ModSoft

Modellbasierte Software-Entwicklung mit UML 2
im WS 2014/15

Teil IV: Object Constraint Language

Prof. Dr. Joachim Fischer
Dr. Markus Scheidgen
Dipl.-Inf. Andreas Blunk

fischer@informatik.hu-berlin.de

ModSoft (UML): Inhalt

Teil I- Einführung

Teil II-Struktur

a)

- Klassen, Assoziationen, Abhängigkeiten
- Interface, Datentypen, Signale, Port,
- Strukturierter Classifier
- Aktive Klassen

b)

- Präzisierungen
- Klassendiagramm, vertiefende Betrachtung
- Operationen

Teil III-Verhalten

a)

- Einfacher Zustandsautomat
- Beispiel: DemonGame
- UML-Modellierungsdefizite

b)

- Aktivitäten

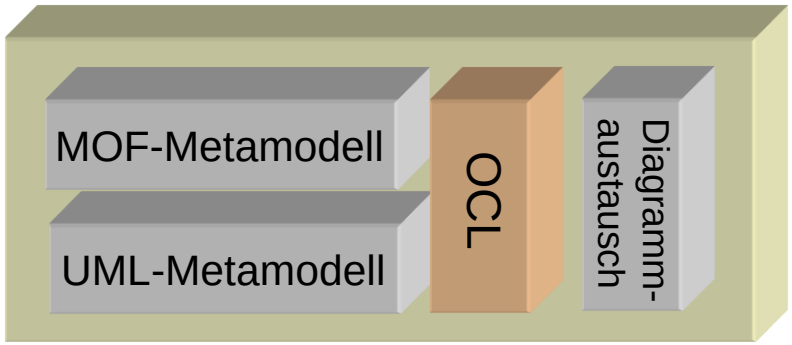
Teil IV- OCL

Inhalt: OCL

~ OCL 2.4
(Feb 2014)

1. Allgemeine Charakterisierung
2. Typen, Metatypen und Werte
3. Typkonformität
4. Standardtypen und Operationen (Überblick)
5. Mächtigkeit von OCL-Ausdrücken
6. OCL-Ausdrucksbildung (anhand von Beispielen)

Einordnung von OCL



Wozu?

- Empfehlung der OMG zur klareren Formulierung von
  1. Randbedingungen/Einschränkungen
  2. Anfragen
  3. Navigationen

  in UML-Modellen

  } in Form von Ausdrucksannotationen

  prinzipiell ist aber jede andere Sprache als Alternative zulässig (inkl. natürliche Sprachen)

- Normative Verwendung
  bei der Festlegung von Regeln zur Wohlgeformheit der MOF- und UML-Metamodelle

OCL ist …

- … ist **keine Programmiersprache**
  - man kann **keinen** Kontrolffluss beschreiben, **keine** Prozesse aktivieren oder andere Seiteneffekte auslösen
- … ist in erster Linie eine **Modellierungssprache**
  - Ausdrücke sind also nicht zwingendend per *definitionem* ausführbar
- … ist eine **formale Sprache**
  - dennoch einfach zu lesen und zu schreiben
  - basiert auf **Prädikatenlogik** (1.Stufe) und **Mengentheorie** in einer (weitgehend) intuitiven Syntax
- … ist eine **textuelle** Sprache

- … ist eine **getypte Sprache**
  - jeder OCL-Ausdrück hat einen Typ
  - jeder in **UML definierte Classifier** repräsentiert einen eigenen unterscheidbaren OCL-Typ
  - darüber hinaus stellt OCL über eine Bibliothek **Standardtypen** (mit Operationen) bereit

OCL als funktionale Spezifikationssprache

- … wurde ursprünglich als eigene Technik entwickelt
  - inzwischen Teil des OMG-Standards für UML
- … erlaubt die Formulierung von Ausdrücken in Gestalt von
  - Prädikaten,
  - logischen Bedingungen/Vergleichen,
    die einzeln gebildet und ausgewertet werden
    und nicht als Gesamtprogramm
    (auch dann nicht, wenn sie nur für ein Diagramm gelten sollen)
- … läßt keine Seiteneffekte zu
  - damit sind ausschließlich nur **Query**-Operationen für UML-Elemente erlaubt
- … dient der Bildung von Ausdrücken über Operanden von
  - OCL- als auch
  - UML-Typen

Ausdrucksberechnung

- Operanden
  - Werte von Container- und Standard-Typen
  - Objekte eines Typs
- Ausführungsreihenfolge
  - von links nach rechts
- Ergebnis
  - Ist nach ausgeführter Berechnung ein Wert oder Objekt eines Typs
- Bestimmung von Nachfolgern des resultierenden Objektes mittels
  - Attribut oder Member-Funktion oder
  - Navigation über bestehende Assoziationen

Inhalt: OCL

~ OCL 2.4
(Feb 2014)

1. Allgemeine Charakterisierung
2. Typen, Metatypen und Werte
3. Typkonformität
4. Standardtypen und Operationen (Überblick)
5. Mächtigkeit von OCL-Ausdrücken
6. OCL-Ausdrucksbildung (anhand von Beispielen)

OCL-Typen: Übersicht

~ OCL 2.4
(Feb 2014)

- OCL ist getypt (aber nicht objektorientiert)

  1. Standardtypen:
     - ~~OclType~~, OclVoid, OclInvalid, OclAny
     - OclMessage
     - UnlimitNatural, Integer, Real,
     - Boolean, String
     - Aufzählungstypen
     - Verbünde/Strukturen (Tupel)
     - Kollektortypen

  2. UML-Nutzer-Typ:
     - jeder UML-Typ (Datentyp, Klass
       spiegelt sich als OCL-Typ wider

OCL-Standardbibliothek

predefined

Primitive Typen

Basistypen

Problem:
wie wird ein UML-Nutzertyp zum OCL-Typ?

- die Typen bilden in OCL eine Hierarchie

  ~ Typkonformitä

Konformität der Basistypen von UML und OCL sichert MOF

OCL-Typen: Kollektortypen

- *Collection*

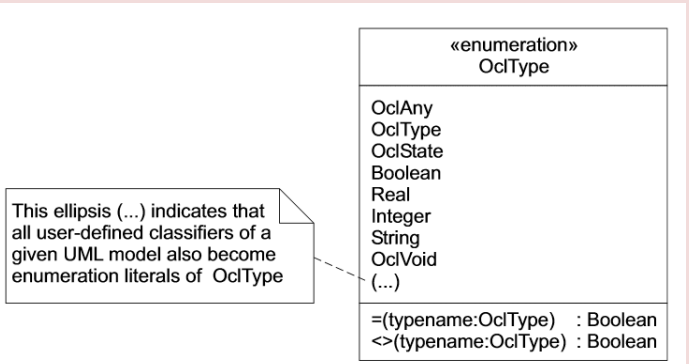
als abstrakter Typ
mit konkreten Untertypen

- Set
  - Menge von Elementen
  - keine Duplikate
- OrderedSet
  - geordnete Menge von Elementen
  - keine Duplikate
- Bag
  - Menge von Elementen
  - kann Duplikate enthalten
- Sequence
  - eine Menge von Elementen
  - kann Duplikate enthalten
  - ist geordnet, aber nicht sortiert

OCL-Typen: OclType

ursprünglich vorhanden, ab Version 2.4 nicht mehr

- spielte eine Sonderrolle unter den Typen von OCL
- er deckte **magisch** alle Typen des OCL-Typ-Systems ab.



- wurde für universellen Typvergleich benötigt
  - einerseits war **OclType** ein MetaElement (M2-Ebene)
  - andererseits wurde der Typ als Parameter von Konformitätsoperatoren benötigt (also direkt nutzbar auf M1-Ebene)

konzeptionelle "Unsauberkeit"

aus dem aktuellen OCL-Standard

An asType expression (v) can be used in cases where static type information is insufficient. It corresponds to the `oclAsType` operation in OCL and can be understood as a cast of a source expression to an equivalent expression of a (usually) more specific target type. The target type must be related to the source type, that is, one must be a subtype of the other. The isTypeOf and isKindOf expressions correspond to the `oclIsTypeOf` and `oclIsKindOf` operations, respectively. An expression (*e* isTypeOf *t'*) can be used to test whether the type of the value resulting from the expression *e* has the type *t'* given as argument. An isKindOf expression (*e* isKindOf *t'*) is not as strict in that it is sufficient for the expression to become true if *t'* is a supertype of the type of the value of *e*. Note that in previous OCL versions these type casts and tests were defined as operations with parameters of type *OclType*. Here, we technically define them as first class expressions, which has the benefit that we do not need the metatype *OclType*. Thus the type system is kept simple while preserving compatibility with standard OCL syntax.

Achtung: OCL-Lehrbücher gehen noch von Existenz von **OclType** aus

OCL-Typen: OclAny

Alle Typen eines UML-OCL-Modells sind **konform** zum Typ OclAny.

