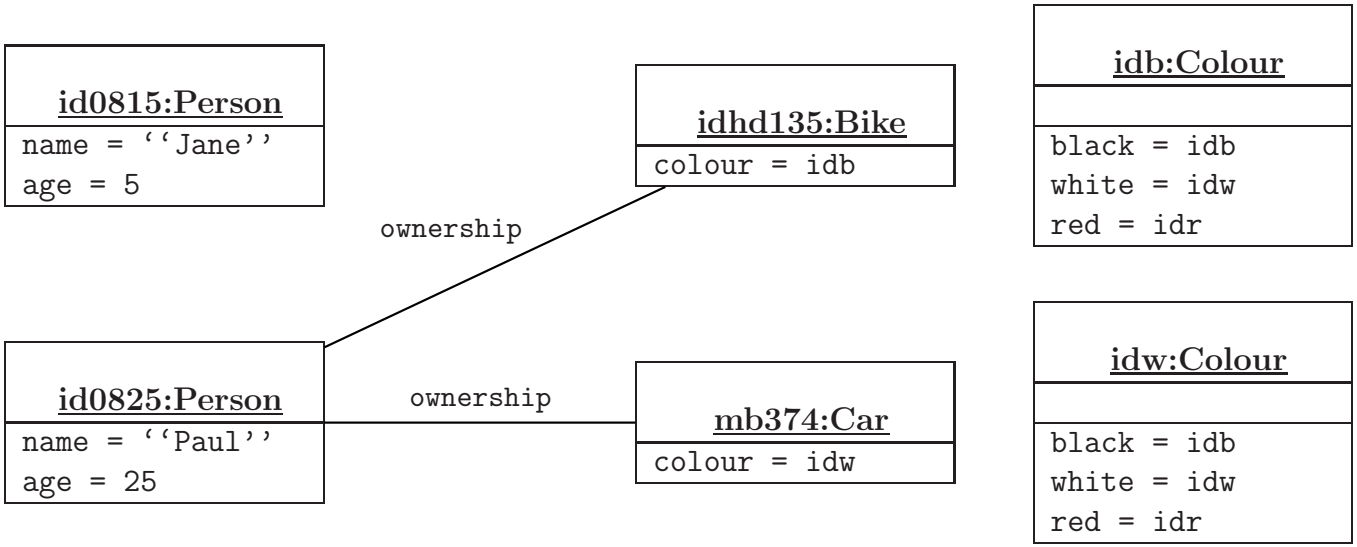


context Vehicle
inv: self.owner.age >= 18 √

context Person
inv: fleet->forAll(colour = Colour.black) ☒

context Person
inv: fleet->select(colour = Colour.black) ->size() <= 3 √
inv: Car.allInstances ->exists(colour = Colour.red) ☒

Snapshots und OCL-Constraints



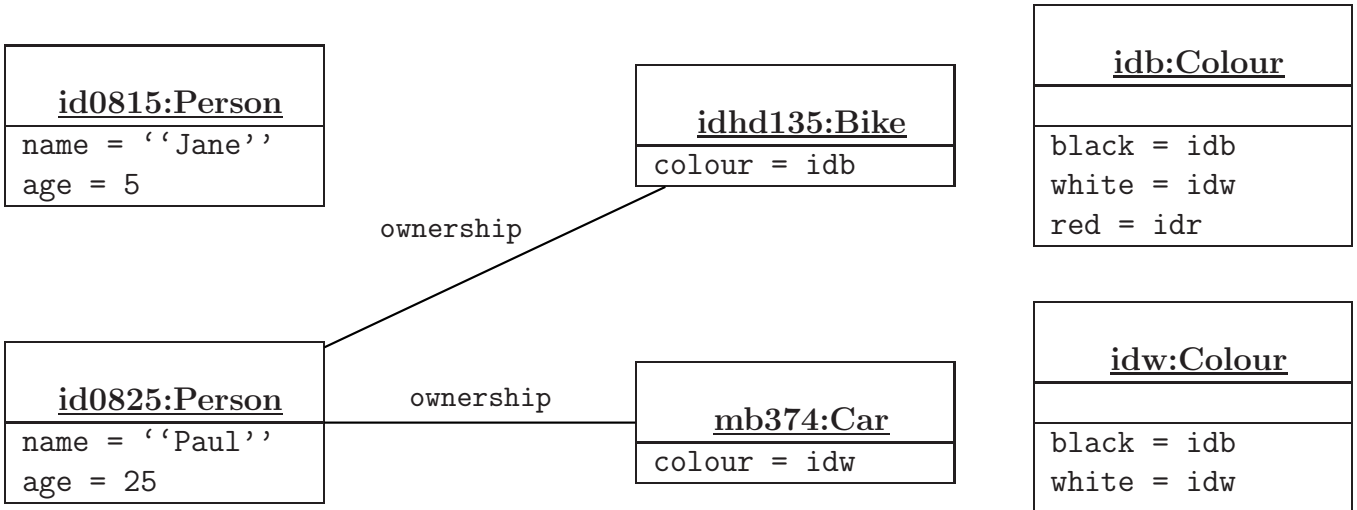
context Vehicle
inv: self.owner.age >= 18 √

