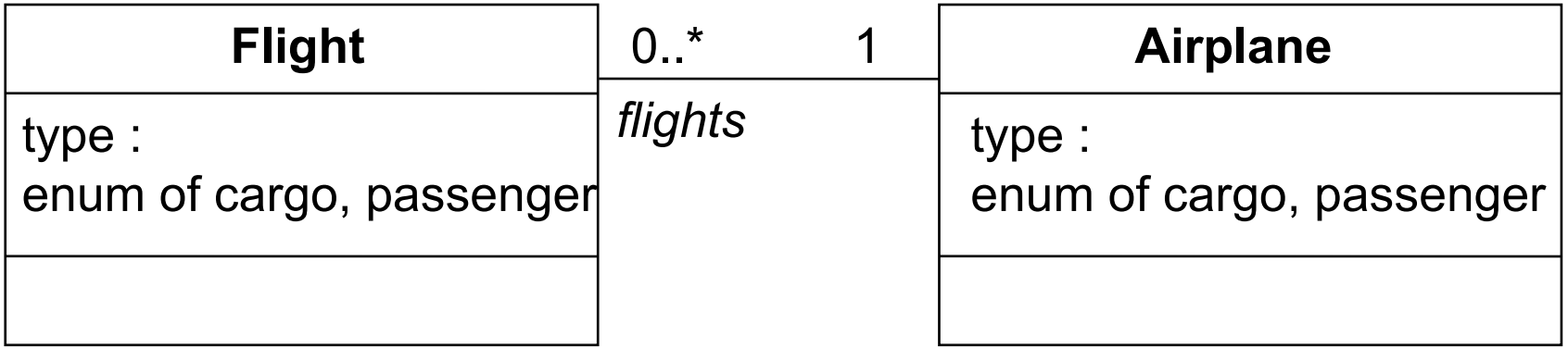
Object Constraint Language

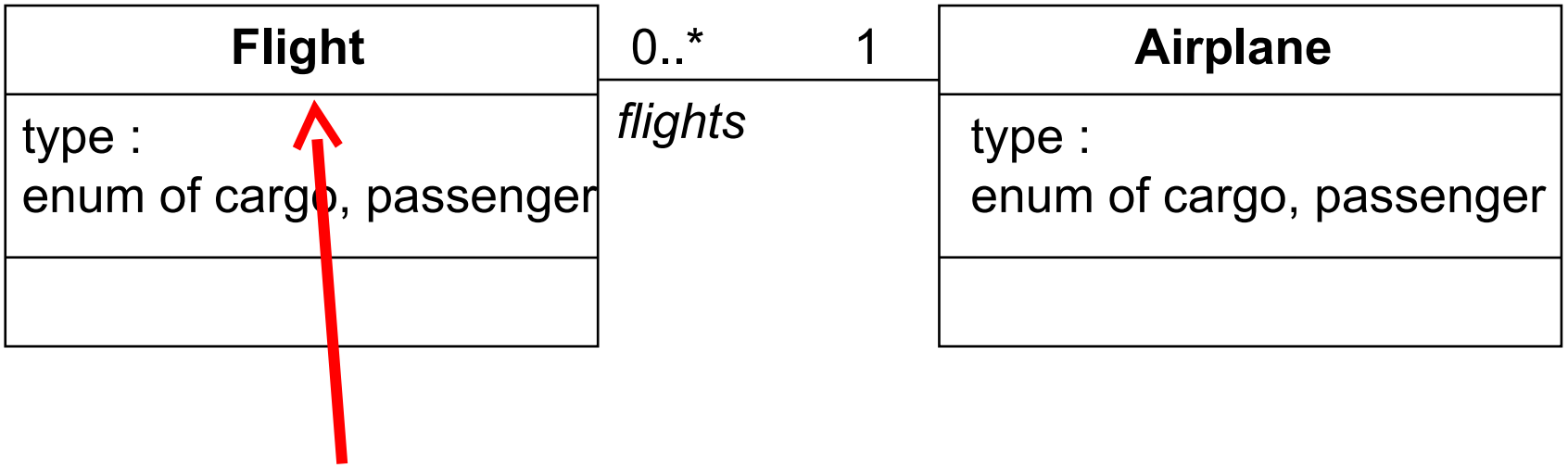
OCL

OCL Intro



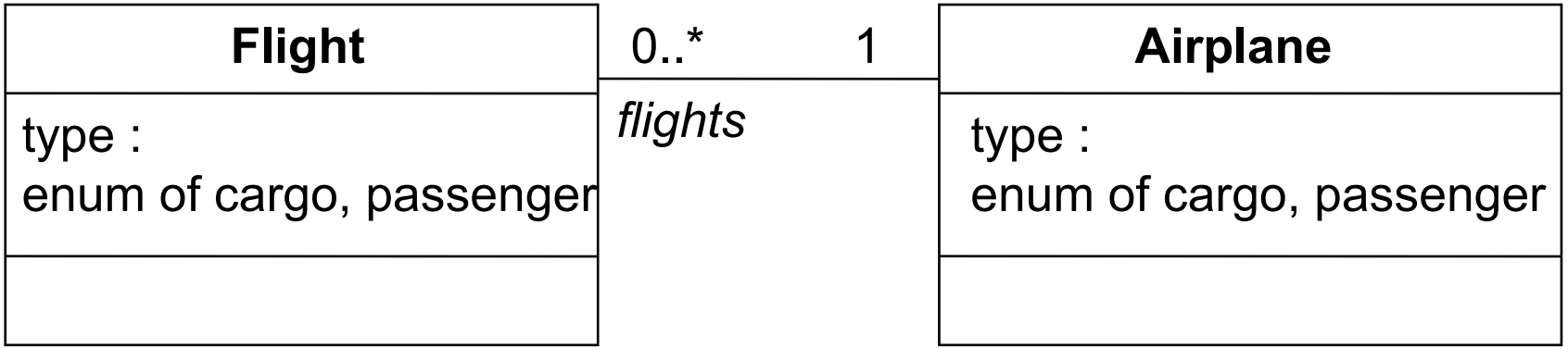

```
{context Flight
inv:  type = #cargo implies airplane.type = #cargo
inv:  type = #passenger implies airplane.type = #passenger}
```

OCL Intro




```
{context Flight
inv:  type = #cargo implies airplane.type = #cargo
inv:  type = #passenger implies airplane.type = #passenger}
```

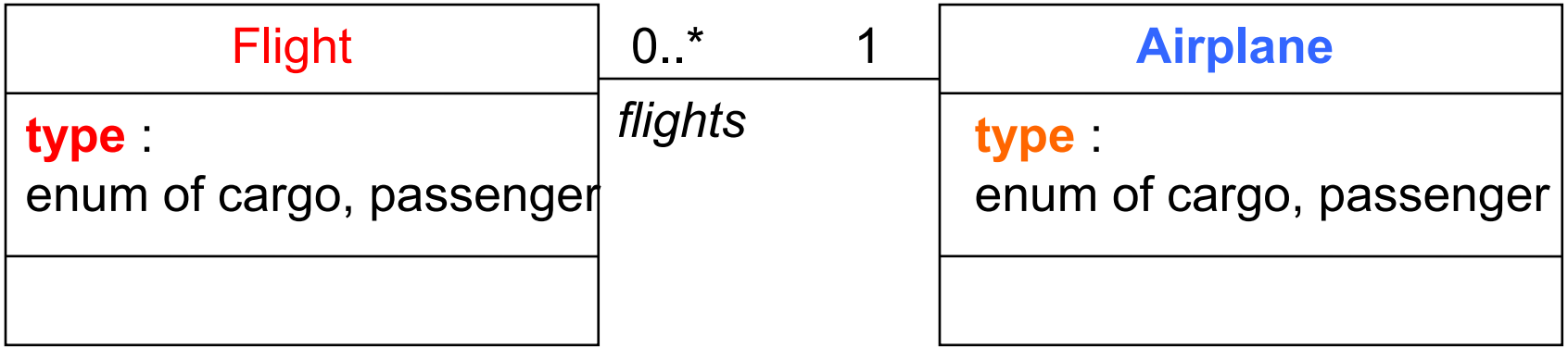
OCL Intro




```
{context Flight
inv:  type = #cargo implies airplane.type = #cargo
inv:  type = #passenger implies airplane.type = #passenger}
```

