Vorlesung (WS 2014/15)

Softwarekonstruktion

Prof. Dr. Jan Jürjens

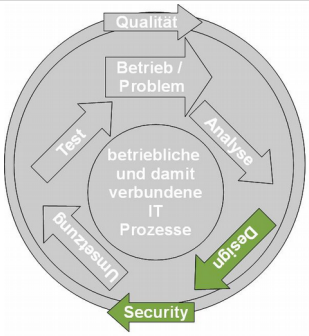
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

1.2: Grundlagen: Object Constraint Language

v. 10.11.2014

Einordnung
Object Constraint Language (OCL)

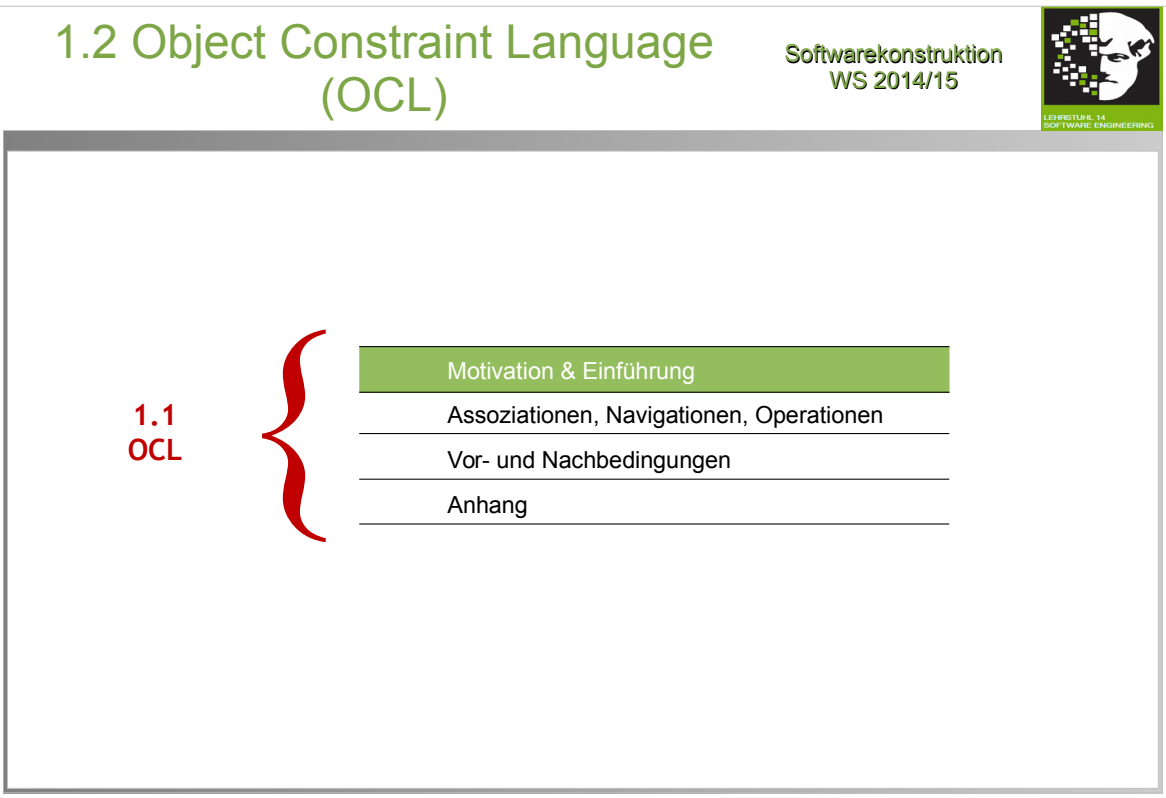
- Modellgetriebene SW-Entwicklung
  - Einführung
  - Modellbasierte Softwareentwicklung
  - Object Constraint Language (OCL)
  - Ereignisgesteuerte Prozesskette (EPK)
  - Petrinetze
  - Eclipse Modeling Foundation (EMF)
- Qualitätsmanagement
- Testen



Abschnitt basiert auf Vortrag "Introduction to OCL" von V. Bembenek, H. Schmidt, M. Heisel.

Literatur (s. Webseite):

- V. Gruhn: **MDA - Effektives Software-Engineering**. Kapitel 3.6.
- J. Seemann, J.W. Gudenberg: **Software-Entwurf mit UML 2**. Kapitel 14.5.
- Object Management Group: **OCL 2.4** http://www.omg.org/spec/OCL/2.4/PDF.



Literatur:

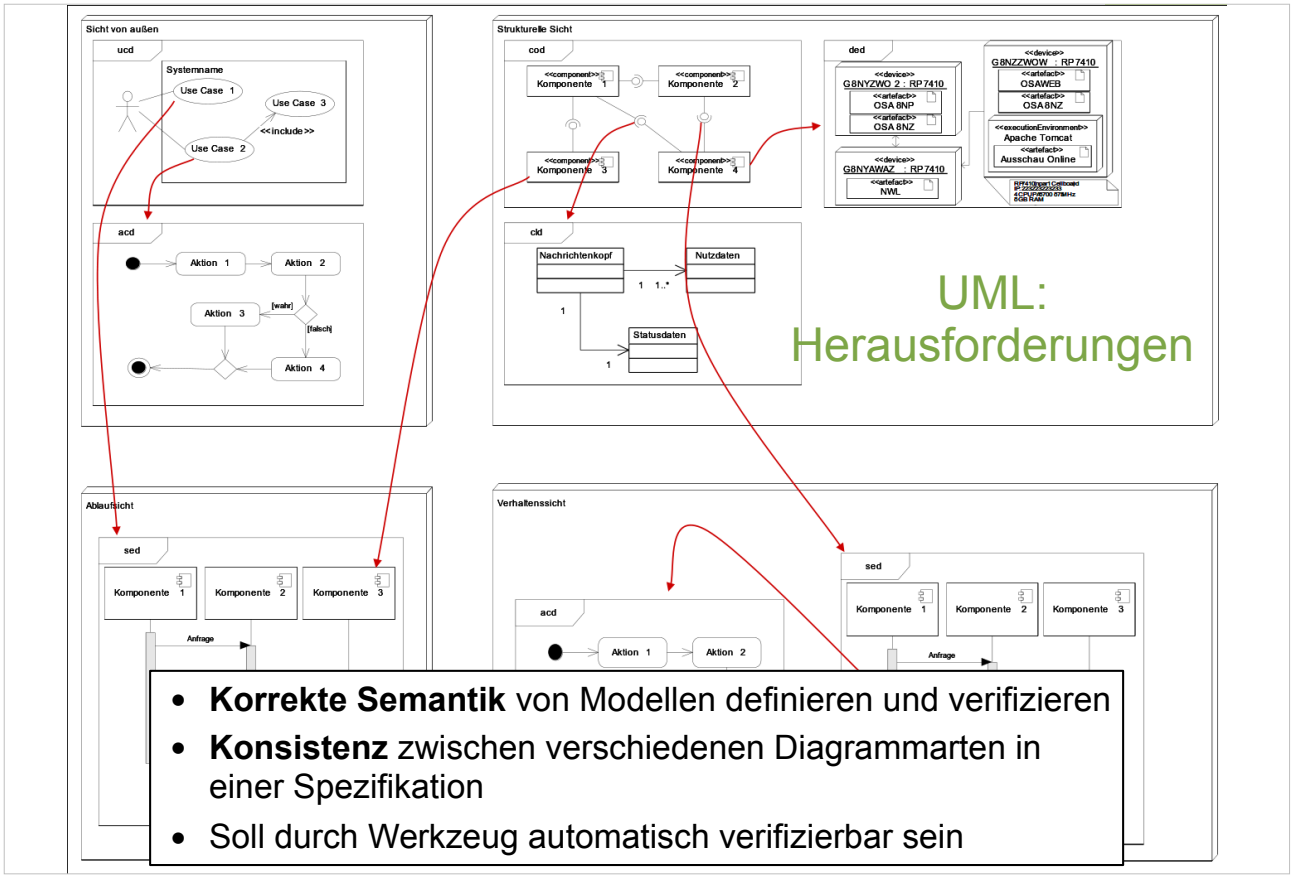
V. Gruhn: **MDA - Effektives Software-Engineering**

http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 3.6

Einleitung OCL

- **Vorheriger Abschnitt:** Modellbasierte Softwareentwicklung
- **Dieser Abschnitt:** Von Modellen zu Objekten
  - Einführung in Object Constraint Language
    → Syntax und Semantik

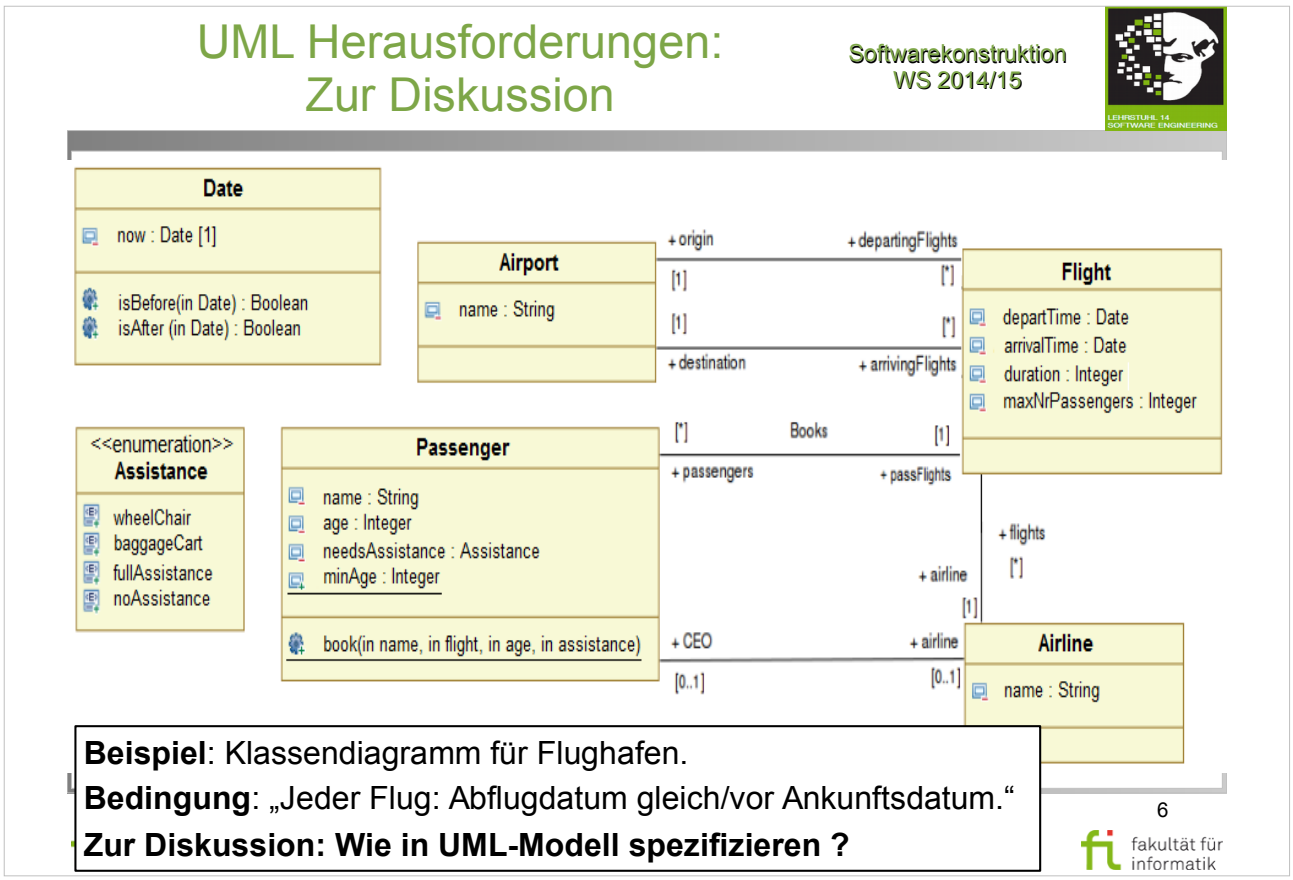


- **Korrekte Semantik** von Modellen definieren und verifizieren
- **Konsistenz** zwischen verschiedenen Diagrammarten in einer Spezifikation
- Soll durch Werkzeug automatisch verifizierbar sein

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 3.6.1 (S.106-107)



Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 3.6.1 (S.106-107)

Zur Fehlervermeidung: **Einschränkungen** definieren,
die zur Laufzeit eingehalten werden müssen.

NB (Methoden in Papyrus): *in* spezifiziert input Parameter;
Typen können weggelassen werden.

Bedingung: „Jeder Flug: Abflugdatum gleich/vor Ankunftsdatum.“
=> implizite Annahme: Flüge gehen nicht („rückwärts“) über Datumsgrenze

OMG Spezifikation: **„Object Constraint Language 2.0“.**

Teil von UML (http://www.omg.org/spec/OCL/2.2/PDF)

Logik-basierte Notation für **Einschränkungen** (Constraints) in UML-Modellen:

- Welche **Klassen erreichbar**; welche Attribute, Operationen und Assoziationen für Objekte dieser Klassen vorhanden.
- **Bedingungen** an Wertebelegungen während Ausführung der modellierten Systemteile (vgl. Assertions in Programmquellcode).

Unterstützt QS bei modellbasierter Softwareentwicklung.

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129
- Abschnitt 3.6.1 (S.106-107)

Seemann, Gudenberg: **Software-Entwurf mit UML 2**
http://www.ub.tu-dortmund.de/katalog/titel/1223020
- Abschnitt 14.5.1 (S.281-282)

OCL: Ziele

Vorteile:

- **Automatische Verifikation der Bedingung.**
- **Präzise Spezifikation der Bedingungen beseitigt Mehrdeutigkeit.**
- **Werkzeuge** erzeugen aus OCL **Assertions** in **Java**.

Keine Programmiersprache:
→ Modellierung von Programmlogik nicht vorgesehen und möglich !

- Insbesondere: OCL-Ausdrücke **ohne Seiteneffekte:**
  - Kann nichts im Modell verändern.
  - OCL-Ausdruck verursacht keine Zustandsänderung, auch wenn er sie spezifizieren kann (z.B. in einer Nachbedingung).