context Person
inv: fleet–>forAll(colour = Colour.black)

context Person
inv: fleet–>select(colour = Colour.black) –>size() <= 3 √

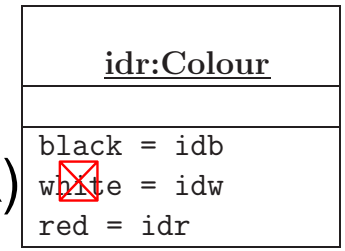
inv: Car.allInstances –>exists(colour = Colour.red) ☒

Snapshots und OCL-Constraints



context Vehicle
inv: self.owner.age >= 18 √

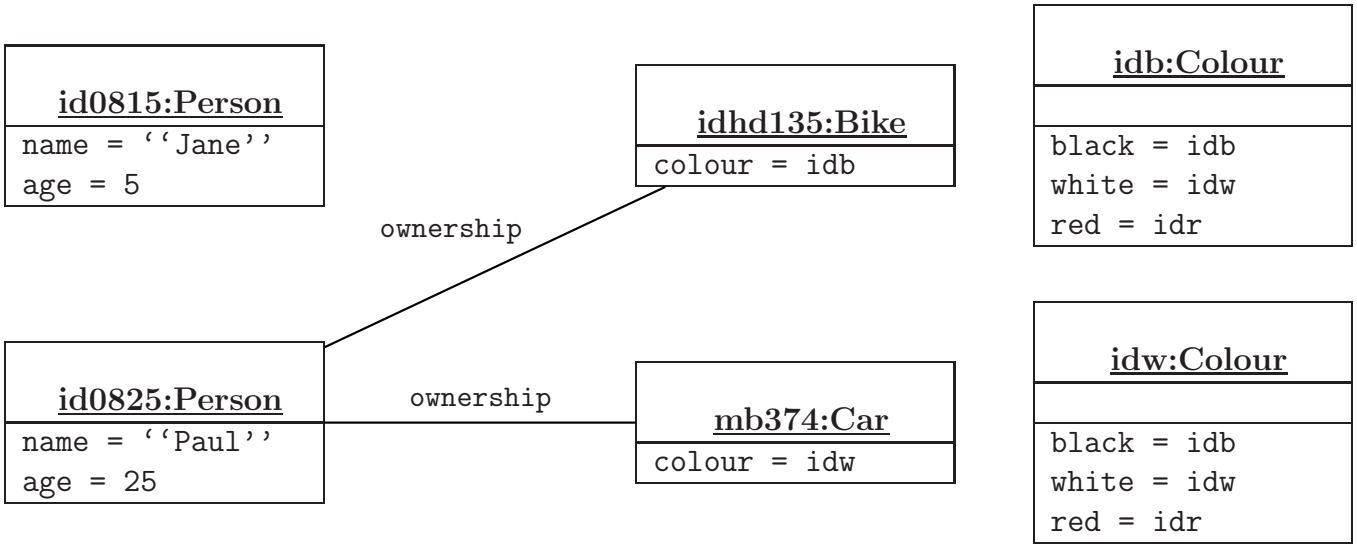
context Person
inv: fleet->forAll(colour = Colour.black)



