

Willkommen zur Vorlesung
Softwarekonstruktion
im Wintersemester 2011/2012

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

07. Einführung in OCL

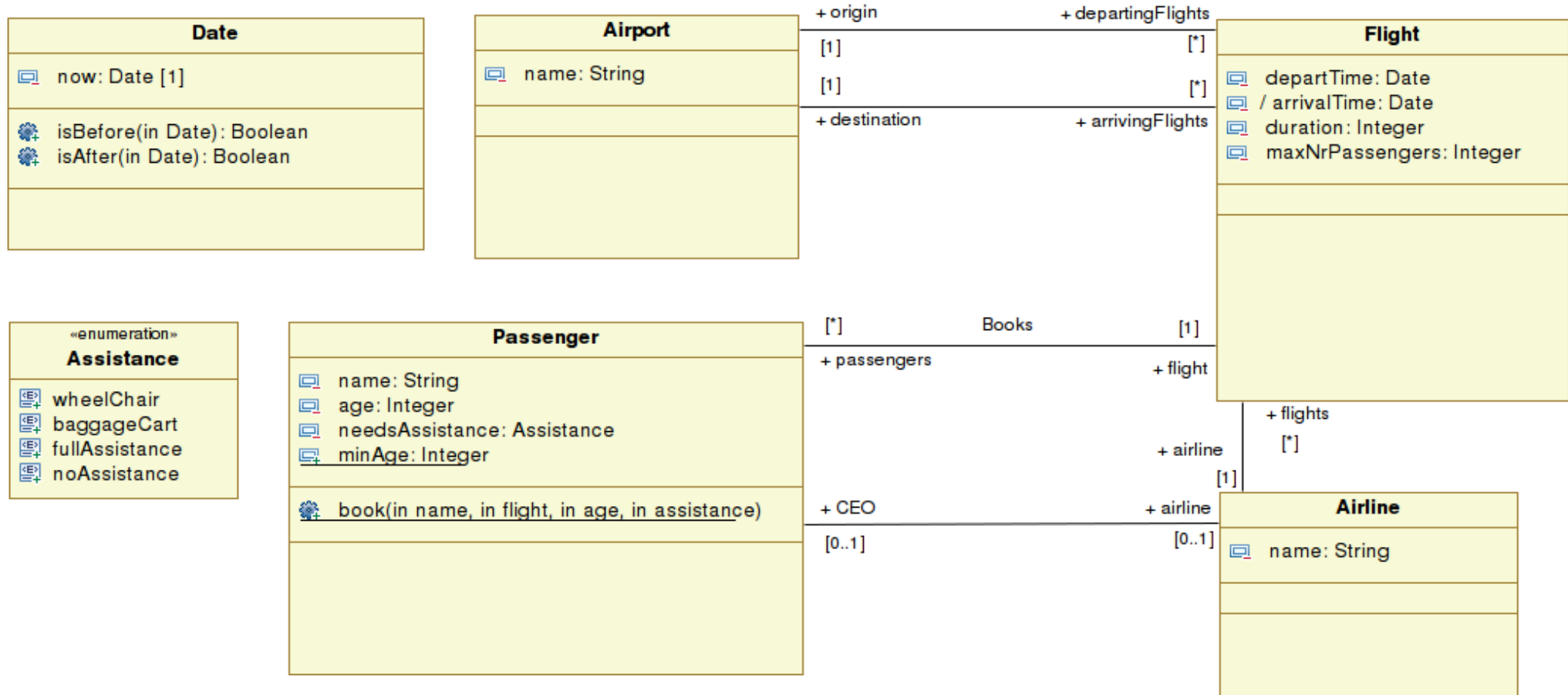
[Diese Folien basieren auf der Vorlesung "Muster- und Komponenten-basierte Softwareentwicklung" von Prof. Dr. Maritta Heisel]

- **UML (Unified Modeling Language) 2.0** [UML09] ist eine von der OMG (Object Management Group)¹ vorgestellte (semi formale) Modellierungssprache
- UML ist die **de facto Industriestandard-Notation** um Softwareanalyse und -design Artefakte zu modellieren
- Die UML- Spezifikation 2.2² beschreibt 14 **(semi) formale Diagrammtypen** wie z.B Klassen- und Use-Case-Diagramme
- Einschränkungen
 - **Nicht präzise** genug, **automatische Verifikation** nur schwer umsetzbar
 - **Schlechte** Möglichkeiten **Code zu generieren** (normalerweise nur Code-Gerüst, kein voll funktionierender Code)

¹<http://www.omg.org/>

²<http://www.omg.org/spec/UML/2.2>

Praktisches Beispiel: Klassendiagramm für einen Flughafen



Wie viele Fluggäste können sich für einen Flug anmelden?