OCL unterstützt Prüfung von **strenger Typisierung.**

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 3.6.1 (S.106-107)

Seemann, Gudenberg: **Software-Entwurf mit UML 2**
http://www.ub.tu-dortmund.de/katalog/titel/1223020

- Abschnitt 14.5.1 (S.281-282)

Wie verwende ich OCL ?

OCL-Ausdrücke: an UML-Modell gebunden.

Beschreiben Einschränkungen für Elemente des Modells, zu dem sie gehören.

Zwei Arten von **Einschränkungen** spezifizieren und verifizieren:

- **Fortlaufende** Zustandsbeschränkung (mit Invarianten).
- Zustandsbeschränkungen **vor** bzw. **nach Methodenaufruf** (mit Vor- und Nachbedingungen).

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 3.6.1 (S.106-107)

Seemann, Gudenberg: **Software-Entwurf mit UML 2**
http://www.ub.tu-dortmund.de/katalog/titel/1223020

- Abschnitt 14.5.1 (S.281-282)

Basisform eines OCL-Ausdrucks

Syntax:

context <identifier>
<constraintType> [<constraintName>]: <boolean expression>

Wobei:

context	Schlüsselwort
<identifier>	Klassen- oder Operationsname.

- Zugehöriges Modellelement markieren.
- Kann innerhalb <boolean expression> genutzt werden.

<constraintType>	Schlüsselwort *inv*, *pre* oder *post*.
<constraintName>	optionaler Name für diesen Constraint.
<boolean expression>	boolescher Ausdruck.

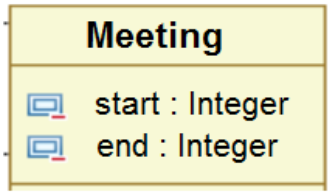
- Kann andere Modellelemente referenzieren.

Literatur:

Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.2.1 – context (S.6)
- Abschnitte 7.3.1 bis 7.3.4 (S.7-9)
- Abschnitt 7.4 (S.10-11)

Basisform eines OCL-Ausdrucks



Das Treffen endet, nachdem es startet.

Beispiel

context Meeting
inv startEndConstraint:
self.end > self.start

context <identifier>
<constraintType> [<constraintName>]:
<boolean expression>

Literatur:
http://st.inf.tu-dresden.de/files/teaching/ss09/stII09/OCL.pdf

- Technische Universität Dresden – Einführung in OCL
- Dr. Birgit Demuth
- F. 13+14

Constraint (Einschränkung):
Einschränkung auf einem oder mehreren Teilen eines UML-Modells.

Literatur:

Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.3 – Invariants (S.8)
- Abschnitt 12.6 – Invariant (S.188)
- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)
- Abschnitt 12.7 – Precondition (S.188-189)
- Abschnitt 12.7.2 – Postcondition (S.189-190)
- Abschnitt 12.11 – Guard (S.192-193)

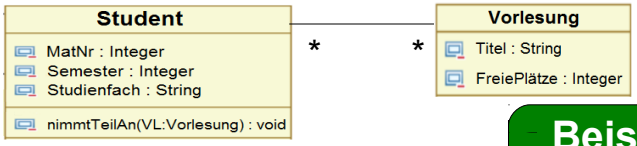
Einschränkungen (Constraints)

Constraint (Einschränkung):
Einschränkung auf einem oder mehreren Teilen eines UML-Modells.

Es gibt die folgenden wichtigsten **Arten** von Einschränkungen:

Class Invariant (Klasseninvariante):
Muss fortwährend von allen Instanzen einer Klasse erfüllt sein

Anhand des Schlüsselwortes *inv* im Kontext der Instanz eines Klassifikators (Klasse, Rollenname...) spezifizierbar



Die Matrikelnummer muss mindestens 5-stellig sein.

Beispiel

context Student

inv: self.MatNr >= 10000

Literatur:

Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.3 – Invariants (S.8)
- Abschnitt 12.6 – Invariant (S.188)
- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)
- Abschnitt 12.7 – Precondition (S.188-189)
- Abschnitt 12.7.2 – Postcondition (S.189-190)
- Abschnitt 12.11 – Guard (S.192-193)

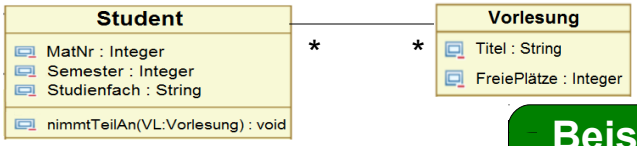
Einschränkungen (Constraints)

Constraint (Einschränkung):
Einschränkung auf einem oder mehreren Teilen eines UML-Modells.

Es gibt die folgenden wichtigsten **Arten** von Einschränkungen:

Class Invariant (Klasseninvariante):
Muss *immer* von allen Instanzen einer Klasse erfüllt sein.

Pre-condition (Vorbedingung):
Muss erfüllt sein, *bevor* Operation ausgeführt wird.



Der Student kann sich nur zu einer Vorlesung anmelden, wenn noch min. ein Platz frei ist.

Beispiel

context Student :: nimmtTeilAn (VL:Vorlesung): void

pre: VL.FreiePlätze >= 1

Literatur:

Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.3 – Invariants (S.8)
- Abschnitt 12.6 – Invariant (S.188)
- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)
- Abschnitt 12.7 – Precondition (S.188-189)
- Abschnitt 12.7.2 – Postcondition (S.189-190)
- Abschnitt 12.11 – Guard (S.192-193)

Einschränkungen (Constraints)

Constraint (Einschränkung):
Einschränkung auf einem oder mehreren Teilen eines UML-Modells.

Es gibt die folgenden wichtigsten **Arten** von Einschränkungen:

Class Invariant (Klasseninvariante):
Muss *immer* von allen Instanzen einer Klasse erfüllt sein.

Pre-condition (Vorbedingung):
Muss erfüllt sein, *bevor* Operation ausgeführt wird.

Post-condition (Nachbedingung):
Muss *nach* Ausführen einer Operation erfüllt sein.

Wenn der Student sich zur Vorlesung angemeldet hat, ist ein Platz weniger frei als vorher.

Beispiel

context Student :: nimmtTeilAn (VL:Vorlesung): void
post: VL.FreiePlätze = VL.FreiePlätze @pre -1

Literatur:

Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.3 – Invariants (S.8)
- Abschnitt 12.6 – Invariant (S.188)
- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)
- Abschnitt 12.7 – Precondition (S.188-189)
- Abschnitt 12.7.2 – Postcondition (S.189-190)
- Abschnitt 12.11 – Guard (S.192-193)

OCL Typen

Folgende **Typen** im OCL-Ausdruck benutzbar:

- **Vordefinierte Typen:**
  - Primitive Typen: String, Integer, Real, Boolean.
  - Kollektions-Typen: Set, Bag, Sequence, OrderedSet.
  - Tupel-Typen: Tuple.
  - Spezielle Typen: OclType, OclAny, …
- **Klassifikatoren** vom UML-Modell und seinen Eigenschaften:
  - Klassen, Enumerationsklassen und Rollennamen.
  - Attribute und Operationen.

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.4 – Basic Values and Types (S.10-11)
- Abschnitt 7.4.11 – Keywords (S.16)

OCL Schlüsselwörter

Folgende **Schlüsselwörter** im OCL-Ausdruck benutzbar:

- Konditionalausdrücke: *If-then-else-endif*
- Boolesche Operatoren: *Not, or, and, xor, implies*
- Globale Definitionen: *Def*
- Lokale Definitionen: *Let-in*

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.4 – Basic Values and Types (S.10-11)
- Abschnitt 7.4.11 – Keywords (S.16)

NB: In den Übungen ist nur die Verwendung einer (dort definierten) Teil-Notation von OCL vorgesehen (insbesondere keine *If-then-else-endif, Def, Let-in* Statements).

Gebräuchliche Typen von Invarianten

Beschränkung von Domänen:

- Beschränkung der Werte, die Attribut annehmen kann.

Beschränkung auf Einmaligkeit:

- Attribut oder Attributmenge einer Klasse für die gilt:
  - Für zwei unterschiedliche Instanzen dieser Klasse dürfen keine gleichen Werte zugewiesen werden.

Regeln für Existenz:

- Bestimmte Objekte / Werte müssen existieren / definiert sein...
- … bevor andere Objekte / Werte definiert / erzeugt werden können.

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.3 – Invariants (S.8)

Invarianten auf Attributen

- Klasse, die von Invarianten referenziert wird: **Kontext der Invarianten.**
- **Gefolgt vom booleschen Wert**, der Invariante angibt.
- Alle Attribute der Kontextklasse in Invarianten nutzbar.

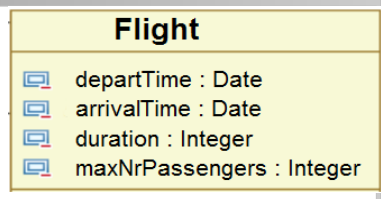
context <identifier>
<constraintType>: <boolean expression>

<identifier> Klassen- oder Operationsname
<constraintType> Schlüsselwort inv, pre, post
<boolean expression> boolescher Ausdruck

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.3.3 – Invariants (S.8)

Invarianten auf Attributen

- Klasse, die von Invarianten referenziert wird: **Kontext der Invarianten.**
- **Gefolgt vom booleschen Wert**, der Invariante angibt.
- Alle Attribute der Kontextklasse in Invarianten nutzbar.
- Beispiel: „Jeder Flug dauert weniger als 4 Stunden.“



Beispiel

context

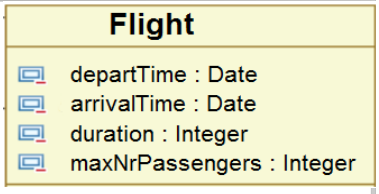
context <identifier>
<constraintType>: <boolean expression>

<identifier> Klassen- oder Operationsname
<constraintType> Schlüsselwort inv, pre, post
<boolean expression> boolescher Ausdruck

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.3.3 – Invariants (S.8)

Invarianten auf Attributen

- Klasse, die von Invarianten referenziert wird: **Kontext der Invarianten.**
- **Gefolgt vom booleschen Wert**, der Invariante angibt.
- Alle Attribute der Kontextklasse in Invarianten nutzbar.
- Beispiel: „Jeder Flug dauert weniger als 4 Stunden.“



Beispiel

context Flight

inv: self.duration < 4

context <identifier>
<constraintType>: <boolean expression>

<identifier> Klassen- oder Operationsname
<constraintType> Schlüsselwort inv, pre, post
<boolean expression> boolescher Ausdruck

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.3.3 – Invariants (S.8)

Invarianten auf Attributen: Abfrageoperationen

- Wenn Attributtyp Klasse ist:

  → Attribute und **Abfrageoperationen** dieser Klasse für Erstellung der Invarianten nutzbar (anhand Punkt-Notation).

- Abfrageoperation: Operation, die **Wert von Attributen nicht ändert.**

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

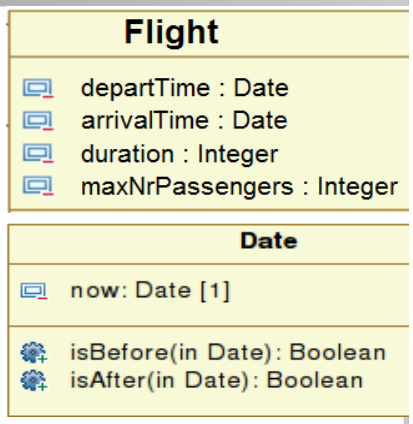
- Abschnitt 7.3.3 – Invariants (S.8)

Invarianten auf Attributen: Abfrageoperationen

- Wenn Attributtyp Klasse ist:

  → Attribute und **Abfrageoperationen** dieser Klasse für Erstellung der Invarianten nutzbar (anhand Punkt-Notation).

- Abfrageoperation: Operation, die **Wert von Attributen nicht ändert.**
- Beispiel: „Abflugdatum eines Fluges ist vor Ankunftsdatum.“



Beispiel

context

context <identifier>
<constraintType>: <boolean expression>

<identifier> Klassen- / Operationsname.
<constraintType> Schlüsselwort inv, pre, post.
<boolean expression> boolescher Ausdruck

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

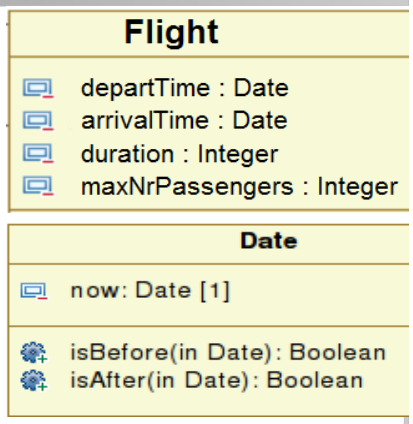
- Abschnitt 7.3.3 – Invariants (S.8)

Invarianten auf Attributen: Abfrageoperationen

- Wenn Attributtyp Klasse ist:

  → Attribute und **Abfrageoperationen** dieser Klasse für Erstellung der Invarianten nutzbar (anhand Punkt-Notation).

