Métodos de Desenvolvimento de Software (MDS)
2011/2012

Miguel Goulão
mgoul@fct.unl.pt
http://ctp.di.fct.unl.pt/~mgoul/

Aula 9

Object Constraint Language (OCL)

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 **<u>not</u>** 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

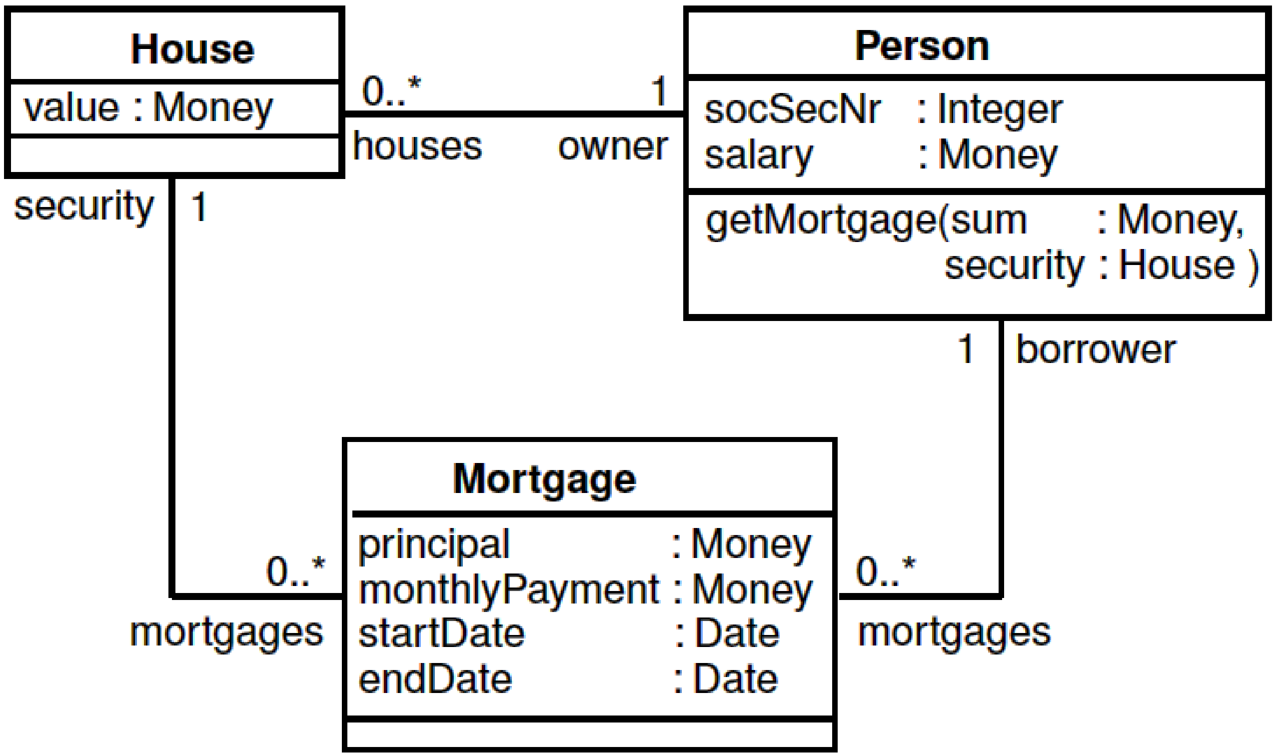
Combining UML and OCL

- 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

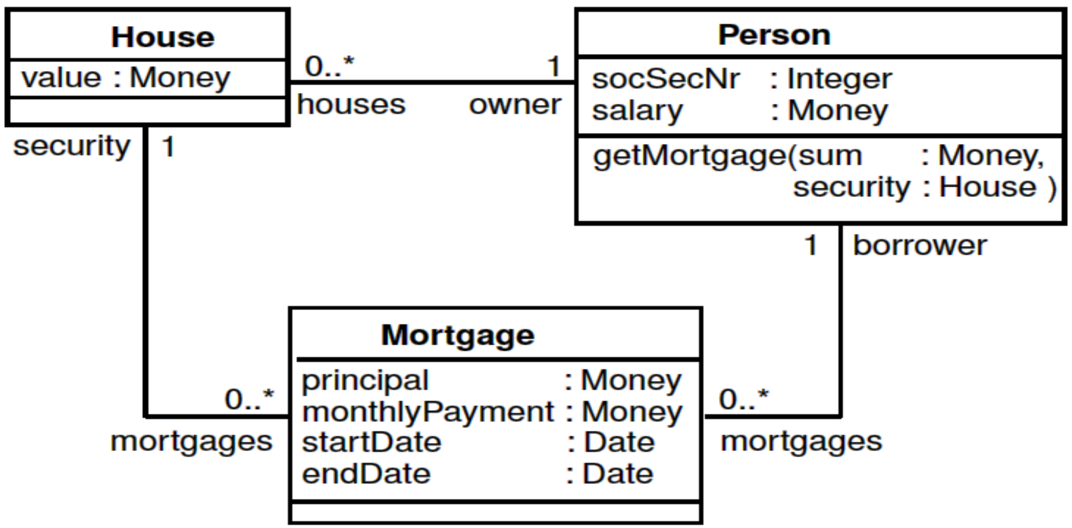
- basic types: String, Boolean, Integer, Real
- from the UML model:
  - classes and their attributes
  - enumeration types
  - associations

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.

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*
   invariant: *self.security.owner = self.borrower*

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

The start date of a mortgage is before its end date

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

context *Mortgage*
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.