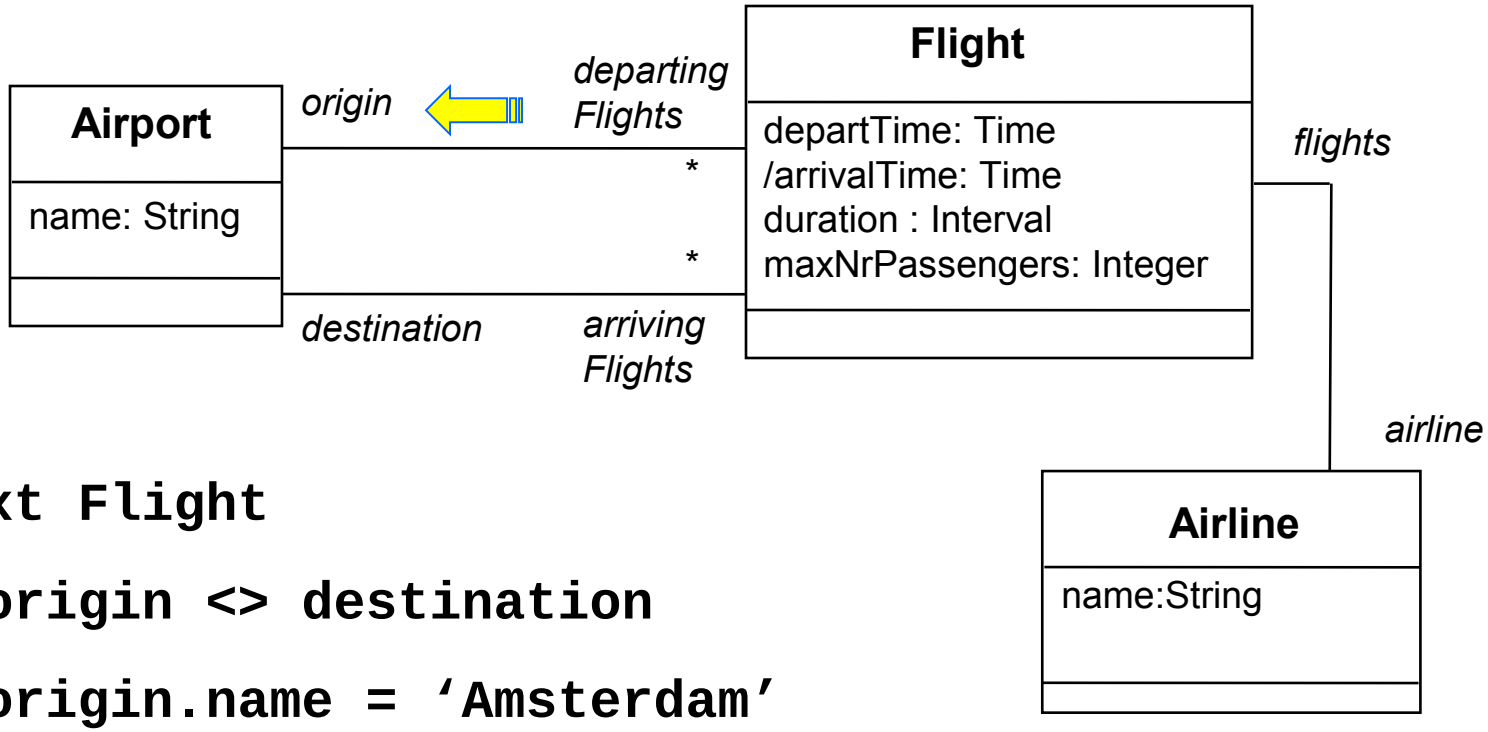
Always true

OCL Intro




```
{context Flight
inv:  type = #cargo implies airplane.type = #cargo
inv:  type = #passenger implies airplane.type = #passenger}
```

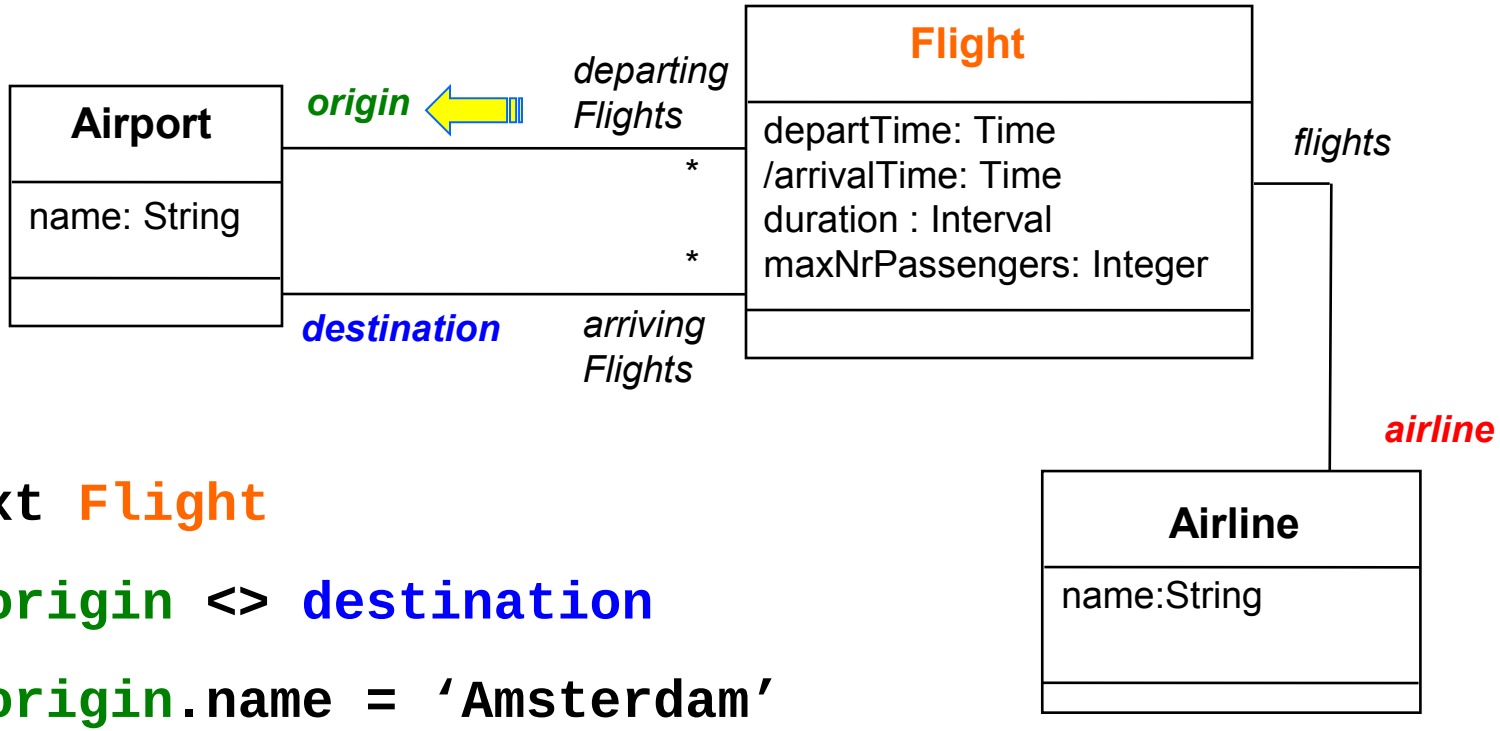
OCL Intro II




```
context Flight
inv: origin <> destination
inv: origin.name = 'Amsterdam'

context Flight
inv: airline.name = 'KLM'
```

OCL Intro II

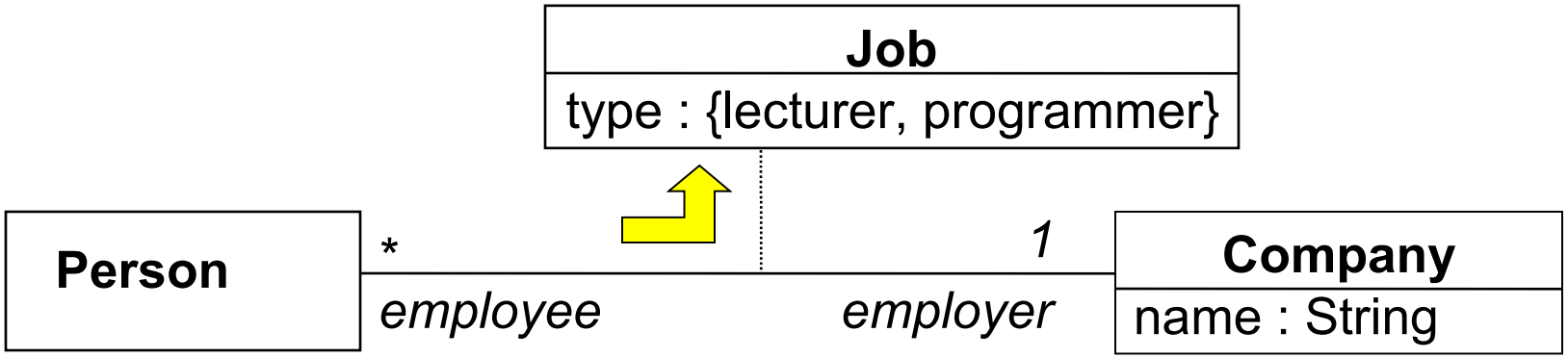



```
context Flight
inv: origin <> destination
inv: origin.name = 'Amsterdam'

context Flight
inv: airline.name = 'KLM'
inv: airline.name.toLower = 'klm'
```

OCL Intro III

```
context Person inv:
if employer.name = ‘CVUT’ then
        job.type = #lecturer
else
    job.type = #programmer
endif
```



Definition of constraint

- **“A constraint is a restriction on one or more values of (part of) an object-oriented model or system.”**

History

- **1995 IBEL IBM**
- **Součást UML 1.1**

UML diagram nedostačuje

- **Potřebujeme jazyk pro specifikaci**
- **Potřebujeme něco pro Objektový návrh**
  - ne nový jazyk
- **Potřebujeme něco standardizovaného**

Účel OCL

- **Jazyk formální specifikace**
  - -> implementovatelný
- **Objektový návrh**
- **Intuitivní syntax pro OO jazyk**
- **Pozor není to programovací jazyk!**

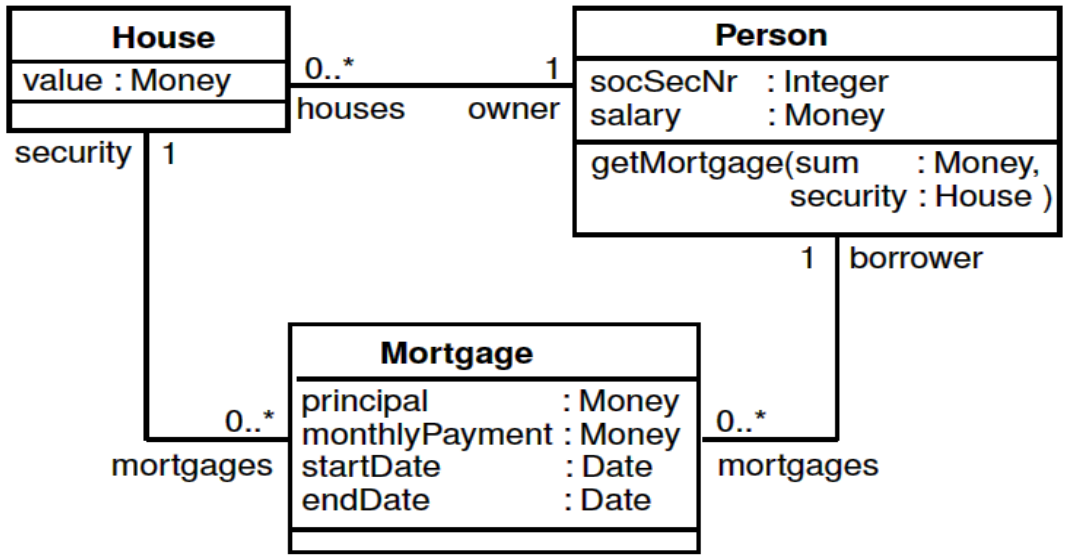
Výhody formální specifikace

- **Lepší dokumentace**
  - Omezení modelu
  - Informace navíc
- **Přesnost**
  - Zamezí dvojznačnosti
- **Komunikace bez nedorozumění**
  - Komunikace mezi vývojářem a analytikem bez nedorozumění

