Object-Oriented Analysis and Design

Session 4: OCL

UML Diagrams are NOT Enough

- We need a language to help with the specification.
- We look for some “add-on” instead of a brand new language with full specification capability.
- Why not first order logic?
    - Not OO.

Object Constraint Language

- A formal language used to express constraints.
- OCL is :
  - A textual language to describe constraints.
  - The constraint language of the UML.
- Formal but easy to use
  - Unambiguous
  - No side effects
- OCL is used to specify constraints on OO systems.
- OCL expressions are always bound to a UML model.
- OCL is the only constraint language that is standardized.

OCL – Fills the Missing Gap

- Formal **specification language** → implementable.
- Supports object concepts.
- "Intuitive" syntax – reminds OO programming languages.
- But – OCL is not a programming language:
  - No control flow.
  - No side-effects.

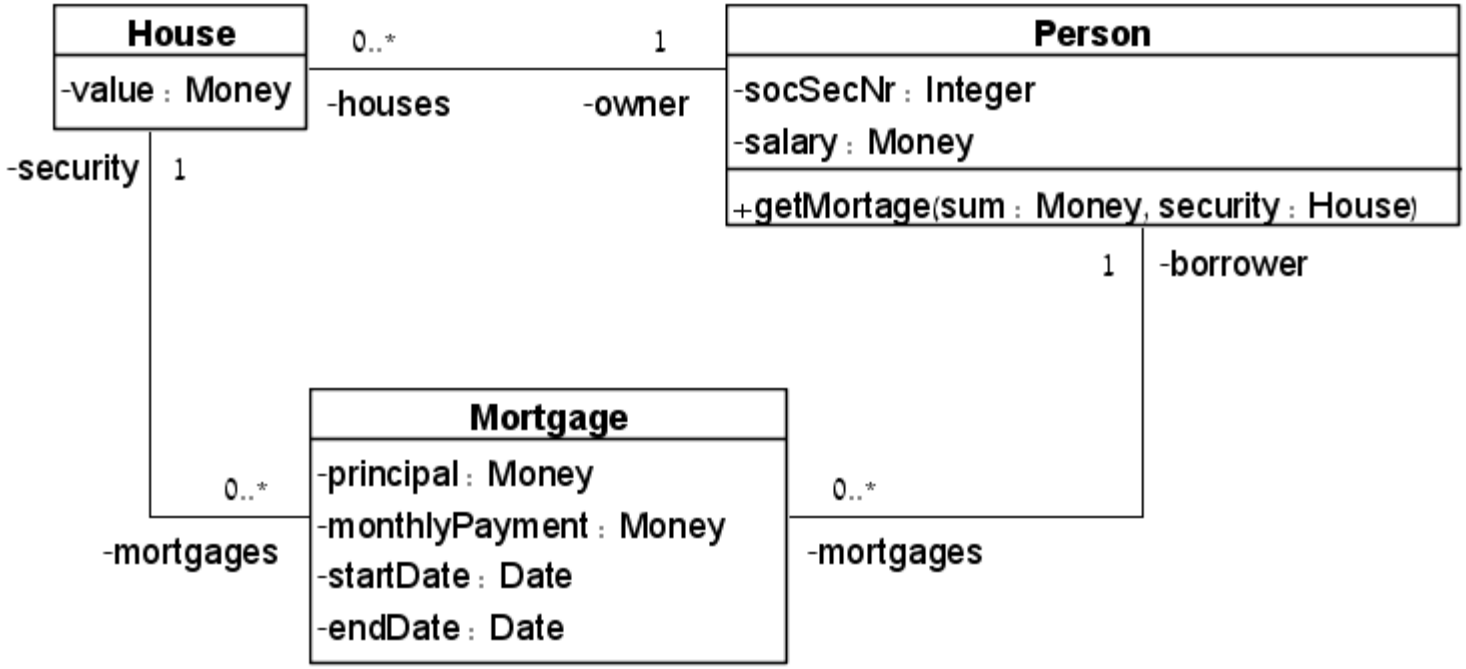
Advantages of Formal Constraints

- Better Documentation:
  - Constraints add information about the model elements and their relationships to the visual models used in UML.
  - It is way of documenting the model.
- More Precision
  - OCL constraints have formal semantics, hence, can be used to reduce the ambiguity in the UML models.
- Communication without misunderstanding
  - UML models are used to communicate between developers, using OCL constraints modelers can communicate unambiguously.

Where to use OCL?