Konzeptuell verhält sich OclAny wie ein **Supertyp** für all diese Typen.

Operationen von OclAny sind demnach für jedes Objekt in OCL-Ausdrücken anwendbar.

OclAny selbst ist eine Instanz der Metaklasse **AnyType**.

Table 7.1 - - Basic OCL types and their values

type	values	co
OclInvalid	invalid	
OclVoid	null, invalid	
Boolean	true, false	(M
Integer	1, -5, 2, 34, 26524, ...	(M
Real	1.5, 3.14, ...	htt
String	'To be or not to be...'	(M
UnlimitedNatural	0, 1, 2, 42, ..., *	htt

primitive OCL-Typen

Set
Sequence
Bag
OrderedSet

Tupel
Klassen

Signale, Zustände
Ports, …

OclMessage

Ausnahmen

Standard ist dazu nicht ganz verständlich

Formale Semantik sagt:

OCLAny ist nicht konform zu Collector-Typen und Tupel

OCL-Typen: OclVoid, OclInvalid

OclVoid

- Instanz von Metaklasse **VoidType**
- ist konform zu allen Typen, bis auf OclInvalid
- hat einen einzigen Wert null (entspricht dem UML Null-Literalwert)
- jede Property-Ruf angewendet auf null resultiert in OclInvalid
  einzige Ausnahme: oclIsUndefined() resultiert in True

OclInvalid

- Instanz von Metaklasse **InvalidType**
- ist konform zu allen Typen(!)
- hat einen einzigen Wert invalid
- jede Property-Ruf angewendet auf invalid resultiert in OclInvalid
  einzige Ausnahmen:
  oclIsUndefined() → True
  oclIsInvalid() → True

Table 7.1 - - Basic OCL types and their values

type	values	co
OclInvalid	invalid	
OclVoid	null, invalid	
Boolean	true, false	(M
Integer	1, -5, 2, 34, 26524, ...	(M
Real	1.5, 3.14, ...	htt
String	'To be or not to be...'	(M
UnlimitedNatural	0, 1, 2, 42, ..., *	htt

primitive OCL-Typen

Tupel
Klassen

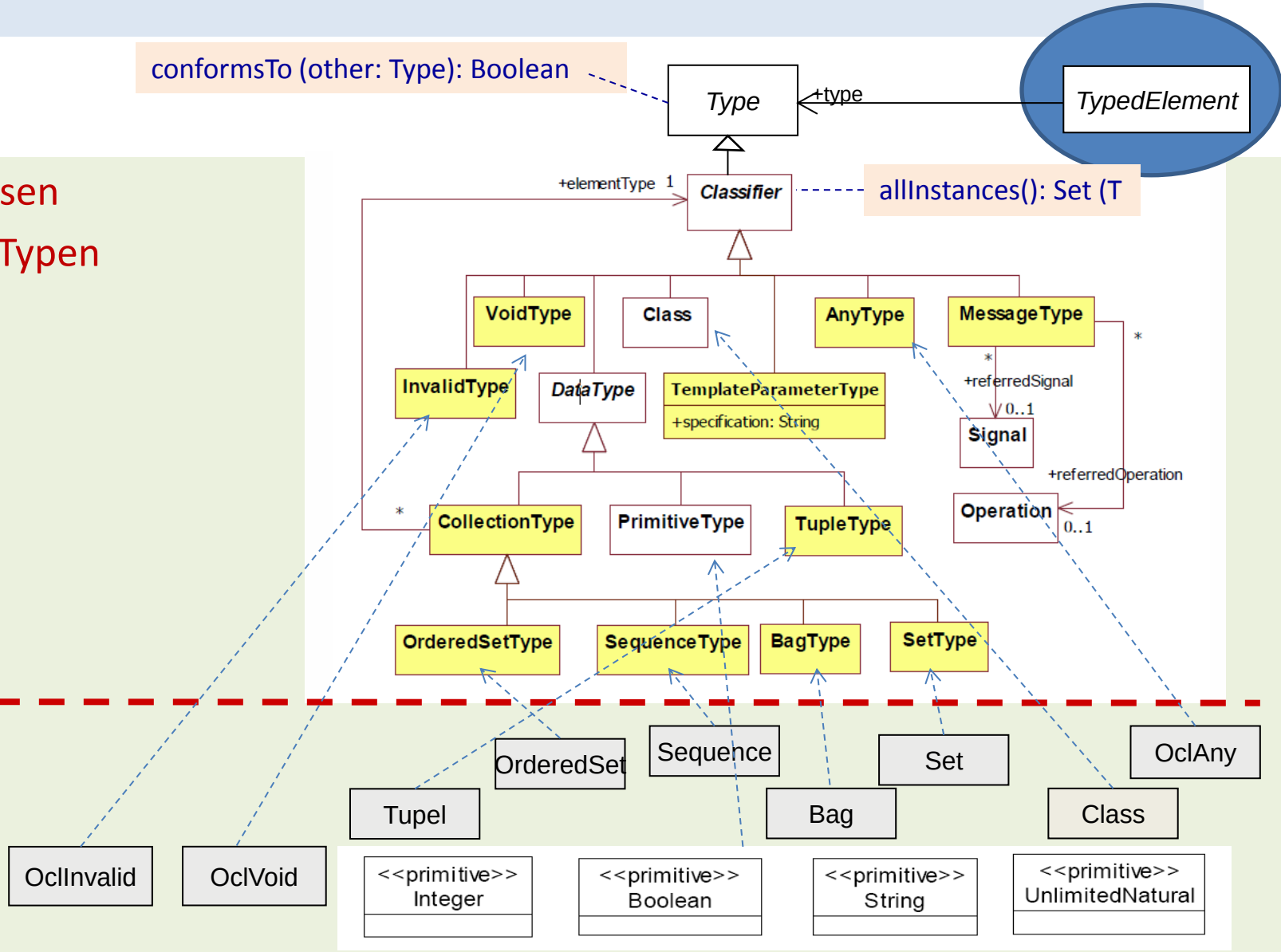
Set
Sequence
Bag
OrderedSet

Signale, Zustände
Ports, …

OclMessage

Metamodell

Metaklassen
von OCL-Typen



Allgemeine Objekt-Operationen

- allInstances(): Set (T) /* T ist Typ von Self */

alle Classifier

- = (OclAny): Boolean
- <> (OclAny): Boolean

auch Signaturen für Collection-SubTypen

- oclAsSet() : Set(T)
- oclIsNew (): Boolean /* Objekterzeugung während einer Op-Ausführung */
  /* als Differenz von Pre- und Po
- oclIsUndefined (): Boolean /* Test auf null */
- oclIsInvalid (): Boolean
- oclAsType (t : Classifier): instance of Classifier /* cast */
- oclIsTypeOf (t : Classifier): Boolean
- oclIsKindOf (t : Classifier): Boolean /* transitive Hülle*/
- oclType() : Classifier
- oclIsInState (s: OclState): Boolean

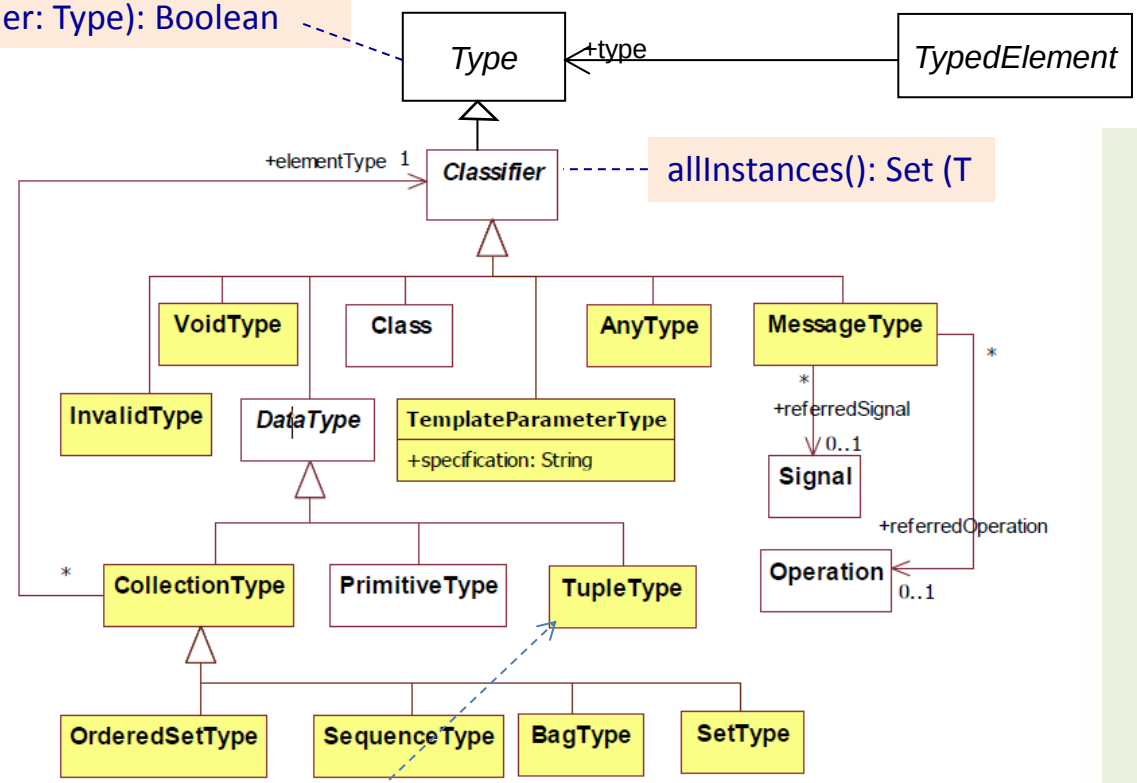
Tools:
anwendbar auf Operanden deren Typ nicht konform zu OclAny ist. ???