Kam s OCL

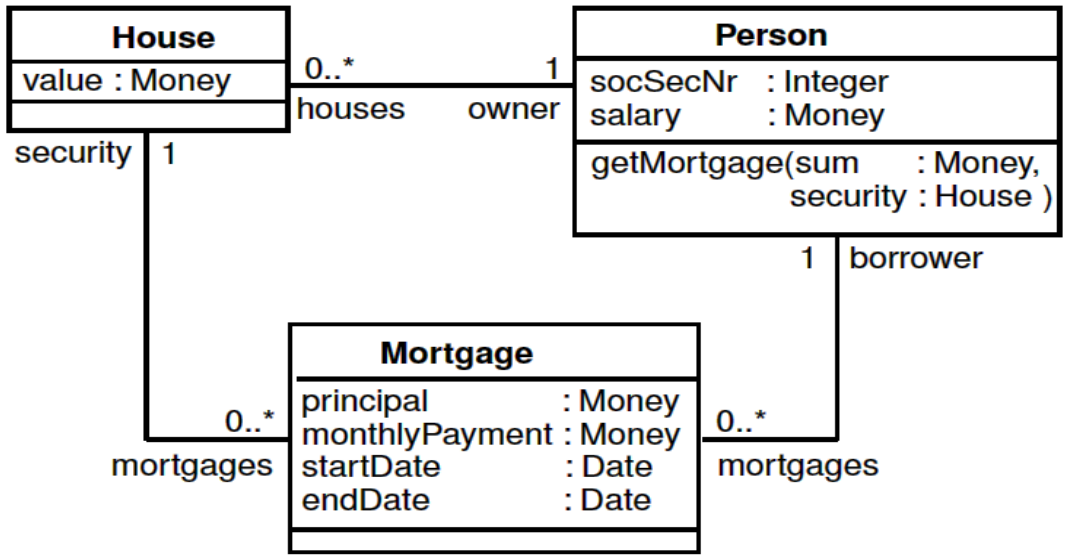
- **Specifikace invariantů tříd a typů**
  - Co smí a co ne
  - Vymezení legálních hodnot
  - …
- **Pre- a post-conditions pro metody**
- **Navigační jazyk**
- **Omezení/vymezení operací**
- **Test požadavku a specifikace**

Example: A Mortgage System (Hypoteka)



1. A person may have a mortgage only on a house she owns.
2. The start date of a mortgage is before its end date.

OCL specification of the constraints:



1. **context** *Mortgage*
invariant: *self.security.owner = self.borrower*

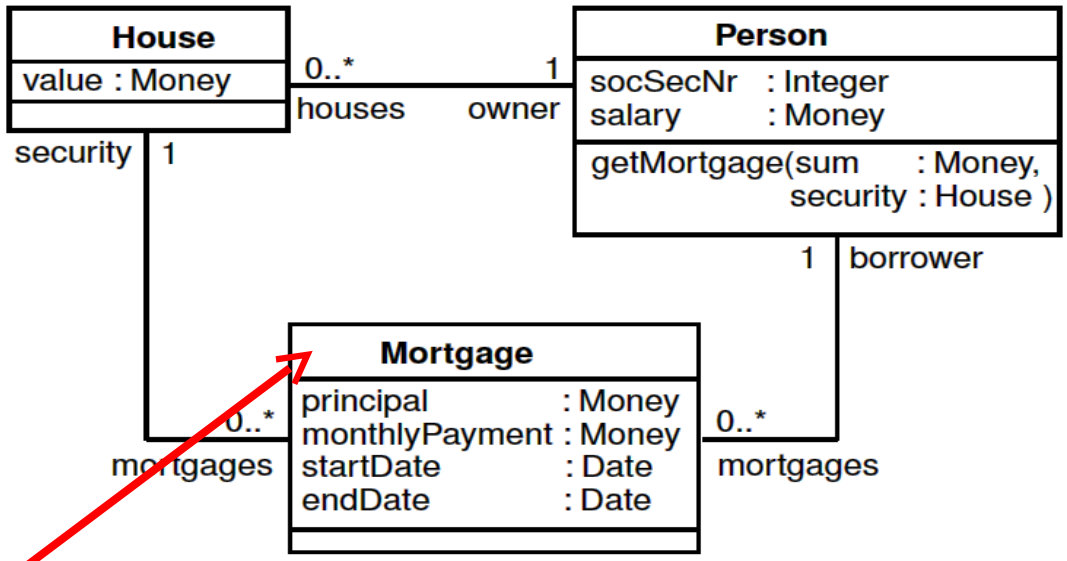
context *Mortgage*
invariant: *security.owner = borrower*

2. **context** *Mortgage*
invariant: *self.startDate < self.endDate*

context *Mortgage*
invariant: *startDate < endDate*

1. **A person may have a mortgage only on a house she owns.**
2. **The start date of a mortgage is before its end date.**

OCL specification of the constraints:



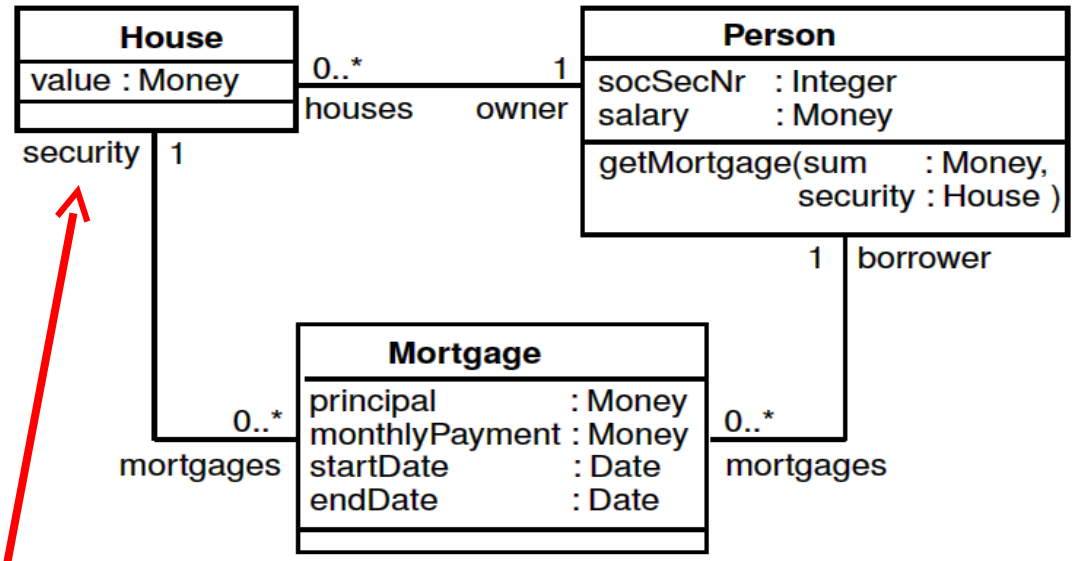
1. **context** *Mortgage*
invariant: *self.security.owner = self.borrower*

context *Mortgage*
invariant: *security.owner = borrower*

2. **context** *Mortgage*
invariant: *self.startDate < self.endDate*

context *Mortgage*
invariant: *startDate < endDate*

OCL specification of the constraints:



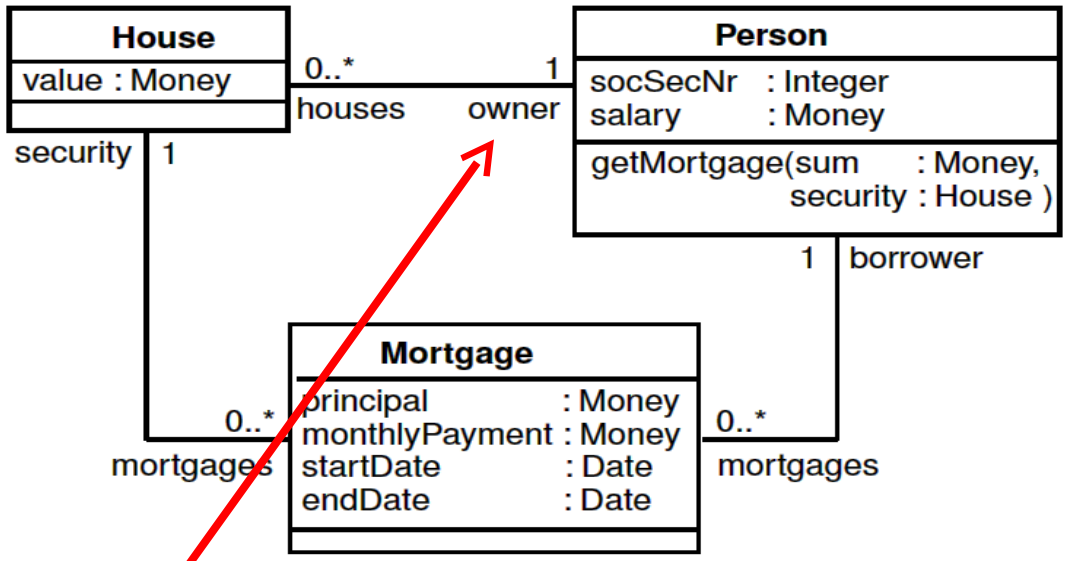
1. **context** *Mortgage*
invariant: *self.security.owner = self.borrower*

context *Mortgage*
invariant: *security.owner = borrower*

2. **context** *Mortgage*
invariant: *self.startDate < self.endDate*

context *Mortgage*
invariant: *startDate < endDate*

OCL specification of the constraints:



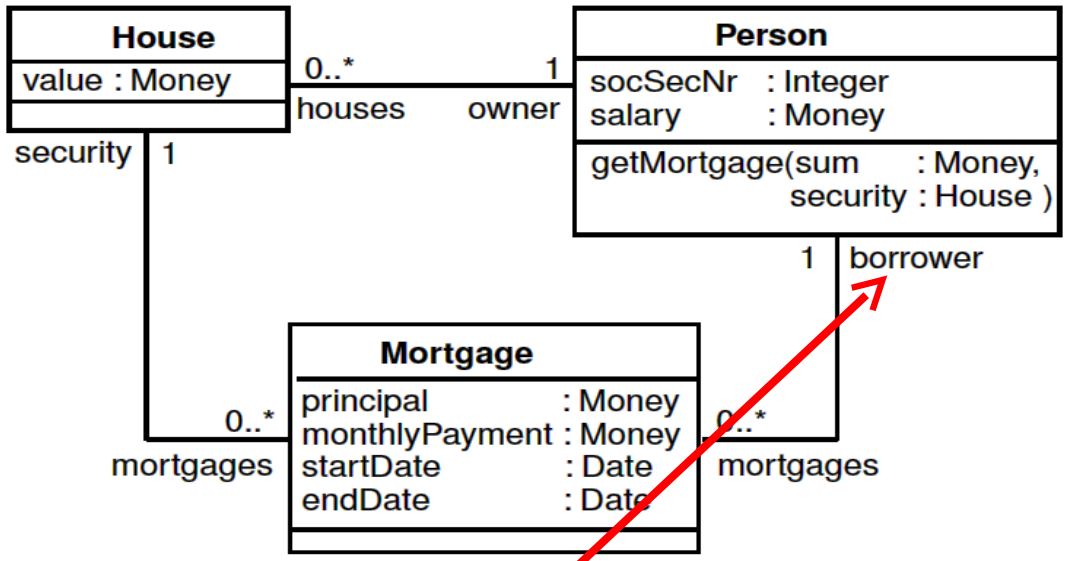
1. **context** *Mortgage*
invariant: *self.security.owner = self.borrower*

context *Mortgage*
invariant: *security.owner = borrower*

2. **context** *Mortgage*
invariant: *self.startDate < self.endDate*

context *Mortgage*
invariant: *startDate < endDate*

OCL specification of the constraints:



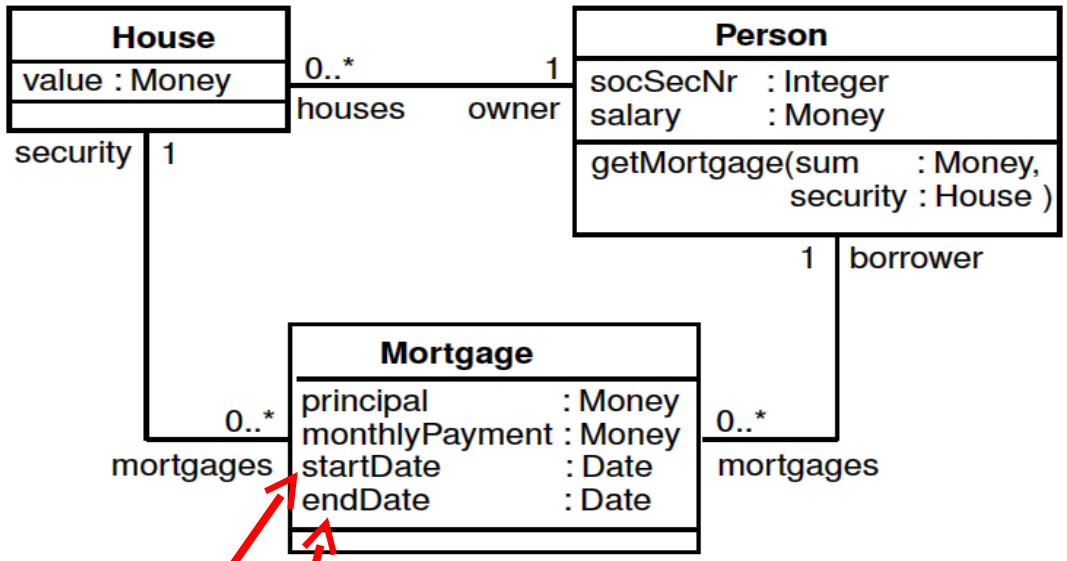
1. **context** *Mortgage*
invariant: *self.security.owner = self.borrower*

context *Mortgage*
invariant: *security.owner = borrower*

2. **context** *Mortgage*
invariant: *self.startDate < self.endDate*

context *Mortgage*
invariant: *startDate < endDate*

OCL specification of the constraints:



1. **context** *Mortgage*
invariant: *self.security.owner = self.borrower*

context *Mortgage*
invariant: *security.owner = borrower*

2. **context** *Mortgage*
invariant: *self.startDate < self.endDate*

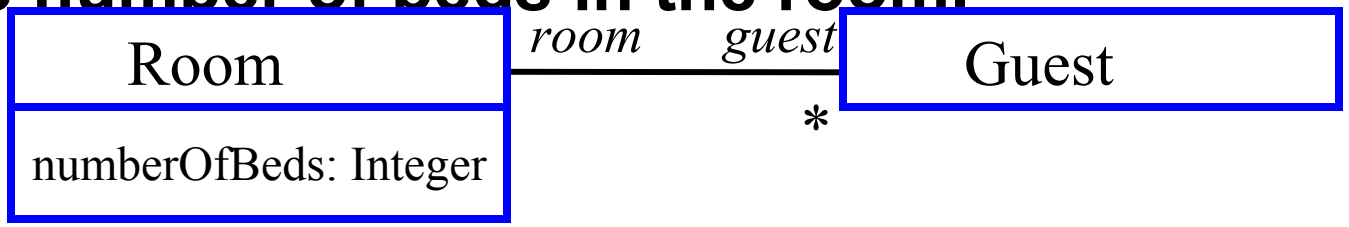
context *Mortgage*
invariant: *startDate < endDate*

More Constraints Examples

- **All players must be over 18.**

```
context Player invariant: self.age >=18
```

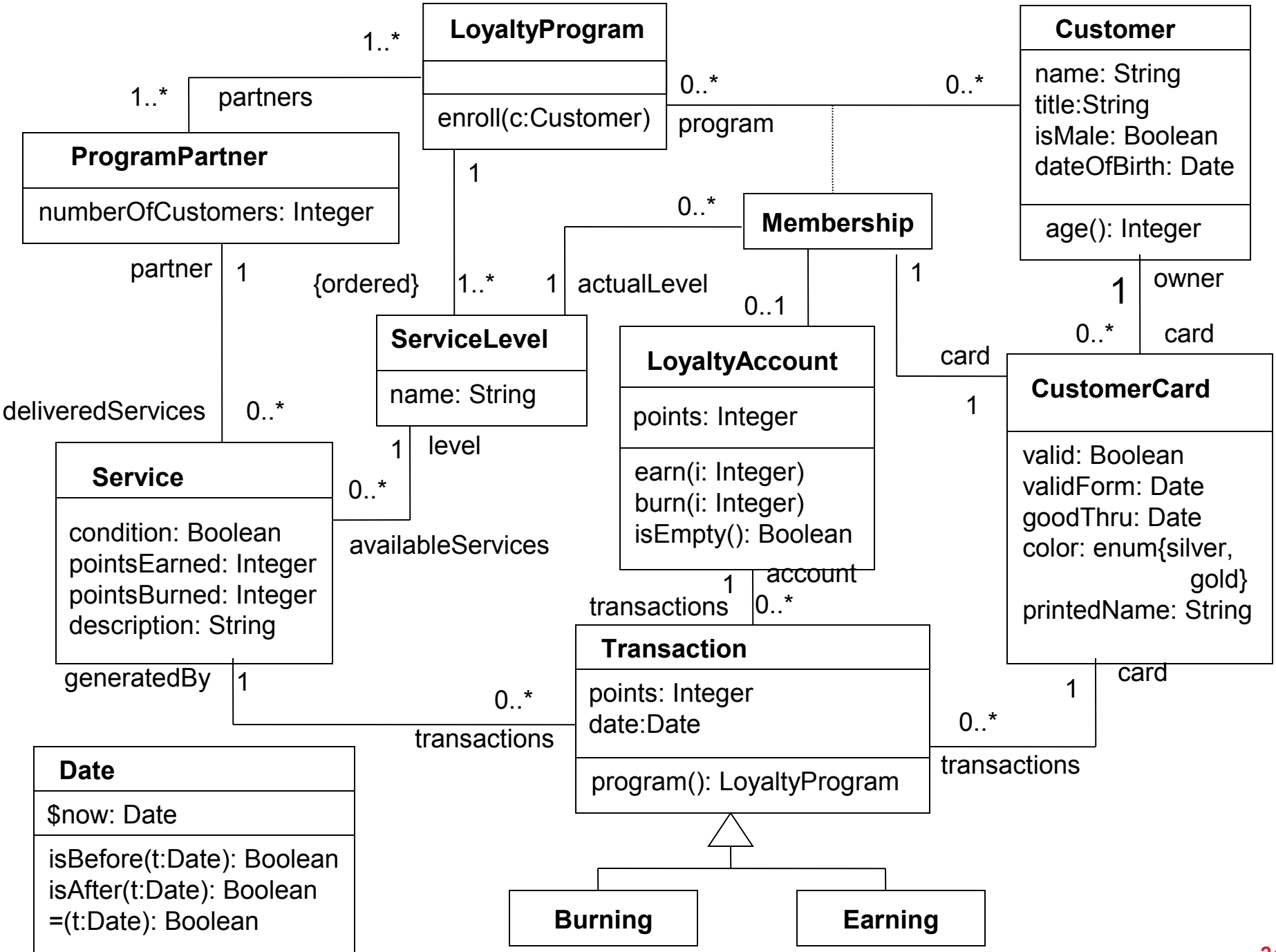
- **The number of guests in each room doesn't exceed the number of beds in the room.**