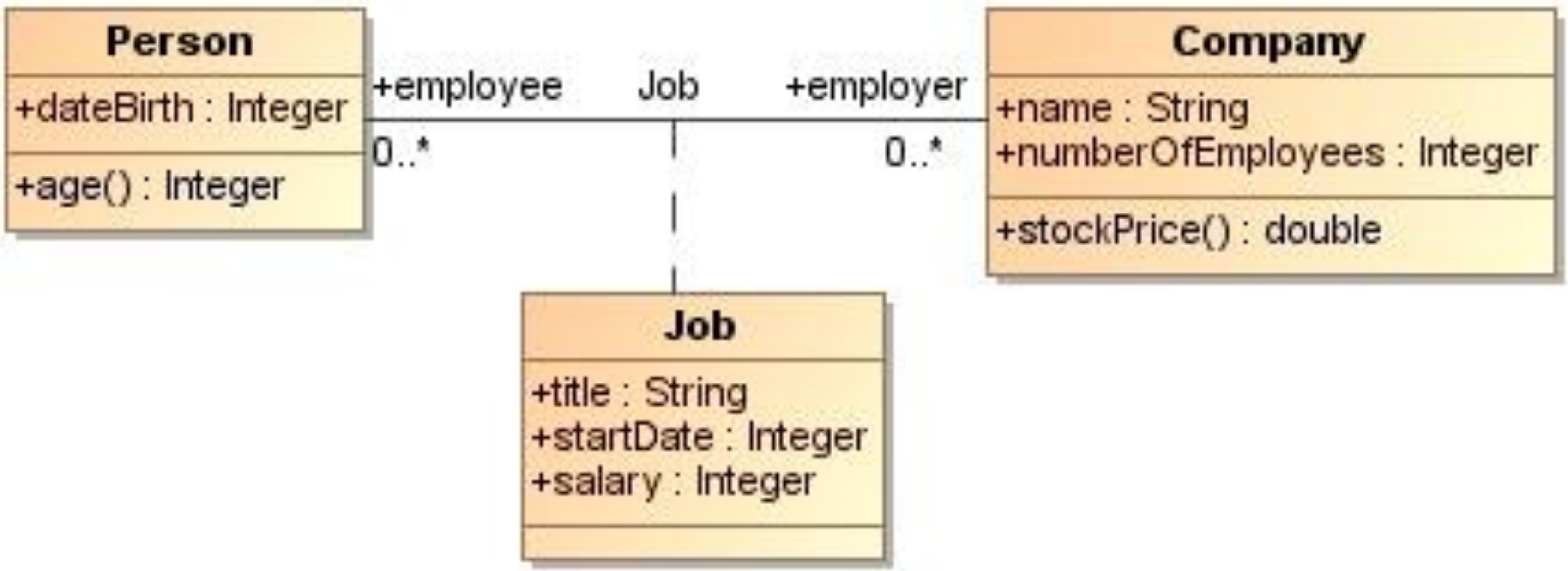
OCL Expressions and Constraints

- 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.
- A constraint is a valid OCL expression of type Boolean.

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.

Self: examples

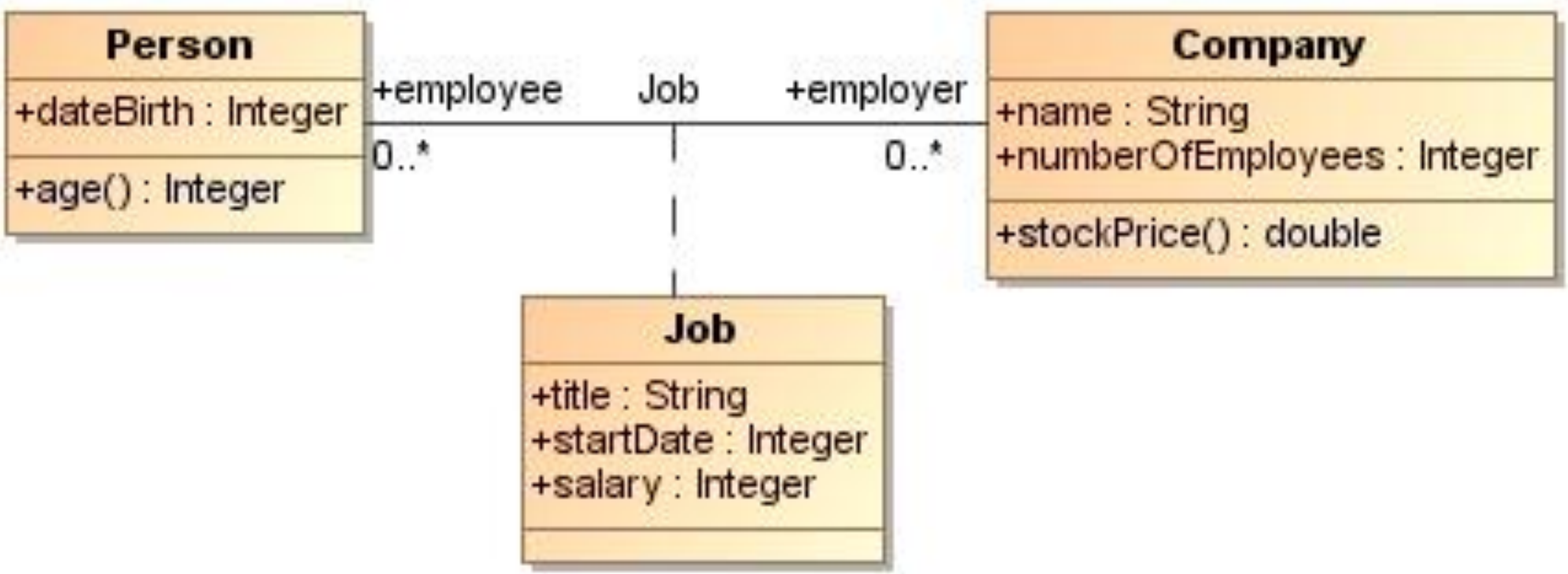


Example 1:
context Company **inv**: self.numberOfEmployees > 50

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

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Self: examples

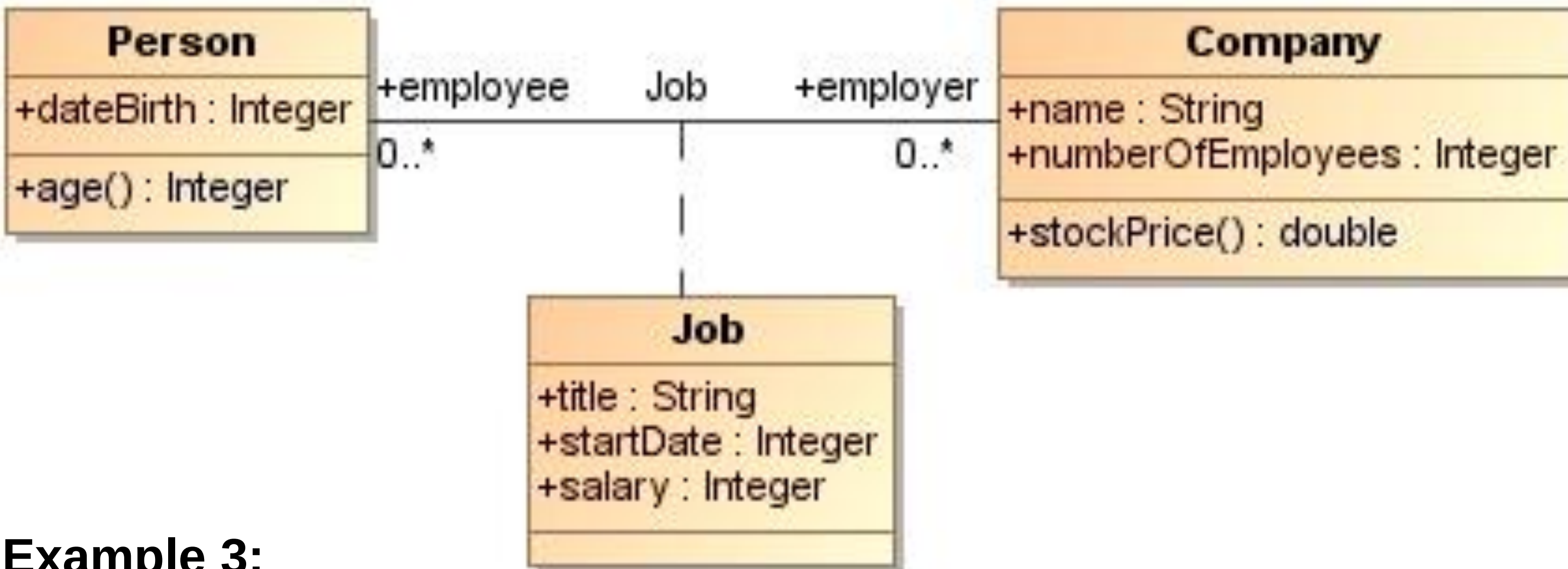


Example 2:
context c: Company **inv:** c.numberOfEmployees > 50

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

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Self: examples



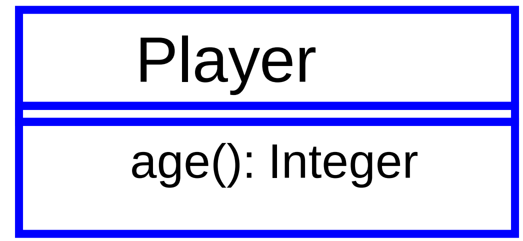
Example 3:
context Job
  inv: self.employer.numberOfEmployees >= 1
  inv: self.employee.age > 21