Konfliktvermeidung
- alle ererbten Eigenschaften von OclAny beginnen mit Präfix ocl !!!

Tupel

conformsTo (other: Type): Boolean

Metaklassen
von OCL-Typen



M2

M1

Tuple

- **Besonderheit**:
  Instanzen sind UML-Klassen
  ohne Operationen!
- nach Konformitätsregel kein Subtyp von OclAny

Tupel

- Werte können als Tupel-Strukturen komponiert werden

  Beispiele

  Tuple { name: String= 'John', age: Integer= 10 }

  Tuple { name = 'John', age = 10 }

  Tuple {age = 10, name = 'John' }

  Tuple { a: Collection(Integer)= Set{1,2,3},
  b: String= 'foo',
  c: String= 'bar' }

Wertnotationen

Set ist Subtyp von Collection

- Definition von Tupel-Typen

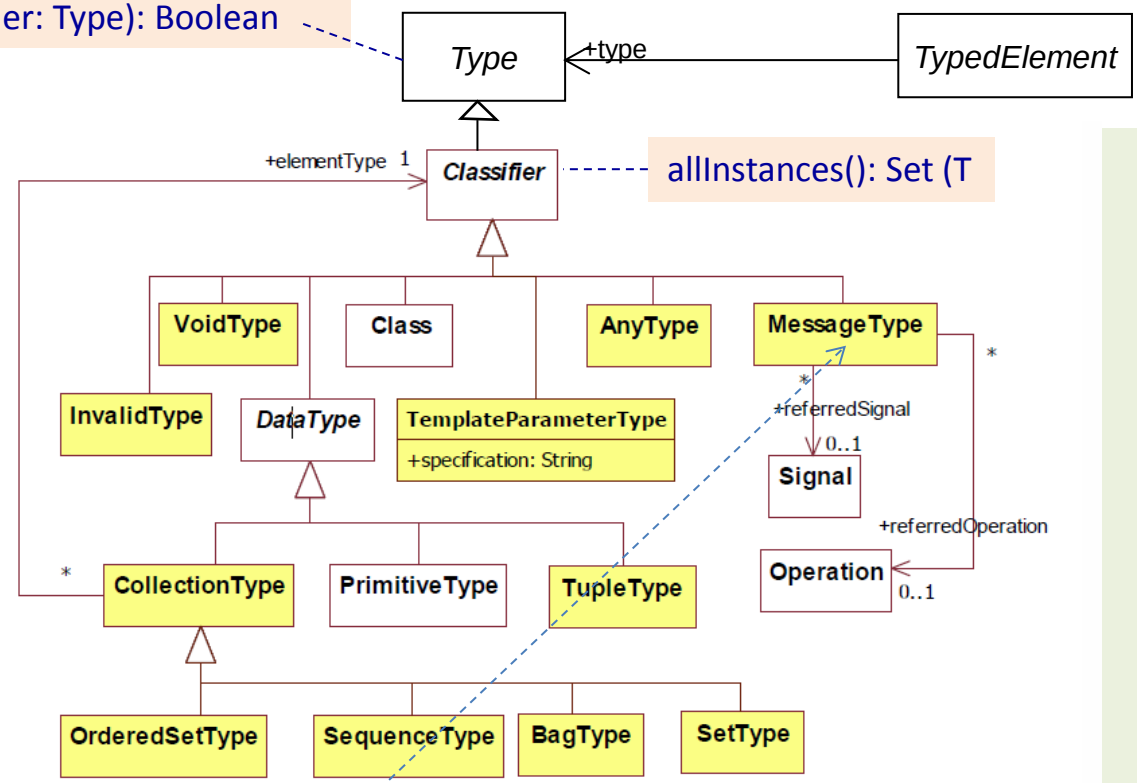
  TupleType (name: String, age: Integer)

  Tuple (a: Collection(Integer), b: String, c: String)

OclMessage

conformsTo (other: Type): Boolean

Metaklassen
von OCL-Typen



M2

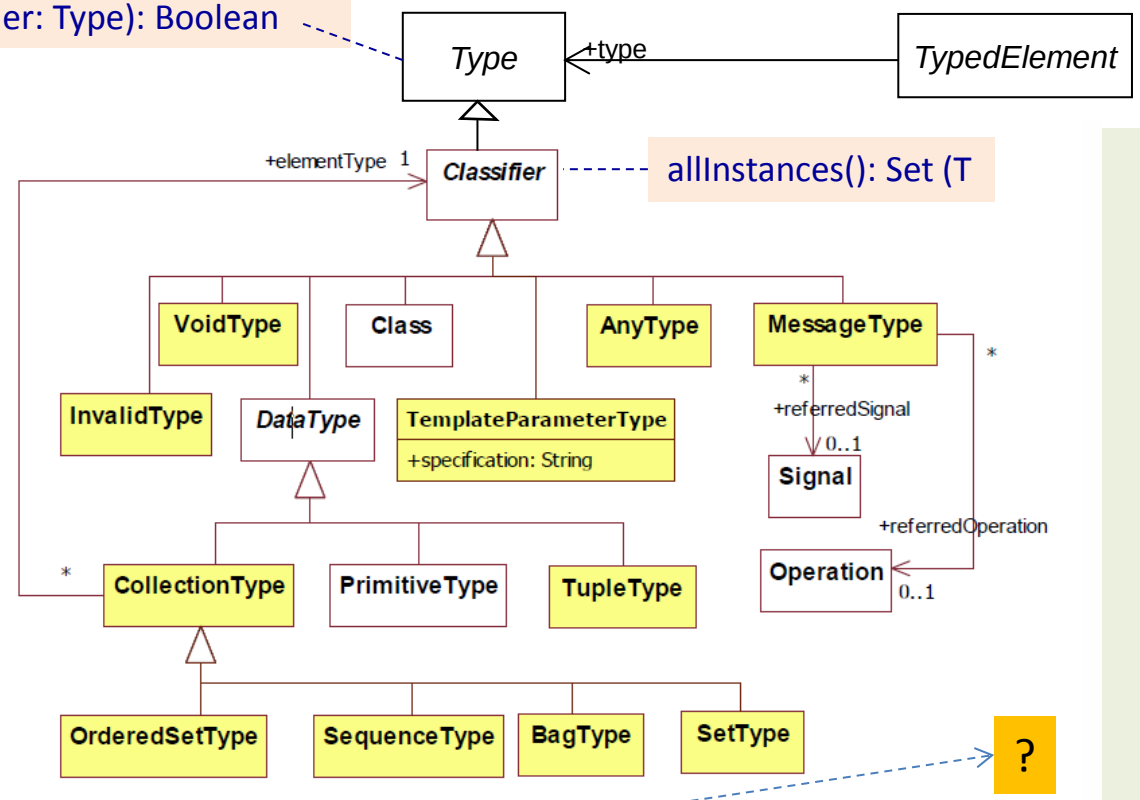
M1

OclMessage:
T substitiert durch
entweder eine Operation oder ein Signal

folg. Operationen sind verfügbar:
hasReturned (): Boolean
result (): << Typ der gerufenen Operation>>
isSignalSent(): Boolean
isOperationCall): Boolean