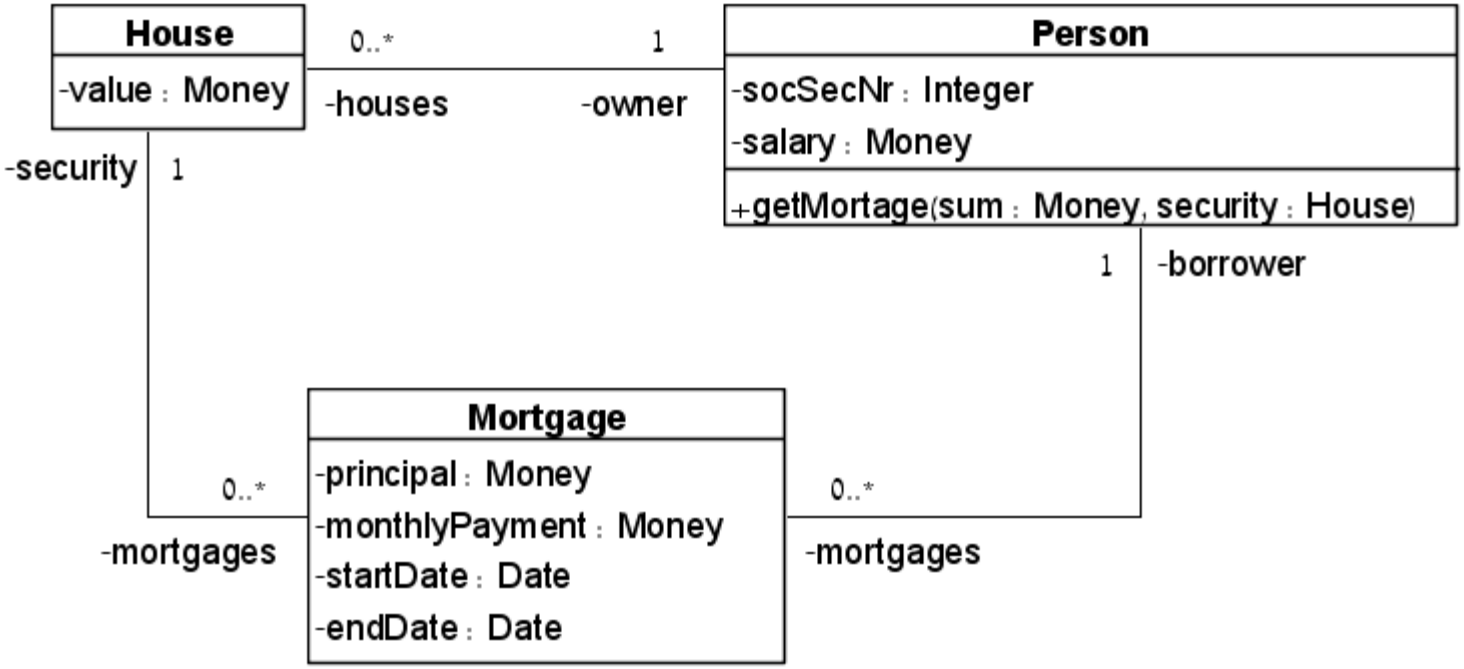
- Specify invariants for classes and types.
- Specify pre- and post-conditions for methods.
- As a navigation language.
- To specify constraints on operations.
- Test requirements and specifications.

Example – A Mortgage System



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

OCL Specification of the Constraints



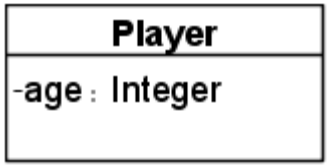
context *Mortgage*
invariant: *self.security.owner =self.borrower*
context *Mortgage*
invariant: *self.startDate < self.endDate*

context *Mortgage*
invariant: *security.owner = borrower*
context *Mortgage*
invariant: *startDate < endDate*

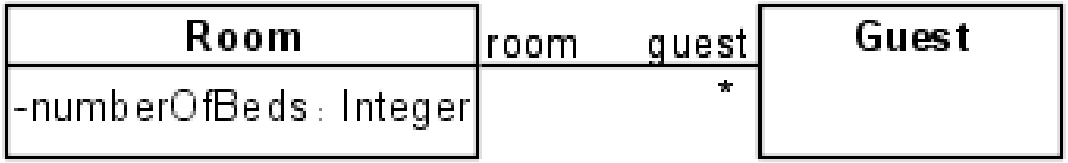
More Constraints Examples

- *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 (invariants),Contexts and self

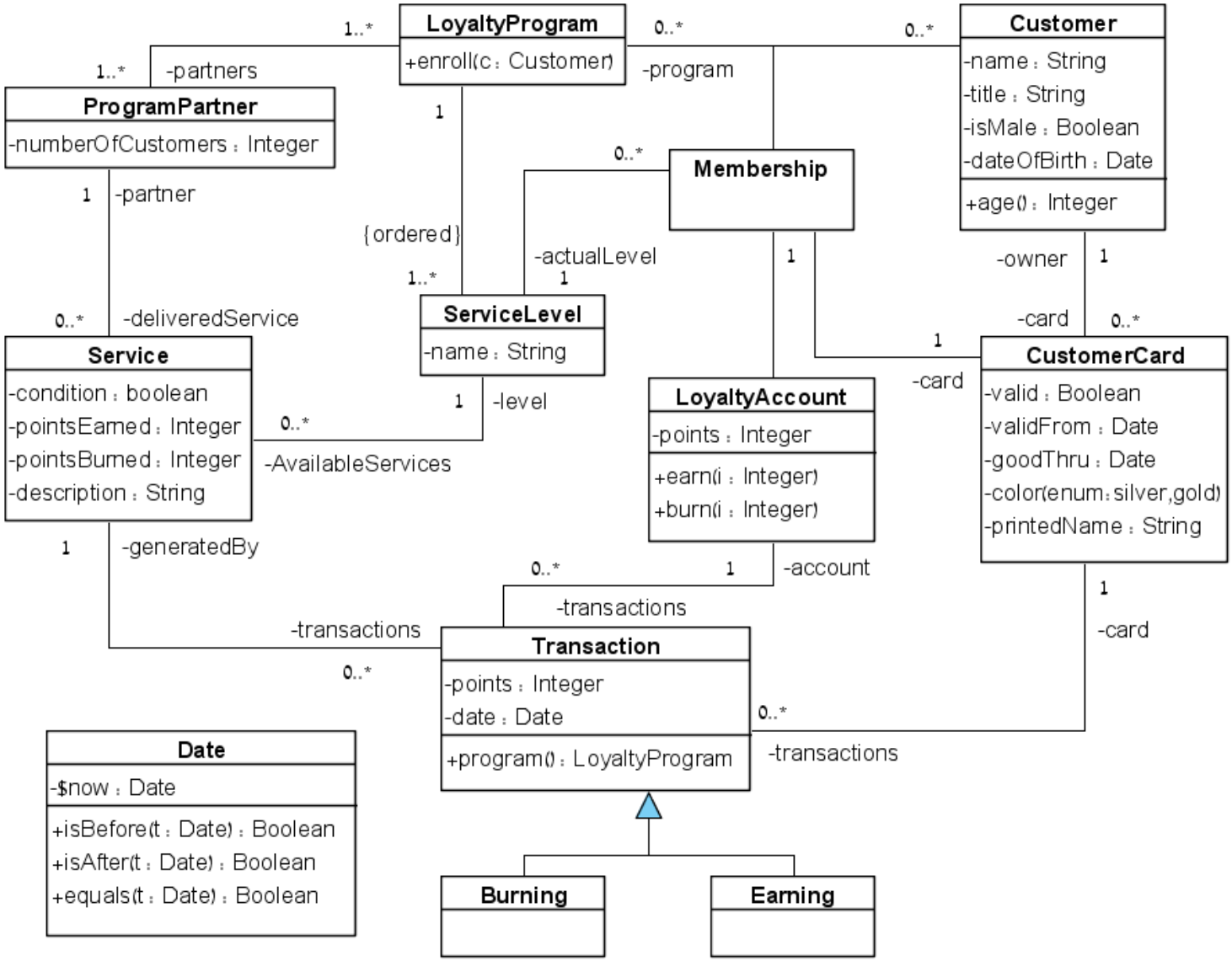
- A **constraint** (**invariant**) is a boolean OCL expression – evaluates to true/false.
- Every constraint is bound to a specific type (class, association class, interface) in the UML model – its context.
- The context objects may be denoted within the expression using the keyword '**self**'.
- The 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**.