- Abfrageoperation: Operation, die **Wert von Attributen nicht ändert.**
- Beispiel: „Abflugdatum eines Fluges ist vor Ankunftsdatum.“



Beispiel

context Flight

inv: self.departTime.isBefore(arrivalTime)

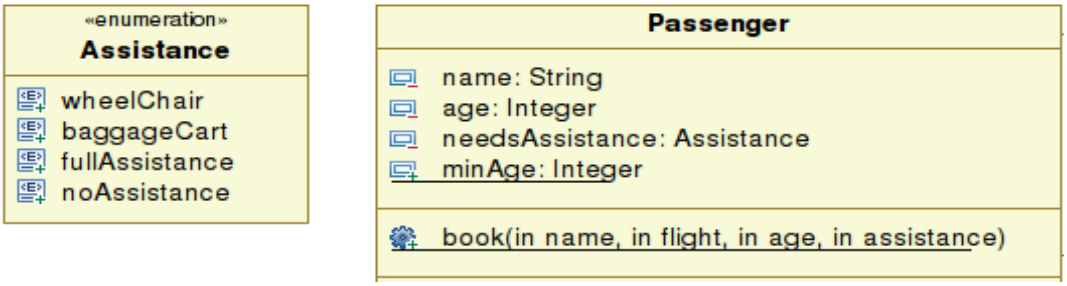
context <identifier>
<constraintType>: <boolean expression>

<identifier> Klassen- / Operationsname.
<constraintType> Schlüsselwort inv, pre, post.
<boolean expression> boolescher Ausdruck

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.3 – Invariants (S.8)

Aufzählungstypen



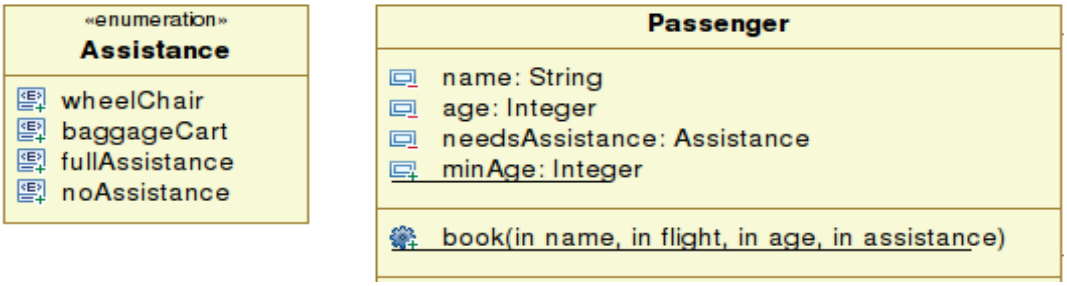
Aufzählung nutzt Datentypen gefolgt von :: und einem Wert.

Beispiel

context Passenger
inv: self.age > 95 implies
self.needsAssistance = Assistance :: wheelChair

Bedeutung: ?

Aufzählungstypen

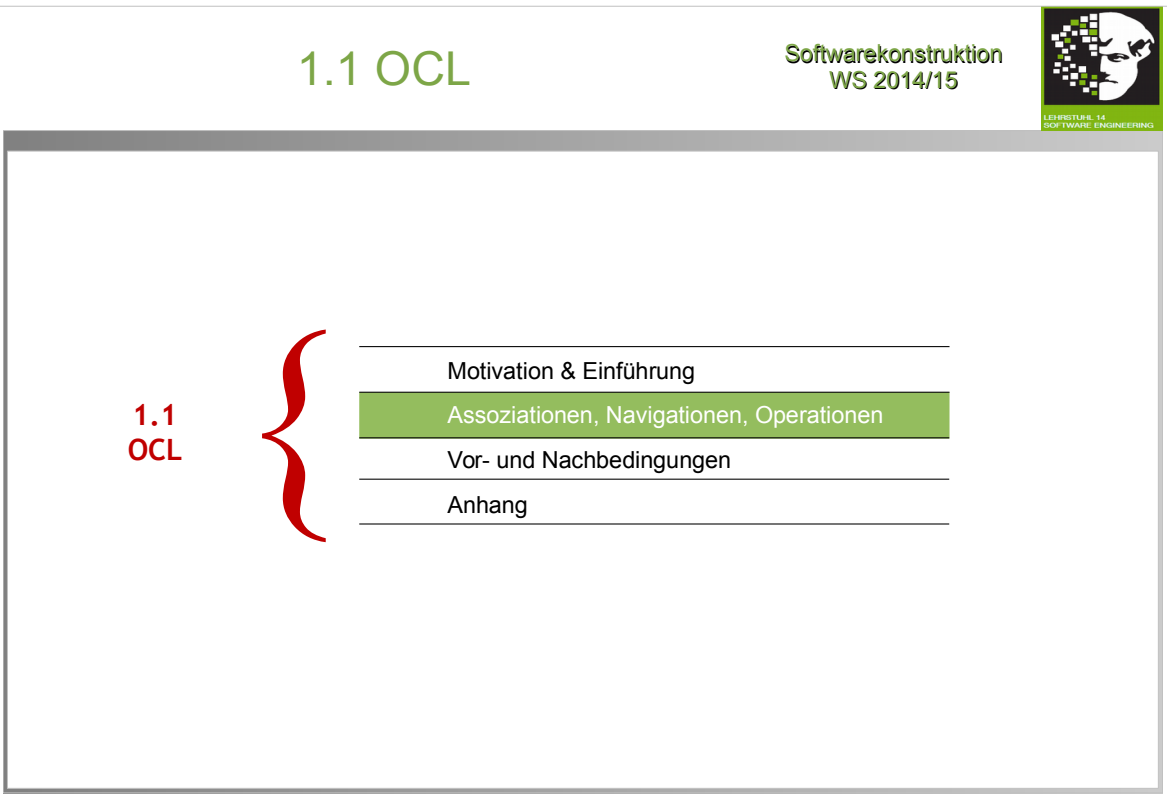


Aufzählung nutzt Datentypen gefolgt von :: und einem Wert.

Beispiel

context Passenger
inv: self.age > 95 implies
self.needsAssistance = Assistance :: wheelChair

Bedeutung: Jeder Passagier über 95 braucht einen Rollstuhl.



Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.5 – Objects and Properties (ab S.17)

Assoziation und Navigation

- Jede Assoziation ist **Navigationspfad.**
- Kontext des Ausdrucks ist **Startpunkt.**
- Rollennamen (oder Assoziationsenden) werden genutzt, um **navigierte Assoziationen** zu identifizieren.

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.5.3 (S.18-20)
- Abschnitt 7.5.4 (S.21)
- Abschnitt 7.5.5 (S.22)
- Abschnitt 7.5.6 (S.22)

Assoziation und Navigation: Beispiel

Beispiel: „Abflugort eines Fluges ist immer ungleich Ziel.“

Beispiel
context

(Annahme: verschiedene Flughäfen haben verschiedene Namen, s. später.)

Beispiel: „Abflugort eines Fluges ist immer Dortmund.“

Beispiel
context

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.5.3 (S.18-20)
- Abschnitt 7.5.4 (S.21)
- Abschnitt 7.5.5 (S.22)
- Abschnitt 7.5.6 (S.22)

Assoziation und Navigation: Beispiel

Beispiel: „Abflugort eines Fluges ist immer ungleich Ziel.“

Beispiel

context Flight

inv: origin <> destination

(Annahme: verschiedene Flughäfen haben verschiedene Namen, s. später.)

Beispiel: „Abflugort eines Fluges ist immer Dortmund.“

Beispiel

context

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.5.3 (S.18-20)
- Abschnitt 7.5.4 (S.21)
- Abschnitt 7.5.5 (S.22)
- Abschnitt 7.5.6 (S.22)

Assoziation und Navigation: Beispiel

Beispiel: „Abflugort eines Fluges ist immer ungleich Ziel.“

Beispiel

context Flight

inv: origin <> destination

(Annahme: verschiedene Flughäfen haben verschiedene Namen, s. später.)

Beispiel: „Abflugort eines Fluges ist immer Dortmund.“

Beispiel

context Flight

inv: origin.name = 'Dortmund'

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.5.3 (S.18-20)
- Abschnitt 7.5.4 (S.21)
- Abschnitt 7.5.5 (S.22)
- Abschnitt 7.5.6 (S.22)

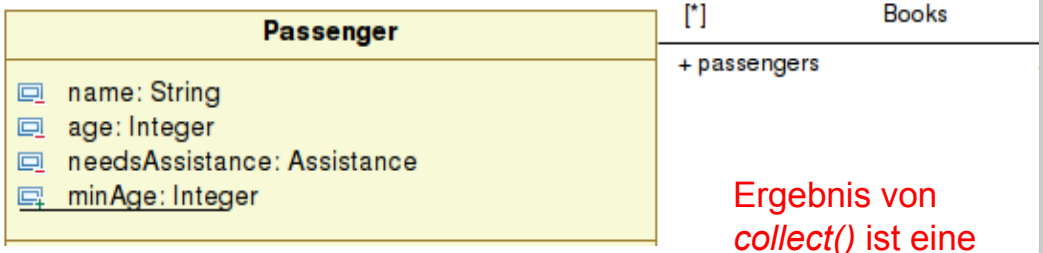
Collection-Operationen

- Assoziationen: Insbesondere **one-to-many** oder **many-to-many** Beziehungen.
  => Navigation zu Assoziationsende resultiert in Collections.
- OCL Ausdrücke geben entweder Fakt über **alle** Objekte in Collection an oder über Collection **selbst**.
- Beschränkungen beziehen sich oft nur auf Teil einer Collection.
  => Mit Collection-Operationen auswählen:
  - **Pfeil** (→) spezifiziert Nutzung vordefinierter Operation für Collections. Z.B. für Klasse *Passenger* und Größe einer Collection *size()*: *Passenger* → *size()*.
- Zur **Abgrenzung:** Verwendung von Operation aus UML-Modell (z.B. *departTime.isBefore(arrivalTime)*).

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.5.11 - Collections (S.24-26)
- Abschnitt 7.6 – Collection Operations (S.28-32)

Collection-Operationen: *collect()*

collect(): Operation, um **Attributwerte** zu **sammeln**, z.B.:
self.passengers → *collect(name).*



Ergebnis von *collect()* ist eine Multimenge !

Bedeutung (in Pseudocode)

```
Collection<String> c = new Collection();
foreach (p: passengers) {c.add(p.name);}
return c;
```


Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

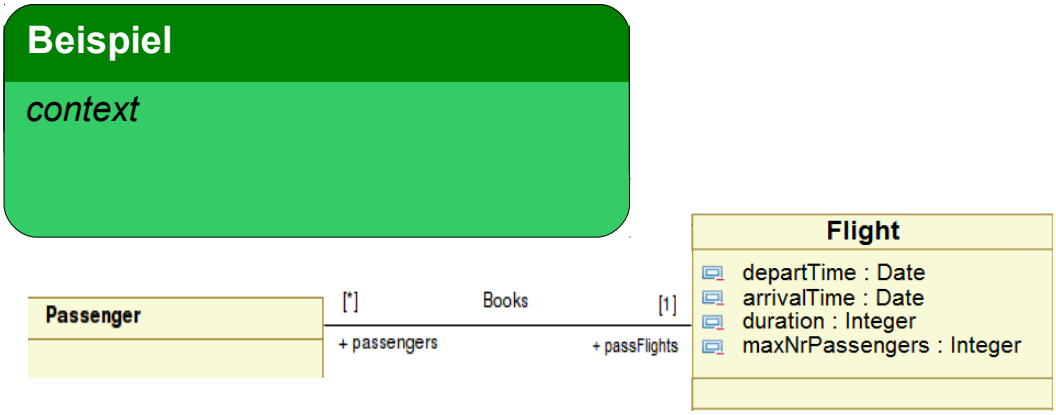
Per Definition erzeugt *collect()* Multimengen / **Bags** (d.h. Elemente können **mehrfach** auftreten).

Punktnotation als verkürzte Schreibweise, z.B. passengers.name
(nicht verwechseln mit Attributzugriff !).

Collection-Operationen: *size()*

size() spezifiziert die Größe von *Collections*.

Beispiel: Anzahl der Passagiere eines Fluges ist kleiner oder gleich der vorgegebenen maximalen Anzahl der Passagiere.



Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)

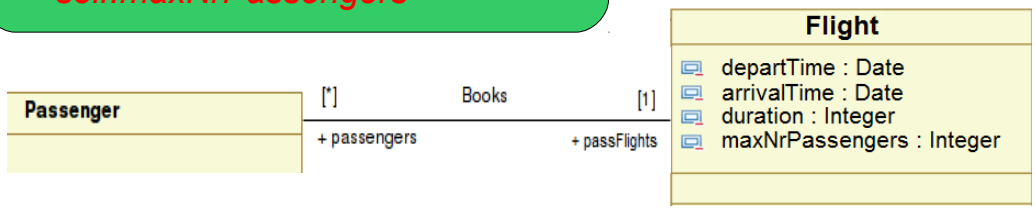
Collection-Operationen: *size()*

size() spezifiziert die Größe von *Collections*.

Beispiel: Anzahl der Passagiere eines Fluges ist kleiner oder gleich der vorgegebenen maximalen Anzahl der Passagiere.

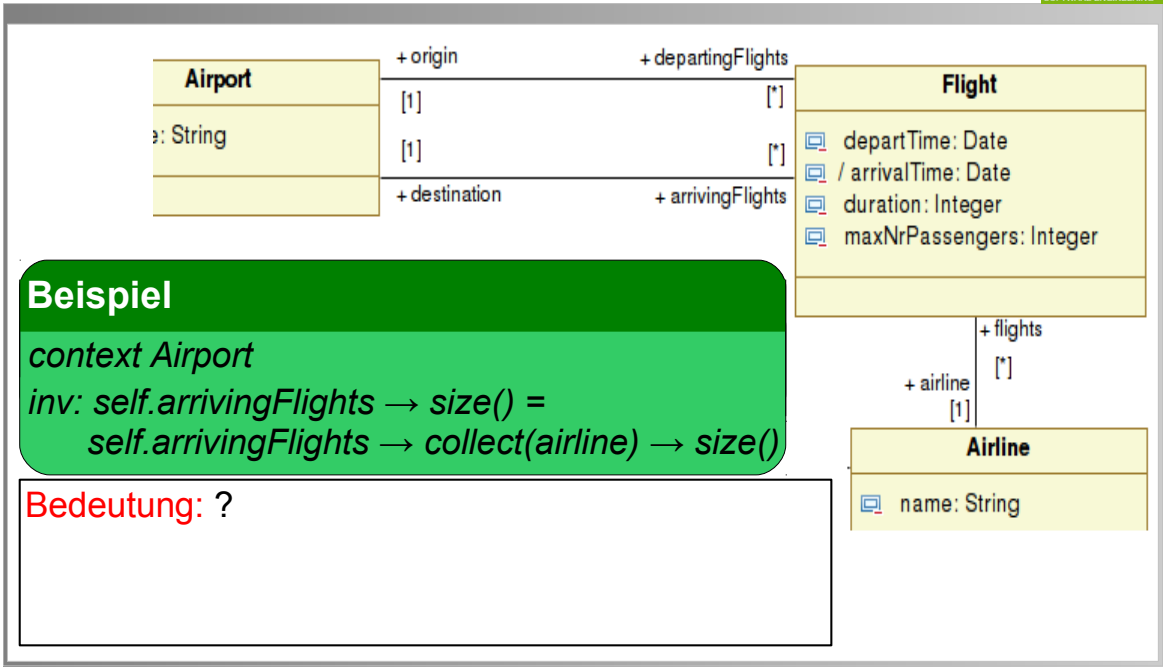
Beispiel

context Flight
inv: self.passengers → size()
<= self.maxNrPassengers



Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.6 – Collection Operations (S.28-32)

Collection-Operationen: *collect()* und *size()*



Beispiel

context Airport

inv: self.arrivingFlights → size() =
self.arrivingFlights → collect(airline) → size()

Bedeutung: ?