OclState

conformsTo (other: Type): Boolean



Metaklassen von OCL-Typen

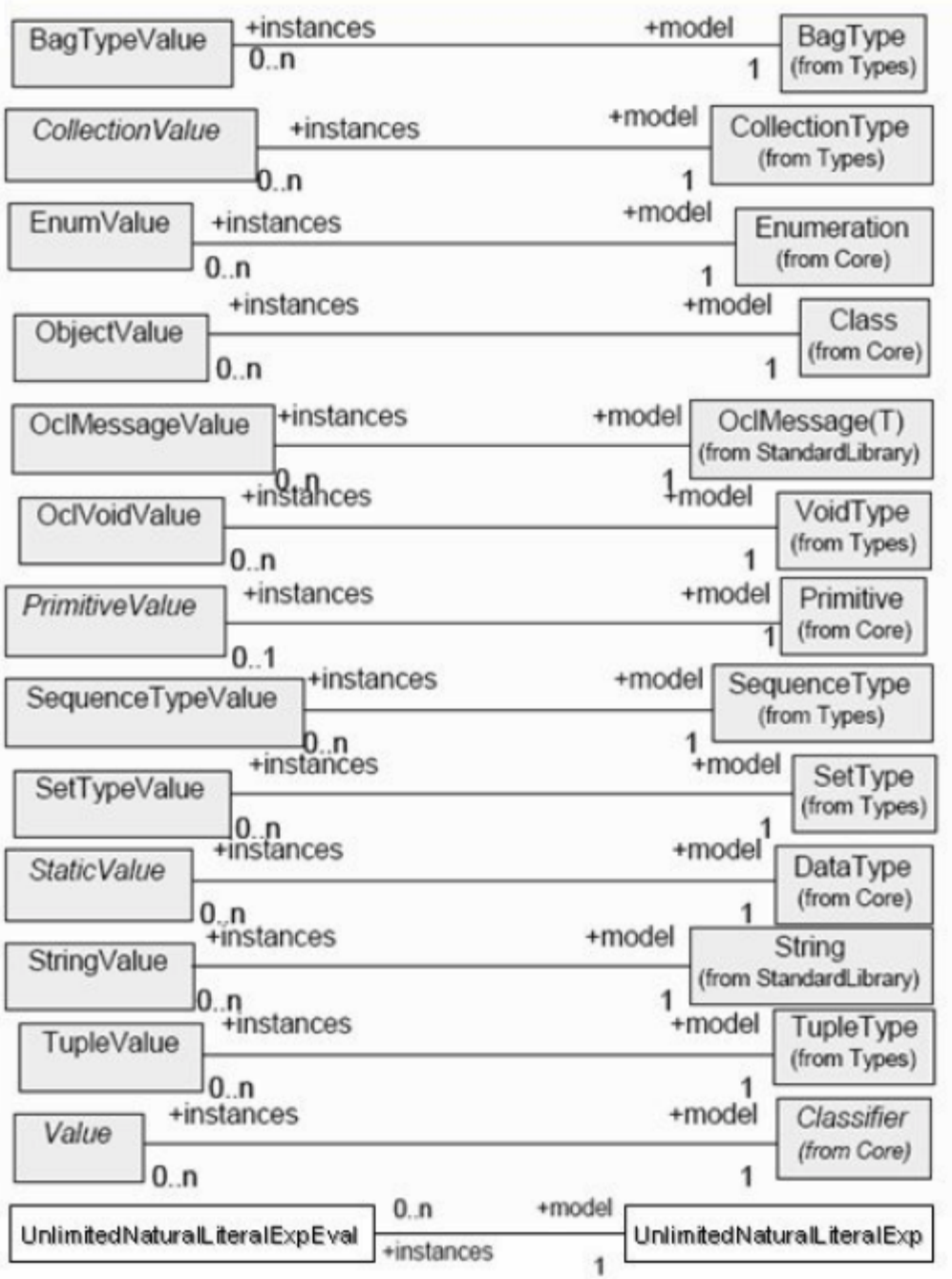
M2

M1

OclState:

- *OclState* is a type very similar to an enumeration type. It is only used in the operation oclIsInState for referring to state names in a state machine. There are no operations defined on this type. *OclState* is therefore not treated specially.

oclIsInState(statespec : OclState) : Boolean


BagTypeValue
+instances
0..n
+model
1
BagType
(from Types)
CollectionValue
+instances
0..n
+model
1
CollectionType
(from Types)
EnumValue
+instances
0..n
+model
1
Enumeration
(from Core)
ObjectValue
+instances
0..n
+model
1
Class
(from Core)
OclMessageValue
+instances
0..n
+model
1
OclMessage(T)
(from StandardLibrary)
OclVoidValue
+instances
0..n
+model
1
VoidType
(from Types)
PrimitiveValue
+instances
0..1
+model
1
Primitive
(from Core)
SequenceTypeValue
+instances
0..n
+model
1
SequenceType
(from Types)
SetTypeValue
+instances
0..n
+model
1
SetType
(from Types)
StaticValue
+instances
0..n
+model
1
DataType
(from Core)
StringValue
+instances
0..n
+model
1
String
(from StandardLibrary)
TupleValue
+instances
0..n
+model
1
TupleType
(from Types)
Value
+instances
0..n
+model
1
Classifier
(from Core)
UnlimitedNaturalLiteralExpEval
0..n
+instances
+model
1
UnlimitedNaturalLiteralExp

Inhalt: OCL

~ OCL 2.4
(Feb 2014)

1. Allgemeine Charakterisierung
2. Typen, Metatypen und Werte
3. Typkonformität
4. Standardtypen und Operationen (Überblick)
5. Mächtigkeit von OCL-Ausdrücken
6. OCL-Ausdrucksbildung (anhand von Beispielen)

Typ-Konformität

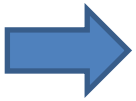
Wird von OCL-Parsern getestet

- Typ A ist konform zu B, falls eine Instanz von A alle Stellen des Auftretens von B-Instanzen im Ausdruck ersetzen kann
- Typ A ist konform zum Typ B, wenn A=B ist
- Typ A ist konform zu B, wenn A ein Untertyp von B ist

Aus der **Typkonformitätseigenschaft** ergibt sich, dass
- eine Operation für ein Objekt eines Basistyps
- auch für ein Objekt des Untertyps

angewendet werden kann

Regeln im OCL-Standard



OCL-Typhierarchie

Regeln (laut OCL-Standard)

1. UnlimitedNatural ist ein Untertyp von Integer
2. Integer ist Untertyp von Real
3. alle Typen (bis auf Tupel und Collection) sind Untertypen von OclAny
4. Set(T), Sequence(T), Bag(T) sind Untertypen von Collection(T)
5. OclVoid ist Untertyp von allen Typen (außer von OclVoid, OclInvalid)
6. OclInvalid ist Untertyp von allen Typen
7. die Typ-Hierarchie spiegelt die Generalisierungshierarchie von UML wider (Typrelationen als Teilmengenrelationen)

Typkonformität ist reflexiv und transitiv

Beispiel: Integer konform zu Real

```
context PrimitiveType
inv: (self.name = 'Integer') implies
    PrimitiveType.allInstances()->forAll (p | (p.name = 'Real') implies
                                                (self.conformsTo(p)))
```

In dieser Art werden im Standard Untertypen definiert

Typ-Konformität (ergänzend)

- Collection(A) ist konform zu Collection(B), falls A konform zu B
- Set(T) ist nicht konform zu OrderedSet(T), Bag(T) und Sequence(T)
- OrderedSet(T) ist nicht konform zu Set(T), Bag(T) und Sequence(T)
- Bag(T) ist nicht konform zu Set(T), OrderedSet(T) und Sequence(T)
- Sequence(T) ist nicht konform zu Set(T), OrderedSet(T) und Bag(T)

Cast mittels OclAsType

sei B Untertyp von A (hier Spezialisierung, A und B sind Klassen)

und

sei eine Objekt(referenz) vom Typ A gegeben

(von der man sicher ist, dass sich dahinter ein Objekt vom Typ B verbirgt)

- Cast:

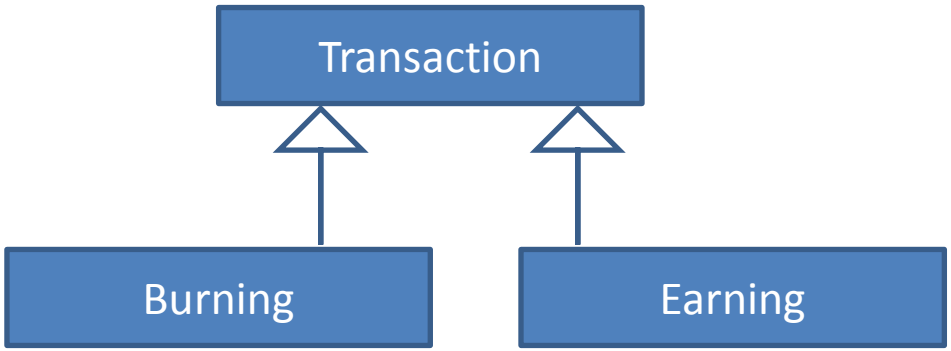
  Object.oclAsType(B)

- Wenn dies nicht gilt -> Resultat= undefined~ null (OclVoid)

Wurden Features einer Klasse bei einer Spezialisierung redefiniert,
kann mit oclAsType auf Features der Basisklasse zugegriffen werden

```
context B
 inv: self.oclAsType(A).f1      -- f1 von A
 inv: self.f1                   -- f1 von B
```

