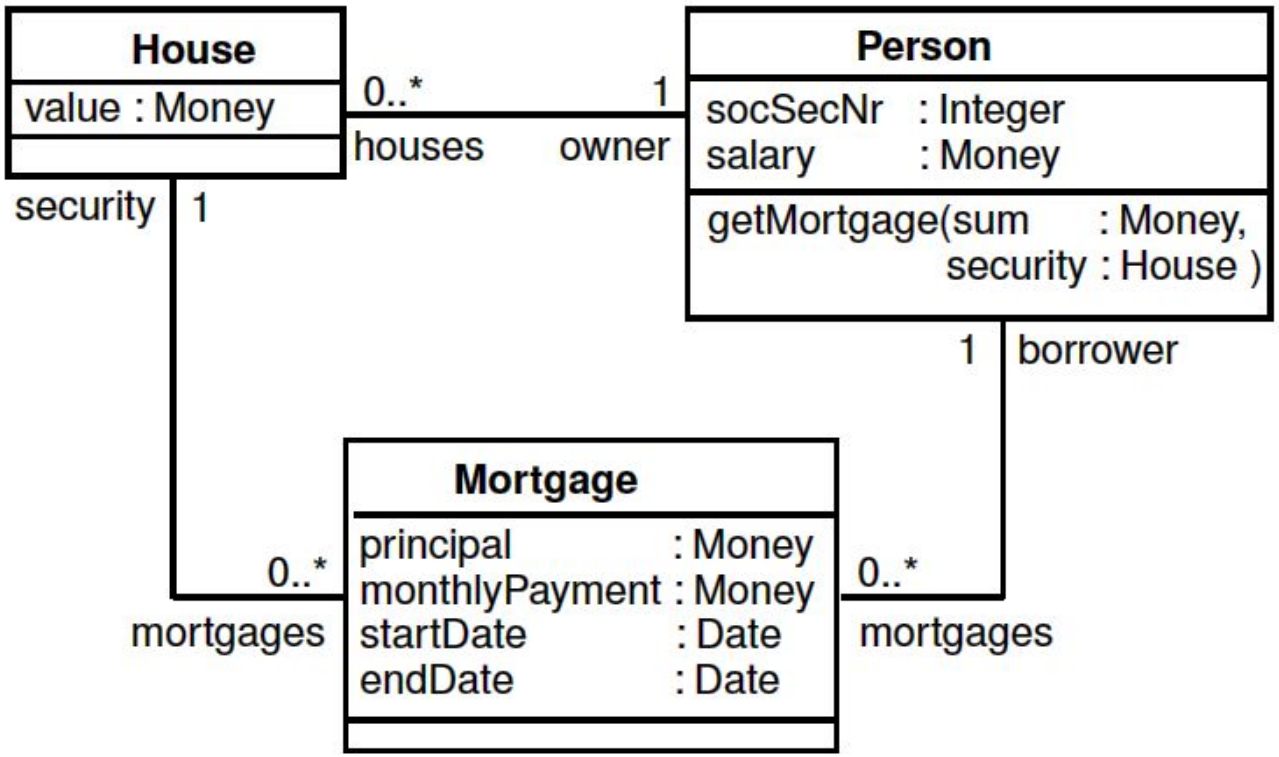
Métodos de Desenvolvimento de Software
Software Development Methods
(MDS)
2016/2017

Object Constraint Language (OCL)

A motivating example: A Mortgage System

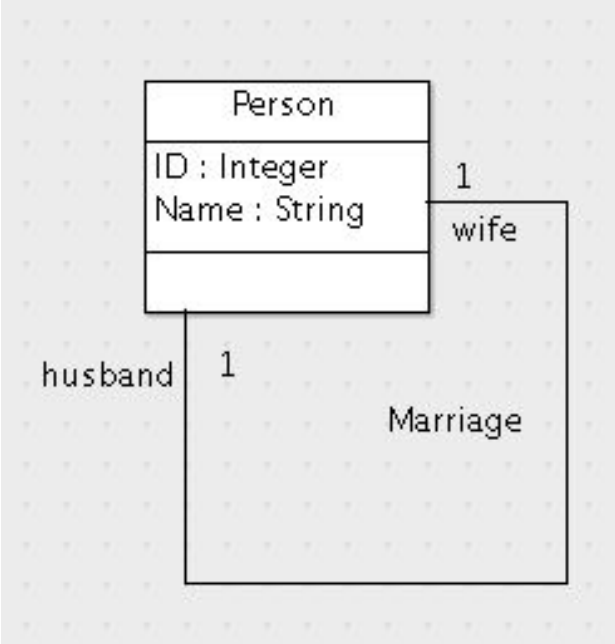


1. A person may have a mortgage only on a house she owns.
2. The start date of a mortgage is before its end date.

How can we modify this diagram to express that?

A motivating example: Marriage

- Person with name and ID
- Relation marriage



How can we modify this diagram to express that a person can not get married with herself?