- (semi-formale) visuelle Modelle können mit folgenden **formalen Spezifikationen angereichert** werden
 - **Zustandsbeschränkung** (mit Invarianten)
 - **Operationelle Semantik** (mit Vor- und Nachbedingungen)
- UML definiert eine Sprache, die mit diesem Ziel genutzt werden kann: Object Constraint Language (OCL)
- Vorteile:
 - UML Diagramme, die OCL- Ausdrücke enthalten führen zu **präzisen Spezifikationen**, die **automatisch verifiziert** werden können
 - Formale Spezifikationen **beseitigen die Mehrdeutigkeit**, die charakteristisch für informelle Spezifikationen ist
 - Es existieren **Werkzeuge, die aus OCL-Spezifikationen** für Zustands-Invarianten und für Vor- und Nachbedingungen von Operationen **Code und Aussagen in Java erzeugen**

- OCL ist eine **formale Sprache für** die Definition von **Einschränkungen** in UML-Modellen
- OCL ist **keine Programmiersprache**; deshalb ist es nicht möglich Programmlogik oder Kontrollflüsse in OCL zu modellieren
- OCL-Ausdrücke sind garantiert **ohne Seiteneffekte**
 - Wenn ein OCL-Ausdruck evaluiert wird, liefert er einen Wert zurück; er kann nichts im Modell verändern
 - Der Zustand des Systems wird sich wegen der Evaluation eines OCL-Ausdrucks nie ändern, auch wenn ein OCL-Ausdruck eine Zustandsänderung spezifizieren kann (z.B. in einer Nachbedingung)
- OCL unterstützt Prüfung von **strenger Typisierung**

OMG Spezifikation

- „Object Constraint Language 2.0“ [UML10]
<http://www.omg.org/spec/OCL/2.2/PDF>

(Teilweise) Basis für unsere „Einführung in OCL“

- The Object Constraint Language:
Getting Your Models Ready for MDA“ [WK03]
<http://www.di.uminho.pt/~jmf/MDSE/u2c.pdf>

OCL-Ausdrücke

- Sind immer **an ein UML-Modell gebunden**
- Beschreiben Einschränkungen für Elemente des Modells zu dem sie gehören; dieses Modell beschreibt welche Klassen genutzt werden können und welche Attribute, Operationen und Assoziationen für die Objekte dieser Klassen vorhanden sind
- Werden im zugehörigen UML-Modell oder in einem separaten Dokument angegeben

Constraint (Einschränkung)

Eine **Einschränkung** auf einem oder mehreren Teilen eines UML-Modells

Class Invariant (Klasseninvariante)

Eine Bedingung, die (fast) **immer** von allen Instanzen einer Klasse erfüllt werden muss

Pre-condition (Vorbedingung)

Eine Bedingung, die erfüllt sein muss, **bevor** eine Operation ausgeführt werden kann

Post-condition (Nachbedingung)

Eine Bedingung, die **nach** dem Ausführen einer Operation erfüllt sein muss

Guard condition (Abdeckungsbedingung)

Eine Bedingung, die **vor** einer Transaktion in einem Zustandsdiagramm, vor einer Nachricht in einem Sequenzdiagramm oder vor anderen UML-Modellen, die Verhalten modellieren, erfüllt sein muss

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

Context Ein Schlüsselwort um das zugehörige Modellelement, welches mittels <identifizier> angegeben wird, zu markieren. Dieses kann andere Modellelemente referenzieren. Das Schlüsselwort selbst kann innerhalb <boolean expression> genutzt werden, um den Kontext zu betreten.