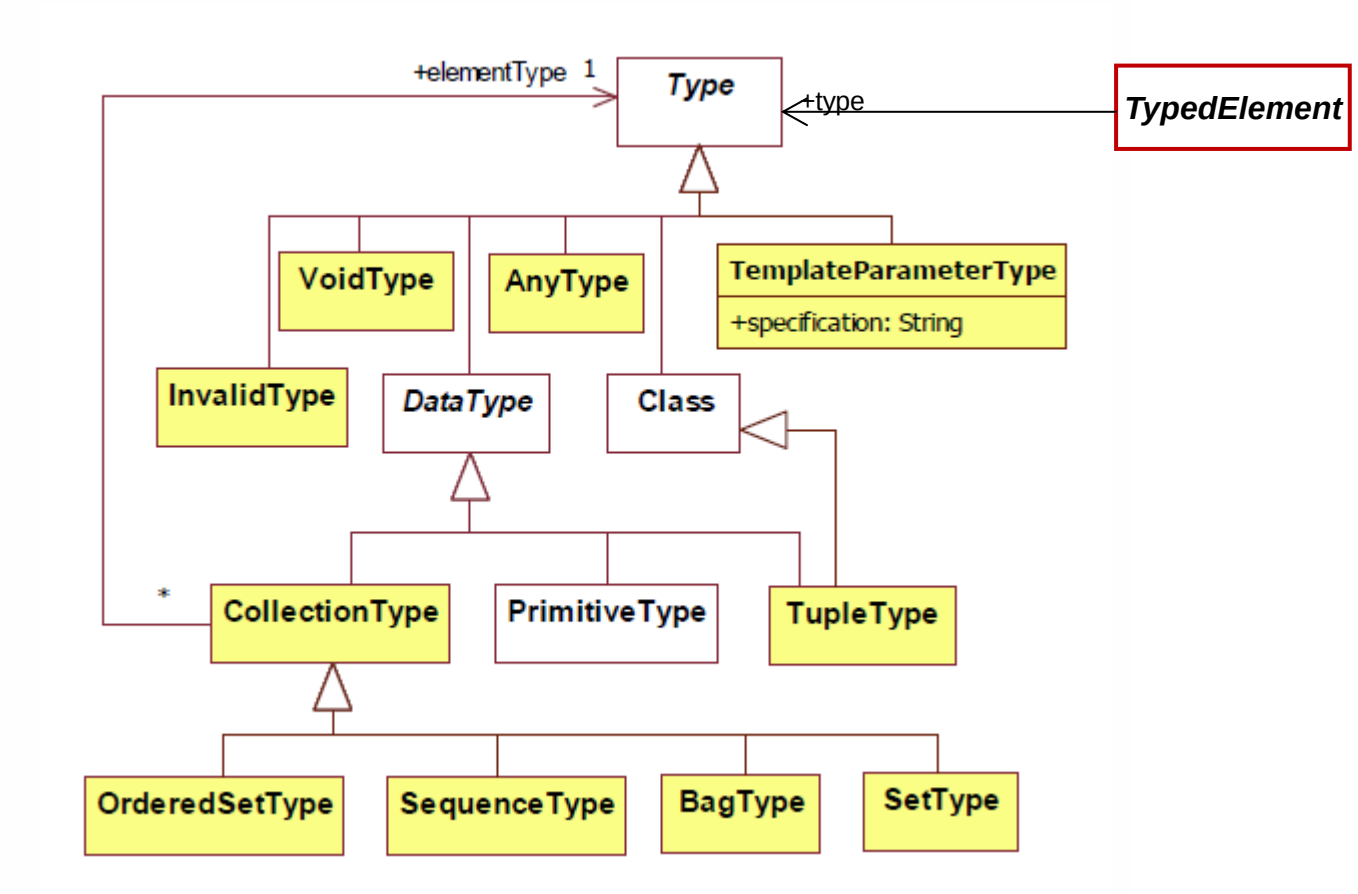
Typtest



context Transaction
inv: self.oclIsKindOf(Transaction) = true
inv: self.oclIsTypeOf(Transaction) = true
inv: self.oclIsKindOf(Burning) = false
inv: self.oclIsTypeOf(Burning) = false

context Burning
inv: self.oclIsKindOf(Transaction) = true
inv: self.oclIsTypeOf(Transaction) = false
inv: self.oclIsKindOf(Burning) = true
inv: self.oclIsTypeOf(Burning) = true
inv: self.oclIsKindOf(Earning) = false
inv: self.oclIsTypeOf(Earning) = false

Typhierarchie



elem:
UserClass

elem.oclOperation()

oder

elem.type.oclAsType(UserClass).oclOperation()

bzw. spezielle Subtyp-Operation

Zugriffsoperator: Punkt oder Pfeil

X. op()

X ist keine Kollektion

X-> op()

- X ist eine Kollektion
  oder
- X ist ein Objekt
  wird als ein-elementige Kollektion aufgefasst
  für die ->
  Kollektionsoperationen anwendbar werden

Inhalt: OCL

~ OCL 2.4
(Feb 2014)

1. Allgemeine Charakterisierung
2. Typen, Metatypen und Werte
3. Typkonformität
4. Standardtypen und Operationen (Überblick)
5. Mächtigkeit von OCL-Ausdrücken
6. OCL-Ausdrucksbildung (anhand von Beispielen)

Allgemeine Objekt-Operationen

- allInstances(): Set (T) /* T ist Typ von Self */

- = (OclAny): Boolean
- <> (OclAny): Boolean

- oclAsSet() : Set(T)
- oclIsNew (): Boolean /* Objekterzeugung während einer Op-Ausführung */
  /* als Differenz von Pre- und PostCondition */
- oclIsUndefined (): Boolean /* Test auf null */
- oclIsInvalid (): Boolean
- oclAsType (t : Classifier): instance of Classifier /* cast */
- oclIsTypeOf (t : Classifier): Boolean
- oclIsKindOf (t : Classifier): Boolean /* transitive Hülle*/
- oclType() : Classifier
- oclIsInState (s: OclState): Boolean

OclInvalid

- allInstances(): Set (T) /* T ist Typ von Self */

alle Classifier

- = (OclAny): Boolean
- <> (OclAny): Boolean

auch Signaturen für Collection-SubTypen

- oclAsSet() : Set(T)
- oclIsNew (): Boolean /* Objekterzeugung während einer Op-Ausführung */
  /* als Differenz von Pre- und
- oclIsUndefined (): Boolean /* Test auf null */
- oclIsInvalid (): Boolean
- oclAsType (t : Classifier): instance of Classifier /* cast */
- oclIsTypeOf (t : Classifier): Boolean
- oclIsKindOf (t : Classifier): Boolean /* transitive Hülle*/
- oclType() : Classifier
- oclIsInState (s: OclState): Boolean

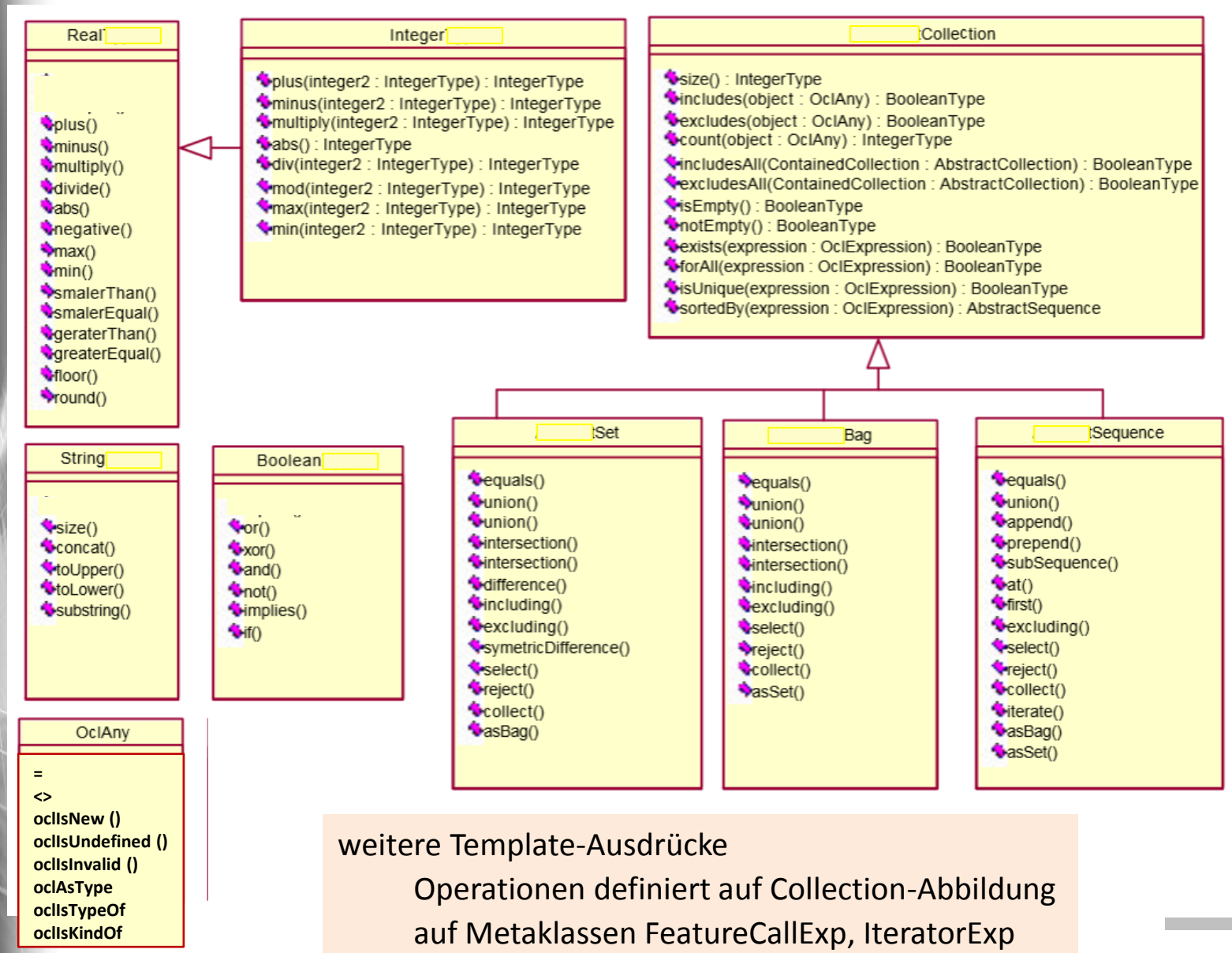
anwendbar auf Operanden deren Typ nicht konform zu OclAny ist ???