Example of a Static UML Model - 1

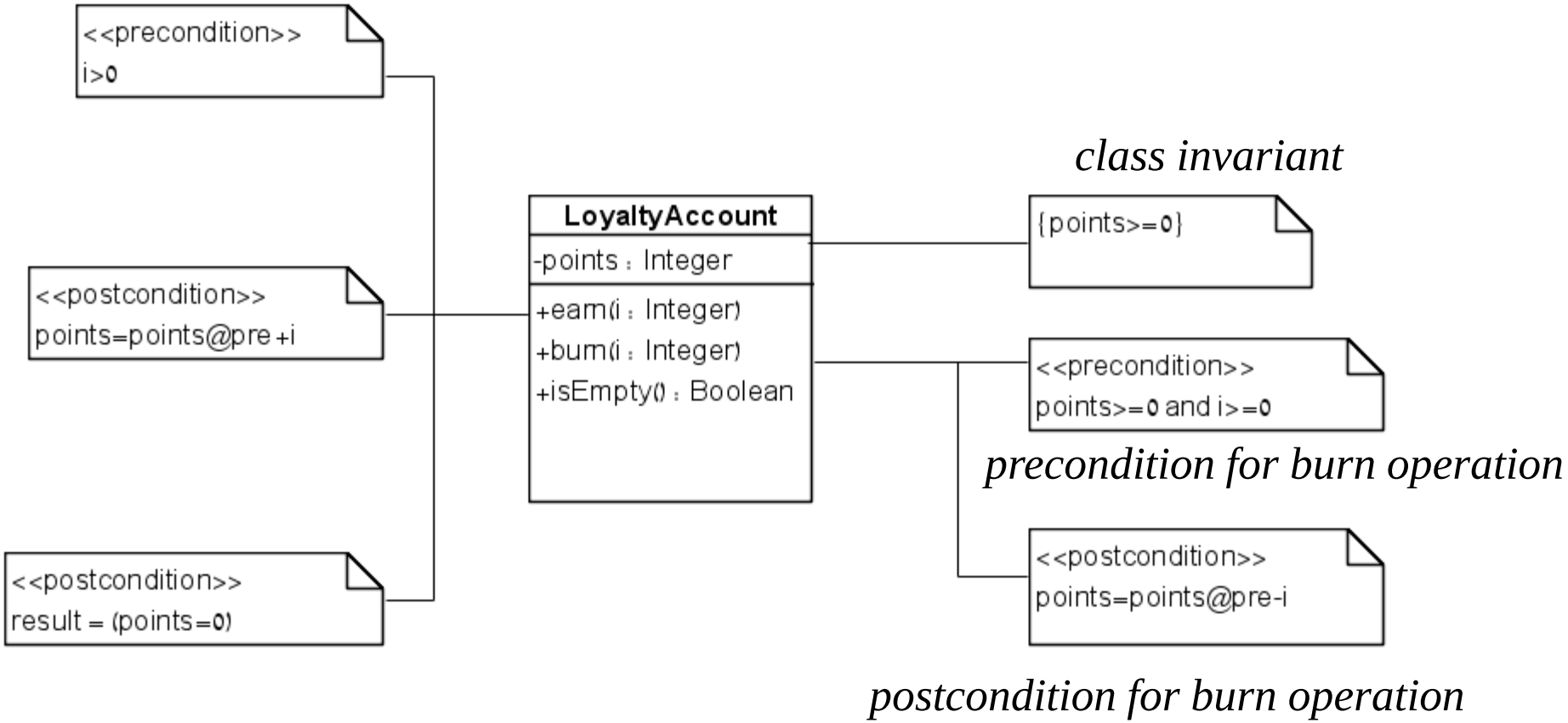
Problem story:

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****).*

Example of a Static UML Model - 2

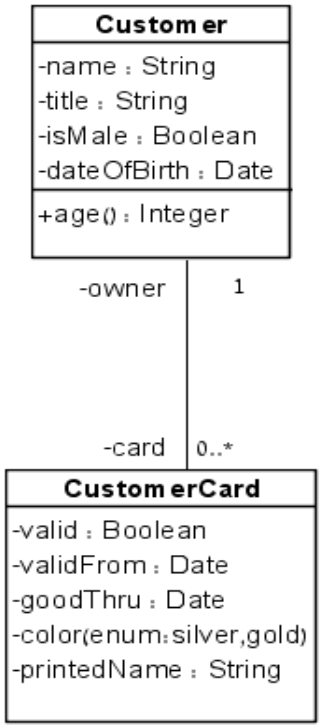


Using OCL in Class Diagrams



Invariants on Attributes

- Invariants on attributes:



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 CustomerCard class.

Invariants using Navigation over Association ends Roles - 1

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

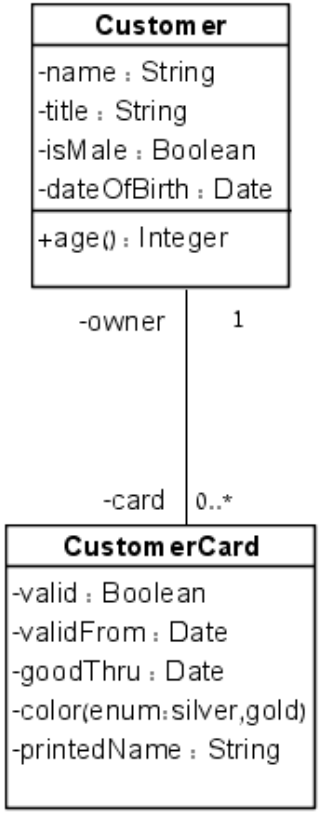
context *CustomerCard*
invariant: *owner.age()* >= 18

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.

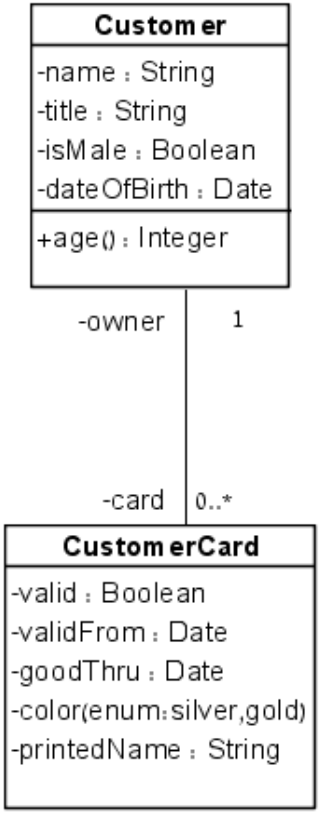
• **Preferred**: Always give role names.



Invariants using Navigation over Association ends Roles - 2

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*.



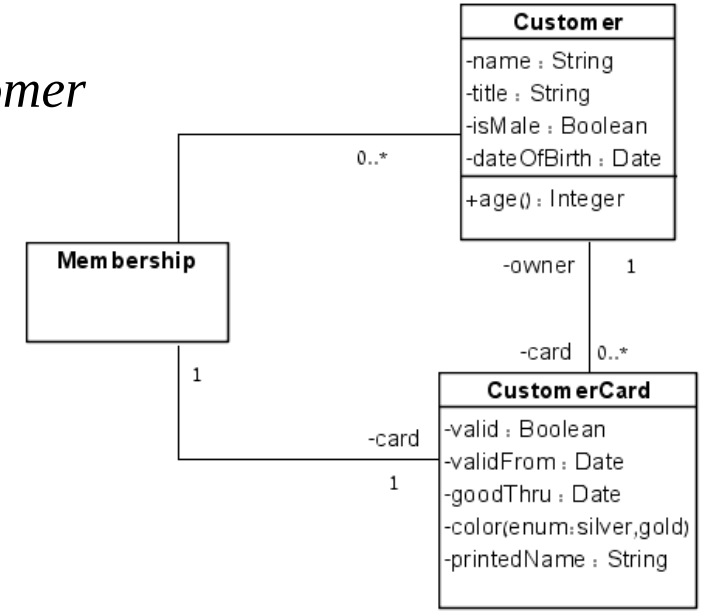
Invariants using Navigation from Association Classes

- 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*



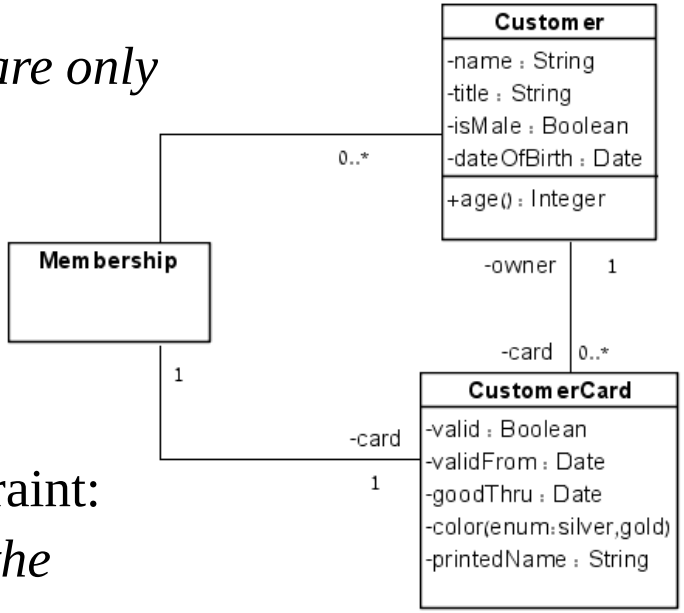
Invariants using Navigation through Association Classes