History

- First developed in 1995 as IBEL by IBM's Insurance division for business modelling
- IBM proposed it to the OMG's call for an object-oriented analysis and design standard. OCL was then merged into UML 1.1.
- OCL was used to define UML 1.2 itself.

Companies behind OCL

- Rational Software, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies, Softeam

UML Diagrams are NOT Enough!

- We need a language to help specifying additional information in UML models.
  - We look for some “add-on”, not a new language with full specification capability.
  - Why not first order logic? – Not OO.
- OCL is used to specify constraints on OO systems.
  - OCL is not the only one.
  - But OCL is the only one that is standardized.
  - Attention: OCL is **not** a programming language:
    - No control flow, no side-effects.

Advantages of formal constraints

- Better documentation
  - Constraints add information about the model elements and their relationships to the UML models
- More precision
  - OCL constraints have formal semantics; used to reduce the ambiguity in the UML models
- Communication without misunderstanding
  - Using OCL constraints modelers can communicate unambiguously

Where to use OCL?

- Specify invariants for classes and types
- Specify pre- and post-conditions for methods
- As a navigation language
- To specify constraints on operations
- Test requirements and specifications

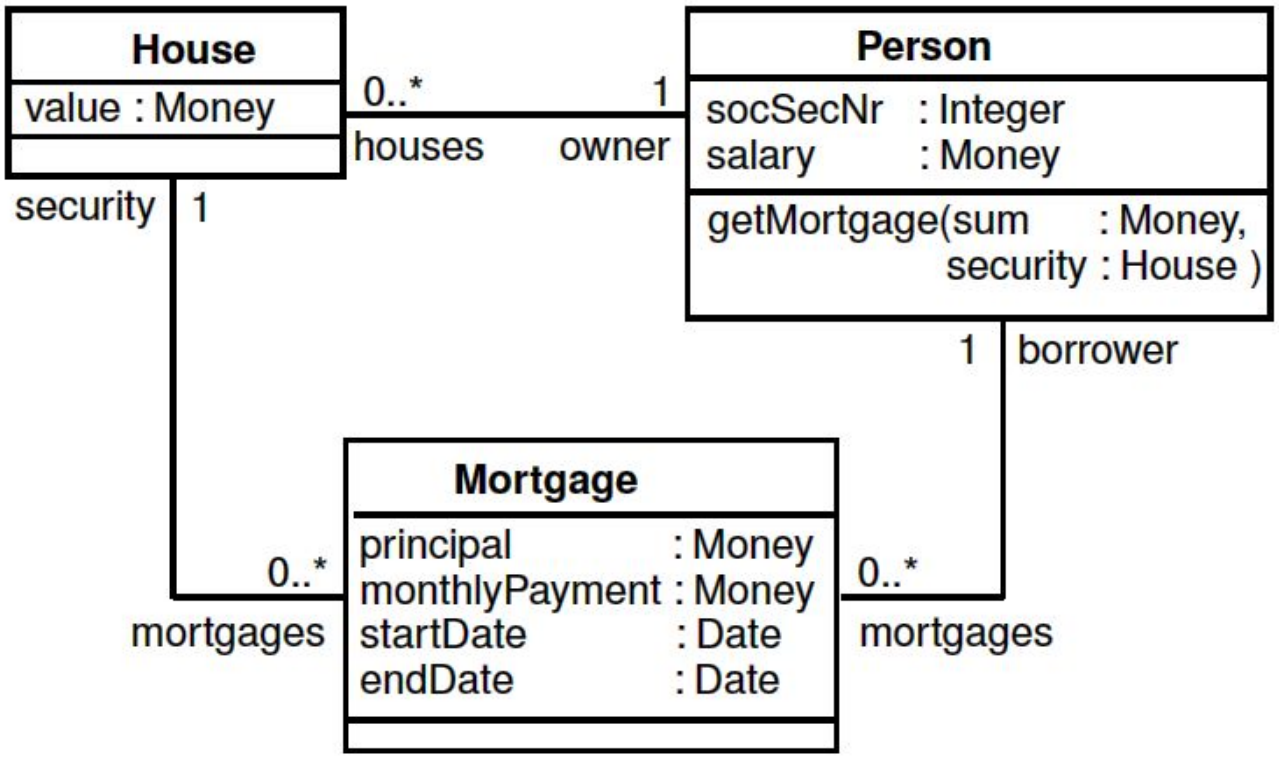
Combining UML and OCL

- Without OCL expressions, many models would be severely underspecified;
- Without the UML diagrams, the OCL expressions would refer to non-existing model elements,
  - there is no way in OCL to specify classes and associations.
- Only when we combine the diagrams and the constraints can we completely specify the model.