Konfliktvermeidung
- alle ererbten Eigenschaften von OclAny beginnen mit Präfix ocl !!!

OclMessage

- hasReturned() : Boolean
- Result(): <Return-Typ der gerufenen OP>
- isSignalSent (): Boolean
- isOperationCall (): Boolean

Operationen der OCL-SL (Auszug)



weitere Template-Ausdrücke

Operationen definiert auf Collection-Abbildung

auf Metaklassen FeatureCallExp, IteratorExp

Collection-Operationen (vollständig)

= (c: Collection(T)): Boolean
<> c: Collection(T)): Boolean
size() : Integer
includes(object : T) : Boolean
excludes(object : T) : Boolean
count(object : T) : Integer
includesAll(c2 : Collection(T)) : Boolean
excludesAll(c2 : Collection(T)) : Boolean
isEmpty() : Boolean
notEmpty() : Boolean
max() : T
min() : T
sum() : T
product(c2: Collection(T2)) :
Set(Tuple(first: T, second: T2))
selectByKind(type : Classifier) :
Collection(T)
selectByType(type : Classifier) :
Collection(T)
asSet() : Set(T)
asOrderedSet() : OrderedSet(T)
asSequence() : Sequence(T)
asBag() : Bag(T)
flatten() : Collection(T2)

forAll (expr oclExpr): Boolean

exists (expr oclExpr): Boolean

isUnique (): Boolean

any (expr BooleanExpr): T

one(expr BooleanExpr): Boolean

collect …

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

Arbeit mit Kollektionen

- A->select (Bedingung über Elemente der Kollektion)

Bildet eine Teilmenge von A, wobei für jedes Element der Teilmenge die Bedingung (OCL-Ausdruck erfüllt ist. (Konstrukt kann auch leere Menge erzeugen)

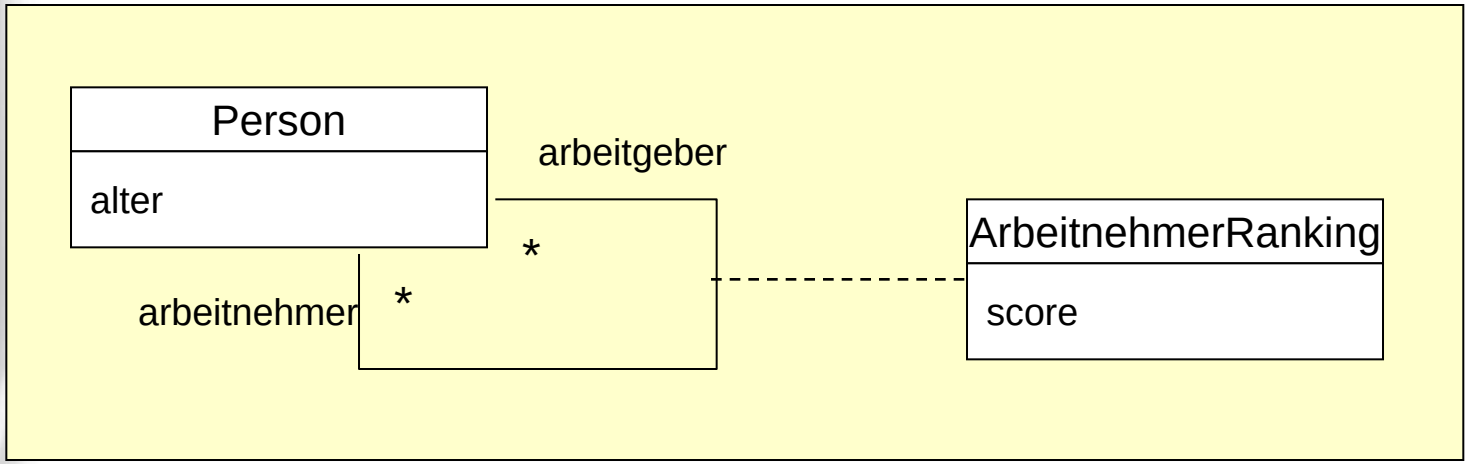
```
context Firma
        inv: arbeitnehmer->select (istVerheiratet) -> notEmpty()
```

A->collect(Projektion auf ein Attribut der Elemente von A)

bildet eine Menge,
die aus den Attribut-Werten der Elemente von A besteht

```
context Firma
        inv: arbeitnehmer->collect (monatsGehalt) -> notEmpty()
```

Rekursive Navigation über Assoziationsklassen (2)



abgekürzte Schreibweise !!!!

self in der Rolle von

aus: OCL-Standard

```
context Person inv:
        self.arbeitnehmerRanking [arbeitgeber]->sum() >0

context Person inv:
        self.arbeitnehmerRanking [arbeitnehmer]->sum() >0

context Person inv:
        self.arbeitnehmerRanking->sum() >0 – unqualifizierter Zugriff
                                             ist nicht erlaubt
```

```
context Person inv:
   self.arbeitnehmerRanking [arbeitgeber]->collect(score)->sum() >0
```

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)

Sortierung geordneter Kollektionen

für geordnete Kollektionen

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

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

-- nach kleiner-als-Elementevergleich
-- Voraussetzung: für Property ist < -Operator definiert

Beispiel (aus dem Standard)

```
context Person
def:
   statistics :
      Set(  Tuple(
                        company: Company,
                        numEmployees: Integer,
                        wellpaidEmployees: Set(Person),
                        totalSalary: Integer
              )
       ) = managedCompanies
              ->collect(c | Tuple {
                        company: Company = c,
                        numEmployees: Integer = c.employee->size(),
                        wellpaidEmployees: Set(Person) = c.Job->select(salary>10000).employee->asSet(),
                        totalSalary: Integer = c.Job.salary->sum()
                        }
                   )
```

context Person **inv**: statistics->sortedBy(totalSalary)->last().wellpaidEmployees->includes(self)

This asserts that a person is one of the best-paid employees of the company with the highest total salary that he manages. In this expression, both 'totalSalary' and 'wellpaidEmployees' are accessing tuple parts.