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

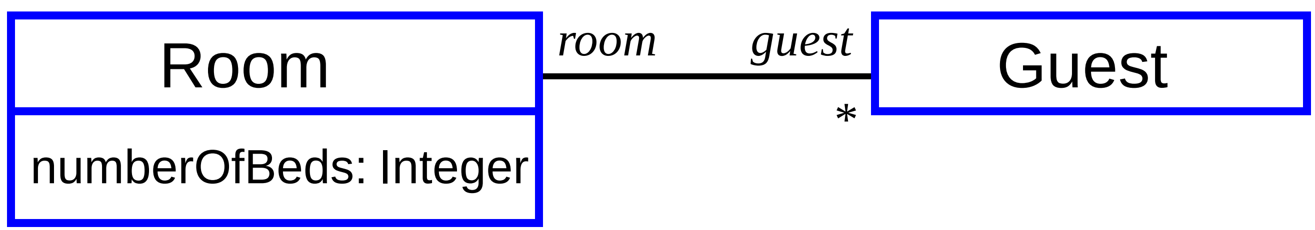
More Constraints

- All players must be over 18.

context Player **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

The number of elements in the collection self

Pre conditions, post conditions and previous values

Account
balance : Real = 0
deposit(amount : Real) Withdraw(amount : Real) getBalance() : Real

Balance before execution of operation

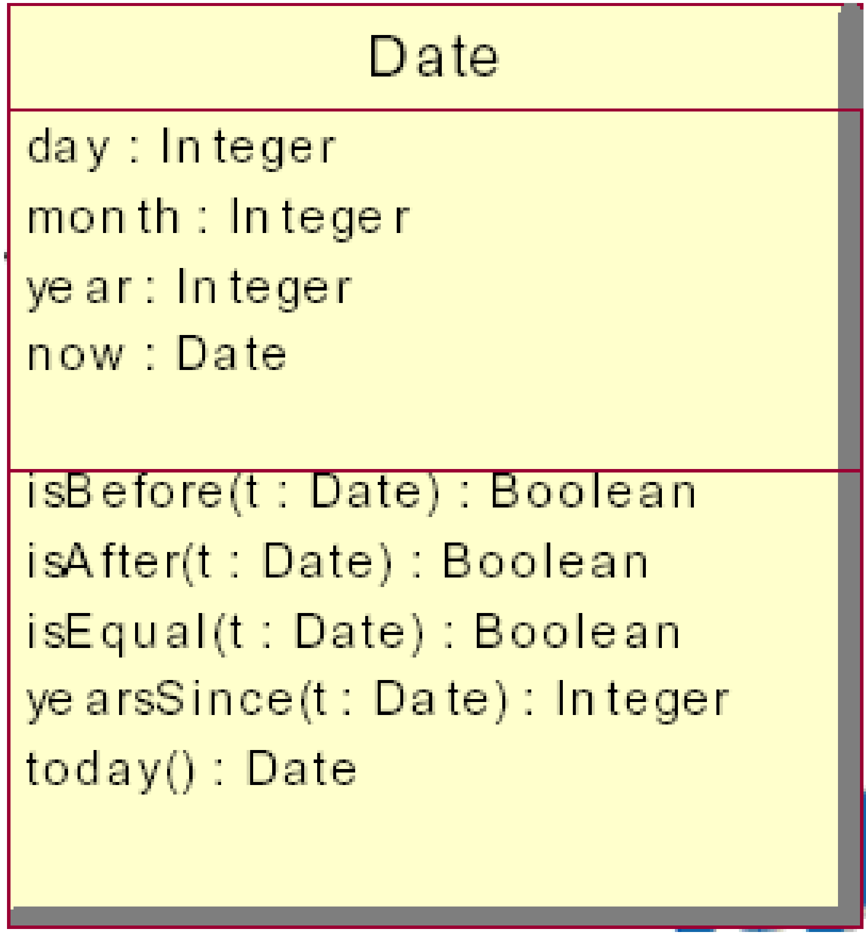
context Account::withdraw(amount : Real)
pre: amount <= balance
post: balance = balance@pre – amount