context Person
inv: fleet->select(colour = Colour.black) ->size() <= 3 √

inv: Car.allInstances ->exists(colour = Colour.red) ☒

Snapshots und OCL-Constraints

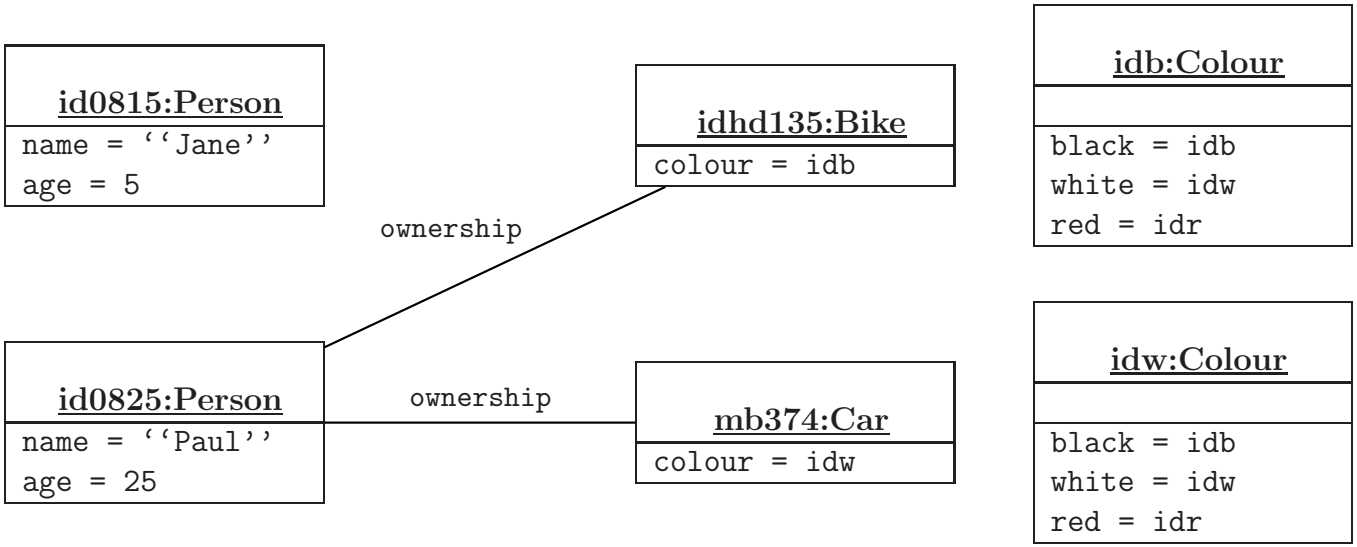


context Vehicle
inv: self.owner.age >= 18 √

context Person
inv: fleet->forAll(colour = Colour.black) ☒

context Person
inv: fleet->select(colour = Colour.black) ->size() <= 3 √
inv: Car.allInstances ->exists(colour = Colour.red) ☒