Elements of an OCL expression that are associated with a UML model

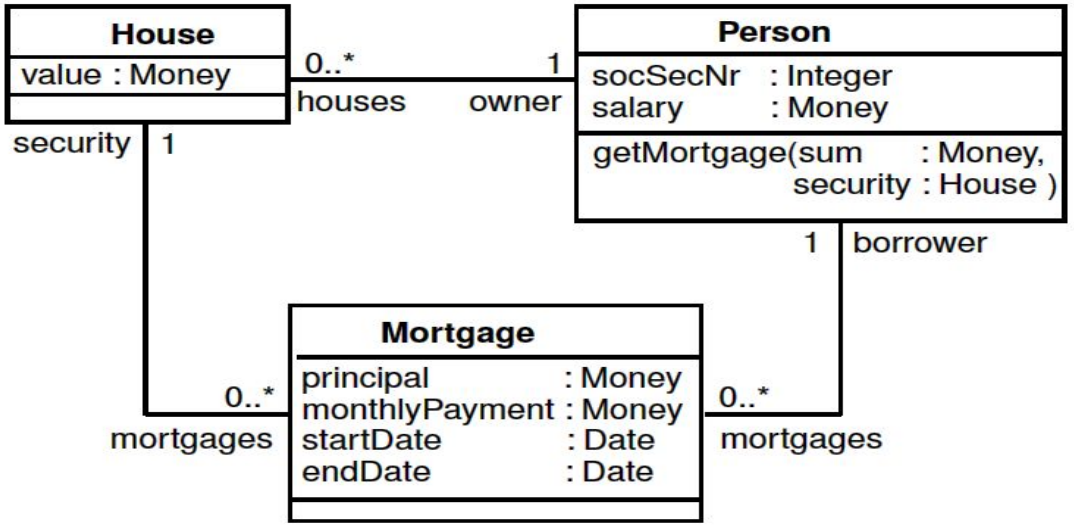
- OCLAny is the supertype of all types in OCL
- basic types (direct subtypes of OCLAny):
  - String, Boolean, Integer, Real
- from the UML model:
  - classes and their attributes
  - enumeration types
  - associations

Motivational Example: A Mortgage System



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.

Motivational Example: OCL specification of the constraints



When the name of an association-end is missing at one of the ends of an association, the name of the type at the association end is used as the role name.
If this results in an ambiguity, the role name is mandatory.

If the role name is ambiguous, then it cannot be used in OCL.

A person may have a mortgage only on a house she owns

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

The start date of a mortgage is before its end date

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

OCL Constraints

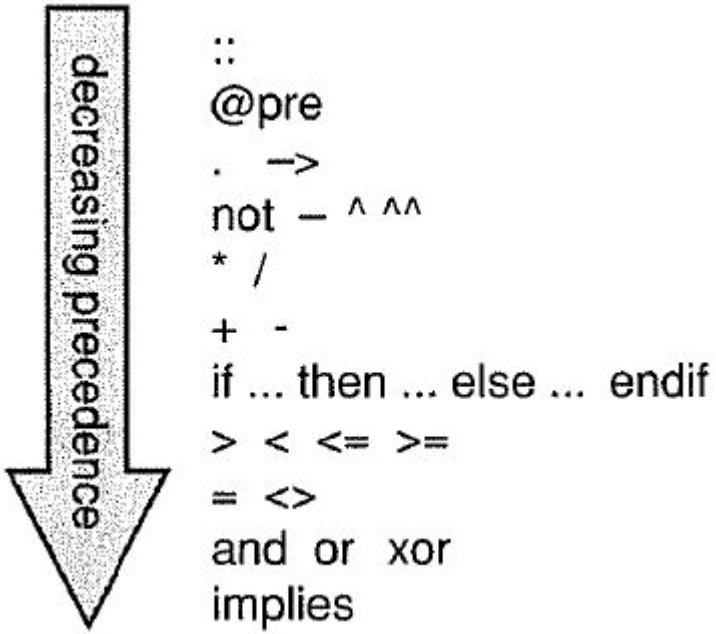
- A constraint is a restriction on one or more values of (part of) an object model/system.
- Constraints come in different forms:
  - invariant
    - constraint on a class or type that must always hold.
  - pre-condition
    - constraint that must hold before the execution of an op.
  - post-condition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

OclAny - Supertype

OclAny operation	Semantics
Comparison operations	
a = b	Returns true if a is the same object as b, otherwise returns false
a <> b	Returns true if a is *not* the same object as b, otherwise returns false
a.oclIsTypeOf(b : OclType) : Boolean	Returns true if a is the same type as b, otherwise returns false
a.oclIsKindOf(b : OclType) : Boolean	Returns true if a is the same type as b, or a subtype of b
a.oclInState(b : OclState) : Boolean	Returns true if a is in the state b, otherwise returns false
a.oclIsUndefined() : Boolean	Returns true if a = OclUndefined
Query operations	
A::allInstances( ) : Set(A)	This is a class scope operation that returns a Set of all instances of type A

OCL Expressions and Constraints

- Each OCL expression has a type.
- Every OCL expression indicates a value or object within the system.
  - 1+3 is a valid OCL expression of type Integer, which represents the integer value 4.
  - An OCL expression is valid if it is written according to the rules (formal grammar) of OCL.