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
```



It is not our aim in MDS

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

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>: **<expressionBody>** expression

<expressionType> <expressionName>: **<expressionBody>** expression

…

endpackage

OCL expression syntax

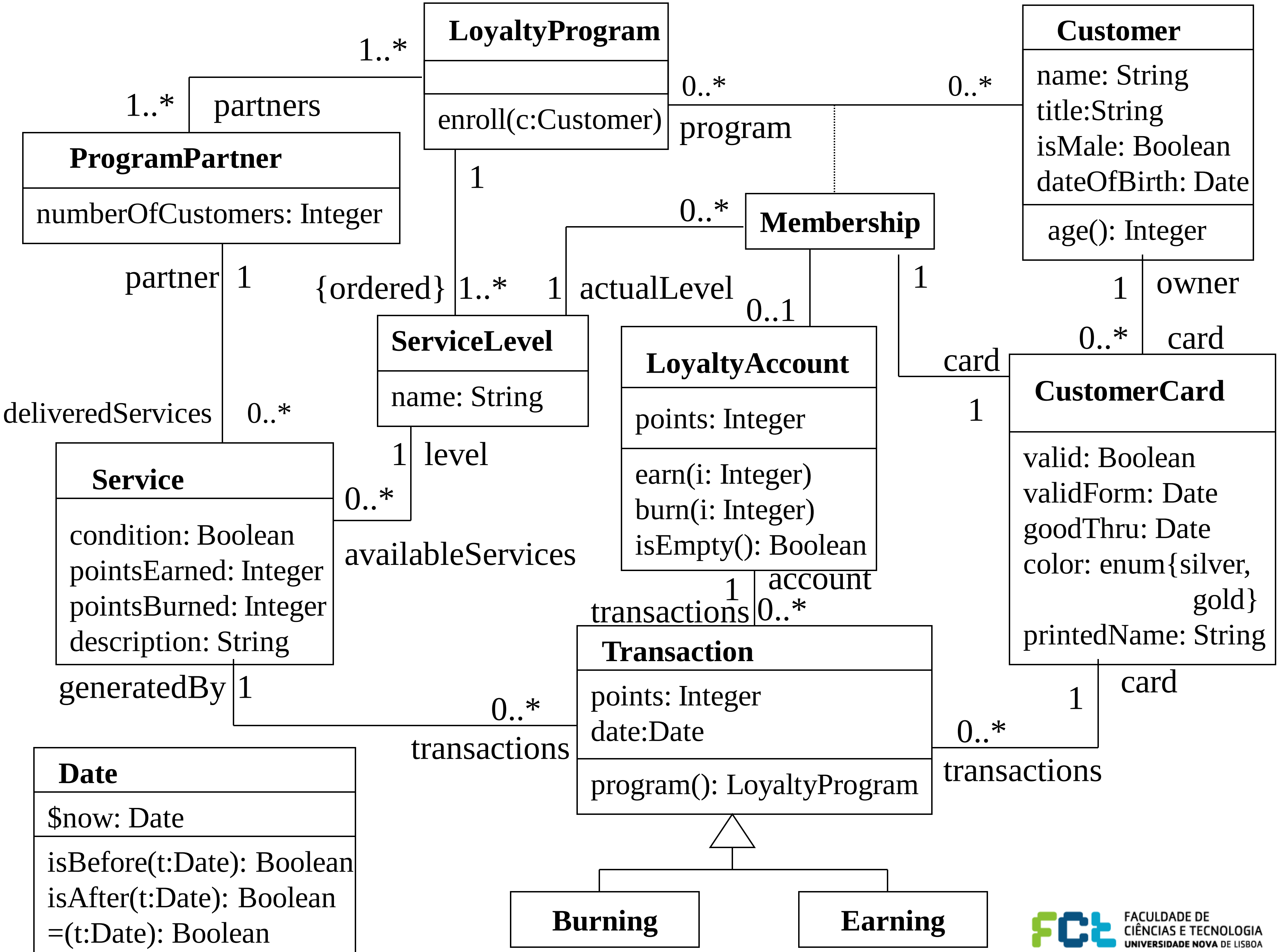
- The context keyword introduces the context for the expression
  - The keywords inv, pre, and post denote the stereotypes, respectively «invariant», precondition», and «postcondition» of the constraint.

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

Example of a static UML Model

- Problem story:

  A company handles loyalty programs (class LoyaltyProgram) for companies (class ProgramPartner) that offer their customers various kinds of bonuses. Often, the extras take the form of bonus points or air miles, but other bonuses are possible. Anything a company is willing to offer can be a service (class Service) rendered in a loyalty program. Every customer can enter the loyalty program by obtaining a membership card (class CustomerCard). The objects of class Customer represent the persons who have entered the program. A membership card is issued to one