Snapshots und OCL-Constraints

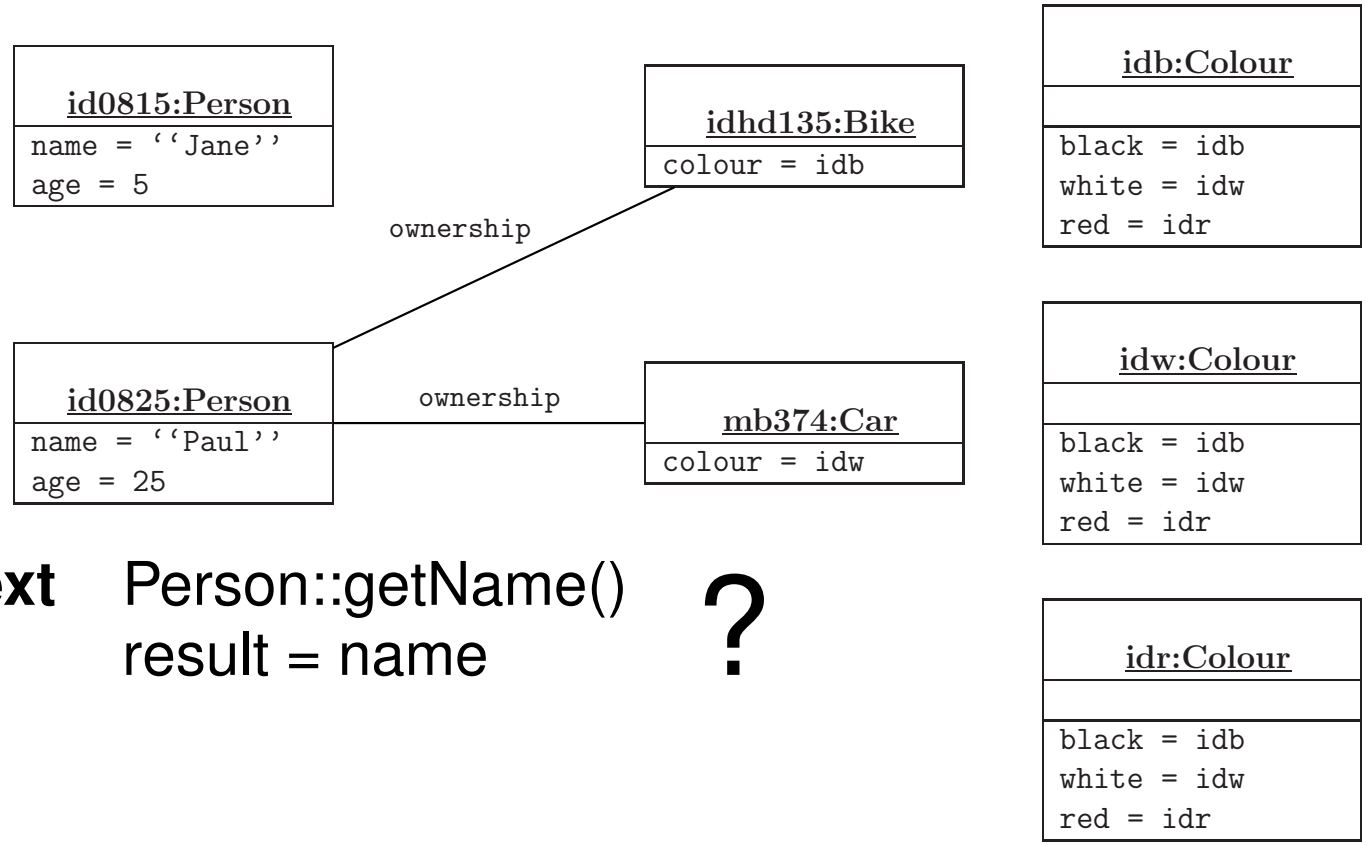


context Vehicle
inv: self.owner.age >= 18 √

context Person
inv: fleet->forAll(colour = Colour.black) ☒

context Person
inv: fleet->select(colour = Colour.black) ->size() <= 3 √
inv: Car.allInstances ->exists(colour = Colour.red) ☒

Snapshots und OCL-Constraints



context Person::getName()
post: result = name ?

Vergleich

OCL	PK1
semantische Unklarheiten	klare Semantik

Vergleich

OCL	PK1
semantische Unklarheiten	klare Semantik
Notation muß gelernt werden (leichter?!)	Notation muß gelernt werden

Vergleich

OCL	PK1
semantische Unklarheiten	klare Semantik
Notation muß gelernt werden (leichter?!)	Notation muß gelernt werden
Ausdrücke eher länger	Ausdrücke eher kürzer

Vergleich

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

Vergleich

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

Vergleich

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