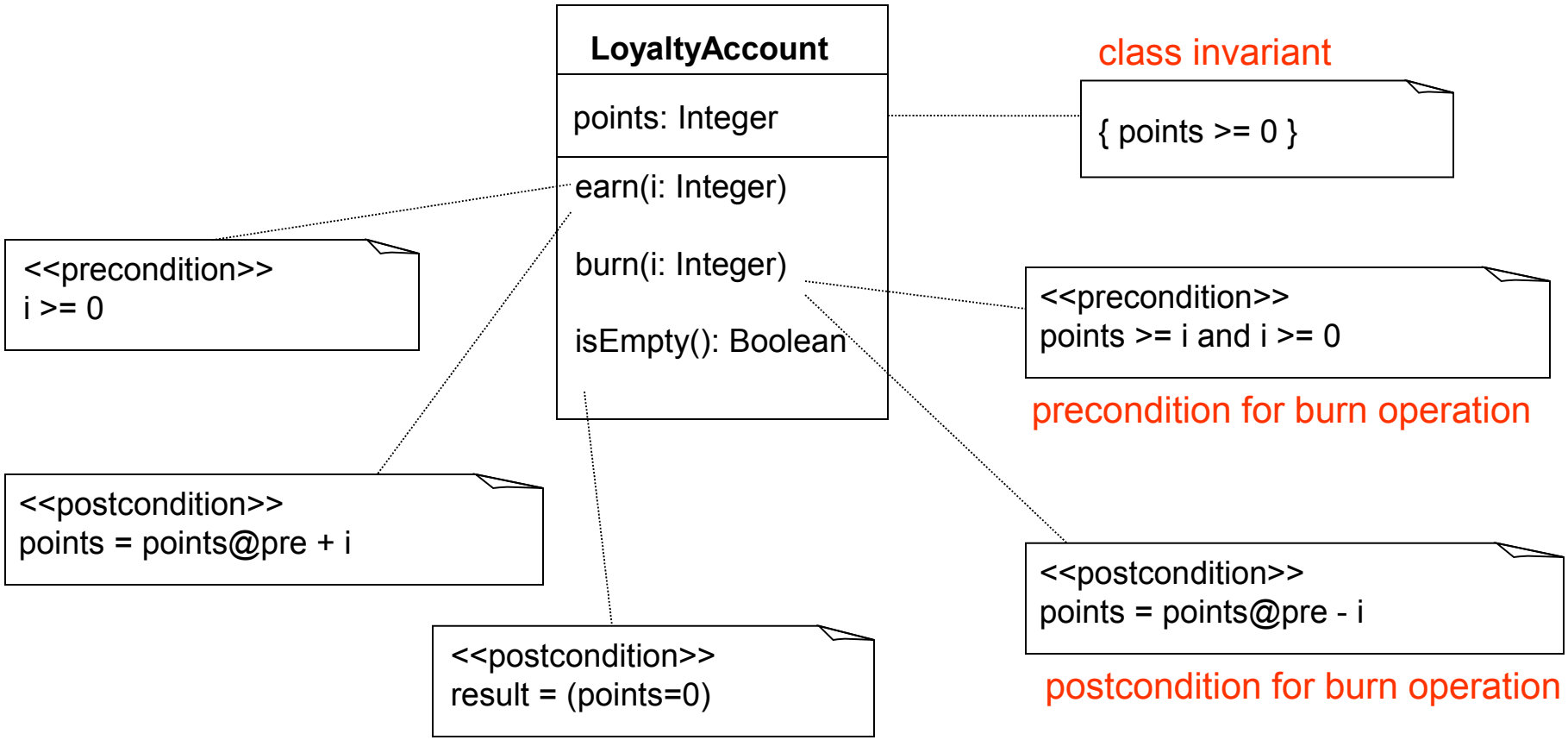

```
context Room invariant: guest -> size() <= numberOfBeds
```

Constraints/Omezení (invariants), Contexts and Self

- **Constraint (invariant)**
  - Boolean OCL výraz – vyhodnotitelný do true/false
- **Každý constraint je vázán na specifický typ**
  - (class, association class, interface) v UML modelu
  - Má svůj context.
- **Context objektu může být vytyčen přes keyword 'self'.**
- **Context lze specifikovat přes:**
  - Context <context name>
  - Přerušovanou čárou spojující objekt v UML modelu
- **Constraint může mít název za klíčovým slovem invariant.**


LoyaltyProgram
enroll(c:Customer)
1..*
1..*
partners
ProgramPartner
numberOfCustomers: Integer
0..*
program
0..*
Customer
name: String
title:String
isMale: Boolean
dateOfBirth: Date
age(): Integer
1
0..*
Membership
partner
1
{ordered}
1..*
1
actualLevel
0..1
1
1
owner
0..*
card
card
1
ServiceLevel
name: String
LoyaltyAccount
points: Integer
earn(i: Integer)
burn(i: Integer)
isEmpty(): Boolean
CustomerCard
valid: Boolean
validForm: Date
goodThru: Date
color: enum{silver, gold}
printedName: String
deliveredServices
0..*
1
level
Service
condition: Boolean
pointsEarned: Integer
pointsBurned: Integer
description: String
0..*
availableServices
1
account
transactions
0..*
Transaction
points: Integer
date:Date
program(): LoyaltyProgram
generatedBy
1
0..*
transactions
1
card
0..*
transactions
Date
$now: Date
isBefore(t:Date): Boolean
isAfter(t:Date): Boolean
=(t:Date): Boolean
Burning
Earning

Using OCL in Class Diagrams



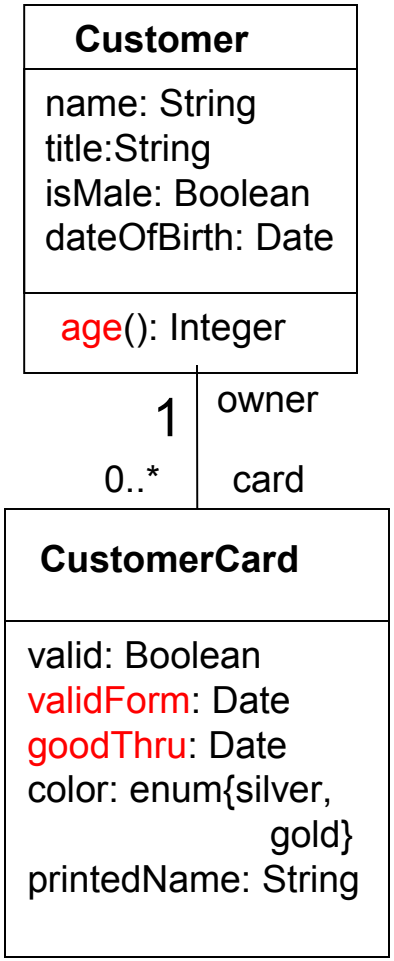
Invarianty Atributů

- Invarianty atributů:

```
context Customer
invariant age-restriction: age >= 18
```

```
context CustomerCard
invariant
 correctDates: validFrom.isBefore(goodThru)
```

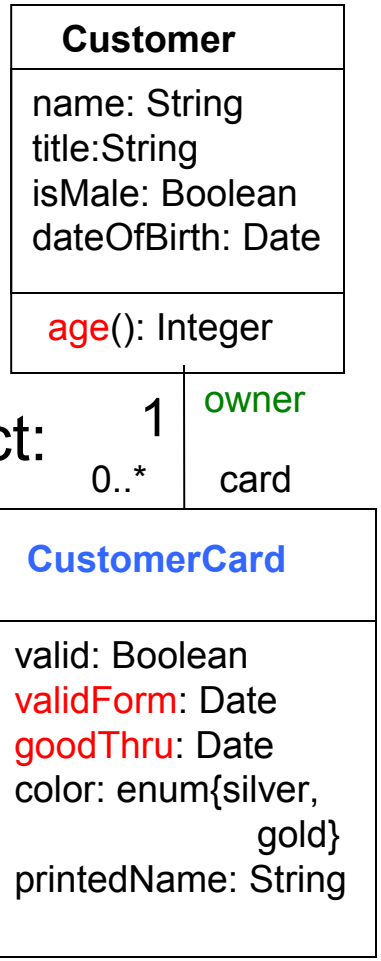
The type of *validFrom* and *goodThru* is *Date*.
isBefore(*Date*):*Boolean* is a *Date* operation.



- The class on which the invariant must be put is the invariant **context**.
- For the above example, this means that the expression is an invariant of the **Customer** class.

Invarianty : Navigace přes associace – role

Navigation over associations is used to refer to Associated objects, starting from the context object:




```
context CustomerCard
invariant: owner.age >= 18
```

owner → a *Customer* instance.
owner.age → an *Integer*.

Note: This is not the "right" context for this constraint!
If the *role name is missing* – use the *class name* at the other end of the association, starting with a lowercase letter.
Preferred: Always give role names.