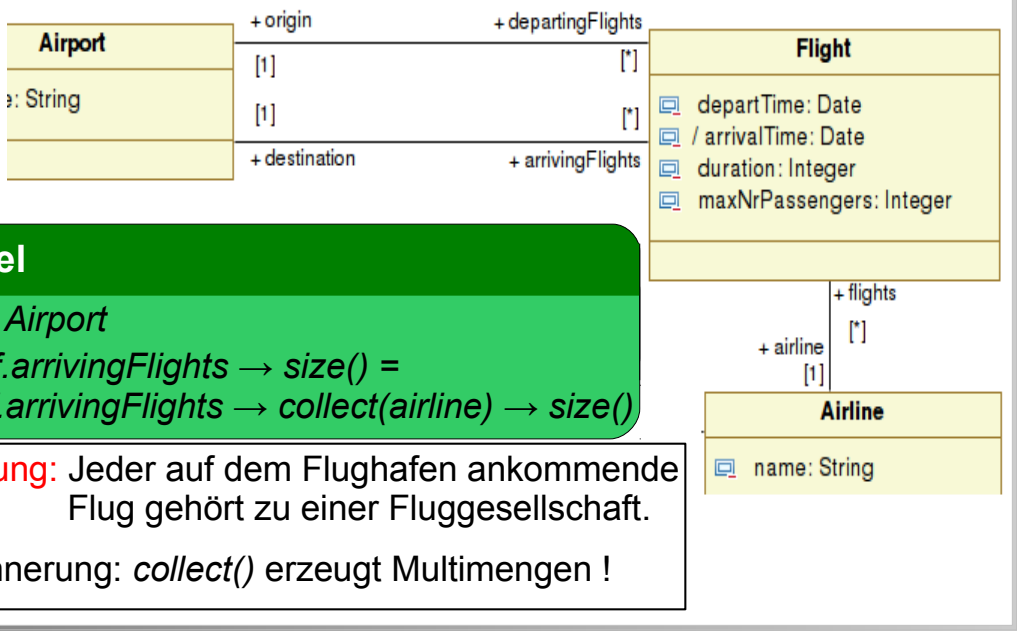
Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

Collection-Operationen: *collect()* und *size()*



Beispiel

context Airport

inv: self.arrivingFlights → size() =
self.arrivingFlights → collect(airline) → size()

Bedeutung: Jeder auf dem Flughafen ankommende Flug gehört zu einer Fluggesellschaft.

Zur Erinnerung: *collect()* erzeugt Multimengen !

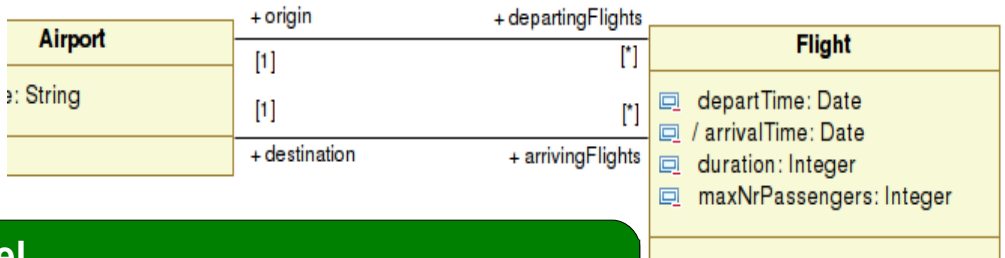
Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

Collection-Operationen: *collect()* und *size()*



Beispiel

context Airport

inv: self.arrivingFlights → size() =
self.arrivingFlights → collect(airline) → size()

Zur Diskussion: Welche Einschränkung in diesem Klassendiagramm ist besonders wichtig, damit dieser OCL- Ausdruck die erwünschte Bedeutung hat ?

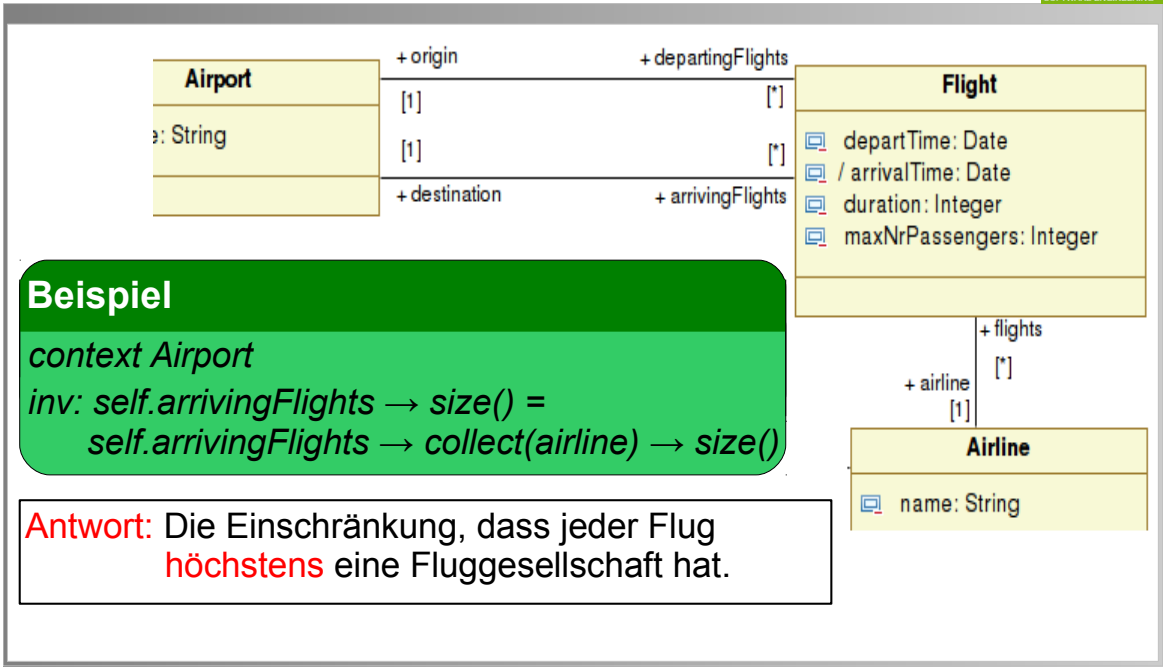
Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

Collection-Operationen: *collect()* und *size()*



Beispiel

context Airport

inv: self.arrivingFlights → size() =
self.arrivingFlights → collect(airline) → size()

Antwort: Die Einschränkung, dass jeder Flug höchstens eine Fluggesellschaft hat.

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

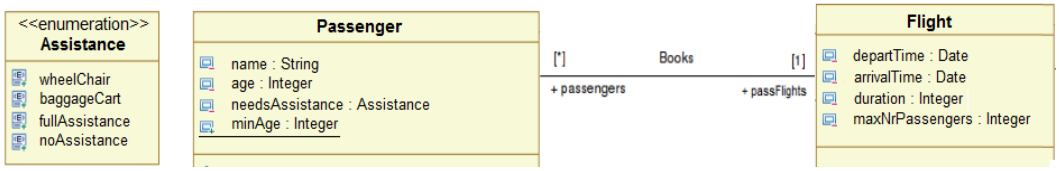
Operation *select()*

- Bekommt OCL-Ausdruck als Parameter übergeben.
- **Ergebnis: Subcollection** der verwendeten Collection.
- Liefert alle Elemente einer Collection, für die der Ausdruck **wahr** ist.

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.6 – Collection Operations (S.28-32)
- Absch. 7.6.1 – Select and Reject Operations (S.28-29)

Operation *select()*

- Bekommt OCL-Ausdruck als Parameter übergeben.
- **Ergebnis: Subcollection** der verwendeten Collection.
- Liefert alle Elemente einer Collection, für die der Ausdruck **wahr** ist.



Beispiel

context Flight

inv: self.passengers → select(needsAssistance) <> Assistance::noAssistance) → size() <= 10

Bedeutung: ?

Literatur:

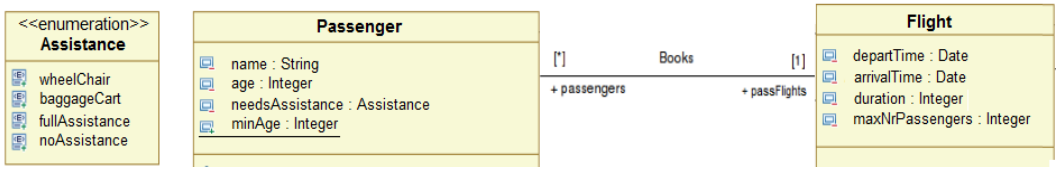
Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Absch. 7.6.1 – Select and Reject Operations (S.28-29)

Operation *select()*

- Bekommt OCL-Ausdruck als Parameter übergeben.
- **Ergebnis: Subcollection** der verwendeten Collection.
- Liefert alle Elemente einer Collection, für die der Ausdruck **wahr** ist.



Beispiel

context Flight

inv: self.passengers → select(needsAssistance) <> Assistance::noAssistance) → size() <= 10

Bedeutung: Die Anzahl der Passagiere eines Fluges, die Hilfe brauchen, ist kleiner oder gleich 10.

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Absch. 7.6.1 – Select and Reject Operations (S.28-29)

Operation *reject()*

- Verhält sich mengentheoretisch komplementär zu *select()*.
- Liefert alle Elemente einer Collection, für die der Ausdruck **falsch** ist.

Beispiel

context Flight
inv: passengers → select(needsAssistance <>
Assistance::noAssistance) → size() <= 10

- Beispiel: Anzahl der Passagiere, die Hilfe brauchen, ist kleiner oder gleich 10. Wie mit *reject()* spezifizieren ?

Beispiel

context Flight

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.6 – Collection Operations (S.28-32)
- Absch. 7.6.1 – Select and Reject Operations (S.28-29)

Operation *reject()*

- Verhält sich mengentheoretisch komplementär zu *select()*.
- Liefert alle Elemente einer Collection, für die der Ausdruck **falsch** ist.

Beispiel

```
context Flight
inv: passengers → select(needsAssistance <>
     Assistance::noAssistance) → size() <= 10
```

- Beispiel: Anzahl der Passagiere, die Hilfe brauchen, ist kleiner oder gleich 10. Wie mit *reject()* spezifizieren ?

Beispiel

```
context Flight
inv: passengers → reject(needsAssistance =
     Assistance::noAssistance) → size() <= 10
```


Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Absch. 7.6.1 – Select and Reject Operations (S.28-29)

Operation *forAll()*

- Nutzbar, um **Bedingung** zu **definieren.**

  → Muss von allen Elementen in Collection eingehalten werden.
- **Erhält OCL-Ausdruck** als Parameter.
- **Liefert booleschen Wert** zurück:
  - Wahr, wenn Bedingung von allen Elementen erfüllt wird.
  - Falsch, sonst.

Literatur:

Object Management Group: **OCL 2.4**

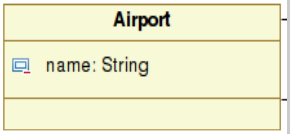
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Absch. 7.6.1 – ForAll Operation (S.30-31)

Operation *forAll()*: Beispiel

Beispiel

context Airport
inv: Airport.allInstances() → forAll (a1, a2 |
a1 <> a2 implies a1.name <> a2.name)



class.allInstances(): Collection mit allen Instanzen einer Klasse.

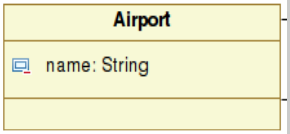
Bedeutung OCL-Ausdruck: *?*

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.6 – Collection Operations (S.28-32)
- Absch. 7.6.1 – ForAll Operation (S.30-31)

Operation *forAll()*: Beispiel

Beispiel

context Airport
inv: Airport.allInstances() → forAll (a1, a2 |
a1 <> a2 implies a1.name <> a2.name)



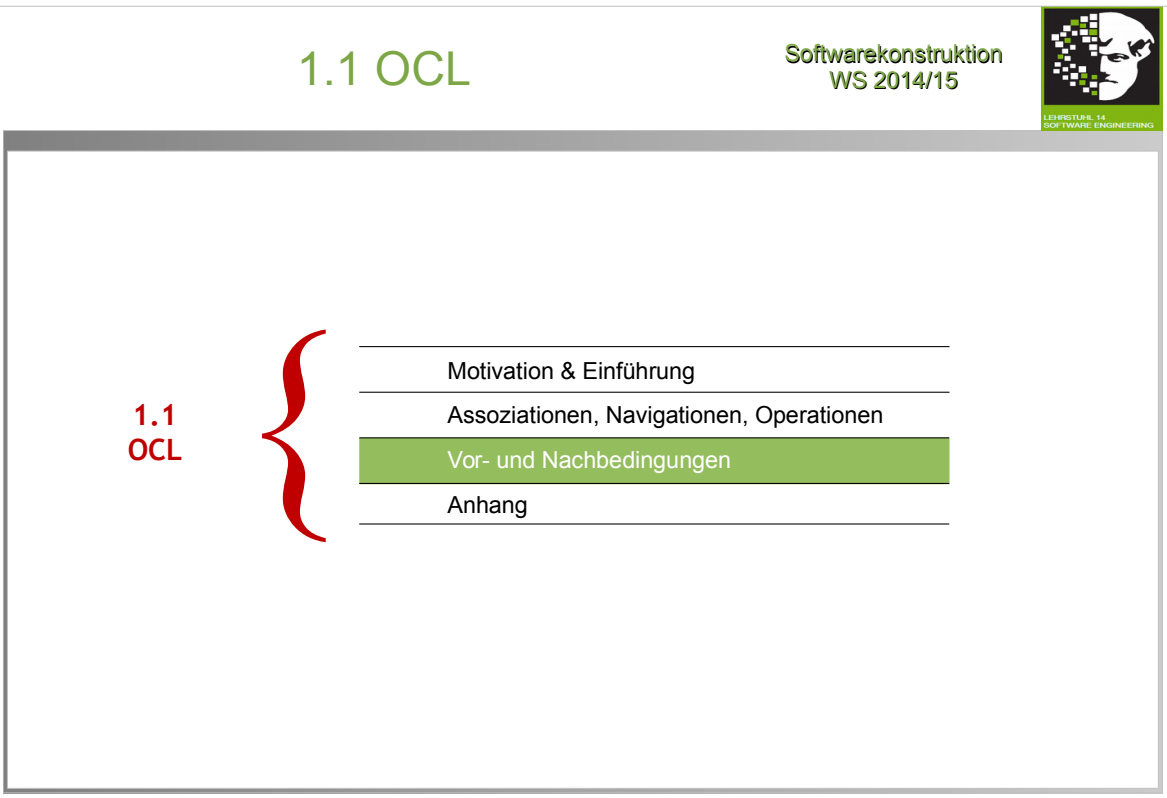
class.allInstances(): Collection mit allen Instanzen einer Klasse.