OCL Standard Types and operators

Type	Operations
Boolean	=, not, and, or, xor, implies, if-then-else
Real	=, +, -, *, /, abs, floor, max, min, <, >, <=, >=
Integer	=, +, -, *, /, abs, div, mod, max, min, <, >, <=, >=
String	=, size, toLower, toUpper, concat, substring

Let Expressions

```
let <variableName>:<variableType> = <letExpression> in
  <usingExpression>
```

Constraints (invariants), Contexts and Self

- A **constraint (invariant)** is a boolean OCL expression
  - evaluates to **true/false**.
- Every **constraint is bound** to a specific type (class, association class, interface) in the UML model – its context.
- The context objects may be denoted within the expression using the **keyword 'self'**.
- The context can be specified by:
  - Context <context name>
  - A dashed note line connecting to the context figure in the UML models
- A constraint might have a name following the keyword invariant.

OCL expression syntax

- OCL expression may be broken down into three parts:
  - The package context (optional)
  - The expression context (mandatory)
  - One or more expressions

package <packagePath> — Package context

expression context { **context** <contexualInstanceName>: **<modelElement>**

<expressionType> <expressionName>: — expression
<expressionBody>

<expressionType> <expressionName>: — expression
<expressionBody>

…

endpackage

OCL expression syntax

- The context keyword introduces the context for the expression
  - The keywords inv, pre, and post denote the stereotypes, respectively «invariant», precondition», and «postcondition» of the constraint.

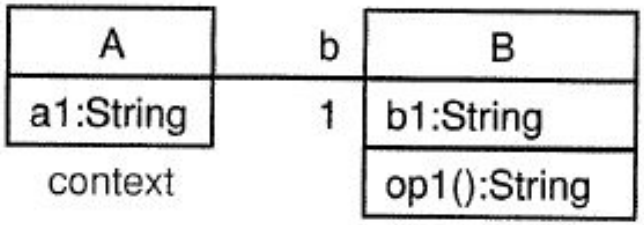
```
package Package::SubPackage
  context X inv:
      ... some invariant ...
  context X::operationName(..)
     pre: ... some precondition ...
endpackage
```

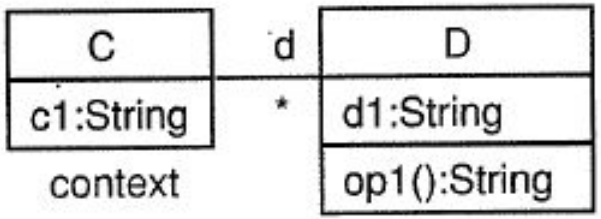
Navigation and naming rules

- **Rule 0** - Class names start with an uppercase letter and role names with a lowercase letter
- **Rule 1** - While navigating from a class to another, if the role of the destination class is defined then use it. Otherwise apply rule 2
- **Rule 2** - While navigating from a class to another, if the role of the destination class is not defined, then use the name of the destination class starting with a lowercase