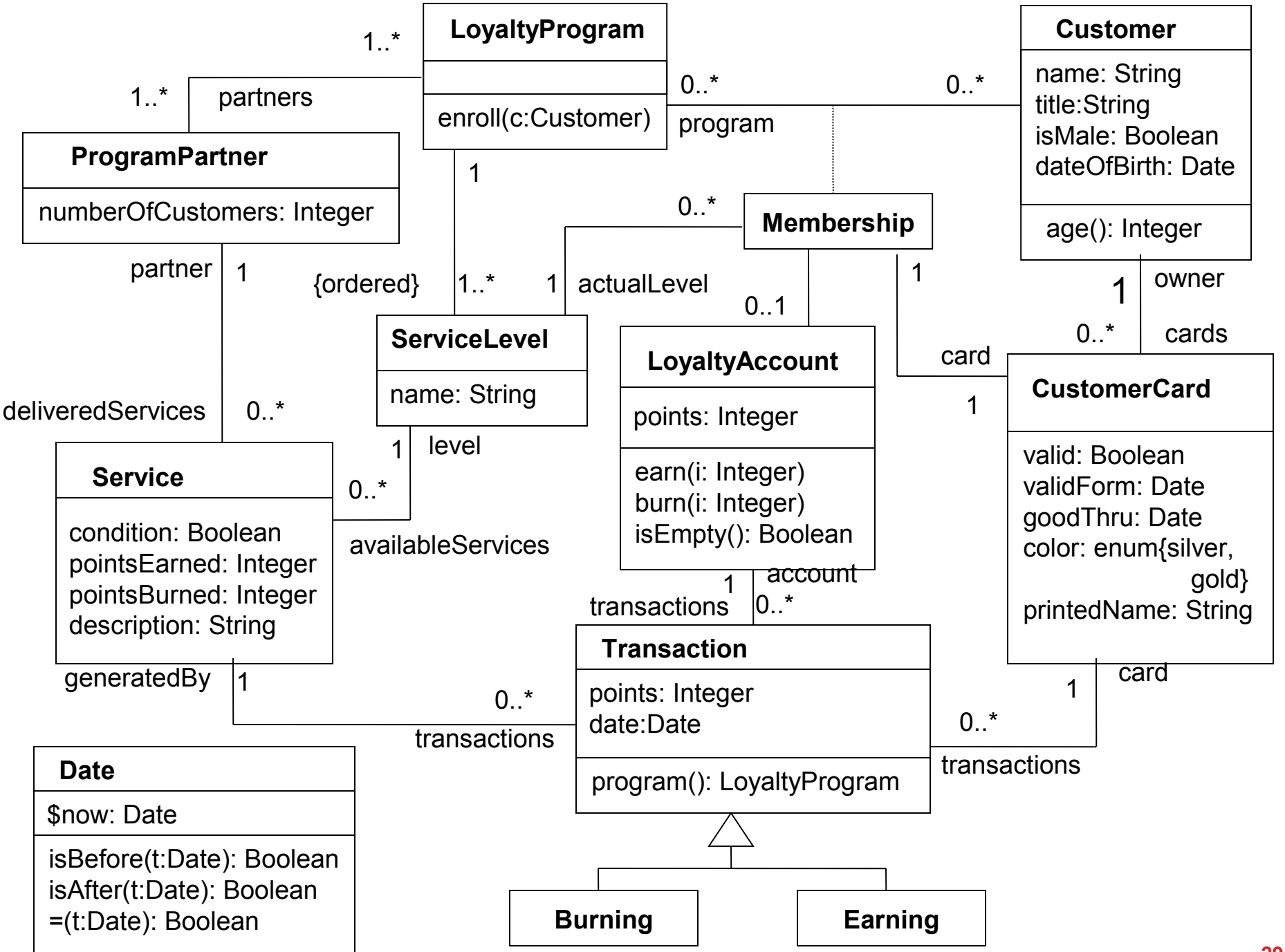
Invarianty : Navigace přes associace – roles

```
context CustomerCard
invariant printedName:
printedName = owner.title.concat(' ').concat(owner.name)
```

printedName → a String.
owner → a Customer instance.
owner.title → a String.
owner.name → a String.

String is a recognized OCL type.
concat is a *String* operation, with the signature

concat(*String*): *String*.


LoyaltyProgram
enroll(c:Customer)
1..*
1..*
partners
ProgramPartner
numberOfCustomers: Integer
partner
1
deliveredServices
0..*
Service
condition: Boolean
pointsEarned: Integer
pointsBurned: Integer
description: String
generatedBy
1
0..*
transactions
Date
$now: Date
isBefore(t:Date): Boolean
isAfter(t:Date): Boolean
=(t:Date): Boolean
1
{ordered}
1..*
1
ServiceLevel
name: String
level
1
0..*
availableServices
0..*
program
0..*
Membership
actualLevel
0..1
LoyaltyAccount
points: Integer
earn(i: Integer)
burn(i: Integer)
isEmpty(): Boolean
account
1
transactions
0..*
Transaction
points: Integer
date:Date
program(): LoyaltyProgram
Burning
Earning
0..*
Customer
name: String
title:String
isMale: Boolean
dateOfBirth: Date
age(): Integer
1
owner
0..*
cards
card
1
CustomerCard
valid: Boolean
validForm: Date
goodThru: Date
color: enum{silver, gold}
printedName: String
card
1
0..*
transactions

Invarianty : Navigace přes associace

Navigation from a class through an association class uses the association class name to obtain all tuples of an object:

"The cards of the memberships of a customer are only the customer's cards":

```
context Customer
invariant correctCard:
cards->includesAll(Membership.card)
```

This is exactly the same as the previous constraint:

"The owner of the card of a membership must be the customer in the membership":

```
context Membership
invariant correctCard: card.owner = customer
```

The Membership correctCard constraint is better!

Invariants using Navigation through Associations with "Many" Multiplicity

Navigation over associations roles with **multiplicity greater than 1** yields a Collection type. Operations on collections are accessed using an arrow **"->"**, followed by the operation name.

"A customer card belongs only to a membership of its owner":

```
context CustomerCard
invariant correctCard:
owner.Membership->includes(membership)
```

owner → a *Customer* instance.
owner.Membership → a set of *Membership instances*.
membership → a *Membership instance*.
includes is an operation of the OCL *Collection* type.

Navigating to collections

```
context Customer
account
```

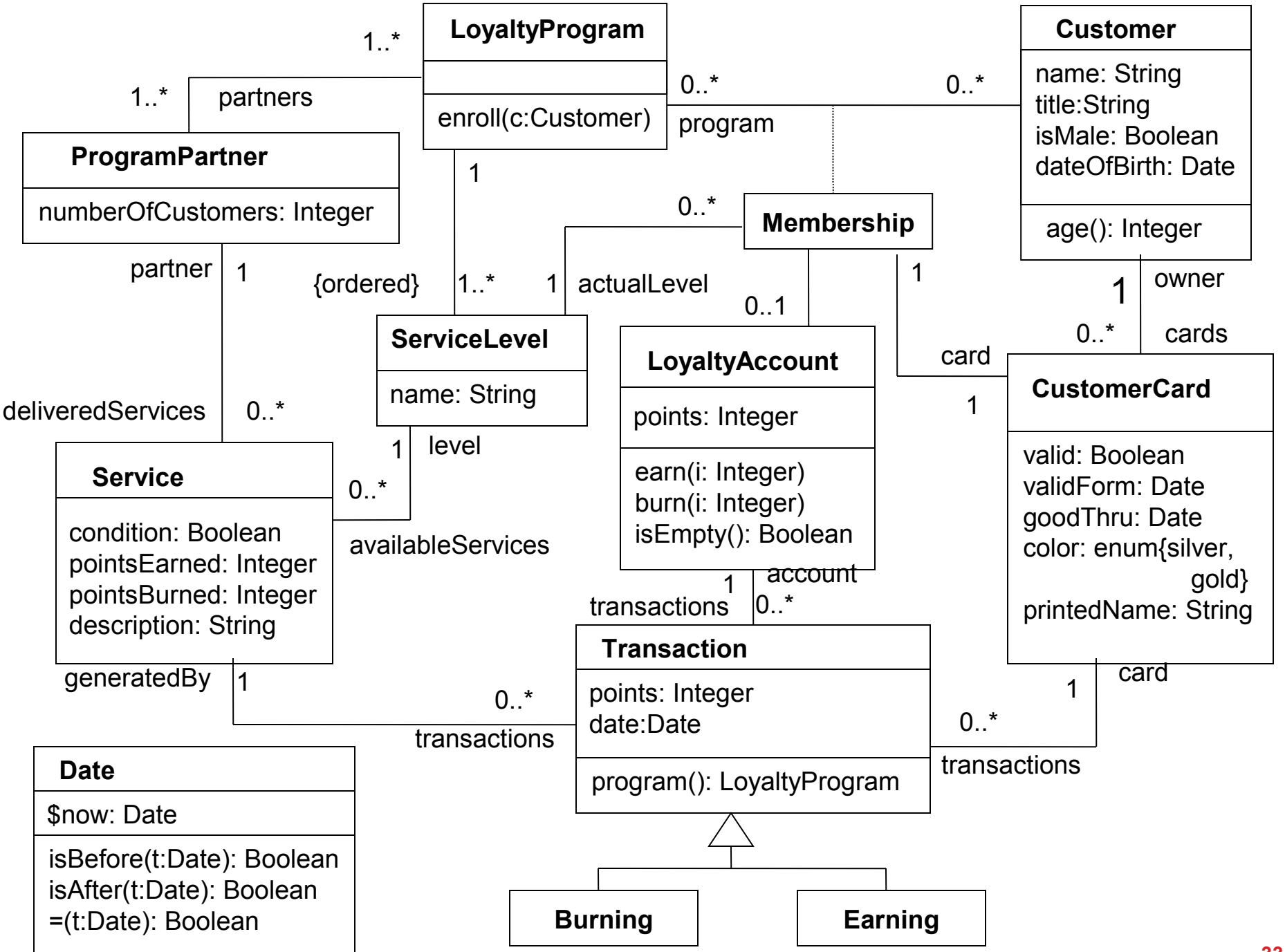
produces a **set** of Accounts

```
context Customer_
account.transaction
```

produces a **bag** of transactions

If we want to use this as a set we have to do the following

```
account.transaction -> asSet
```