person, but can be used for an entire family or business. Loyalty programs can allow customers to save bonus points (class loyaltyAccount) , with which they can "buy" services from program partners. A loyalty account is issued per customer membership in a loyalty program (association class Membership). Transactions (class Transaction) on loyalty accounts involve various services provided by the program partners and are performed per single card. There are two kinds of transactions: Earning and burning. Membership durations determine various levels of services (class serviceLevel).



Adapted example from [Warmer & Kleppe 1999]

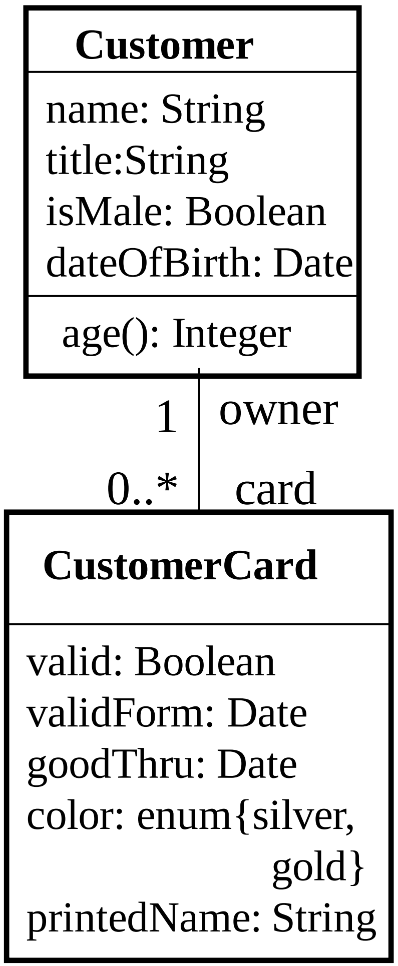
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Invariants on Attributes

- Invariants on attributes:

Named invariant

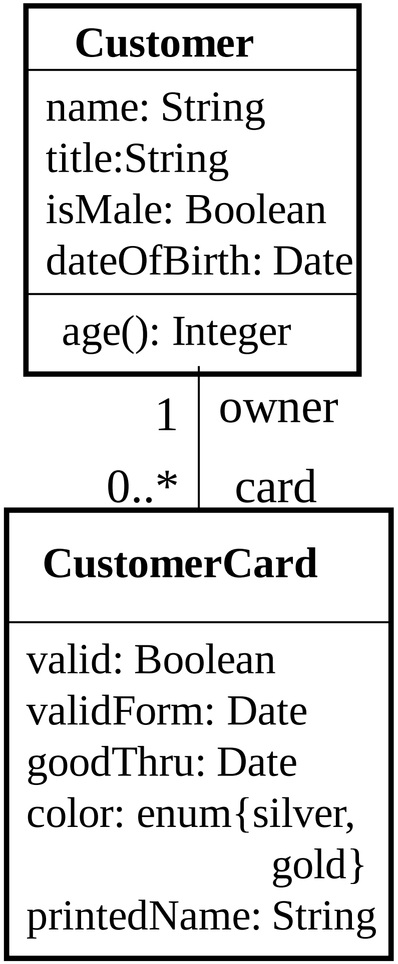
```
context Customer
  invariant agerestriction: age >= 18
context CustomerCard
  invariant correctDates:
  validFrom.isBefore(goodThru)

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.

Invariants using Navigation over Association Ends – Roles

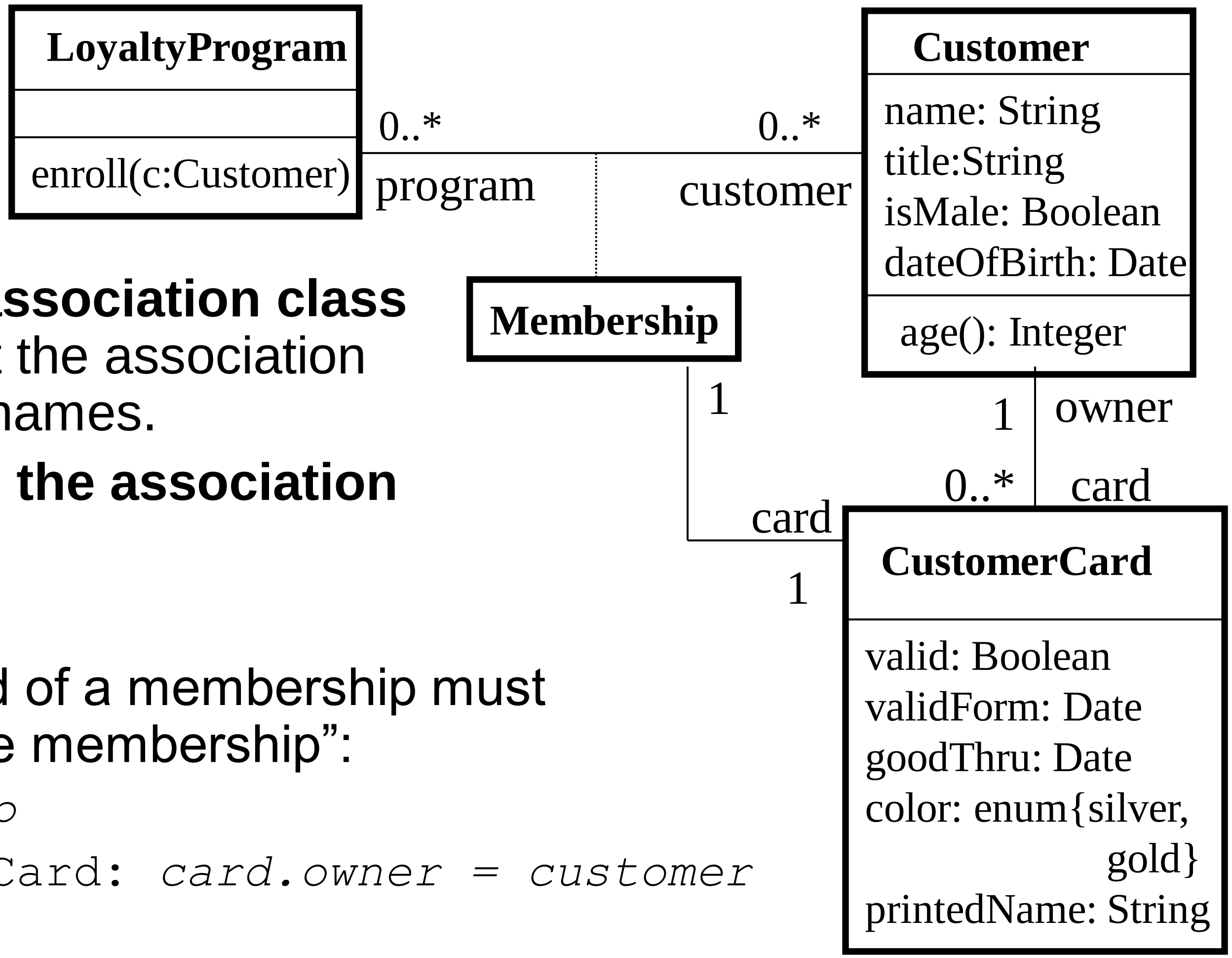

```
context CustomerCard
invariant printedName:
 printedName = owner.title.concat(‘ ’).
     concat(owner.name)
```