- 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**:***

  card->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 constrain is better!**

Invariants using Navigation through Associations with "Many" Multiplicity

Navigation over associations roles with multiplicity greater than 1 yields 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":

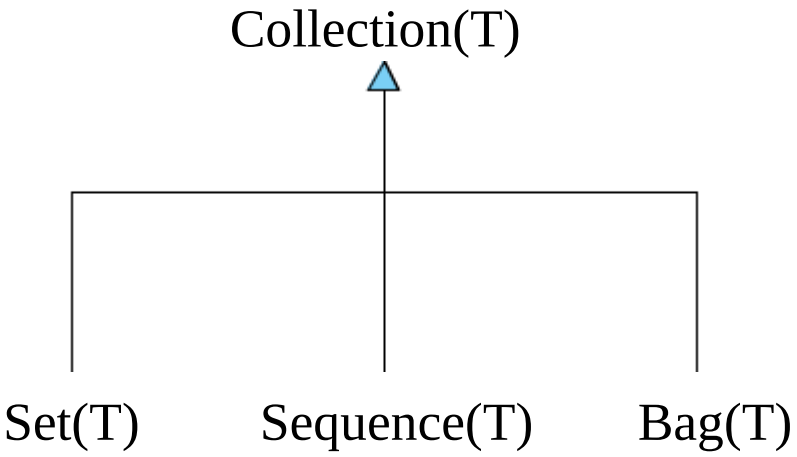
context *CustomerCard*
invariant *correctCard**:***
owner.Membership->includes(membership)

owner → a Customer instance.
owner.Membership → a set of Membership instances.
membership → a Membership instance.

includes is an operation of the OCL *Collection* type.

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 - 1

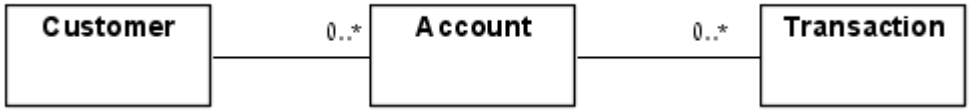
- <collection> → **size ( )**
  → **isEmpty( )**
  → **notEmpty ( )**
  → **sum ( )**
  → **count (** object **)**
  → **excludes (** object **)**
  → **includes (** object **)**
  → **includesAll (** collection **)**