Bedeutung OCL-Ausdruck: Jeder Flughafenname ist einzigartig.

Äquivalent mit Operation *isUnique()*:

context Airport
inv : Airport.allInstances() → isUnique(name)

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.6 – Collection Operations (S.28-32)
- Absch. 7.6.1 – ForAll Operation (S.30-31)



Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)

Vor- und Nachbedingungen

- In Klassendiagrammen nur Syntax und Signatur einer Operation definierbar.
- **Semantik** einer Operation mittels **Vor- und Nachbedingungen** in OCL spezifizierbar.
- Vorbedingung:
  - Bedingungen von Argumenten und dem initialen Objektzustand müssen erfüllt werden.

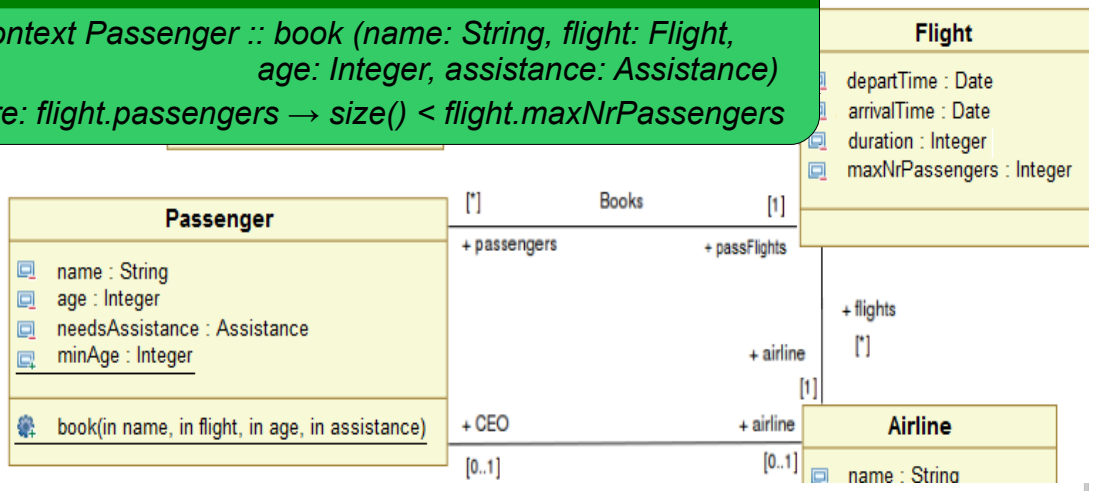
Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)

Vorbedingung: Beispiel

Beispiel

context Passenger :: book (name: String, flight: Flight, age: Integer, assistance: Assistance)

pre: flight.passengers → size() < flight.maxNrPassengers



Bedeutung: ?

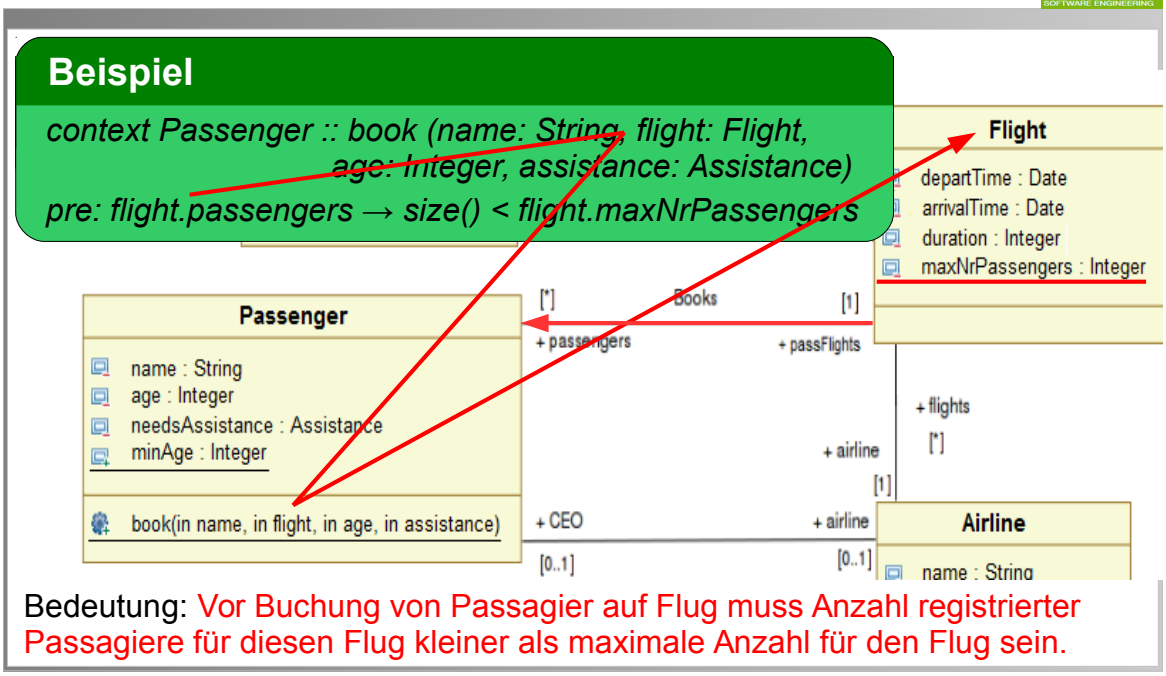
Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)

Vorbedingung: Beispiel



Bedeutung: Vor Buchung von Passagier auf Flug muss Anzahl registrierter Passagiere für diesen Flug kleiner als maximale Anzahl für den Flug sein.

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)

Vorbedingung: Beispiel

Beispiel

context Passenger :: book (name: String, flight: Flight, age: Integer, assistance: Assistance)

pre: flight.passengers → not exists (p: Passenger | p.age = age and p.name = name and p.needsAssistance = assistance)

Bedeutung: ?

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)

Vorbedingung: Beispiel

Beispiel

context Passenger :: book (name: String, flight: Flight, age: Integer, assistance: Assistance)

pre: flight.passengers → not exists (p: Passenger | p.age = age and p.name = name and p.needsAssistance = assistance)

Bedeutung:

Vor Ausführung von *book()*:

- existiert kein *Passenger*-Objekt, dessen Attribute die Werte aus *book* enthalten.

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)

Nachbedingungen

Nachbedingung:

- Bedingung, die **am Ende einer Operationsausführung** vom Rückgabewert, finalem Objektzustand, Argumenten und initialem Objektzustand erfüllt sein muss.
  - Unter Annahme, dass Vorbedingungen erfüllt sind.
- **Spezifiziert** beabsichtigte Ergebnisse und **Zustandsänderungen** (was), aber nicht wie sie geschehen (wie).
- Kann initialen Zustand eines Objektfelds mit Postfix-Notation *@pre* referenzieren (z.B. *flight.passengers@pre*).
- Kann Rückgabewert mit Schlüsselwort *result* referenzieren.

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)

Nachbedingung: Beispiel

Beispiel

context Passenger :: book (name: String, flight: Flight, age: Integer, assistance: Assistance)

post: flight.passengers → size() - flight.passengers@pre → size() = 1 and
flight.passengers → exists (p: Passenger | p.age = age and p.name = name and p.needsAssistance = assistance)

Bedeutung: ?

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)

Nachbedingung: Beispiel

Beispiel

context Passenger :: book (name: String, flight: Flight, age: Integer, assistance: Assistance)

post: flight.passengers → size() - flight.passengers@pre → size() = 1 and
flight.passengers → exists (p: Passenger | p.age = age and p.name = name and p.needsAssistance = assistance)

Bedeutung:

Nach Ausführung von *book()*:

- erreicht die Assoziation *passengers* ein zusätzliches Objekt und
- existiert ein *Passenger*-Objekt, dessen Attribute die Werte aus *book* enthalten.

Literatur:

Object Management Group: **OCL 2.4**

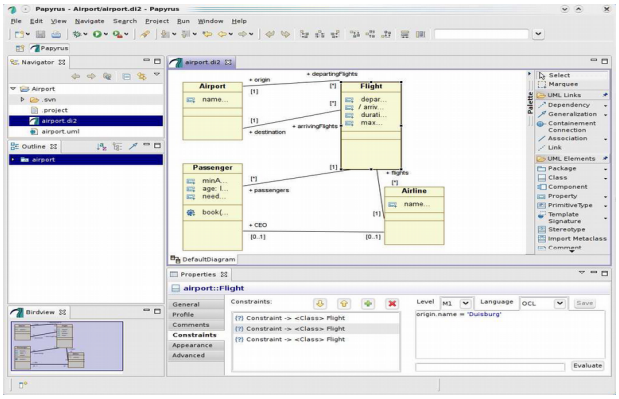
http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.3.4 – Pre- and Postconditions (S.8-9)

Werkzeugunterstützung

Verschiedene UML-Editoren unterstützen Einbindung von OCL-Bedingungen.

Zum Beispiel: Papyrus.



- Frei erhältliches Open-Source Werkzeug für Modellierung mit UML 2.0, s. http://www.papyrusuml.org/
- Basiert auf Entwicklungs-umgebung Eclipse.
- Erweiterbare Architektur von Papyrus erlaubt Hinzufügen von Diagrammen, neuen Codegeneratoren, etc.
- Erlaubt Einbinden von OCL-Bedingungen.

Zusammenfassung: Object Constraint Language (OCL)

Logik-basierte Notation für **Einschränkungen** in **UML**-Modellen.

- **Bedingungen** an Ausführung der modellierten Systemteile formulieren und analysieren: **Invarianten, Collections, Vor-** und **Nachbedingungen, Wächterbedingung**.
- Erlaubt **Qualitätssicherung** der Software bereits im Design auf **Modellebene**, wo Fehler kostengünstig repariert werden können.

Vorteile:

- Automatische Verifikation der Bedingung.
- **Werkzeuge** erzeugen aus OCL **Assertions** in **Java**.

Zusammenfassung: Object Constraint Language (OCL)

Nächste Abschnitte:
Alternative Modellierungsnotationen im Rahmen der modellbasierten Softwareentwicklung.

Abschnitt 1.3: Modellierung von Geschäftsprozessen mit Ereignisbasierte Prozessketten (EPKs).

- Als Alternative zu UML Aktivitätsdiagrammen.

1.1 OCL: Anhang

Weitere Informationen zum selbständigen Nachlesen / Nachschlagen.

1.1 OCL

- Motivation & Einführung
- Assoziationen, Navigationen, Operationen
- Vor- und Nachbedingungen
- Anhang

Wie verwende ich OCL ?

OCL-Ausdrücke: an UML-Modell gebunden.

Beschreiben Einschränkungen für Elemente des Modells, zu dem sie gehören.

Zwei Arten von **Einschränkungen** spezifizieren und verifizieren:

- **Fortlaufende** Zustandsbeschränkung (mit Invarianten).
- Zustandsbeschränkungen **vor** bzw. **nach Methodenaufruf** (mit Vor- und Nachbedingungen).

Beispiel

context Student
inv: self.MatNr >= 10000

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 3.6.1 (S.106-107)

Seemann, Gudenberg: **Software-Entwurf mit UML 2**
http://www.ub.tu-dortmund.de/katalog/titel/1223020

- Abschnitt 14.5.1 (S.281-282)

Wie verwende ich OCL ?

OCL-Ausdrücke: an UML-Modell gebunden.

Beschreiben Einschränkungen für Elemente des Modells, zu dem sie gehören.

Zwei Arten von **Einschränkungen** spezifizieren und verifizieren:

- **Fortlaufende** Zustandsbeschränkung (mit Invarianten).
- Zustandsbeschränkungen **vor** bzw. **nach Methodenaufruf** (mit Vor- und Nachbedingungen).

Beispiel

context Student :: nimmtTeilAn (VL:Vorlesung) : void
pre: VL.FreiePlätze > 0

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 3.6.1 (S.106-107)

Seemann, Gudenberg: **Software-Entwurf mit UML 2**
http://www.ub.tu-dortmund.de/katalog/titel/1223020

- Abschnitt 14.5.1 (S.281-282)

Wie verwende ich OCL ?

OCL-Ausdrücke: an UML-Modell gebunden.

Beschreiben Einschränkungen für Elemente des Modells, zu dem sie gehören.

Zwei Arten von **Einschränkungen** spezifizieren und verifizieren:

- **Fortlaufende** Zustandsbeschränkung (mit Invarianten).
- Zustandsbeschränkungen **vor** bzw. **nach Methodenaufruf** (mit Vor- und Nachbedingungen).

Beispiel

context Student :: nimmtTeilAn (VL:Vorlesung):void
post: VL.FreiePlätze = VL.FreiePlätze@pre - 1

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 3.6.1 (S.106-107)

Seemann, Gudenberg: **Software-Entwurf mit UML 2**
http://www.ub.tu-dortmund.de/katalog/titel/1223020

- Abschnitt 14.5.1 (S.281-282)

OCL Syntax

Folgende **Operationen** im OCL-Ausdruck benutzbar:

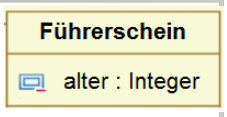
- **+ (r:Real) : Real :** Summe von *self* und r
- **- (r:Real) : Real :** Division von *self* und r
- *** (r:Real) : Real :** Produkt von *self* und r
- **/ (r:Real) : Real :** Quotient von *self* und r
- **abs() : Real :** Absoluter Wert von *self*
- **max (r : Real) :** Maximum von *self* und r
- **min (r : Real) :** Minimum von *self* und r
- **< (r : Real) : Boolean :** True falls *self* kleiner als r ist.
- **> (r : Real) : Boolean :** True falls *self größer* als r ist.
- ...