<**identifizier**> ist ein Klassen- oder Operationsname

<**constraintType**> ist eins der Schlüsselwörter inv, pre oder post

<**constraintName**> ist ein optionaler Name für die Einschränkung

<**boolean expression**> ist irgendein boolescher Ausdruck,
oft ein Vergleich

Die folgenden **Typen** können in einem OCL-Ausdruck benutzt werden

Vordefinierte Typen

- Primitive Typen: String, Integer, Real, Boolean
- Collection Types: Set, Bag, Sequence, OrderedSet
- Tupel Typen: Tuple
- Spezielle Typen: OclType, OclAny, ...

Klassifikatoren von einem UML-Modell und seiner Eigenschaften

- **Klassen**, **Enumerationsklassen** und **Rollennamen**
- **Attribute** und **Operationen**

Folgende **Schlüsselwörter** können in einem OCL-Ausdruck benutzt werden:

- *If-then-else-endif* Konditionalausdrücke
- *Not, or, and xor, implies* boolesche Operatoren
- *Def* globale Definitionen
- *Let-in* lokale Definitionen

- Eine Invariante

- Ist eine Bedingung, die vor und nach der Ausführung einer Methode erfüllt sein muss, die jedoch während der Ausführung verletzt werden kann
- Wird anhand des Schlüsselwortes *inv* im Kontext der Instanz eines Klassifikators (Klasse, Rollenname...) spezifiziert

- Beschränkung von Domänen:
 - Beschränkung der Werte, die ein Attribut annehmen kann
- Beschränkung auf Einmaligkeit:
 - Ein Attribut oder eine Attributmenge einer Klasse für die gilt, dass für zwei unterschiedliche Instanzen dieser Klasse diesen Attributen keine gleichen Werte zugewiesen werden dürfen
- Zeitliche Beschränkung
- Bedingungen, die abgeleitete Modellelemente definieren (z.B abgeleitet Attribute)
- Regeln für die Existenz:
 - Regeln, die angeben, dass bestimmte Objekte/Werte existieren/ definiert sein müssen, bevor andere Objekte/ Werte definiert/ erzeugt werden

- Die Klasse, die von einer Invarianten referenziert wird, ist der Kontext der Invarianten
- Es ist gefolgt von einem booleschen Wert, der die Invariante angibt
- Alle Attribute der Kontextklasse können in der Invarianten genutzt werden

Beispiel

context: Flight

Inv: duration < 4

- Bedeutung: ?

- Die Klasse, die von einer Invarianten referenziert wird, ist der Kontext der Invarianten
- Es ist gefolgt von einem booleschen Wert, der die Invariante angibt
- Alle Attribute der Kontextklasse können in der Invarianten genutzt werden

Beispiel

context: Flight

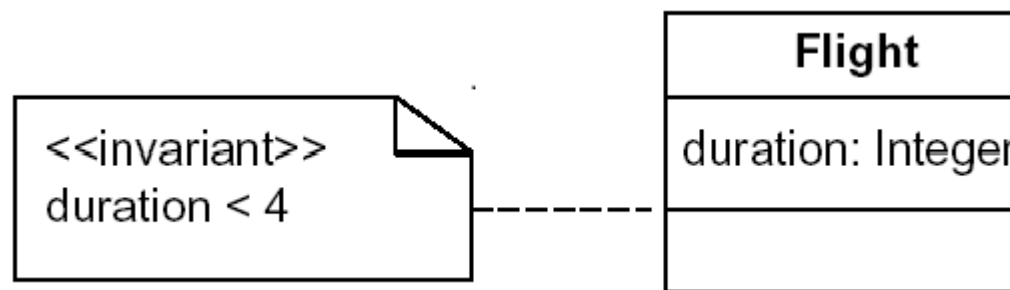
Inv: duration < 4

- Bedeutung: Jeder Flug dauert weniger als 4 Stunden

Die folgenden Notationen sind äquivalent

Context Flug
Inv: self.dauer < 4

Context Flug
Inv: dauer < 4



- Wenn der Attributtyp eine Klasse ist, können die Attribute und **Abfrageoperationen** dieser Klasse für die Erstellung der Invarianten genutzt werden (anhand der Punkt-Notation)
- Abfrageoperation:
Eine Operation, die den Wert von Attributen nicht ändert

Beispiel

context: Flight

Inv: departTime.isBefore(arrivalTime)

- Bedeutung: ?