Inhalt: OCL

~ OCL 2.4
(Feb 2014)

1. Allgemeine Charakterisierung
2. Typen, Metatypen und Werte
3. Typkonformität
4. Standardtypen und Operationen (Überblick)
5. Mächtigkeit von OCL-Ausdrücken
6. OCL-Ausdrucksbildung (anhand von Beispielen)

Mächtigkeit von OCL-Ausdrücken

Anwendung vordefinierter Operationen (Standardbibliothek)

- insbesondere für Kollektionen
  - Teilmengen- und Projektionsbildungen,
  - Zusammenfassungen,
  - All- und Existenz-Quantoren für Iterationen

- mit Möglichkeit einer
  - kombinierten und
  - rekursiven Anwendung

OCL- Einsätzfälle

```
// Anwendung auf M1-Ebene
Class.allInstanzes() -> forAll (c | c.name <> null )
// Alle Klassen im Modell müssen einen Namen haben
```

- Definition von
  - Invarianten für Klassen u. Typen in Klassendiagrammen
  - Invarianten für Stereo-Typen
  - Vor- und Nachbedingungen für Operationen u. Methoden
  - Wächterbedingungen
  - Ziele (Zielmengen) von Nachrichten und Aktionen
  - Constraints von Operationen
  - Ableitungsregeln von Attributen
- Navigationsbeschreibung (im Objektmodell)
  - Anfragesprache
- ab UML 1.3: Built-In-Nutzung zur
  - Definition von *Well-Formedness Rules* für
    - Invarianten und Metaklassen

Präzisierung von UML-Modellen

```
Context Interface
Inv: features -> select (f | f. oclIsKindOf (Attribute) )-> isEmpty()
// Interfaces dürfen keine Attribute enthalten
```

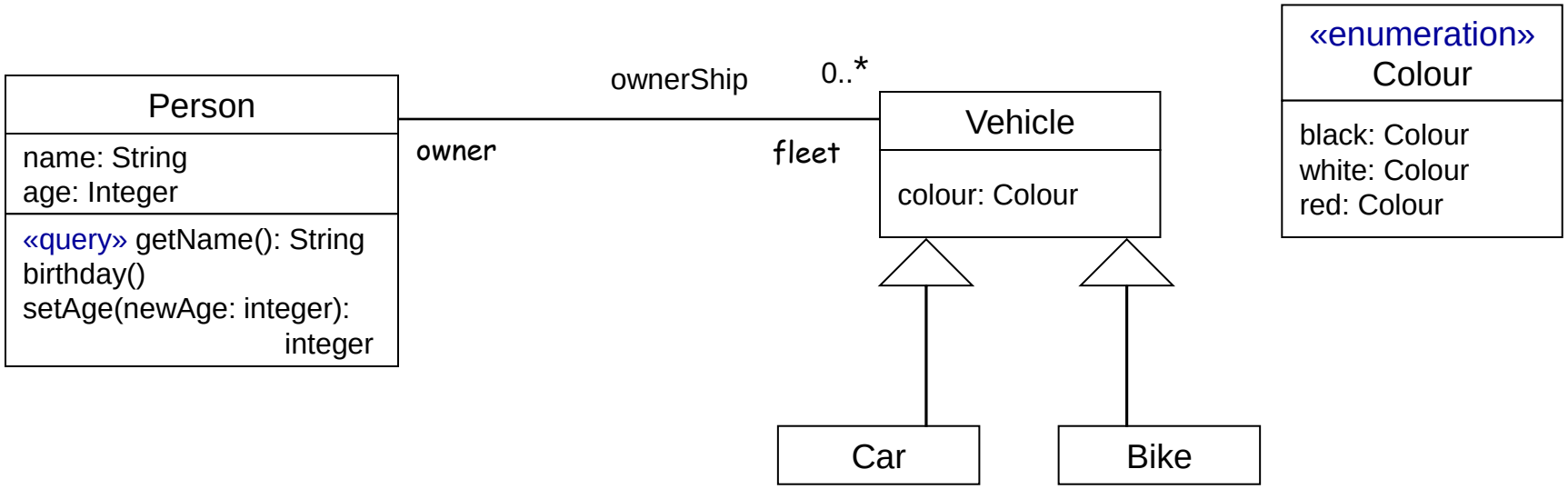
Präzisierung der UML-Sprach-definition

Inhalt: OCL

~ OCL 2.4
(Feb 2014)

1. Allgemeine Charakterisierung
2. Typen, Metatypen und Werte
3. Typkonformität
4. Standardtypen und Operationen (Überblick)
5. Mächtigkeit von OCL-Ausdrücken
6. OCL-Ausdrucksbildung (anhand von Beispielen)

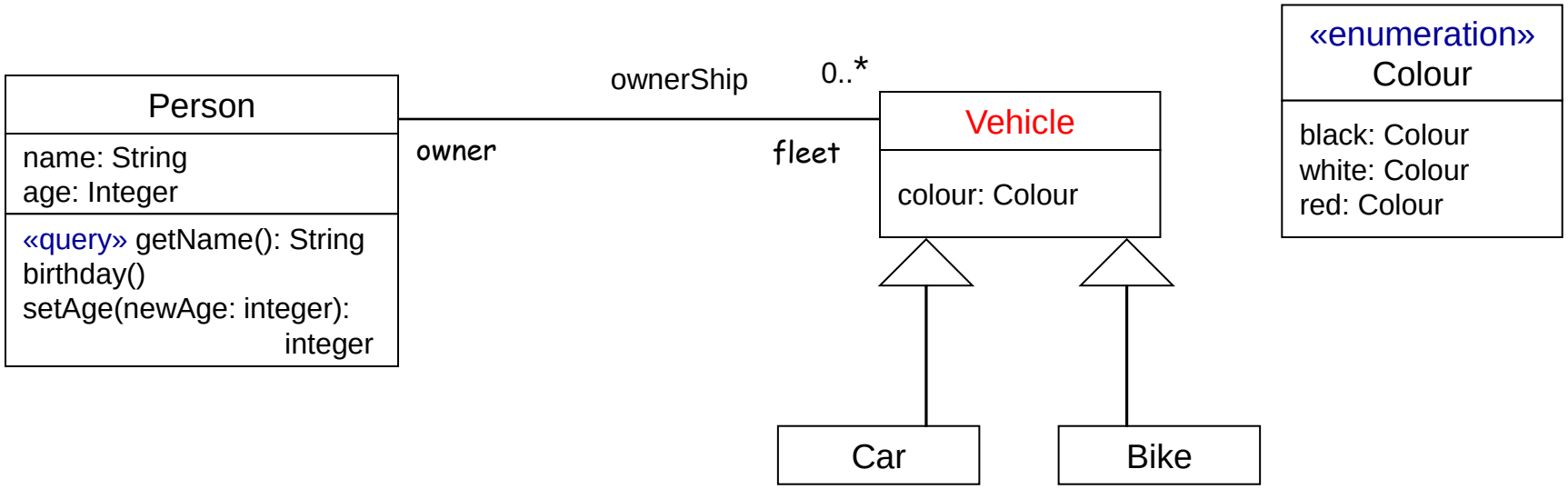
Unzulängliches UML-Modell



Besitzer von Fahrzeugen müssen ein gewisses Alter haben

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

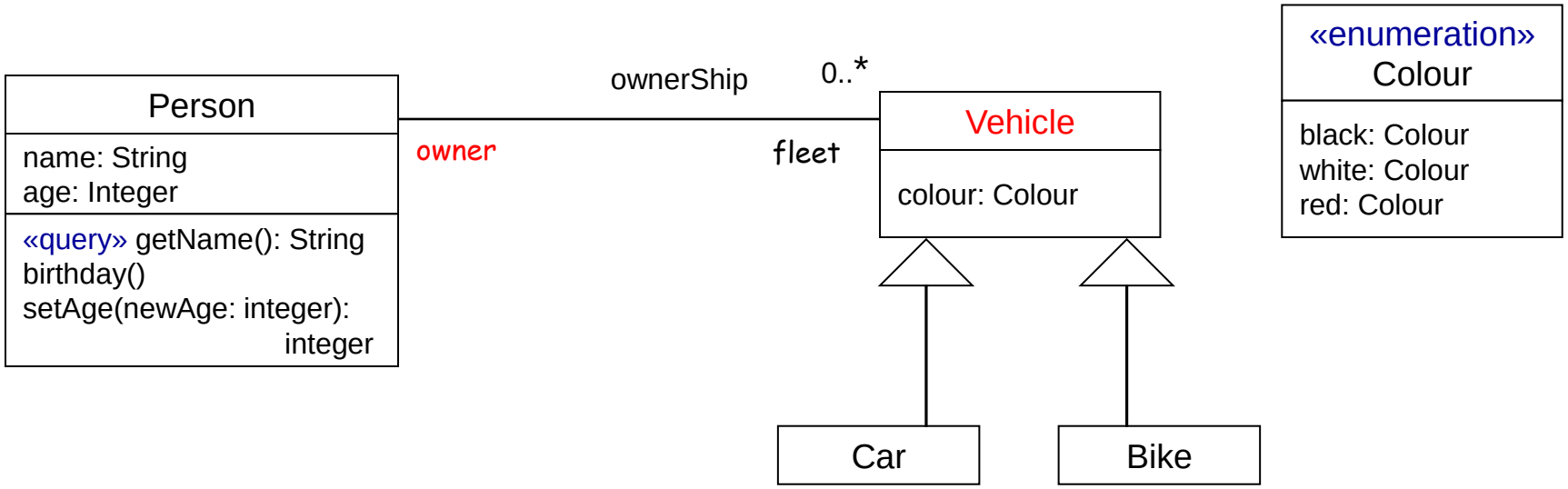
OCL-Interpretation (1)