OCL Spezifikation der OMG Group: S. 157-158

OCL Syntax

Folgende **Operationen** im OCL-Ausdruck benutzbar:

- **< (r : Real) : Boolean :** True falls *self* kleiner als r ist.
- **> (r : Real) : Boolean :** True falls *self größer* als r ist.
- …



Für einen Führerschein muss man älter als 16 Jahre alt sein.

Beispiel

context Führerschein
inv: alter > 16

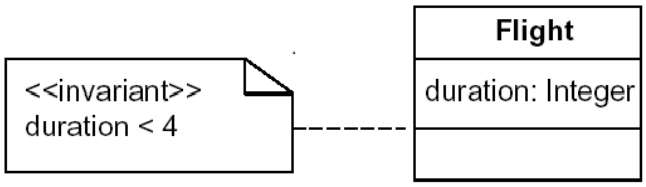
OCL Spezifikation der OMG Group: S. 157-158

Gebrauch von self

Folgende Notationen sind äquivalent:

Beispiel
context Flug
inv: self.duration < 4

Beispiel
context Flug
inv: duration < 4



Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.3.1 – Self (S.7)

Um Missverständnisse zu vermeiden, ist in der Übung die vollständigere Notation (inkl. „self.“) vorgesehen.

Primitive Typen

Type	Description	Values	Operators and Operations
Boolean		true, false	=, <>, and, or, xor, not, implies, if-then-else-endif (note 2)
Integer	A whole number of any size	-1, 0, 1, ...	=, <>, >, <, >=, <=, *, +, - (unary), - (binary), / (real), abs(), max(b), min(b), mod(b), div(b)
Real	A real number of any size	1.5, ...	=, <>, >, <, >=, <=, *, +, - (unary), - (binary), /, abs(), max(b), min(b), round(), floor()
String	A string of characters	'a', 'John'	=, <>, size(), concat(s2), substring(lower, upper) (1<=lower<=upper<=size), toReal(), toInteger()

Notes:
1) Operations indicated with parenthesis are applied with ".", but the parenthesis may be omitted.
2) Example: title = (if isMale then 'Mr.' else 'Ms.' endif)

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.4 – Basic Values and Types (S.10-11)

Wiederholung: Arten von Kollektionen

- Collection von Objekten:
  - **Set**:
    - Jedes Element kommt nur einmal vor.
    - Einfaches Navigieren einer Assoziation liefert Set zurück.
  - **Bag**:
    - Gleiche Elemente dürfen mehrmals vorkommen.
  - **OrderedSet**:
    - Satz von geordneten Elementen.
  - **Sequence**:
    - Bag in dem Elemente geordnet sind.

Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.5.11 – Collections (S.24-25)

Collections und Tuples

Description	Syntax	Examples
Abstract collection of elements of type T	Collection(T)	
Unordered collection, no duplicates	Set(T)	Set{1 , 2}
Ordered collection, duplicates allowed	Sequence(T)	Sequence {1, 2, 1} Sequence {1..4} (same as {1,2,3,4})
Ordered collection, no duplicates	OrderedSet(T)	OrderedSet {2, 1}
Unordered collection, duplicates allowed	Bag(T)	Bag {1, 1, 2}
Tuple (with named parts)	Tuple(field1: T1, ... fieldn : Tn)	Tuple {age: Integer = 5, name: String = 'Joe' } Tuple {name = 'Joe', age = 5}

Note 1: They are *value types*: "=" and "<>" compare values and not references.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.5.11 – Collections (S.24-25)
- Abschnitt 7.5.15 – Tuples (S.27-28)

Operations on Collection (T)

Operation	Description
size(): Integer	The number of elements in this collection (*self*)
isEmpty(): Boolean	size = 0
notEmpty(): Boolean	size > 0
includes(object: T): Boolean	True if *object* is an element of *self*
excludes(object: T): Boolean	True if *object* is not an element of *self*
count(object: T): Integer	The number of occurrences of *object* in *self*
includesAll(c2: Collection(T)): Boolean	True if *self* contains all the elements of *c2*
excludesAll(c2: Collection(T)): Boolean	True if *self* contains none of the elements of *c2*
sum(): T	The addition of all elements in *self* (T must support "+")
product(c2: Collection(T2)) : Set(Tuple(first:T, second:T2))	The cartesian product operation of *self* and *c2*.

Note: Operations on collections are applied with "->" and not "."

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.7 (ab S.165)
- Abschnitt 11.7.1 – Collection (S.165-167)

Iterator Expressions on Collection (T)

Iterator expression	Description
iterate(iterator: T; accum: T2 = init | body) : T2	Returns the final value of an accumulator that, after initialization, is updated with the value of the *body* expression for every element in the *source* collection.
exists(iterators | body) : Boolean	True if *body* evaluates to true for at least one element in the *source* collection. Allows multiple iterator variables.
forAll(iterators | body): Boolean	True if *body* evaluates to true for each element in the source collection. Allows multiple iterator variables.
one(iterator| body): Boolean	True if there is exactly one element in the *source* collection for which *body* is true
isUnique(iterator| body): Boolean	Results in true if *body* evaluates to a different value for each element in the *source* collection.
any(iterator | body): T	Returns any element in the source collection for which *body* evaluates to true. The result is null if there is none.
collect(iterator| body): Collection(T2)	The Collection of elements resulting from applying *body* to every member of the *source* set.

Note: The iterator variable declaration can be omitted when there is no ambiguity.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.9 (ab S.177)
- Abschnitt 11.9.1 – Collection (S.177-179)

Iterator expression on Collection (T)

Iterator expression	Description
select(iterator | body): Collection(T)	The Collection of elements of the *source* collection for which *body* is true. The result collection is of the same type of the *source* collection.
reject(iterator | body): Collection(T)	The Collection of elements of the *source* collection for which *body* is false. The result collection is of the same type of the *source* collection.
collectNested(iterator | body): CollectionWithDuplicates(T2)	The Collection of elements (allowing duplicates) that results from applying *body* (of type T2) to every member of the *source* collection. The result is not flattened. Conversions: Set -> Bag, OrderedSet -> Sequence.
sortedBy(iterator | body): OrderedCollection(T)	Returns an ordered Collection of all the elements of the *source* collection by ascending order of the value of the *body* expression. The type T2 of the *body* expression must support "<". Conversions: Set -> OrderedSet, Bag -> Sequence.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.9 (ab S.177)
- Abschnitt 11.9.1 – Collection (S.177-179)

Operations on Set (T) I

Operation	Description
=(s: Set(T)) : Boolean	Do *self* and *s* contain the same elements?
union(s: Set(T)): Set(T)	The union of *self* and *s*.
union(b: Bag(T)): Bag(T)	The union of *self* and bag *b*.
intersection(s: Set(T)): Set(T)	The intersection of *self* and *s*.
intersection(b: Bag(T)): Set(T)	The intersection of *self* and *b*.
-(s: Set(T)) : Set(T)	The elements of *self*, which are not in *s*.
including(object: T): Set(T)	The set containing all elements of *self* plus *object*.
excluding(object: T): Set(T)	The set containing all elements of *self* minus *object*.
symmetricDifference(s: Set(T)): Set(T)	The set containing all the elements that are in *self* or *s*, but not in both.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.7 (ab S.165)
- Abschnitt 11.7.2 – Set (S.167-169)

Operations on Set (T) II

Operation	Description
flatten() : Set(T2)	If T is a collection type, the result is the set with all the elements of all the elements of *self*; otherwise, the result is *self*.
asOrderedSet(): OrderedSet(T)	OrderedSet with elements from *self* in undefined order.
asSequence(): Sequence(T)	Sequence with elements from *self* in undefined order.
asBag(): Bag(T)	Bag will all the elements from *self*.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.7 (ab S.165)
- Abschnitt 11.7.2 – Set (S.167-169)

Operations on Bag (T)

Operation	Description
=(bag: Bag(T)) : Boolean	True if *self* and *bag* contain the same elements, the same number of times.
union(bag: Bag(T)): Bag(T)	The union of *self* and *bag*.
union(set: Set(T)): Bag(T)	The union of *self* and *set*.
intersection(bag:Bag(T)): Bag(T)	The intersection of *self* and *bag*.
intersection(set: Set(T)): Set(T)	The intersection of *self* and *set*.
including(object: T): Bag(T)	The bag with all elements of *self* plus *object*.
excluding(object: T): Bag(T)	The bag with all elements of *self* without *object*.
flatten() : Bag(T2)	If T is a collection type: bag with all the elements of all the elements of *self*; otherwise: *self*.
asSequence(): Sequence(T)	Seq. with elements from *self* in undefined order.
asSet(): Set(T)	Set with elements from *self*, without duplicates.
asOrderedSet(): OrderedSet(T)	OrderedSet with elements from *self* in undefined order, without duplicates.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.7 (ab S.165)
- Abschnitt 11.7.4 – Bag (S.171-173)

Operations on Sequence (T) I

Operation	Description
=(s: Sequence(T)) : Boolean	True if *self* contains the same elements as *s*, in the same order.
union(s: Sequence(T)): Sequence(T)	The sequence consisting of all elements in *self*, followed by all elements in *s*.
flatten() : Sequence(T2)	If T is a collection type, the result is the set with all the elements of all the elements of *self*; otherwise, it's *self*.
append(object: T): Sequence(T)	The sequence with all elements of *self*, followed by *object*.
prepend(obj: T): Sequence(T)	The sequence with *object*, followed by all elements in *self*.
insertAt(index : Integer, object : T) : Sequence(T)	The sequence consisting of *self* with *object* inserted at position *index* (1<=index<=size+1)
subSequence(lower : Integer, upper: Integer) : Sequence(T)	The sub-sequence of *self* starting at index *lower*, up to and including index *upper* (1<=lower<=upper<=size)

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.7 (ab S.165)
- Abschnitt 11.7.5 - Sequence (S.174-176)

Operations on Sequence (T) II

Operation	Description
at(i : Integer) : T	The *i*-th element of *self* (1<=i<=size)
indexOf(object : T) : Integer	The index of *object* in *self*.
first() : T	The first element in *self*.
last() : T	The last element in *self*.
including(object: T): Sequence(T)	The sequence containing all elements of *self* plus *object* added as last element
excluding(object: T): Sequence(T)	The sequence containing all elements of *self* apart from all occurrences of *object*.
asBag(): Bag(T)	The Bag containing all the elements from *self*, including duplicates.
asSet(): Set(T)	The Set containing all the elements from *self*, with duplicates removed.
asOrderedSet(): OrderedSet(T)	An OrderedSet that contains all the elements from *self*, in the same order, with duplicates removed.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.7 (ab S.165)
- Abschnitt 11.7.5 - Sequence (S.174-176)

Operations on OrderedSet (T)

Operation	Description
append(object: T): OrderedSet(T)	The set of elements, consisting of all elements of *self*, followed by *object*.
prepend(object: T): OrderedSet(T)	The sequence consisting of *object*, followed by all elements in *self*.
insertAt(index : Integer, object : T) : OrderedSet(T)	The set consisting of *self* with *object* inserted at position *index*.
subOrderedSet(lower : Integer, upper : Integer) : OrderedSet(T)	The sub-set of *self* starting at number *lower*, up to and including element number *upper* (1<=lower<=upper<=size).
at(i : Integer) : T	The *i*-th element of *self* (1<=i<=size).
indexOf(object : T) : Integer	The index of *object* in the sequence.
first() : T	The first element in *self*.
last() : T	The last element in *self*.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.7 (ab S.165)
- Abschnitt 11.7.3 – OrderedSet (S.169-171)

Special Types

Type	Description
OclAny	Supertype for all types except for collection and tuple types. All classes in a UML model inherit all operations defined on OclAny.
OclVoid	The type OclVoid is a type that conforms to all other types. It has one single instance called *null*. Any property call applied on *null* results in *OclInvalid*, except for the operation oclIsUndefined(). A collection may have *null's*.
OclInvalid	The type OclInvalid is a type that conforms to all other types. It has one single instance called *invalid*. Any property call applied on *invalid* results in *invalid*, except for the operations oclIsUndefined() and oclIsInvalid().
OclMessage	Template type with one parameter T to be substituted by a concrete operation or signal type. Used in some postconditions that need to constrain the messages sent during the operation execution.
OclType	Meta type.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.2 (S.152)

Operations defined in OclAny

Operation	Description
=(object2 : OclAny) : Boolean	True if *self* is the same object as *object2*.
<>(object2 : OclAny) : Boolean	True if *self* is a different object from *object2*.
oclIsNew() : Boolean	Can only be used in a postcondition. True if *self* was created during the operation execution.
oclAsType(t : OclType) : OclType	Cast (type conversion) operation. Useful for downcast.
oclIsTypeOf(t: OclType) : Boolean	True if *self* is of type *t*.
oclIsKindOf(t : OclType) : Boolean	True if *self* is of type *t* or a subtype of *t*.
oclIsInState(s : OclState) : Boolean	True if *self* is in state *s*.
oclIsUndefined() : Boolean	True if *self* is equal to *null* or *invalid*.
oclIsInvalid() : Boolean	True if *self* is equal to *invalid*.
allInstances() : Set(T)	Static operation that returns all instances of a classifier.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.3 (ab S.153)
- Abschnitt 11.3.1 – OclAny (S.153-154)

Operations defined in OclMessage

Operation	Description
hasReturned() : Boolean	True if type of template parameter is an operation call, and the called operation has returned a value.
result()	Returns the result of the called operation, if type of template parameter is an operation call, and the called operation has returned a value.
isSignalSent() : Boolean	Returns true if the OclMessage represents the sending of a UML Signal.
isOperationCall() : Boolean	Returns true if the OclMessage represents the sending of a UML Operation call.
parameterName	The value of the message parameter.