- Wenn der Attributtyp eine Klasse ist, können die Attribute und **Abfrageoperationen** dieser Klasse für die Erstellung der Invarianten genutzt werden (anhand der Punkt-Notation)
- Abfrageoperation:
Eine Operation, die den Wert von Attributen nicht ändert

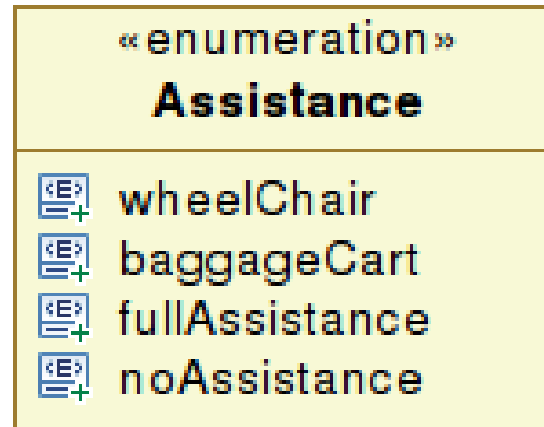
Beispiel

context: Flight

Inv: departTime.isBefore(arrivalTime)

- Bedeutung: Das Abflugdatum ist vor dem Ankunftsdatum.

Eine Aufzählung nutzt Datentypen gefolgt von :: und einem Wert



Beispiel

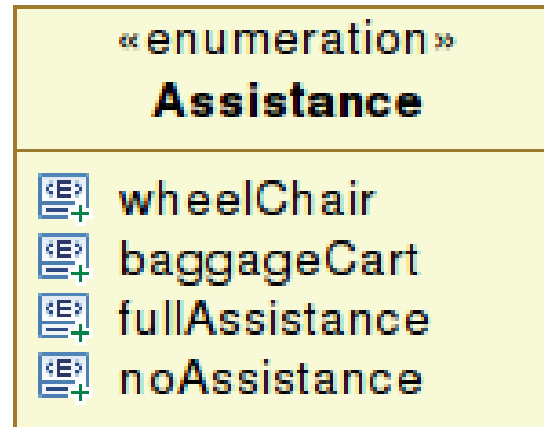
context Passenger

Inv: self.age > 95 implies

Self.NeedsAssistance = Assistance :: wheelchair

Bedeutung: ?

Eine 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.

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

Beispiel

context Flight

Inv: origin <> destination

Bedeutung: ?

Beispiel

context Flight

Inv: origin.name = 'Duisburg'

Bedeutung:?

Beispiel

context Flight

Inv: origin <> destination

Bedeutung: Der Abflugort ist immer ungleich dem Flugziel

Beispiel

context Flight

Inv: origin.name = 'Duisburg'

Bedeutung:?

Beispiel

context Flight

Inv: origin <> destination

Bedeutung: Der Abflugort ist immer ungleich dem Flugziel

Beispiel

context Flight

Inv: origin.name = 'Duisburg'

Bedeutung: Der Abflugort ist immer Duisburg

- Assoziationen sind oft one-to-many oder many-to-many Beziehungen, deshalb werden Beschränkungen für Collections benötigt
- OCL Ausdrücke geben entweder einen Fakt über alle Objekte in der Collection an oder über die Collection selbst

- Eine Collection-Operationen kann immer dann verwendet werden, wenn eine Navigation in einer **Menge von Objekten** endet
- Ein Pfeil (\rightarrow) zwischen dem Rollennamen und der Operation weist auf die Nutzung einer der vordefinierten Operation für Collections hin (z.B. *Passanger* \rightarrow *size()*)
- Ein Punkt (.) zwischen dem Rollennamen und der Operation gibt an, dass eine Operation aus dem UML-Modell verwendet wird (z.B. *departTime.isBefore(arrivalTime)*)

Beispiel

context Flight

Inv: passengers \rightarrow size() \leq maxNrPassengers

Bedeutung: ?