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
invariant correctCard: card.owner = customer
```


FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Métodos de Desenvolvimento de Software (MDS)
2011/2012

Miguel Goulão
mgoul@fct.unl.pt
http://ctp.di.fct.unl.pt/~mgoul/

Object Constraint Language (OCL)

Navigation and naming rules

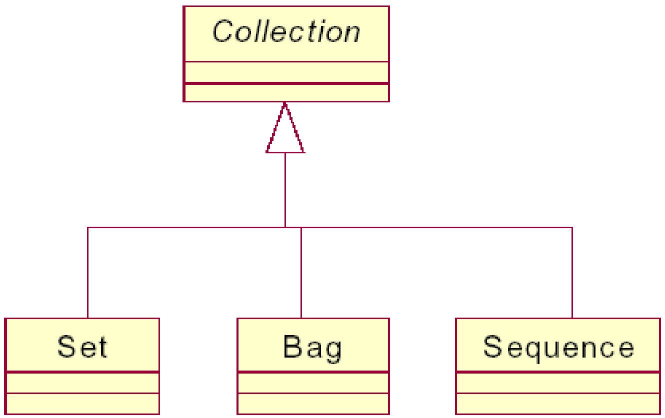
- Rule 0 - Class names start with an uppercase letter and role names with a lowercase letter
- Rule 1 - While navigating from a class to another, if the role of the destination class is defined then use it. Otherwise apply rule 2
- Rule 2 - While navigating from a class to another, if the role of the 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.

The OCL Collection types

- Collection is a predefined OCL type
- 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:T1; r:T2 = <v.e.> | <v.e.>)

- b.e. stands for: boolean expression
- v.e. stands for: value expression

Collection operations

- The number of elements in the collection self: size()
- The information of whether an object is part of a collection: includes()
- The information of whether an object isn't part of a collection: excludes()
- The number of times that object occurs in the collection self: count()
- The information of whether all objects of a given collection are part of a specific collection: includesAll()
- The information of whether none of the objects of a given collection are part of a specific collection: excludesAll()
- The information if a collection is empty: isEmpty()
- The information if a collection is not empty: notEmpty()

Collection operations

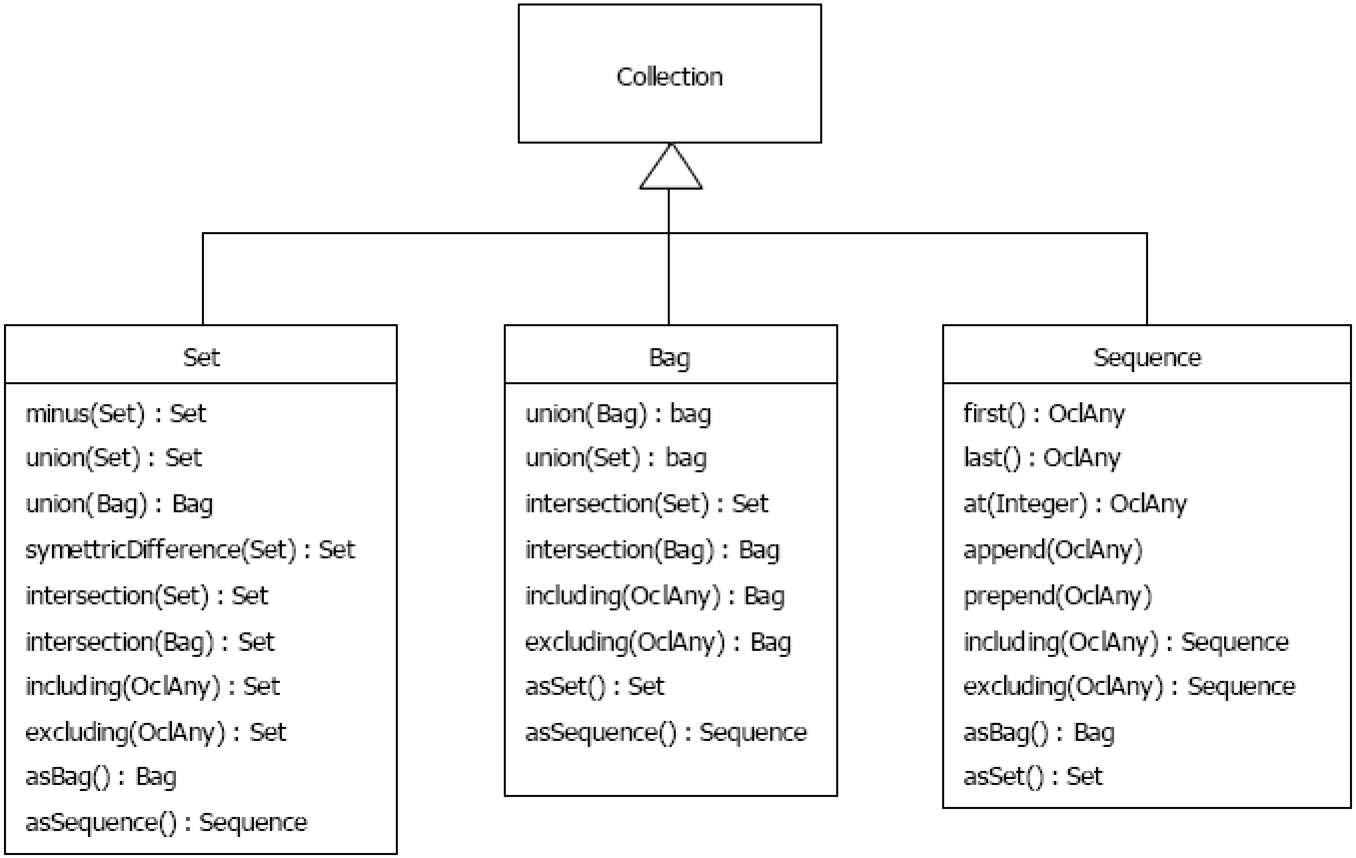
- Iterators over collections
- The selection of a sub-collection: select()