entnommen aus http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf

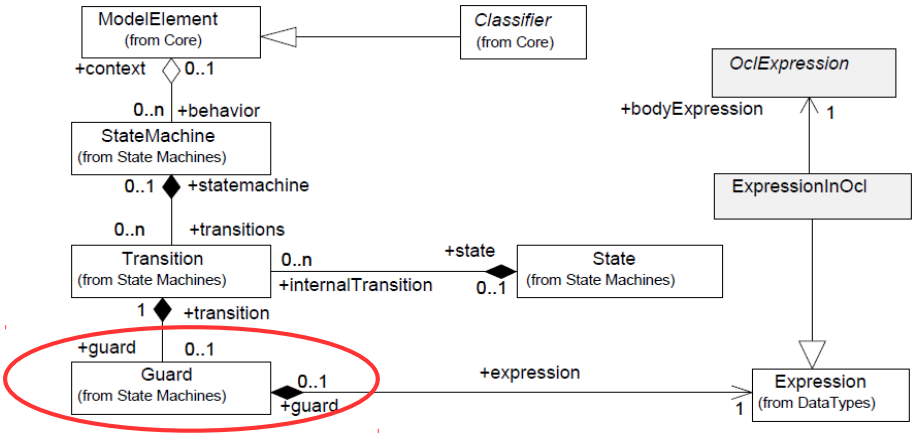
Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 11.3 (ab S.153)
- Abschnitt 11.3.4 – OclMessage (S.153-154)

Wächterbedingung in StateCharts: Definition im UML Metamodell

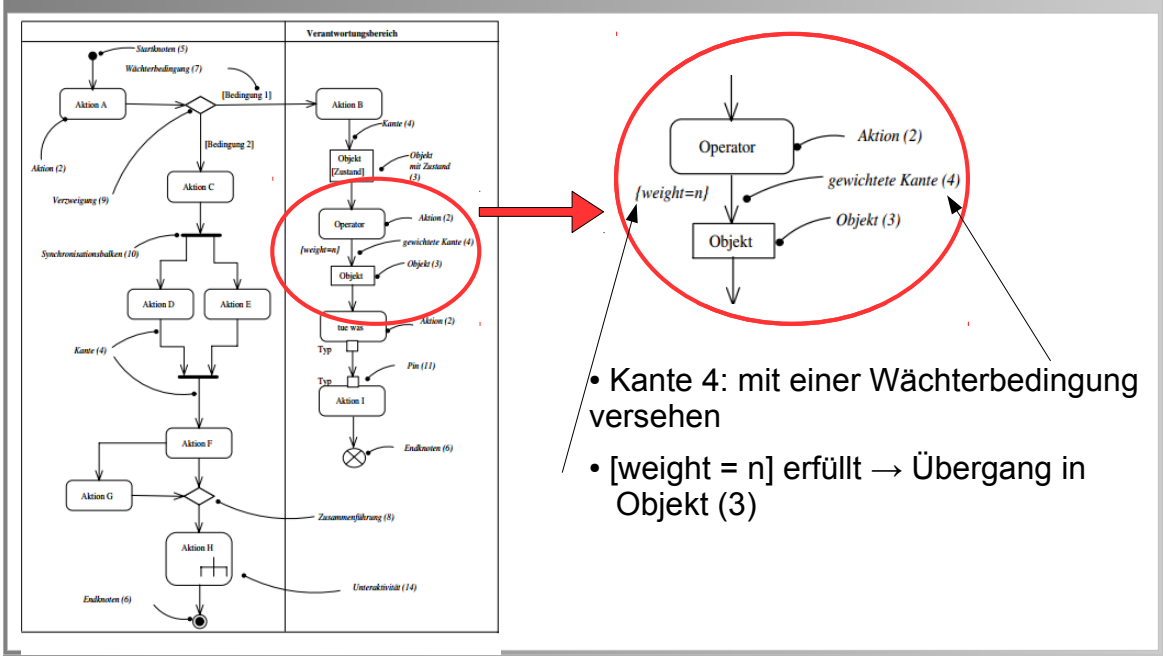


Wächterbedingung (Guard condition):

- Boolesche Bedingung: validiert Auslösen eines Ereignisauftritts.

OCL Spezifikation der OMG Group: S. 192-193

Wächterbedingung in StateCharts: Beispiel



- Kante 4: mit einer Wächterbedingung versehen
  - [weight = n] erfüllt → Übergang in Objekt (3)

Software-Entwurf mit UML 2

- Abbildung: S. 301
- Erläuterung: S. 303

Guard condition: Regeln

```
context ExpressionInOcl
inv: not self.guard.transition.getStateMachine().context.oclIsUndefined()   -- A
   and
   self.guard.transition.getStateMachine().context.oclIsKindOf(Classifier)   -- B
   implies
    contextualClassifier =
      self.guard.transition.getStateMachine().context.oclAsType(Classifier)
   and
   self.bodyExpression.type.name = 'Boolean'
```

Kontext

Klassifier

1. Statemachine, in dem guard auftaucht, muss **Kontext** (A) haben, das ist ein **Klassifier** (B).
   - Kontextueller Klassifier: Besitzer des Statemachine
   - **Typ** des OCL Ausdrucks: **boolean**
2. @pre ist in einem guard nicht erlaubt.

OCL Spezifikation der OMG Group: S. 192-193

Die *collect* Operation II

- **Resultierende Collection beinhaltet andere Objekte** als ursprüngliche Collection.
- Wenn **Quelle Set** ist, so ist **resultierende Collection** kein Set, sondern **Bag.**
- Wenn **Quelle Sequence** oder OrderedSet ist, so ist neue Collection **Sequence**.

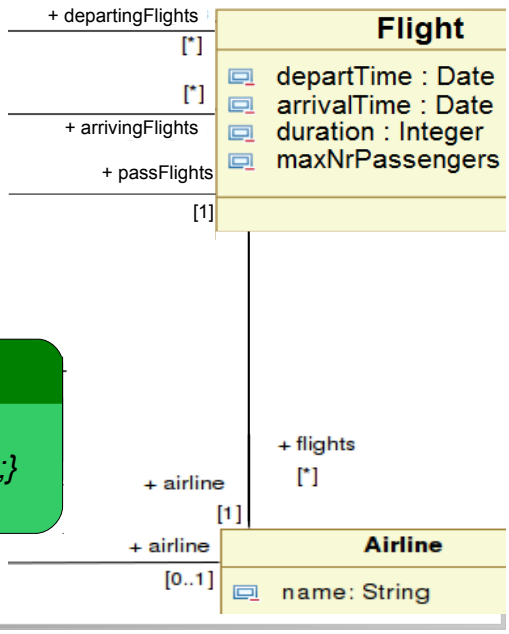
Collection-Operationen: *collect()* - Objekte

Kann benutzt werden, um neue Collections aus Objekten am Ende der Assoziation zu bilden, z.B.:

arrivingFlights → collect(airline).

Bedeutung (in Pseudocode)

```
Collection<Airline> c = new Collection();
foreach (f: arrivingFlights) {c.add(f.airline);}
return c;
```



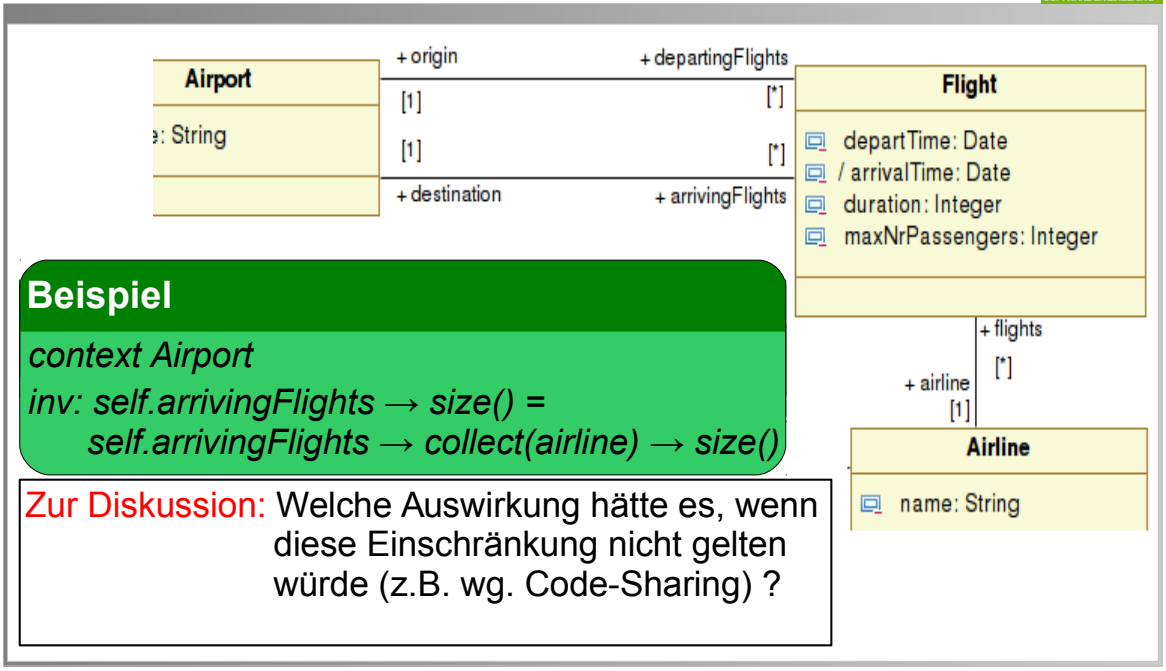
Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

collect(): Beispiel



Beispiel

context Airport

inv: self.arrivingFlights → size() =
self.arrivingFlights → collect(airline) → size()

Zur Diskussion: Welche Auswirkung hätte es, wenn diese Einschränkung nicht gelten würde (z.B. wg. Code-Sharing) ?

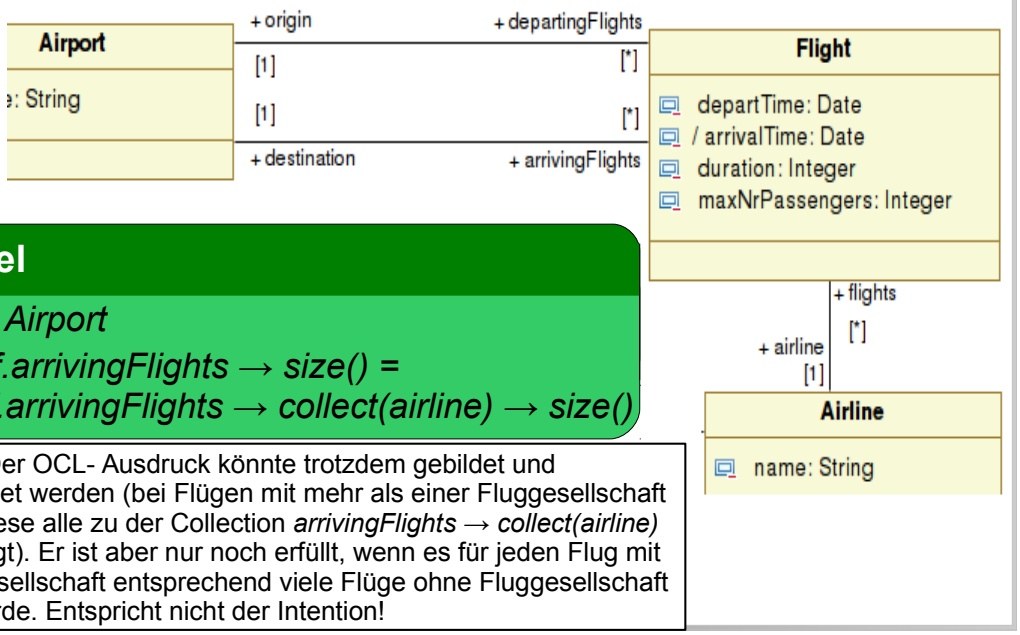
Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

collect(): Beispiel



Beispiel

context Airport

inv: self.arrivingFlights → size() =
self.arrivingFlights → collect(airline) → size()

Antwort: Der OCL- Ausdruck könnte trotzdem gebildet und ausgewertet werden (bei Flügen mit mehr als einer Fluggesellschaft werden diese alle zu der Collection *arrivingFlights → collect(airline)* hinzugefügt). Er ist aber nur noch erfüllt, wenn es für jeden Flug mit >1 Fluggesellschaft entsprechend viele Flüge ohne Fluggesellschaft geben würde. Entspricht nicht der Intention!

Literatur:

Object Management Group: **OCL 2.4**

http://www.omg.org/spec/OCL/2.4/PDF

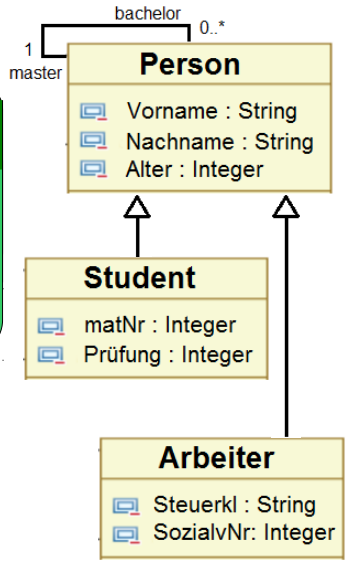
- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

collect(): Beispiel (2)

Beispiel

Context Person
inv:
self.bachelor->select(s|s.isKindOf(Student))->collect(Prüfung)-> size() >= self.Prüfung

Bedeutung: ?



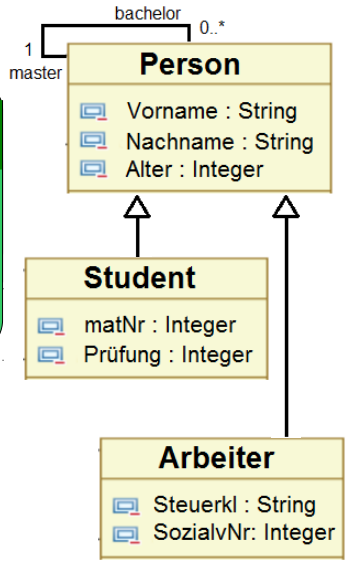
Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

collect(): Beispiel (2)

Beispiel

Context Person
inv:
self.bachelor->select(s|s.isKindOf(Student))->collect(Prüfung)-> size() >= self.Prüfung

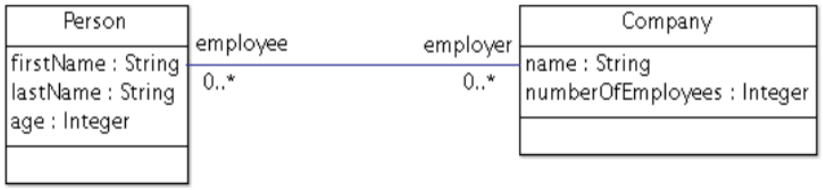
Bedeutung: Mehrere Bachlorstudenten haben mehrere Prüfungen als ein Masterstudent geschrieben.