- Eine Collection-Operationen kann immer dann verwendet werden, wenn eine Navigation in einer **Menge von Objekten** endet
- Ein Pfeil (\rightarrow) zwischen dem Rollennamen und der Operation weist auf die Nutzung einer der vordefinierten Operation für Collections hin (z.B. *Passanger* \rightarrow *size()*)
- Ein Punkt (.) zwischen dem Rollennamen und der Operation gibt an, dass eine Operation aus dem UML-Modell verwendet wird (z.B. *departTime.isBefore(arrivalTime)*)

Beispiel

context Flight

Inv: passengers \rightarrow size() \leq maxNrPassengers

Bedeutung: Die Anzahl der Passagiere ist kleiner oder gleich der maximalen Anzahl von Sitzen

- Eine Collection von Objekten ist:
 - Ein Set:
 - Jedes Element kommt nur einmal vor
 - Single Navigation of an association results in a set ()
 - Ein Bag:
 - Gleiche Elemente dürfen mehrmals vorkommen
 -
 - Ein OrderedSet:
 - Ein Satz von geordneten Elementen
 -
 - Eine Sequence:
 - Ein Bag in dem die Elemente geordnet sind

- Operation um Attributwerte zu sammeln, z. B.
passengeres → *collect(name)*

Bedeutung (in Pseudocode)

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

- Die Operation kann auch benutzt werden, um eine neue Collections aus den 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;
```

- Die resultierende Collection beinhaltet andere Objekte als die ursprüngliche Collection
- Wenn die Quelle ein Set ist, so ist die resultierende Collection kein Set sondern ein Bag
- Wenn die Quelle eine Sequence oder ein OrderedSet ist, so ist die neue Collection eine Sequence
- Die Punktnotation ist eine verkürzte Schreibweise:
passengers.name
arrivingFlights.airline

Beispiel

context Airport

Inv: arrivingFlights → *size()* =

arrivingFlights → *collect(Airline)* → *size()*

- Bedeutung: ?

Beispiel

context Airport

Inv: arrivingFlights → *size()* =
arrivingFlights → *collect(Airline)* → *size()*

- Bedeutung: Jeder ankommende Flug gehört zu einer Airline

- Die *select* Operation bekommt einen OCL-Ausdruck als Parameter übergeben
- Das Ergebnis dieser Operation ist eine Subcollection der verwendeten Collection
- *Select* liefert alle Elemente einer Collection, für die der Ausdruck **wahr** ist

Beispiel

context Flight

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

- Bedeutung: ?

- Die *select* Operation bekommt einen OCL-Ausdruck als Parameter übergeben
- Das Ergebnis dieser Operation ist eine Subcollection der verwendeten Collection
- *Select* liefert alle Elemente einer Collection, für die der Ausdruck **wahr** ist

Beispiel

context Flight

inv: passengeres → select(needsAssistance <> Assistance::noAssistance) → size() <= 10

- Bedeutung: Die Anzahl der Passagiere, die Hilfe brauchen, ist kleiner oder gleich 10

- Die *reject* Operation verhält sich analog zu *select*
- *Reject* liefert alle Elemente einer Collection, für die der Ausdruck **falsch** ist

Beispiel

context Flight

*inv: passengeres → reject(needsAssistance =
Assistance::noAssistance) → size() <= 10*

- Bedeutung: ?

- Die *reject* Operation verhält sich analog zu *select*
- *Reject* liefert alle Elemente einer Collection für die der Ausdruck **falsch** ist

Beispiel

context Flight

*inv: passengeres → reject(needsAssistance =
Assistance::noAssistance) → size() <= 10*

- Bedeutung: Die Anzahl der Passagiere, die Hilfe brauchen, ist kleiner oder gleich 10

- Die forAll Operation kann genutzt werden, um eine **Bedingung** zu definieren, die von allen Elementen in einer Collection eingehalten werden muss
- Die forAll Operation erhält einen **OCL-Ausdruck** als Parameter
- Diese Operation wird verwendet, wenn es ein (Sub-)Set von allen Instanzen einer Klasse gibt und dieses Set überprüft werden soll
- Diese Operation liefert einen booleaschen Wert zurück:
 - Wahr, wenn die Bedingung von allen Elementen erfüllt wird
 - Ansonsten falsch

- *class.allInstances()*: Collection mit allen Elementen einer Klasse

Beispiel

context Airport

*inv : Airport.allInstances()->forall(a1, a2 |
a1 <> a2 implies a1.name <> a2.name)*

- Bedeutung: ?

- *class.allInstances()*: Collection mit allen Elementen einer Klasse

Beispiel

context Airport

*inv : Airport.allInstances()->forall(a1, a2 |
a1 <> a2 implies a1.name <> a2.name)*

- Bedeutung: Jeder Flughafenname ist einzigartig
- Äquivalent:

context Airport

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

- In Klassendiagrammen kann nur die Syntax und Signatur einer Operation definiert werden
- Die **Semantik** einer Operation kann mittels **Vor- und Nachbedingungen** in OCL spezifiziert werden
- Vorbedingung:
 - Bedingungen, die von Argumenten und dem initialen Objektzustand erfüllt werden müssen, damit ein Operationsaufruf gültig ist.

Beispiel

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

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

- Bedeutung: ?

Ein Beispiel für eine Vorbedingung

Beispiel

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

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

- Bedeutung: Die Anzahl der registrierten Passagiere für einen Flug muss vor einer Buchung kleiner als *maxNrPassengers* sein

- Nachbedingung:

- Bedingung die vom Rückgabewert, finalem Objektzustand, den Argumenten und dem initialem Objektzustand erfüllt sein müssen am Ende einer Operationsausführung, in der Annahme, das die Vorbedingungen erfüllt sind
- Spezifiziert beabsichtigte Ergebnisse und Zustandsänderungen (was), aber nicht wie sie geschehen (wie)
- Der initiale Zustand eines Objektfelds wird mit *@pre* angegeben
- Der Rückgabewert wird mit dem Schlüsselwort *result* angegeben

Beispiel

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

post: flight.passengeres \rightarrow size() -

flight.passengeres@pre \rightarrow size() = 1 and

*flight.passengeres \rightarrow exists(p: Passenger | p.age = age
and p.name = name*

and p.needsAssistance = assistance)

- Bedeutung: ?

Beispiel

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