Besitzer von Fahrzeugen müssen ein gewisses Alter haben

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

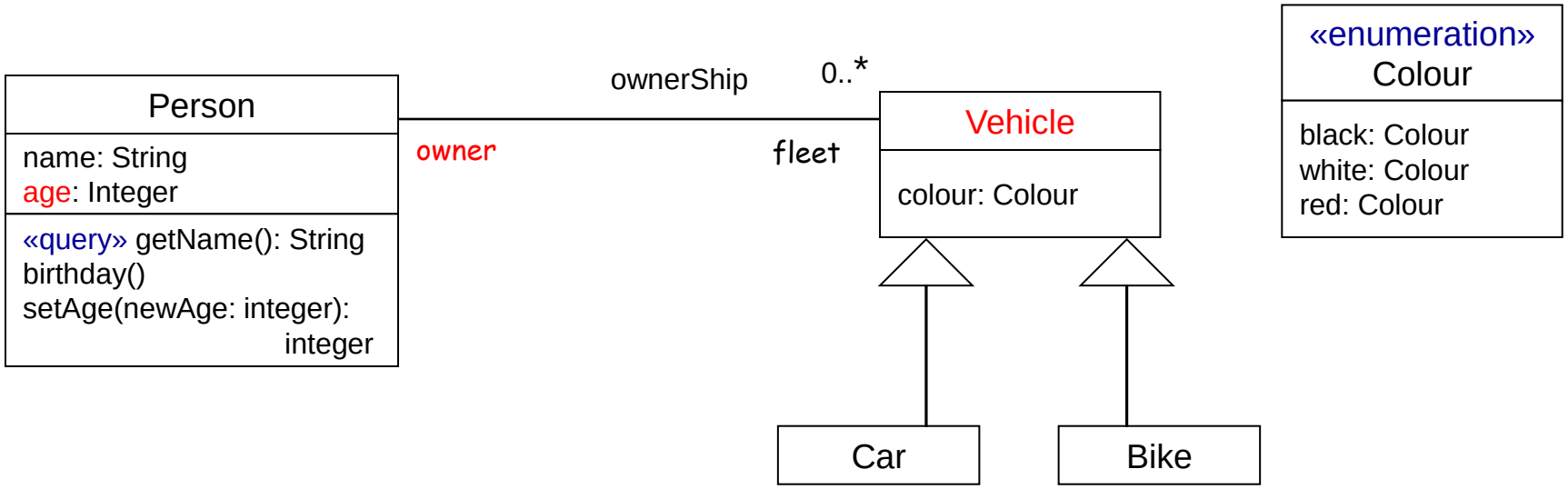
OCL-Interpretation (2)



Besitzer von Fahrzeugen müssen ein gewisses Alter haben

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

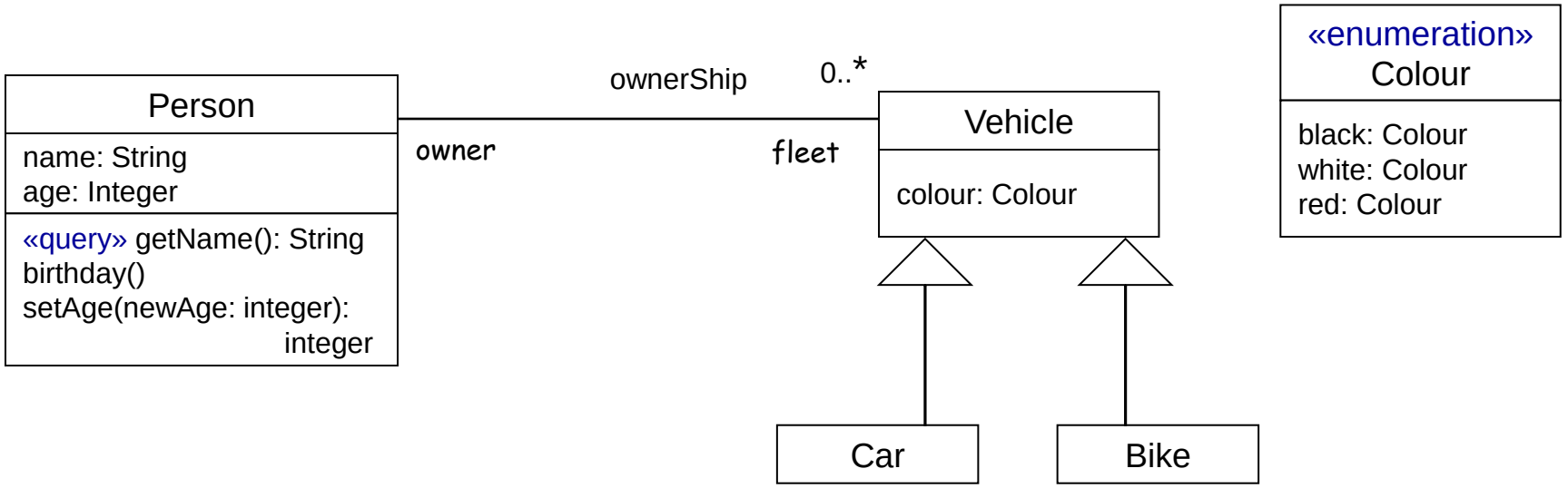
OCL-Interpretation (3)



Besitzer von Fahrzeugen müssen ein gewisses Alter haben

```
context  Car
inv:
    self. owner. age >= 18
```

Unterschiede ähnlicher OCL-Invarianten



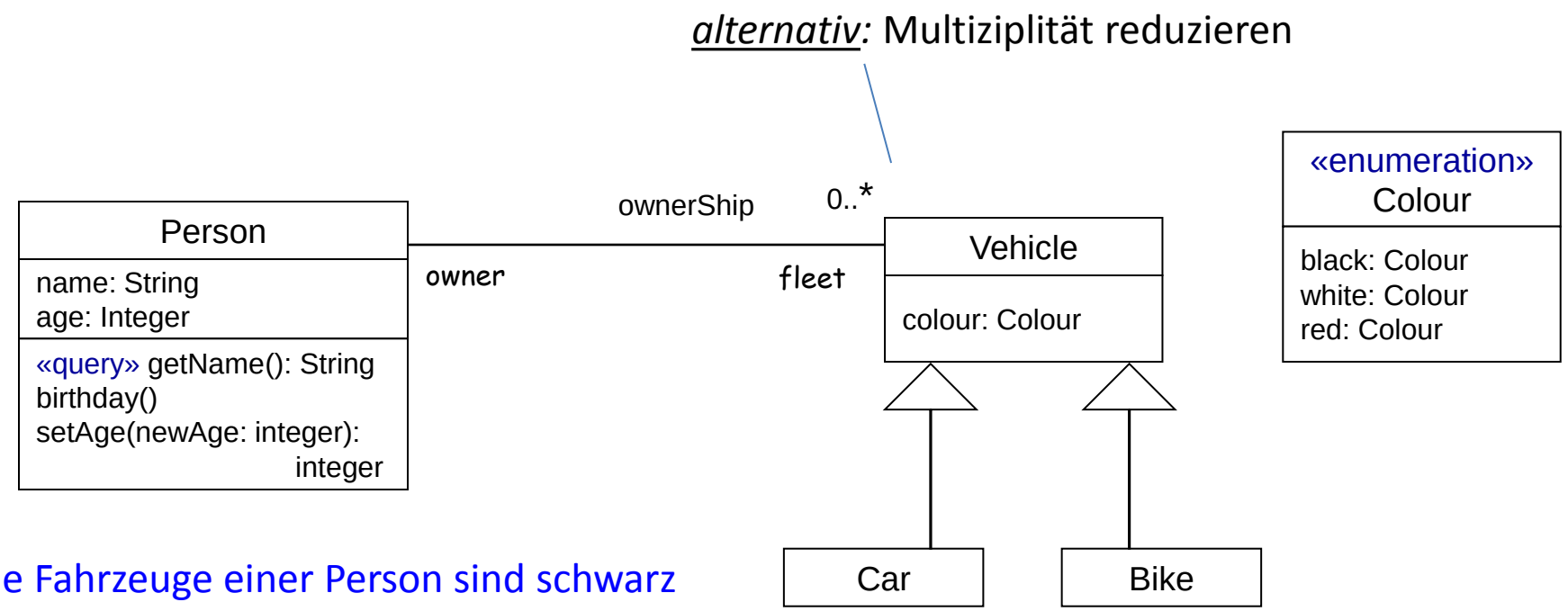
- Was bedeuten die verschiedenen Varianten?

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

context Person
inv:
self. age >= 18

context Car
inv:
self. age >= 18

Weitere OCL-Invarianten



Alle Fahrzeuge einer Person sind schwarz

context Person
inv: self.fleet->**forAll** (v | v.colour = #black)

Keiner besitzt mehr als 3 Fahrzeuge

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

Keiner besitzt mehr als 3 schwarze Fahrzeuge

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