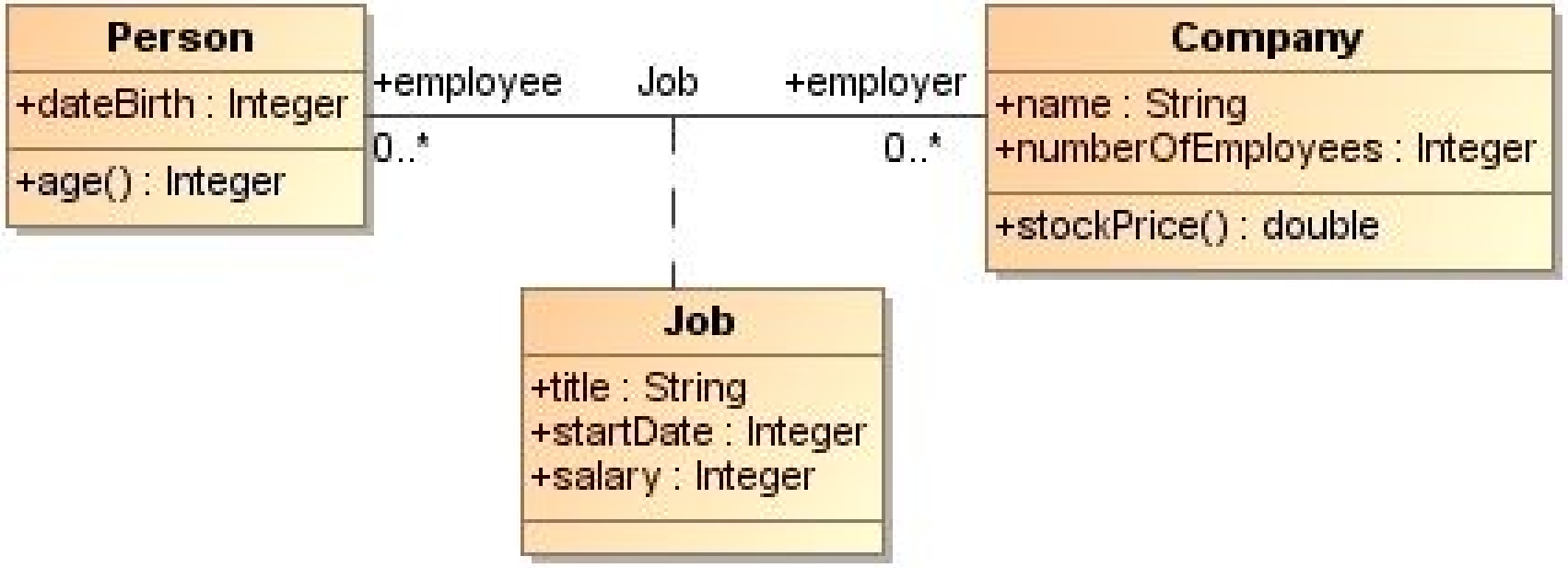
Navigation and collections

- OCL expressions can be built by navigating in the class diagram
- By definition, the result of **navigating through just one association is a Set**
- The result of **navigating through more than one association where at least one has multiplicity many is a Bag**.
  - Exception: if the association is adorned with the {ordered} tag, we get a Sequence.

Example model	Navigation expressions (A is the expression context)	
	Expression	Value
	self	The contextual instance – an instance of A
	self.b	An object of type B
	self.b.b1	The value of attribute B::b1
	self.b.op1()	The result of operation B::op1()

Example model	Navigation expressions	
	Expression	Value
	self	The contextual instance – an instance of C
	self.d	A Set(D) of objects of type D
	self.d.d1	A Bag(String) of the values of attribute D::d1 Shorthand for self.d->collect(d1)
	self.d.op1()	A Bag(String) of the results of operation D::op1() Shorthand for self.d->collect(op1())

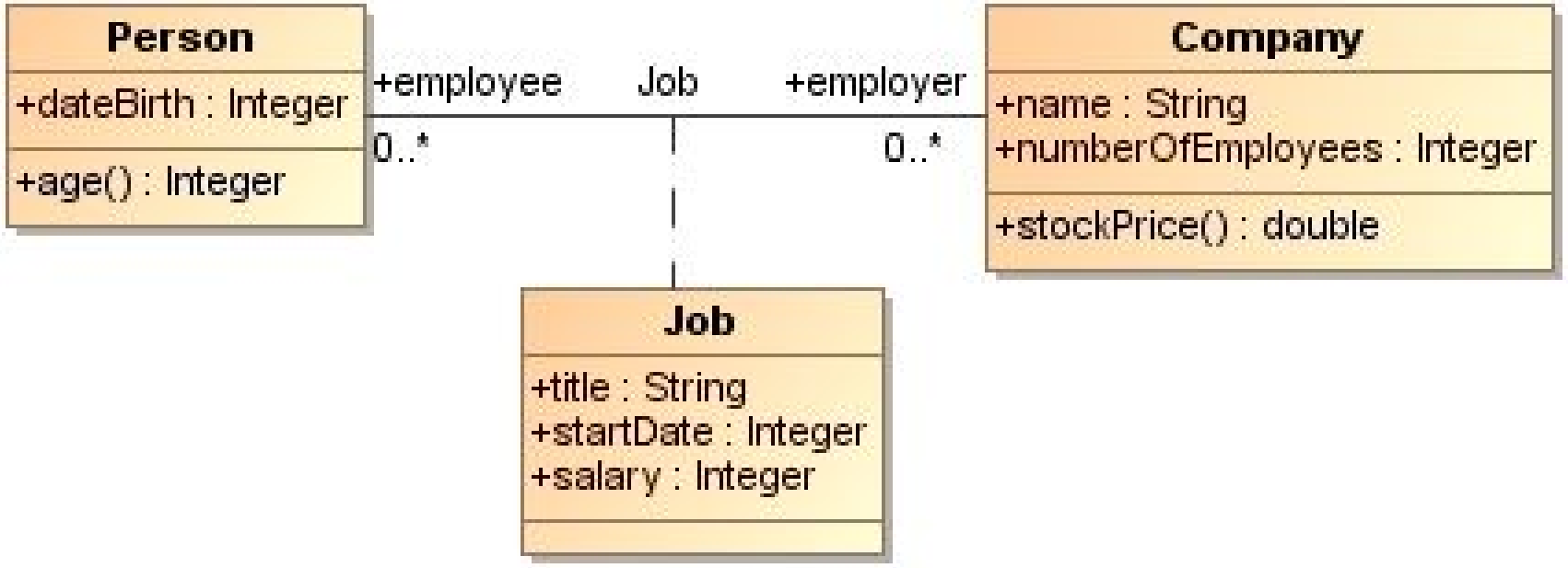
Self: examples



Example 1:
context Company **inv**: self.numberOfEmployees > 50

The label ***inv:*** declares the constraint to be an «invariant» constraint.