- When specifying a collection which is derived from some other collection, but which contains different objects from the original collection (i.e., it is not a sub-collection) use: collect()
- The information of whether an expression is true for all objects of a given collection: forAll()
- The addition of all elements of a collection: sum()
  Elements must be of a type supporting the + operation.

Collections operations summary

Collection

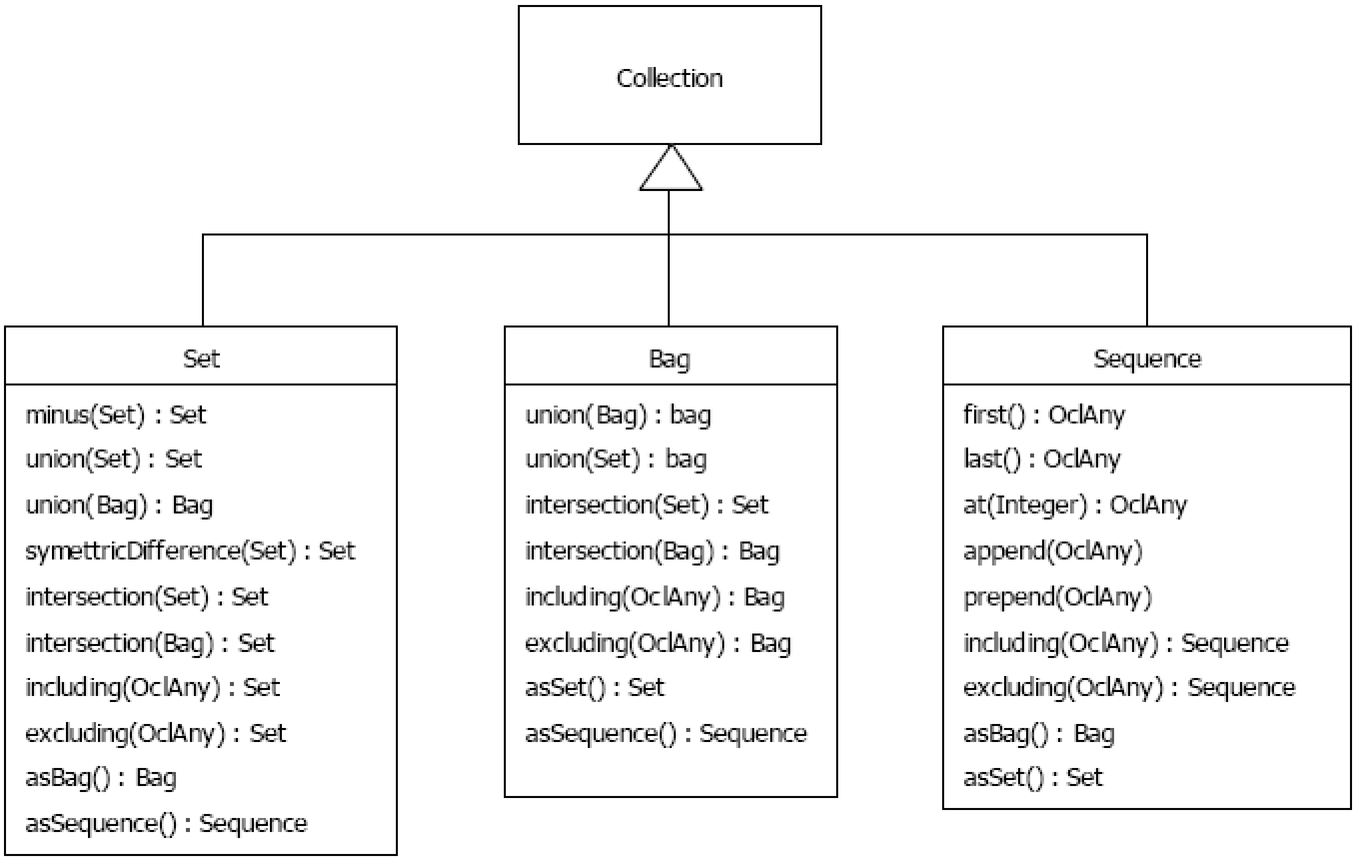
size() : Integer
includes(object : OclAny) : Boolean
count(object : OclAny) : Integer
includesAll(c2 : Collection(T)) : Boolean
isEmpty() : Boolean
notEmpty() : Boolean
sum() : Real
exists(expr : OclExpression) : Boolean
forAll(expr : OclExpression) : Boolean
iterate(expr : OclExpression) : OclType

Specialized Collection Operations



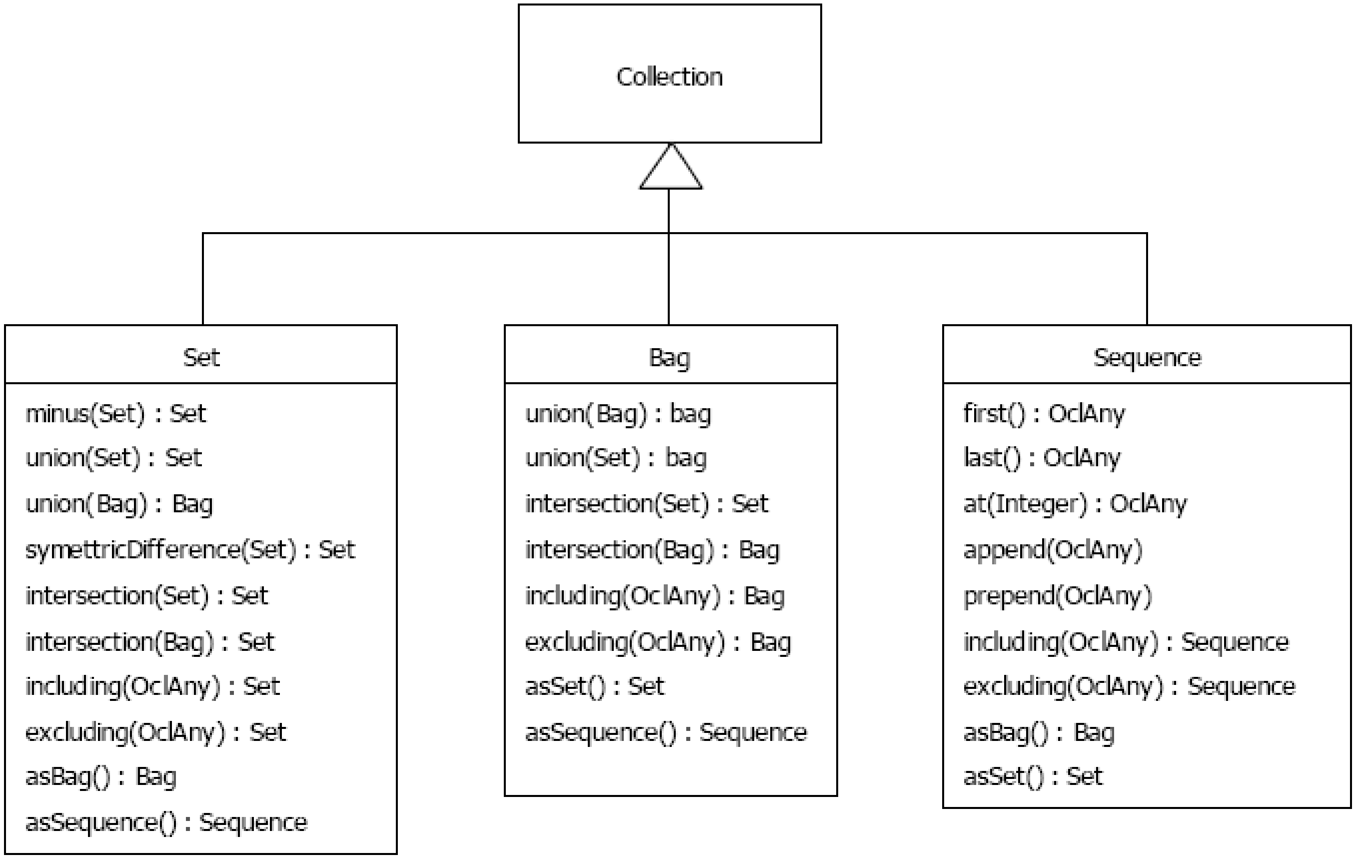
E.g. **Set**{4, 2, 3, 1}.minus(Set{2, 3}) = Set{4, 1}

Specialized Collection Operations



E.g., **Bag**{1, 2, 3, 5}.including(6) = Bag{1, 2, 3, 5, 6}

Specialized Collection Operations



E,g, **Sequence**{1, 2, 3, 4}.append(5) = Sequence{1, 2, 3, 4, 5}

Collection Operations: examples

specifies that the collection of all the employees older than 50 years is not empty.

context Company **inv**:
self.employee->select(age > 50)->notEmpty()

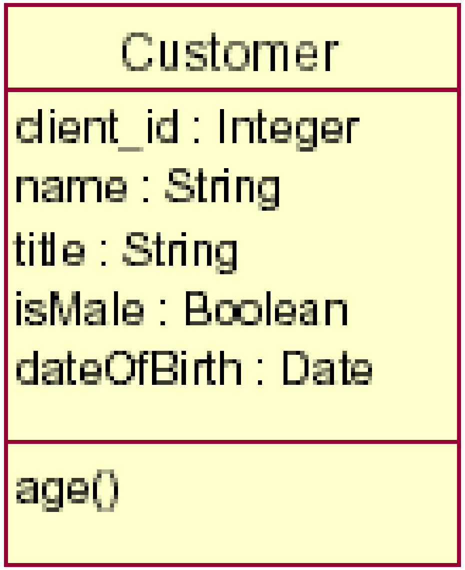
The ***self.employee*** is of type **Set(Person)**. The ***select*** takes each person from *self.employee* and evaluates *age > 50* for this person. If this results in *true*, then the person is in the result Set.

specifies that the collection of all the employees who are **not** married is empty.

context Company **inv**:
self.employee->reject(isMarried)->isEmpty()

Expressing uniqueness constraints

- Constraint: customer identifiers should always be unique



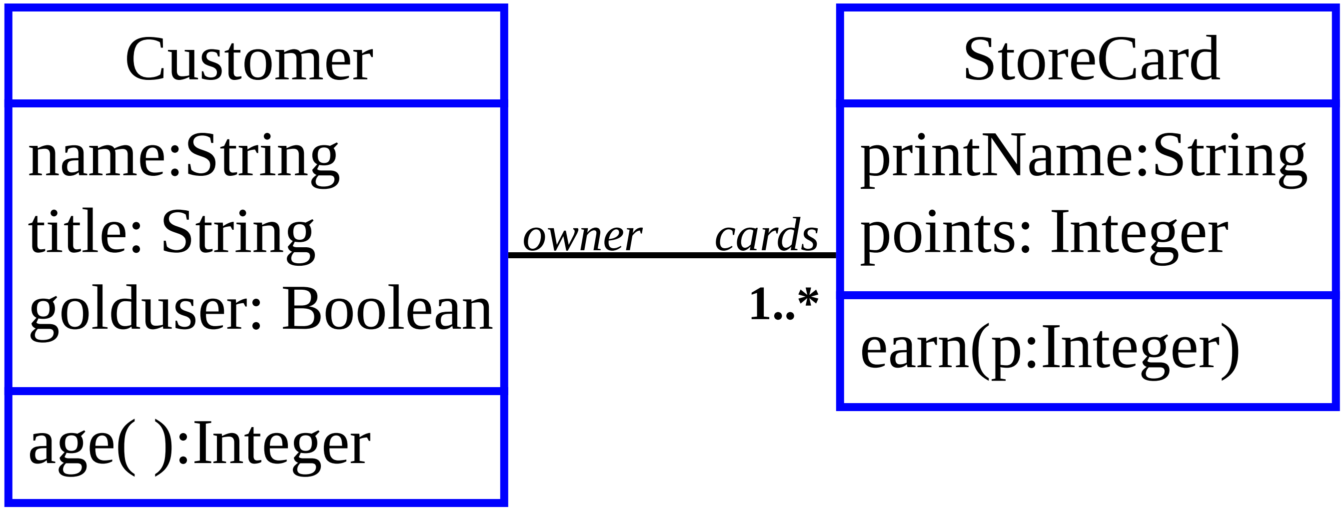
returns all instances of a given type

Context Customer **inv:**