Collection Operations - 2

- <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

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

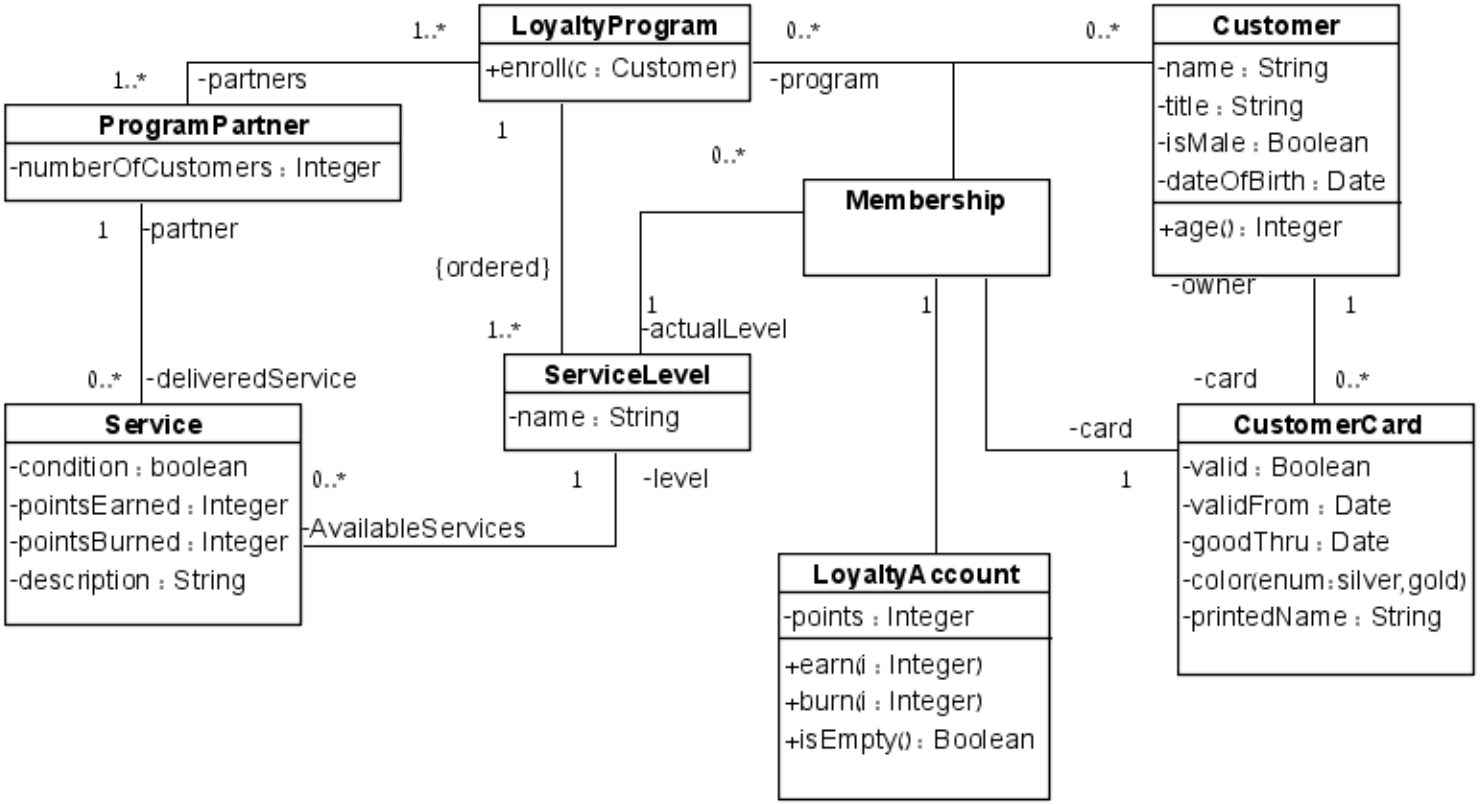
Navigating to Collections

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

***context** LoyaltyProgram*

invariant** minServices**:

partners.deliveredservice->size() >= 1



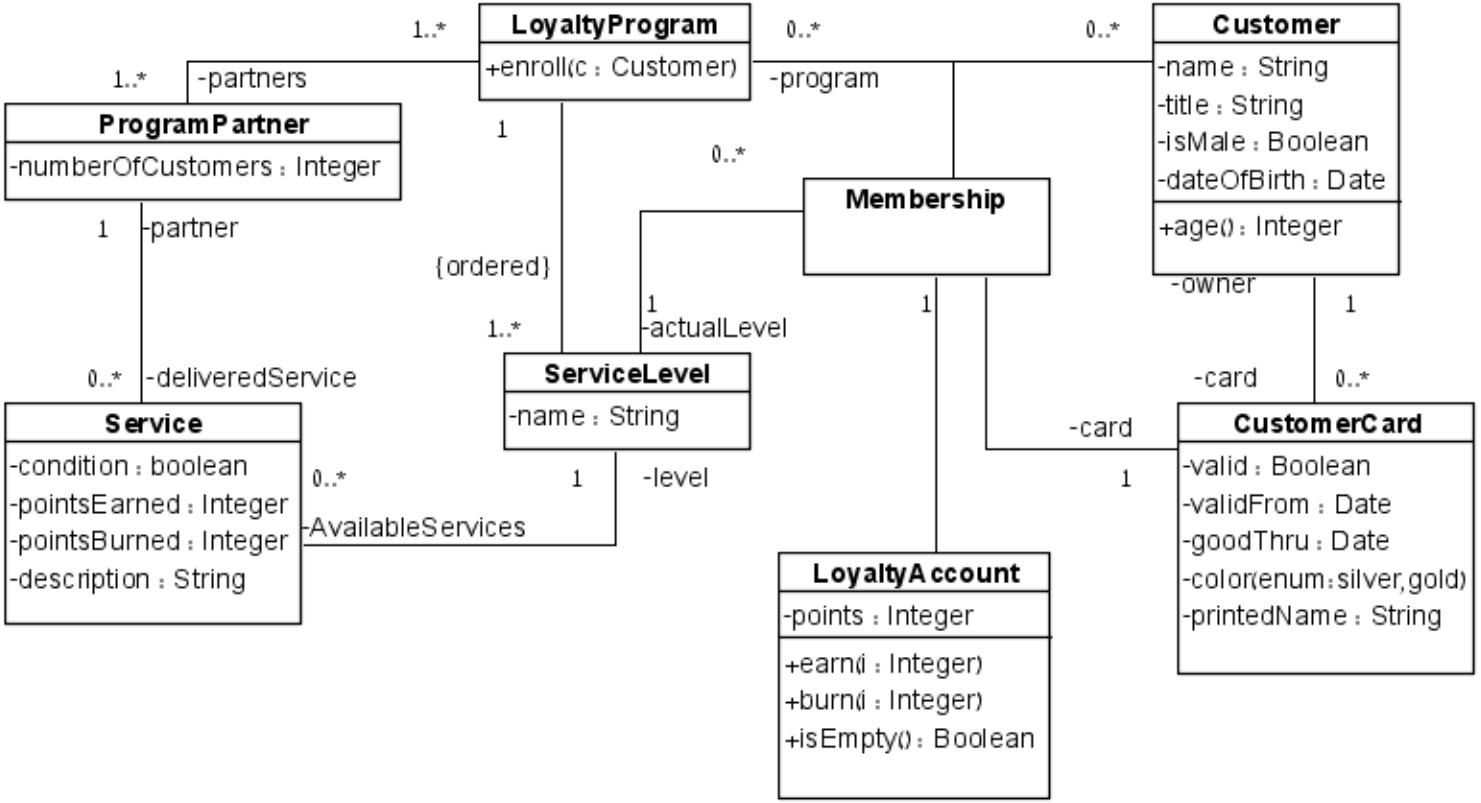
Navigating to Collections

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

context *Customer*

invariant *sizesAgree:*

Programs->size() = card->select(valid=true)->size()