```
post: flight.passengeres → size() -
```

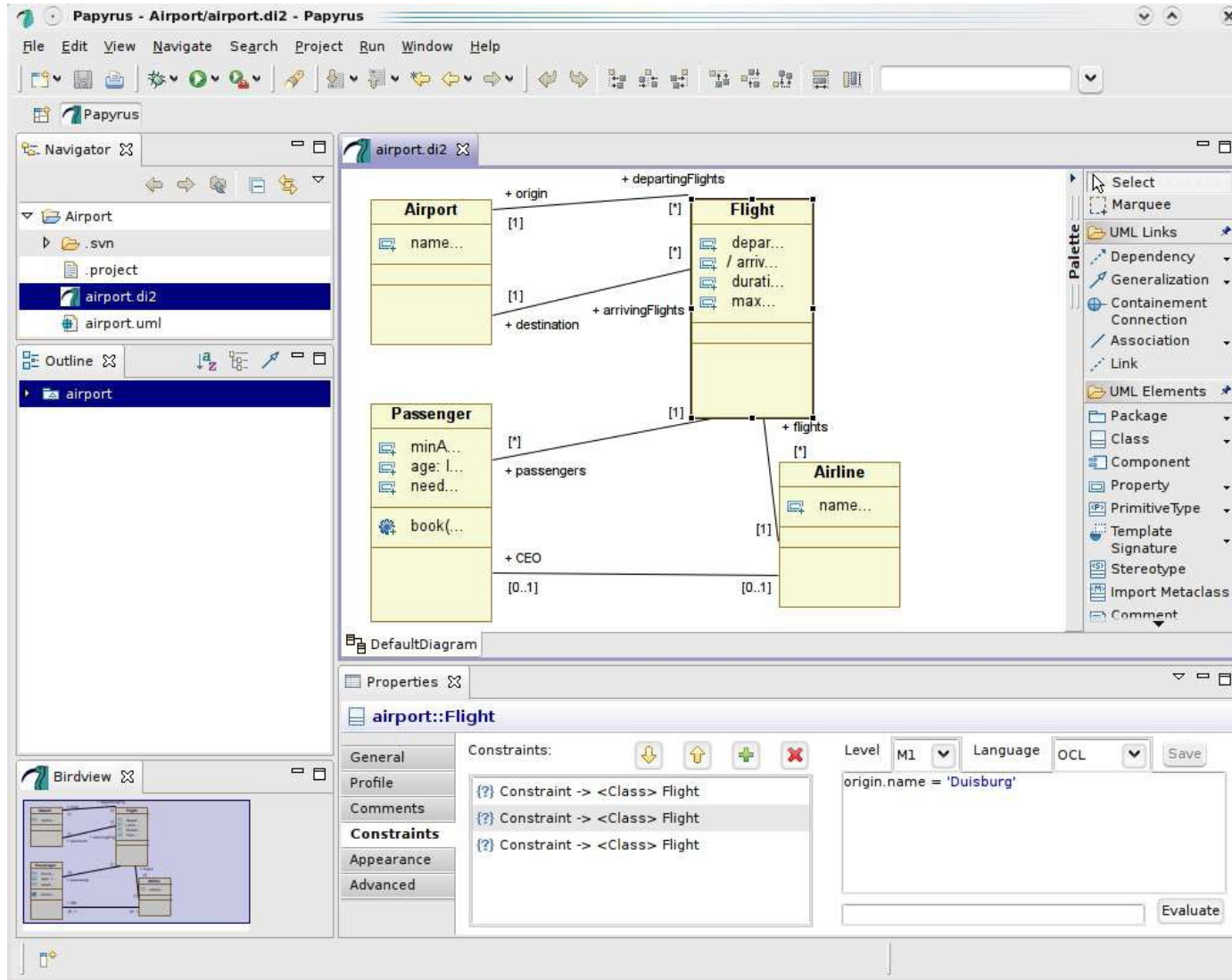
```
flight.passengeres@pre → size() = 1 and
```