LoyaltyProgram
enroll(c:Customer)
Customer
name: String
title:String
isMale: Boolean
dateOfBirth: Date
age(): Integer
ProgramPartner
numberOfCustomers: Integer
Membership
ServiceLevel
name: String
LoyaltyAccount
points: Integer
earn(i: Integer)
burn(i: Integer)
isEmpty(): Boolean
CustomerCard
valid: Boolean
validForm: Date
goodThru: Date
color: enum{silver, gold}
printedName: String
Service
condition: Boolean
pointsEarned: Integer
pointsBurned: Integer
description: String
Transaction
points: Integer
date:Date
program(): LoyaltyProgram
Date
$now: Date
isBefore(t:Date): Boolean
isAfter(t:Date): Boolean
=(t:Date): Boolean
Burning
Earning
1..*
partners
1..*
0..*
program
0..*
1
0..*
partner
1
{ordered}
1..*
1
actualLevel
0..1
1
owner
1
0..*
cards
card
1
deliveredServices
0..*
1
level
0..*
availableServices
account
1
transactions
0..*
generatedBy
1
0..*
transactions
0..*
transactions
card
1

Navigation to Collections

"The partners of a loyalty program have at least one delivered service":

```
context LoyaltyProgram
invariant minServices:
partners.deliveredservices->size() >= 1
```

"The number of a customer's programs is equal to that of his/her valid cards":

```
context Customer
invariant sizesAgree:
Programs->size() = cards->select(valid=true)->size()
```

Navigation to Collections

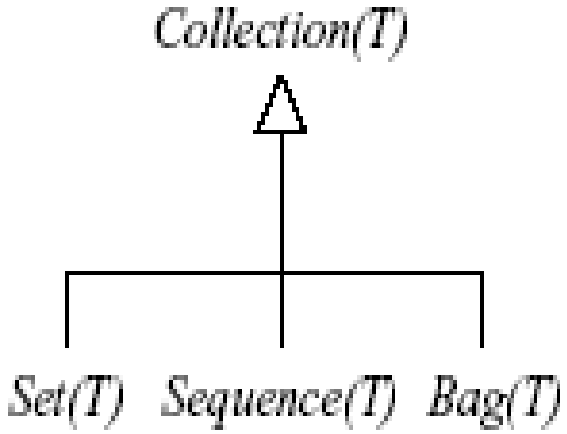
"When a loyalty program does not offer the possibility to earn or burn points, the members of the loyalty program do not have loyalty accounts. That is, the loyalty accounts associated with the Memberships must be empty":

```
context LoyaltyProgram
invariant noAccounts:
   partners.deliveredservices->
   forAll(pointsEarned = 0 and pointsBurned = 0)
            implies Membership.account->isEmpty()

and, or, not, implies, xor are logical connectives.
```

The OCL Collection types

- **Collection is a predefined OCL type**
  - Operations are defined for collections
  - They never change the original
- **Three different collections:**
  - Set (no duplicates)
  - Bag (duplicates allowed)
  - Sequence (ordered Bag)



- **With collections type, an OCL expression either states a fact about all objects in the collection or states a fact about the collection itself, e.g. the size of the collection.**
- **Syntax:**
  - collection->operation

Collection Operations

<collection> → size ( )

→ isEmpty ( )

→ notEmpty ( )

→ sum ( )

→ count (object)

→ includes (object)

→ includesAll (collection)

Collections cont.

<collection> → select (e:T | <b.e.>)

→ reject (e:T | <b.e.>)

→ collect (e:T | <v.e.>)

→ forAll (e:T* | <b.e.>)

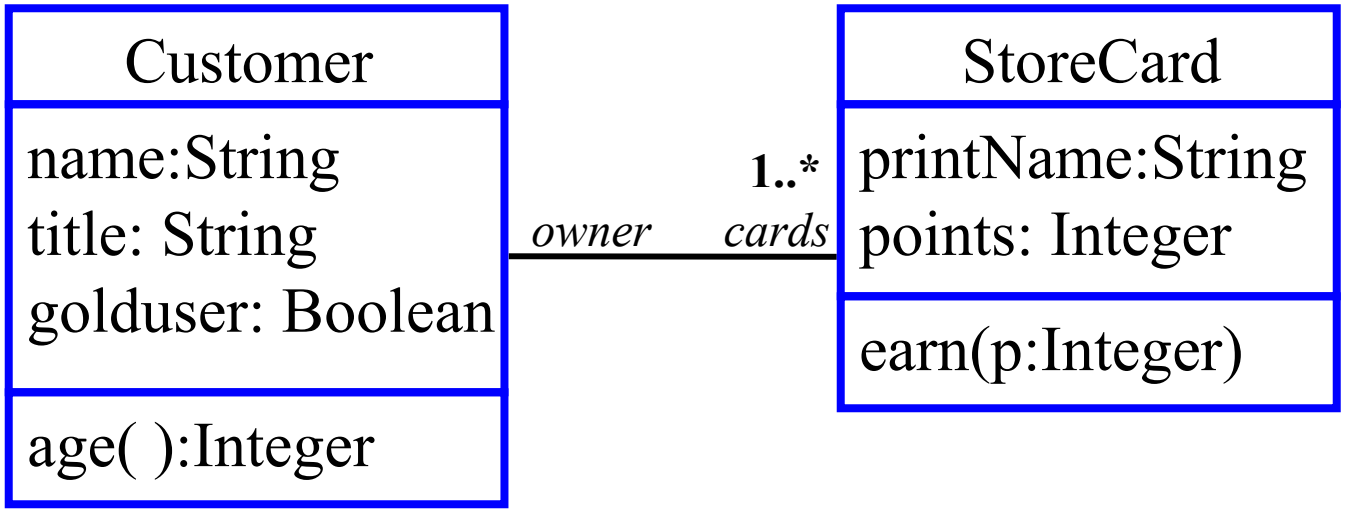
→ exists (e:T | <b.e.>)

→ iterate ($e:T_1$; $r:T_2$ = <v.e.> | <v.e.>)

b.e. stands for: boolean expression

v.e. stands for: value expression

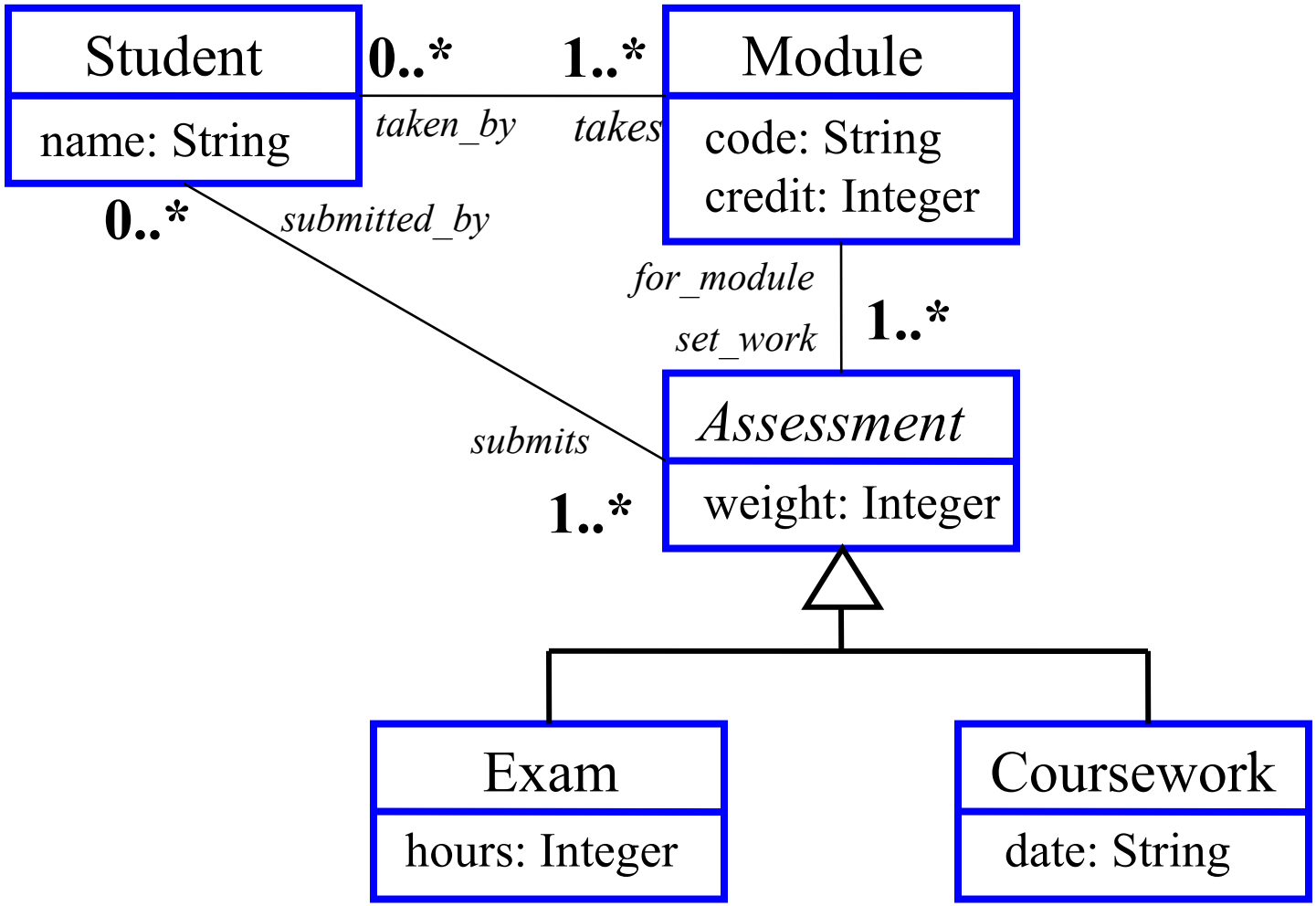
Changing the context



context *StoreCard*
invariant: *printName = owner.title.concat(owner.name)*

context *Customer*
cards →forAll (
printName = owner.title.concat(owner.name))

Example UML diagram



Constraints

a) Modules can be taken iff they have more than seven students registered

b) The assessments for a module must total 100%

c) Students must register for 120 credits each year

d) Students must take at least 90 credits of CS modules each year

e) All modules must have at least one assessment worth over 50%

f) Students can only have assessments for modules which they are taking

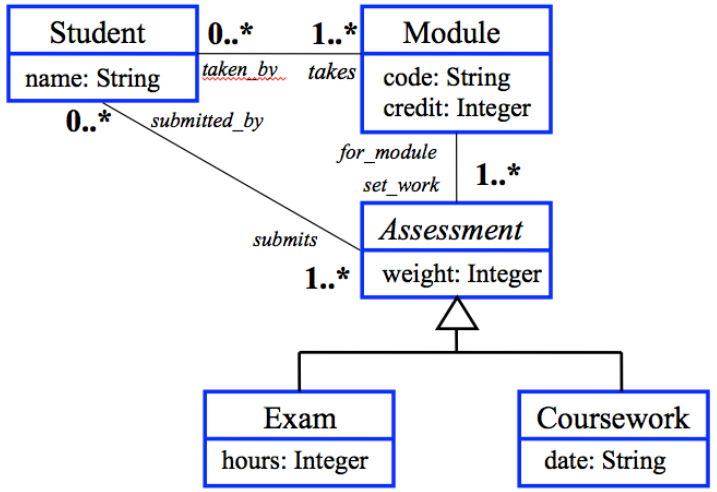
Constraint (a)

Modules can be taken iff they have more than seven students registered

Note: when should such a constraint be imposed?