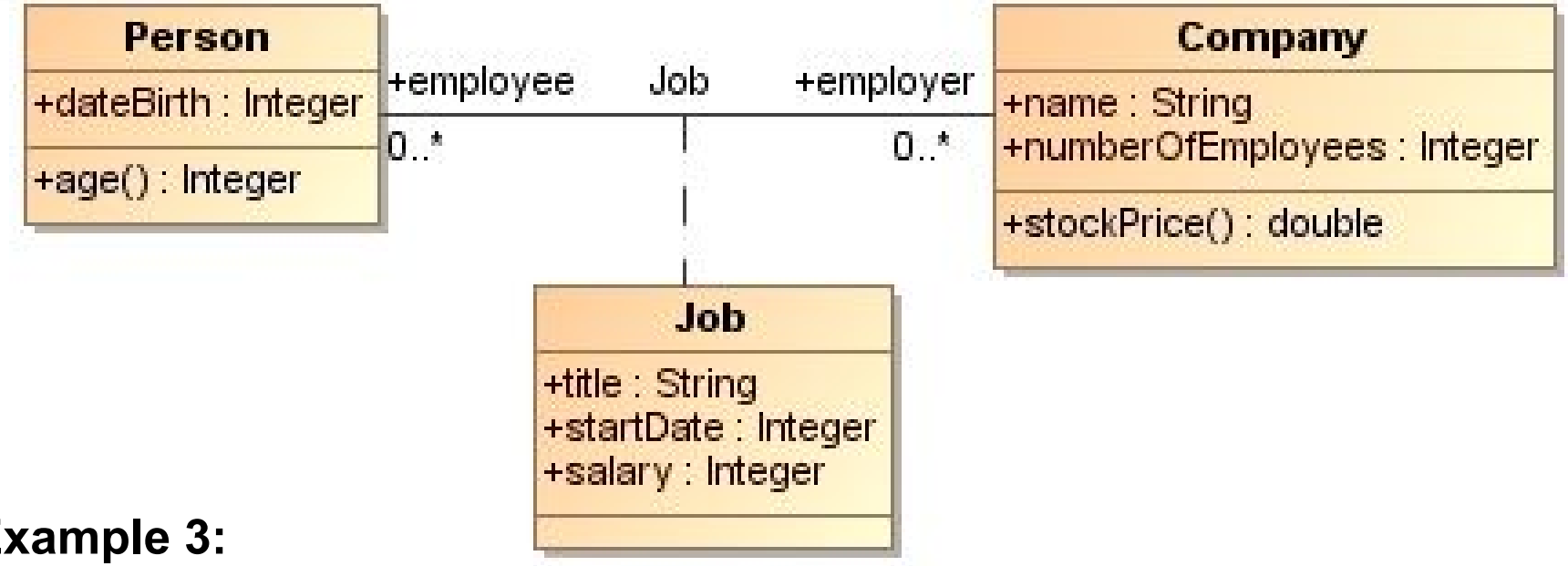
Self: examples



Example 2:
context c: Company
inv: c.numberOfEmployees > 50

The label ***inv:*** declares the constraint to be an «invariant» constraint.

Self: examples



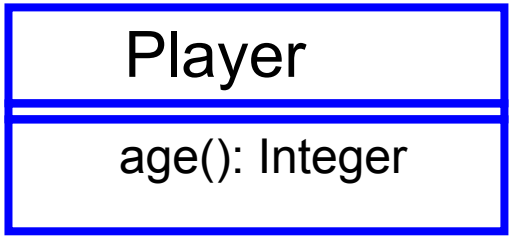
Example 3:
context Job
  inv: self.employer.numberOfEmployees >= 1
  inv: self.employee.age > 21

The label ***inv:*** declares the constraint to be an «invariant» constraint.

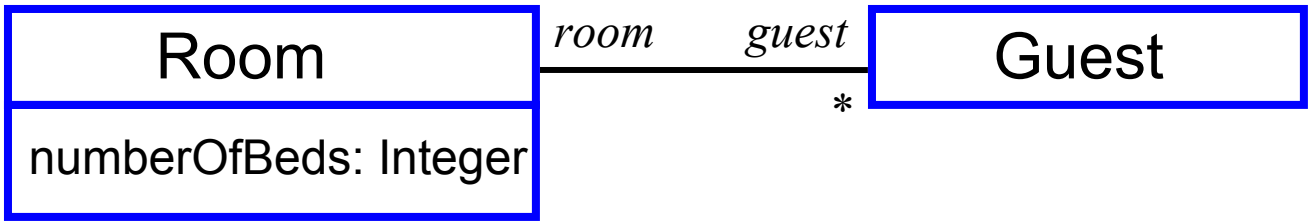
More Constraints

- All players must be over 18.