```
flight.passengeres → exists( p: Passenger | p.age = age  
and p.name = name
```

```
and p.needsAssistance = assistance)
```

- Bedeutung:
 - Nach der Ausführung existiert ein zusätzliches Objekt
 - Die Attribute eines Objekts wurden anhand der Werte aus *book* initialisiert

- Papyrus ist ein frei erhältliches **Open-Source Werkzeug für die Modellierung mit UML 2.0**
- Download: <http://www.papyrusuml.org/>
- Basiert auf der Eclipse-Entwicklungsumgebung
- Gänzliche Einhaltung des von OMG definierten UML 2.0 Standards (gemäß Webseite)
- Erweiterbare Architektur von Papyrus erlaubt das Hinzufügen von Diagrammen, neuen Codegeneratoren, etc.
- Erlaubt das Einbinden von OCL-Bedingungen



| 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)

Collections und Tuples

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

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

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

Note 2: Tuple components can be accessed with “.” as in “t1.name”

Operations on Collection (T)

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

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

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

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

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

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

Operations on Set (T) I

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

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

Operations on Set (T) II

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

Softwarekonstruktion
WS 2011



| 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 <i>self</i> ; otherwise, the result is <i>self</i> . |
| asOrderedSet() : OrderedSet(T) | OrderedSet with elements from <i>self</i> in undefined order. |
| asSequence() : Sequence(T) | Sequence with elements from <i>self</i> in undefined order. |
| asBag() : Bag(T) | Bag will all the elements from <i>self</i> . |

Operations on Bag (T)

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

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

Operations on Sequence (T) I

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

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

Operations on Sequence (T) II

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

Softwarekonstruktion
WS 2011



LEHRSTUHL 14
SOFTWARE ENGINEERING

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

Operations on OrderedSet (T)

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

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

| 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 <i>null</i> . Any property call applied on <i>null</i> results in <i>OclInvalid</i> , except for the operation <i>oclIsUndefined()</i> . A collection may have <i>null</i> 's. |
| OclInvalid | The type OclInvalid is a type that conforms to all other types. It has one single instance called <i>invalid</i> . Any property call applied on <i>invalid</i> results in <i>invalid</i> , except for the operations <i>oclIsUndefined()</i> and <i>oclIsInvalid()</i> . |
| 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. |

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

| 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. |

- [UML09] **UML Revision Task Force**
*OMG Unified Modeling Language:
Sperstructure*, February 2009
<http://www.omg.org/spec/UML/2.2/.2>
- [UML10] **UML Revision Task Force**
Object Constraint Language Specifivation,
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*