```
Customer.allInstances->forAll(c1, c2: Customer | c1 <> c2 implies
   c1.client_id <> c2.client_id)
```

returns all instances of type Customer

Changing the context

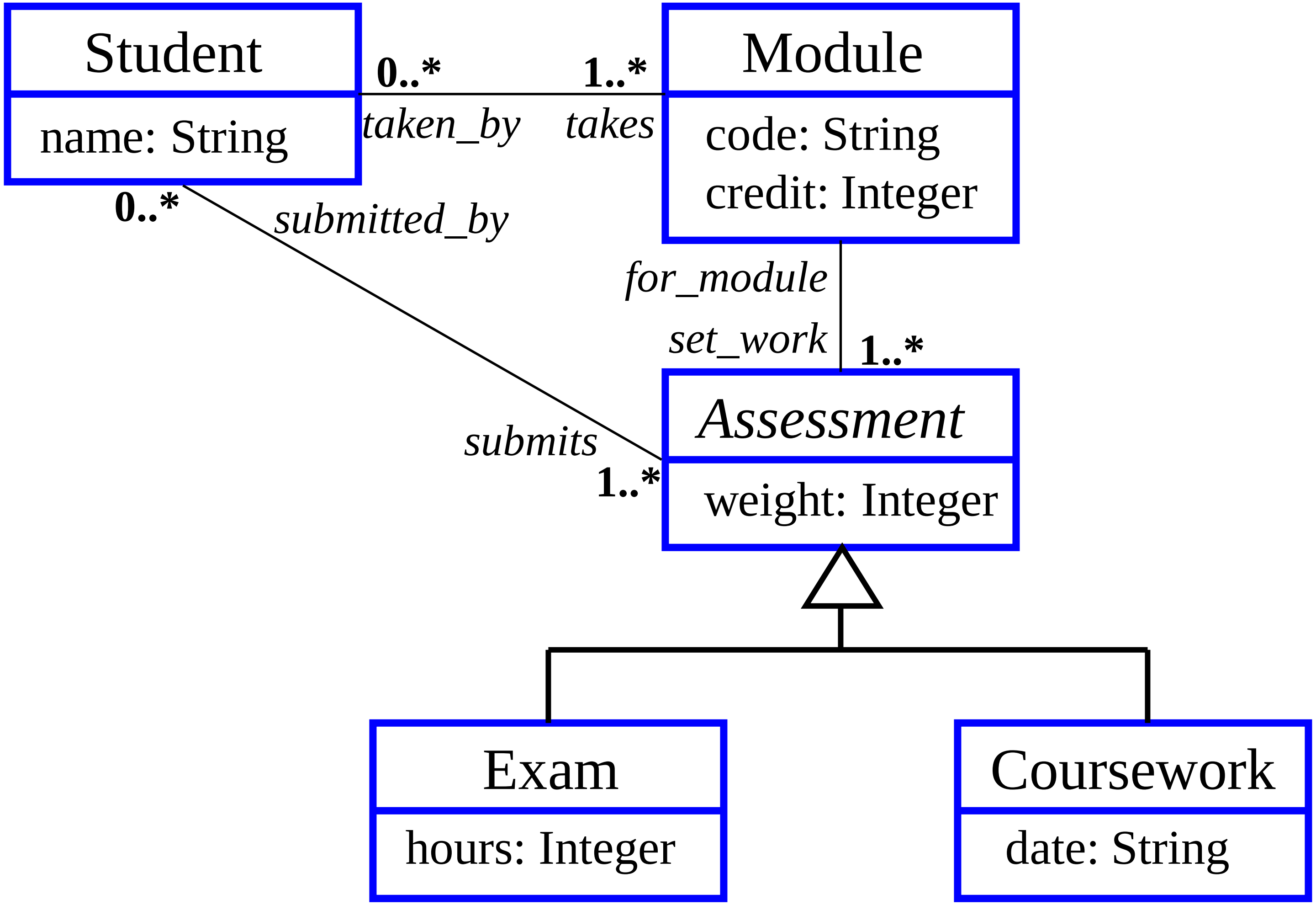


context *StoreCard*
inv: *printName = owner.title.concat(owner.name)*

Note switch of context!

context *Customer*
inv: *cards* $\rightarrow$ *forAll* (
printName = owner.title.concat(owner.name))

Example UML diagram



Constraints

- Modules can be taken iff they have more than seven students registered
- The assessments for a module must total 100%
- Students must register for 120 credits each year
- Students must take at least 90 credits of CS modules each year
- All modules must have at least one assessment worth over 50%
- 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

inv: taken_by→size > 7

Constraint (b)

- The assessments for a module must total 100%

context Module

inv:

set_work.weight→sum( ) = 100

Constraint (c)

- Students must register for 120 credits each year

context Student

inv: takes.credit→sum( ) = 120

Constraint (d)

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

context Student

inv:

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

inv: set_work→exists(weight > 50)

Constraint (f)

- Students can only have assessments for modules which they are taking

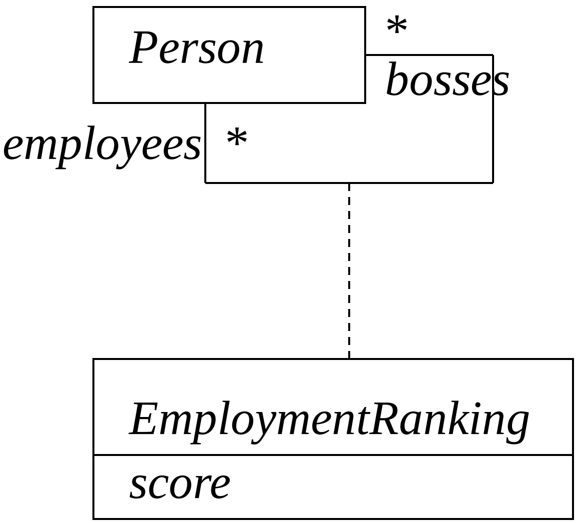
context Student

inv: takes→includesAll(submits.for_module)

Invariants using Navigation through Cyclic Association Classes

- Navigation through association classes that are cyclic requires use of roles to distinguish between association ends:

  object.associationClass[role]



- The accumulated score of an employee is positive:

context Person

inv:

employmentRanking[bosses].score->sum()>0

Every boss must give at least one 10 score:

context Person

inv:

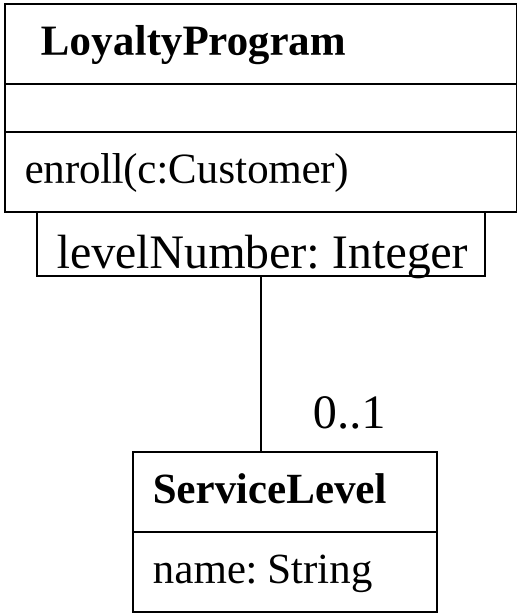
employmentRanking[employees]->exists(score = 10)

Due to unary association, we need to state the direction of the navigation

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Invariants using Navigation through Qualified Association

- To navigate qualified associations you need to index the qualified association using a qualifier
- object.navigation[qualifierValue, ...]
  - If there are multiple qualifiers their values are separated using commas
- Examples



context LoyaltyProgram

    inv: serviceLevel[1].name = ‘basic’

context LoyaltyProgram

    inv: serviceLevel->exists(name = ‘basic’)

Classes and Subclasses

- Consider the following constraint

  context LoyaltyProgram

  inv:

  partners.deliveredServices.transaction.points->sum() < 10,000

- If the constraint applies only to the Burning subclass, we can use the operation oclType of OCL:

  context LoyaltyProgram

  inv:

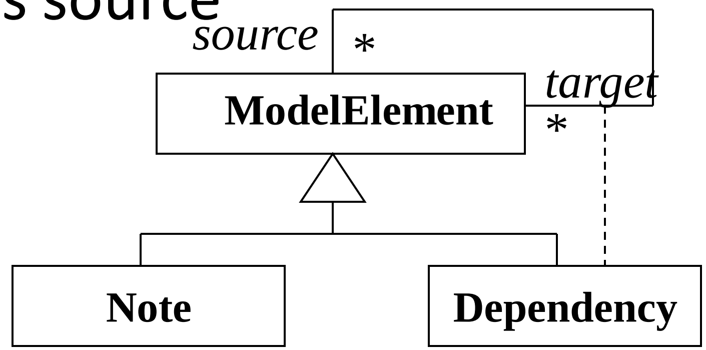
  partners.deliveredServices.transaction
  ->select(oclType = Burning).points->sum() < 10,000

Classes and Subclasses

- “The target of a dependency is not its source”

context Dependency

inv: self.source <> self



- Is ambiguous: Dependency is both

a ModelElement and an Association class.

context Dependency

inv: self.oclAsType(Dependency).source <> self

inv: self.oclAsType(ModelElement).source -> isEmpty()

References

- The Amsterdam Manifesto on OCL, In Object Modeling with the OCL (LNCS2263) p115-149
- The Object Constraint Language, Precise Modeling with UML, Jos Warmer and Anneke Kleppe, Addison-Wesley, 1999.
- 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 M°age
- The UML's Object Constraint Language: OCL Specifying Components, JAOO Tutorial – September 2000, Jos Warmer & Anneke Kleppe
- OCL website: http://www.omg.org/uml/