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



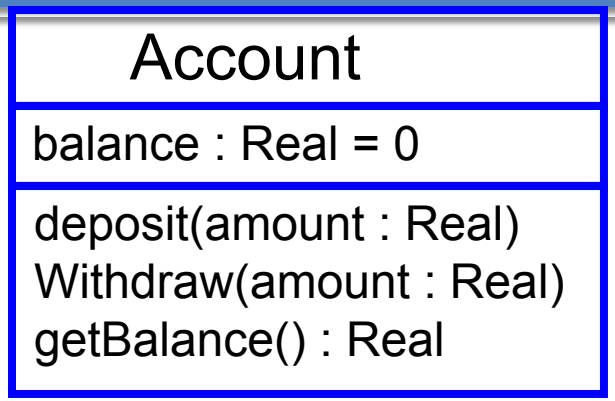
- The number of guests in each room doesn't exceed the number of beds in the room.




```
context Room
   inv:
   guest -> size <= numberOfBeds
```

The number of elements in the collection self

Pre conditions, post conditions and previous values



Balance before execution of operation

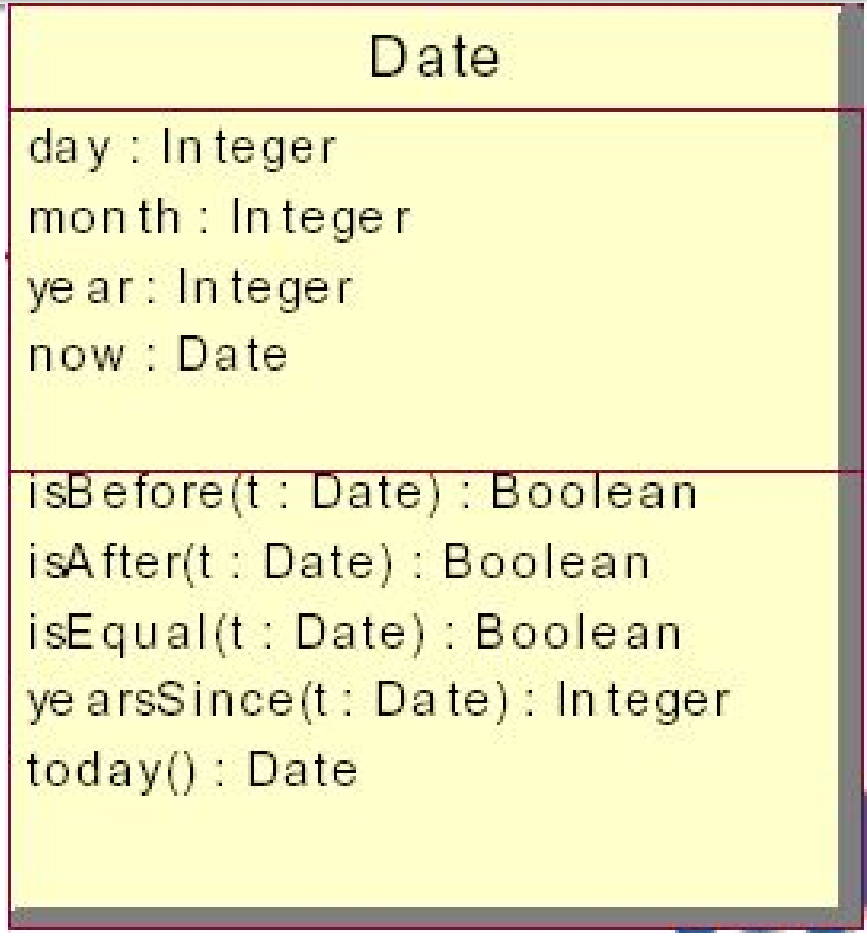
context Account::withdraw(amount : Real)
pre: amount <= balance
post: balance = balance@pre – amount

context Account::getBalance() : Real
post: result = balance

Return value of operation

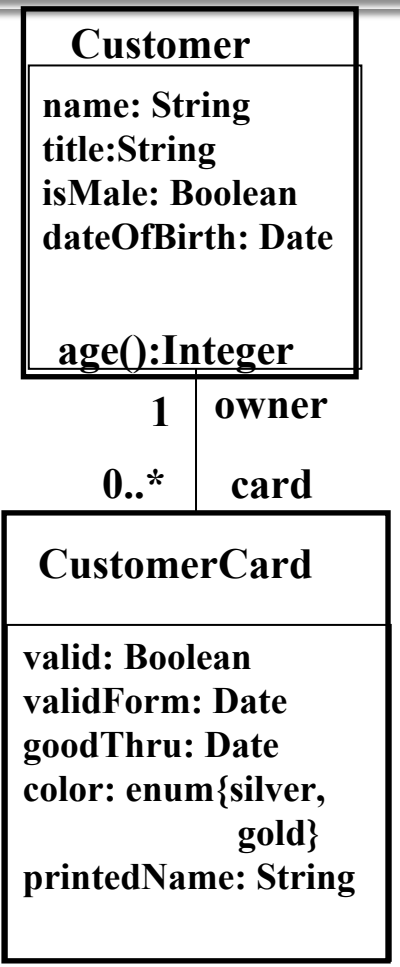
Expressing operation semantics


```
Date::isBefore(t:Date): Boolean =
    if self.year = t.year then
        if self.month = t.month then
            self.day < t.day
        else
            self.month < t.month
        endif
    else
        self.year < t.year
    endif
```