Literatur:
Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF
- Abschnitt 7.6 – Collection Operations (S.28-32)
- Abschnitt 7.6.2 – Collect Operation (S.29-30)

Weiteres select() Beispiel

Keiner der Mitarbeiter einer Firma darf älter als 67 sein.

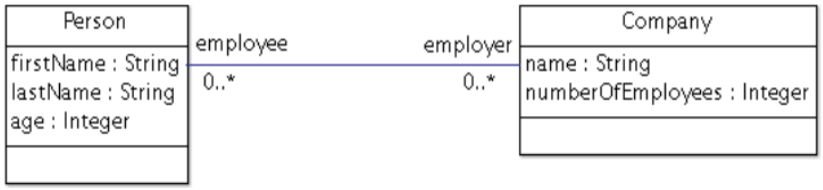


Hilfe:
Mit **isEmpty()** kann man überprüfen, ob eine Collection **leer** ist.

Lösung

Weiteres select() Beispiel

Keiner der Mitarbeiter einer Firma darf älter als 67 sein.



Hilfe:
Mit **isEmpty()** kann man überprüfen, ob eine Collection **leer** ist.

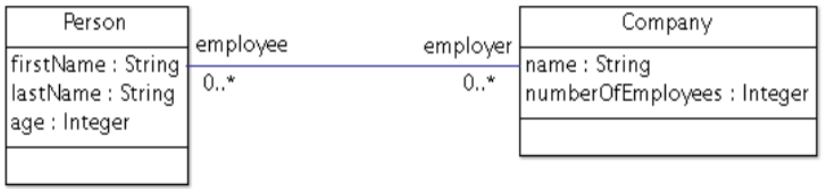
Lösung

context Company

inv: self.employee → select(age > 67) → isEmpty()

Weiteres reject() Beispiel

Keiner der Mitarbeiter einer Firma darf älter als 67 sein.



Lösung mit select()

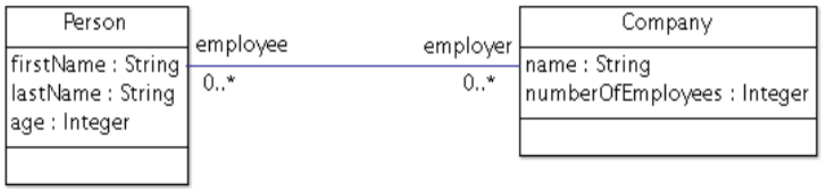
context Company

inv: self.employee → select(age > 67) → isEmpty()

Lösung mit reject() ?

Weiteres reject() Beispiel

Keiner der Mitarbeiter einer Firma darf älter als 67 sein.



Lösung mit select()

context Company

inv: self.employee → select(age > 67) → isEmpty()

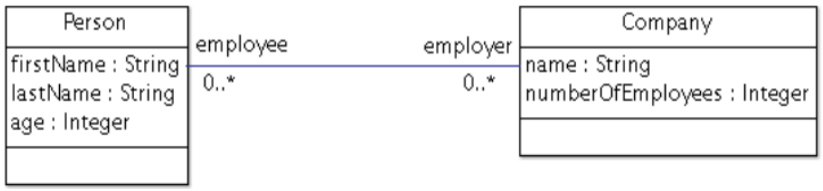
Lösung mit reject() ?

context Company

inv: self.employee → reject(age <= 67) → isEmpty()

Weiteres forAll() Beispiel

Keiner der Mitarbeiter einer Firma darf älter als 67 sein.



Lösung mit select()

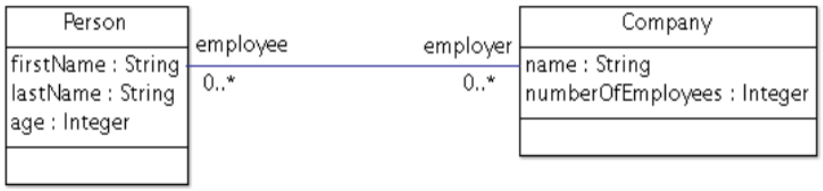
context Company

inv: self.employee → select(age > 67) → isEmpty()

Lösung mit forAll() ?

Weiteres forAll() Beispiel

Keiner der Mitarbeiter einer Firma darf älter als 67 sein.



Lösung mit select()

context Company

inv: self.employee → select(age > 67) → isEmpty()

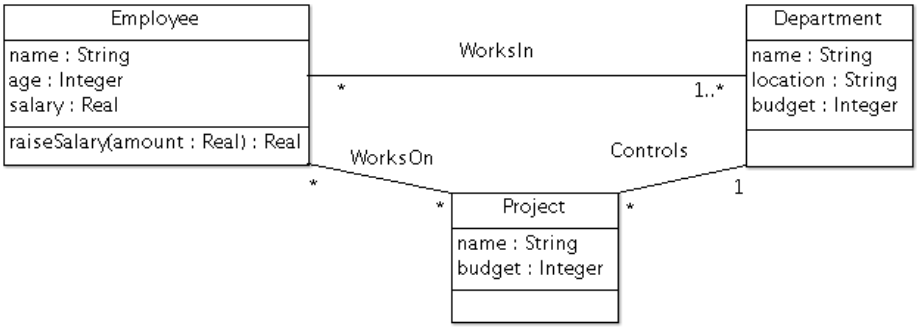
Lösung mit forAll() ?

context Company

inv: self.employee → forAll(age <= 67)

Weitere Beispiele

Mitarbeiter, die an mehr Projekten als andere Mitarbeiter des gleichen Department arbeiten, bekommen eine höhere Vergütung.

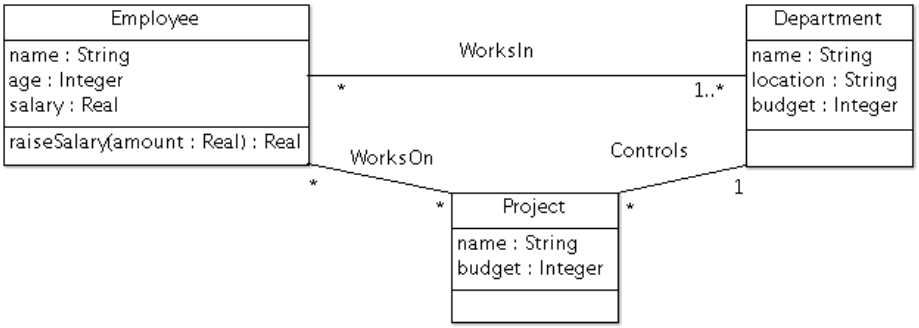


Lösung

context *Department*
inv: *self.employee → forAll(e1, e2 | e1.project → size > e2.project → size* ***implies*** *e1.salary > e2.salary)*

Weitere Beispiele

Wenn der Wert (amount) positiv ist, erhöhe das Gehalt um diesen Wert.

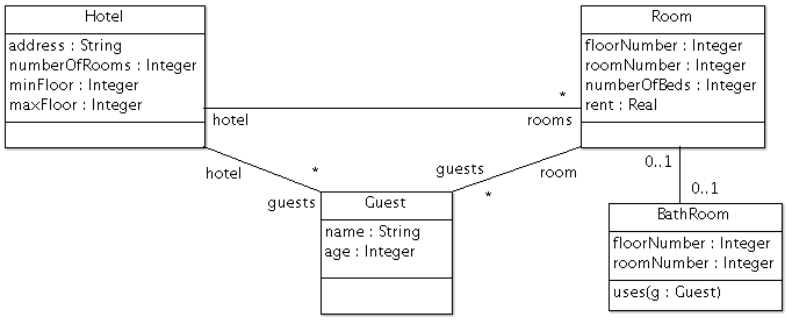


Lösung

context *Employee::raiseSalary(amount: Real): Real*
pre: amount > 0
post: self.salary = self.salary**@pre** + amount **and** result = self. salary

Weitere Beispiele

Die Anzahl der Gäste in einem Raum muss kleiner gleich der Anzahl der Betten sein, es sei denn einer der Gäste ist jünger als 5 Jahre, dann darf die Anzahl der Gäste einen mehr betragen.

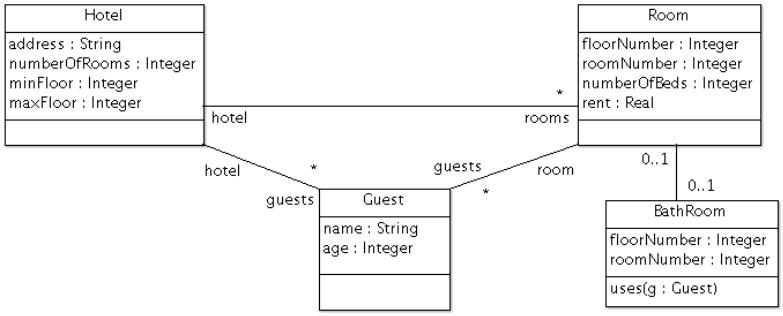


Lösung

***context** Room*
inv:** guests → size <= numberOfBeds **or
*(guests → size = numberOfBeds + 1 **and** guests → exists(g: Guest | g.age <= 4)*

Weitere Beispiele

Was könnte folgender OCL-Ausdruck bedeuten?



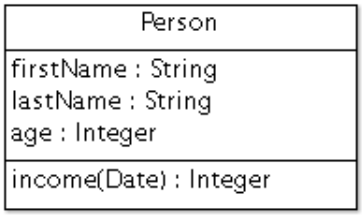
Lösung

context *Bathroom:uses(g: Guest)*
pre: if *room → notEmpty()* ***then*** *room.guests → includes(g)*
else *g.room.floorNumber = self.floorNumber* ***endif***

Antwort: Wenn ein Gast ein Badezimmer benutzt, dann ist es entweder das Badezimmer in seinem Raum oder ein Badezimmer auf dem selben Flur, welches keinem speziellen Raum zugeordnet ist.

Nachbedingung: Beispiel

Das Einkommen einer Person darf nicht 5000 Euro überschreiten.



Zur Erinnerung:

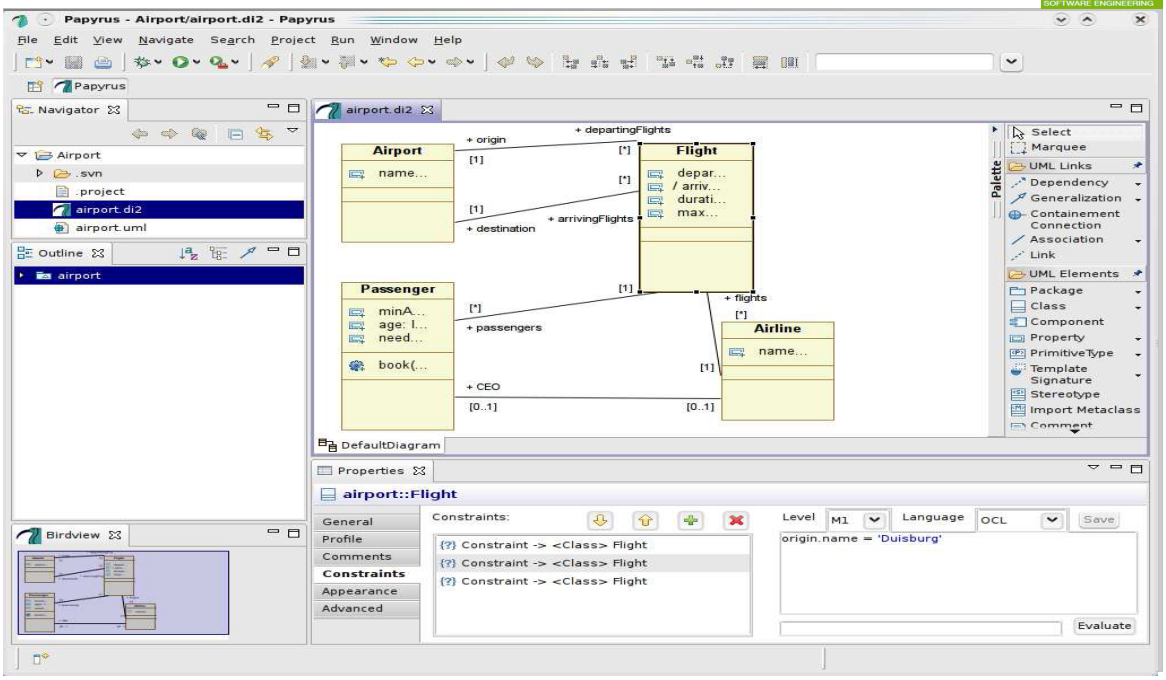
Schlüsselwort **result** referenziert den Rückgabewert.

Lösung

context Person::income(d: Date): Integer

post: result <= 5000

Flughafen-Beispiel in Papyrus



Literatur

- [UML09] UML Revision Task Force: *OMG Unified Modeling Language: Superstructure*, February 2009 http://www.omg.org/spec/UML/2.2/.2
- [UML10] UML Revision Task Force: *Object Constraint Language Specification*, February 2010 http://www.omg.org/spec/OCL/2.2/.6
- [WK03] Jos Warmer and Anneke Kleppe: *The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley Longman Publishing & Co.,Inc., Boston, MA, USA, 2003*

Literatur

Literatur:

• V. Gruhn, D. Pieper, C. Röttgers: **MDA - Effektives Software-Engineering mit UML 2 und Eclipse**. Xpert.press / Springer-Verlag, 2006.
UB e-Book: http://www.ub.tu-dortmund.de/katalog/titel/1223129 .
  • Kapitel 3.6.

• J. Seemann, J.W. Gudenberg: **Software-Entwurf mit UML 2**. Xpert.press / Springer-Verlag, 2006.
UB e-Book: http://www.ub.tu-dortmund.de/katalog/titel/1223020.
  • Kapitel 14.5.

• J. Warmer, A. Kleppe: ***The Object Constraint Language: Getting Your Models Ready for MDA***. *Addison-Wesley Longman Publ. & Co.,Inc., 2003.*
UB: http://www.ub.tu-dortmund.de/katalog/titel/901443
http://www.ub.tu-dortmund.de/katalog/titel/787903

• Object Management Group: **OCL 2.4**
http://www.omg.org/spec/OCL/2.4/PDF