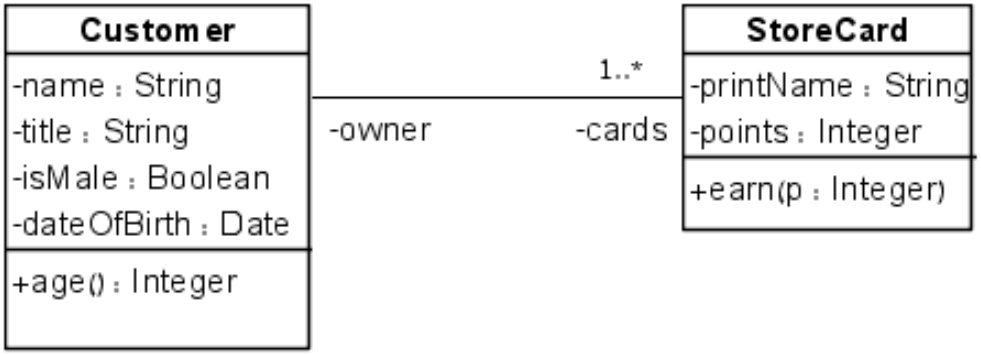
Navigating to Collections

"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.deliveredservice->
    forAll(pointsEarned = 0 and pointsBurned = 0)
        implies Membership.account->isEmpty()
```

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

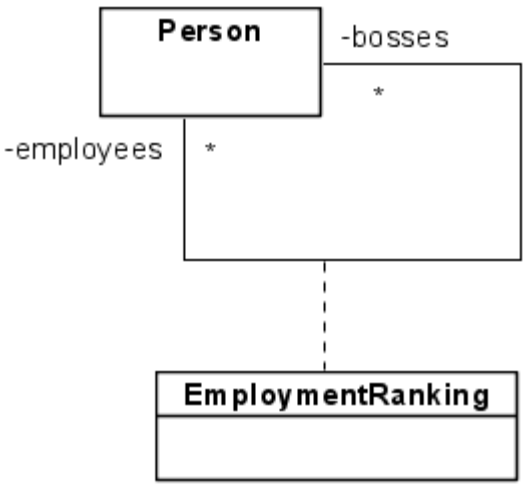
Changing the Context



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

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

Invariants using Navigation through Cyclic Association Classes



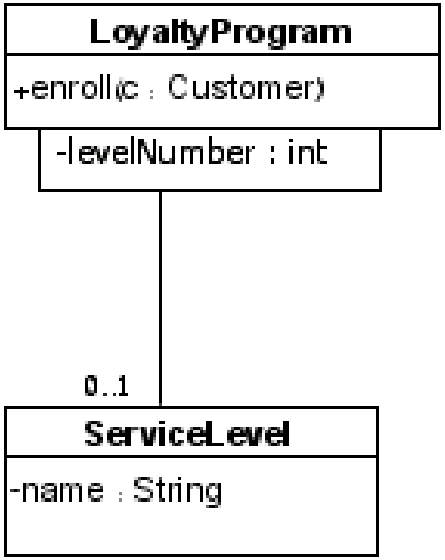
• Navigation through association classes that are cyclic requires use of roles to distinguish between association ends: object.associationClass[role]

• The accumulated score of an employee is positive:
context *Person*
invariant: *employeeRanking*[*bosses*].*score*->*sum*()>*0*

• Every boss must give at least one 10 score:
context *Person*
invariant:
employeeRanking[*employees*]->*exists*(*score = 10*)

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



- ***Example***

context *LoyaltyProgram*
serviceLevel[1].name = 'basic'

context *LoyaltyProgram*
serviceLevel->exists(name = 'basic')

Classes and Subclasses

• Consider the following constraint

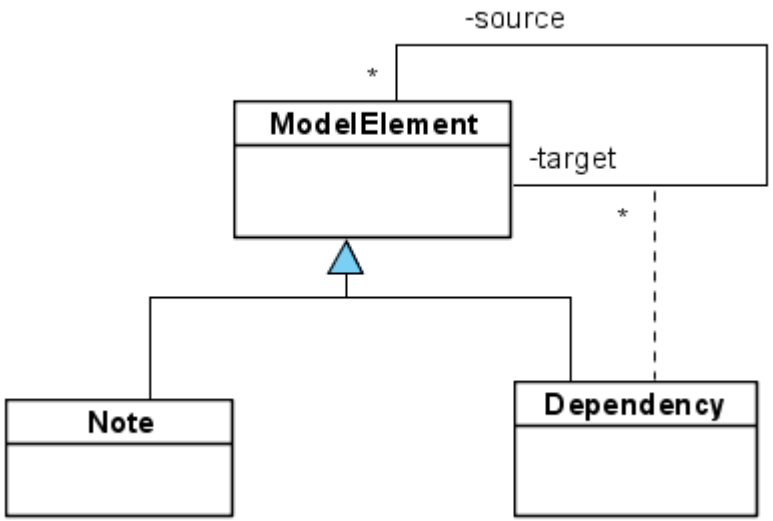
context *LoyaltyProgram*
invariant*:*
partners.deliveredService.transaction.points->sum() < 10,000

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

context *LoyaltyProgram*
invariant*:*
partners.deliveredService.transaction
*->select(**oclType** = Burning).points->sum() < 10,000*

Classes and Subclasses

context *Dependency*
invariant: *self.source* <> *self*



Is ambiguous:
Dependency is both
a *ModelElement* and an Association class.

context *Dependency*
invariant: *self.oclAsType*(*Dependency*).source <> *self*
Or
context *Dependency*
invariant: *self.oclAsType*(*ModelElement*).source <> *self*