Invariants using Navigation over Association Ends – Roles

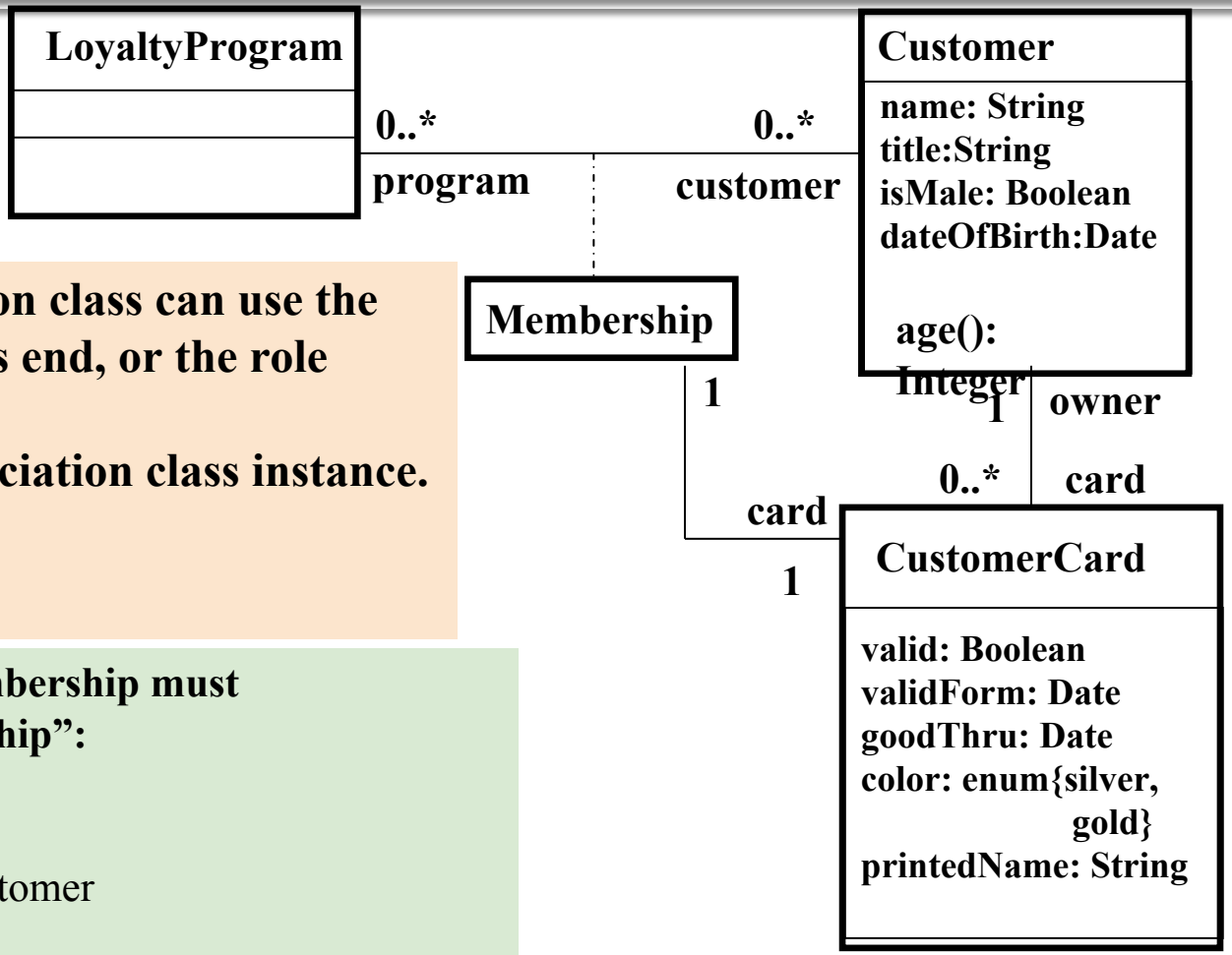

```
context CustomerCard
    inv printedName:
        printedName = owner.title.concat(' ').concat(owner.name)
```



Where:

- **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 signature concat(String): String**

Invariants using Navigation from Association Classes



Navigation from an association class can use the classes at the association class end, or the role names.
The context object is the association class instance.

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

context Membership
inv correctCard: card.owner = customer