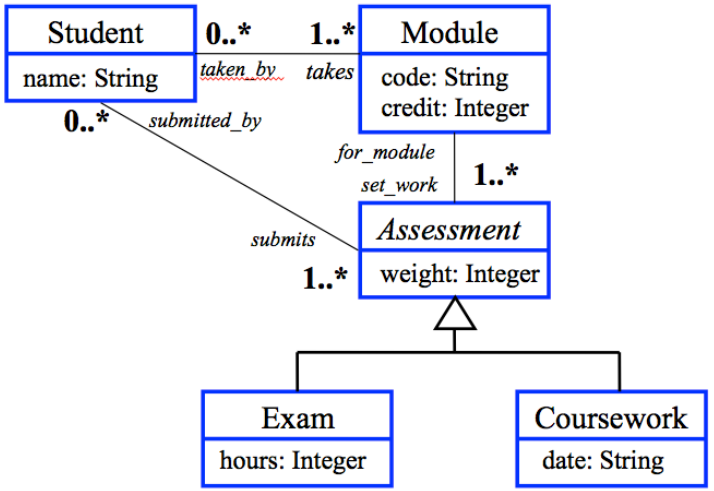
context *Module*

invariant: *taken_by→size* > *7*



Constraint (b)

The assessments for a module must total 100%



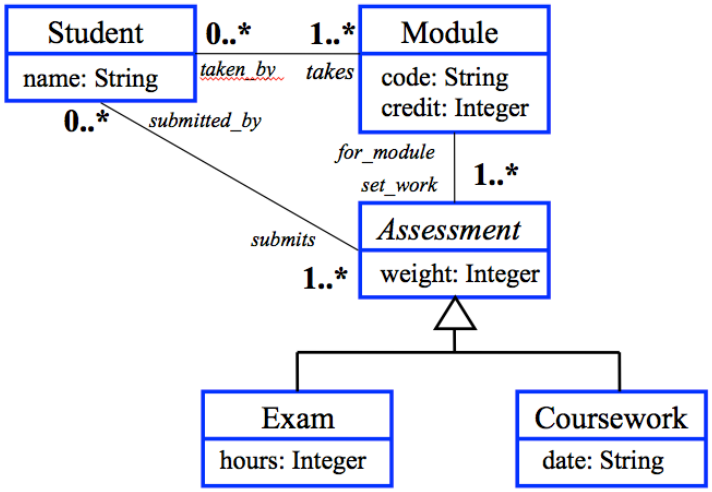
context *Module*

invariant:

set_work.weight→sum( ) = 10(

Constraint (c)

Students must register for 120 credits each year

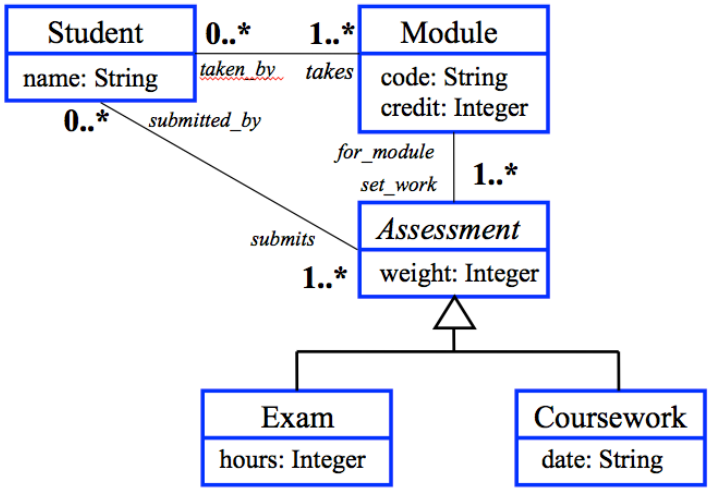


context *Student*

invariant: *takes.credit→sum( ) = 120*

Constraint (d)

Students must take at least 90 credits of CS modules each year



context *Student*

invariant:

takes →

***select*(*code.substring*(*1,2*) = *'CS'*).credit→sum( )**

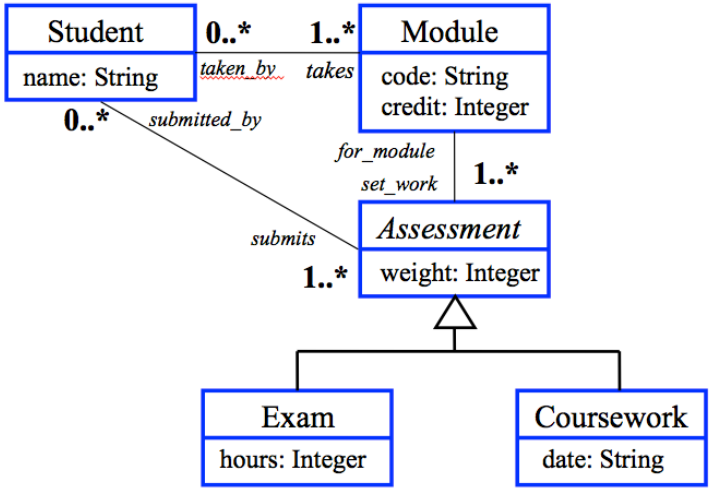
>= 90

Constraint (e)

All modules must have at least one assessment worth over 50%

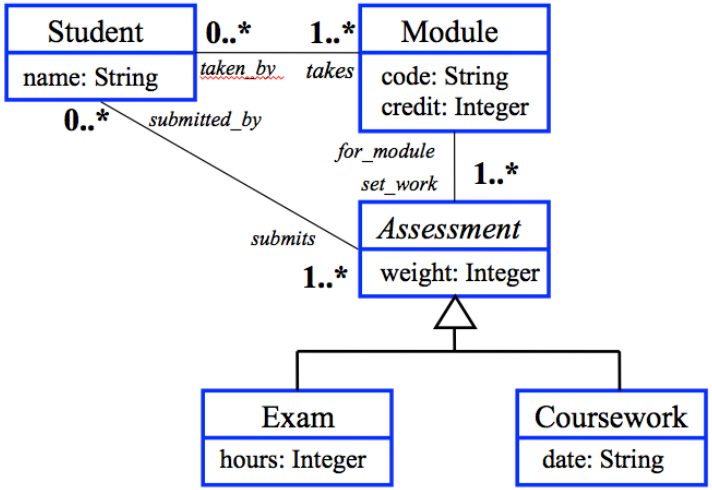
context *Module*

invariant: *set_work→exists(weight > 50)*



Constraint (f)

Students can only have assessments for modules which they are taking



context *Student*

invariant: *takes→includesAll(submits.for_module)*

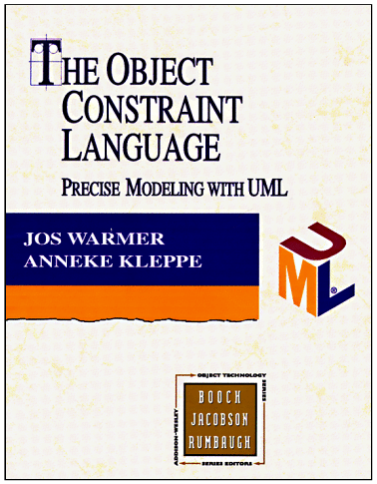
OCL Constraints

- ***A constraint is a restriction on one or more values of (part of) an object model/system.***
- **Constraints come in different forms:**
  - invariant
    - constraint on a class or type that must always hold
  - pre-condition
    - constraint that must hold before the execution of an op.
  - post-condition
    - constraint that must hold after the execution of an op.
  - guard
    - constraint on the transition from one state to another.

We studied only class constraints (invariants).

Further Resources for OCL

- **The Object Constraint Language**
  - ISBN 0-201-37940-6
- **OCL home page**
  - www.klasse.nl/ocl/index.htm



Further Resources for OCL

- **The Amsterdam Manifesto on OCL**
  - In Object Modeling with the OCL (LNCS2263) p115-149
- **The Object Constraint Language, Precise Modeling with UML, Addison-Wesley, 1999.**
- **The Object Constraint Language, Precise Modeling with UML 2nd**
- **Response to the UML 2.0 OCL RfP (ad/2000-09-03) Revised Submission, Version 1.6 January 6, 2003**
- **Some Shortcomings of OCL, the Object Constraint Language of UML**
  - Mandana Vaziri and Daniel Jackson,1999
- **http://www.klasse.nl/english/uml/ UML CENTER**
- **Informal formality? The Object Constraint Language and its application in the UML metamodel**
  - Anneke Kleppe, Jos Warmer, Steve Cook
- **A Pratical Application of the Object Constraint Language OCL**
  - Kjetil Moage
- **The UML's Object Constraint Language: OCL Specifying Components, JAOO Tutorial – September 2000**
  - Jos Warmer & Anneke Kleppe