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.

Combining UML and OCL

- Without OCL expressions, the model 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 is associated with an UML MODEL

- Basic types: String, Boolean, Integer, Real.
- Classes from the UML model and their attributes.
- Enumeration types from the UML model.
- Associations from the UML model.

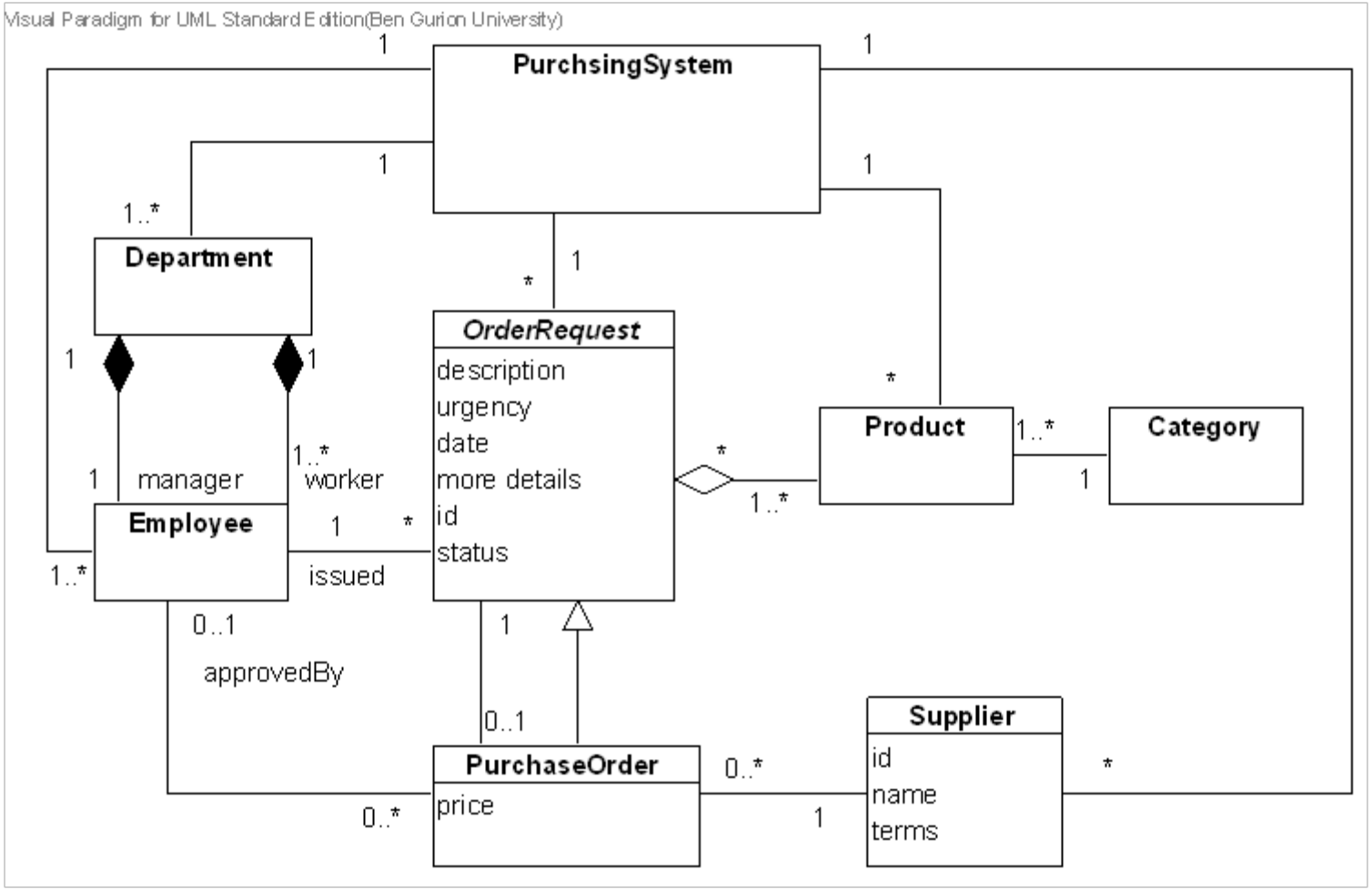
What is OCL anyway?

- A textual specification language
- A expression language
- Is side-effect-free language
- Standard query language
- Is a strongly typed language
  - so expressions can be precise
- Is a formal language
- Is part of UML
- Is used to define UML
- Is Not a programming language
- The OCL is *declarative* rather than imperative
- Mathematical foundation, but no mathematical symbols
  - based on set theory and predicate logic
  - has a formal mathematical semantics

Conclusions and Tips

- OCL invariants allow you to :
  - Model more precisely.
  - Stay implementation independent.
- OCL usage tips :
  - Keep constraints simple.
  - Always combine natural language with OCL.
  - Use a tool to check your OCL.

לפניכם תיאור חלקי של מערכת רכש. במערכת זו עובדים יכולים להזמין מוצרים שונים המסווגים לפי קטגוריות שונות. הבקשה להזמנה כוללת את תיאורה והדחיפות שלה, תאריך ועוד פרטים נדרשים (מלל חופשי). לאחר ההזנה, הבקשה עוברת לאישור מנהל מחלקה. במידה שהמנהל מאשר את הבקשה, היא עוברת למחלקת רכש. בעת הטיפול בבקשות, לכל בקשה בוחרת מחלקת הרכש את הספק הרצוי מרשימת ספקים המתקבלת ממערכת חיצונית (אבל מנוהלת המערכת), מחליטה על המחיר ופותחת בקשת רכישה חדשה. לעיתים, יש גם עריכה של ההזמנה המקורית (לדוגמא שינוי מוצר).


Visual Paradigm for UML Standard Edition(Ben Gurion University)
PurchsingSystem
Department
OrderRequest
description
urgency
date
more details
id
status
Product
Category
Employee
manager
worker
issued
approvedBy
PurchaseOrder
price
Supplier
id
name
terms
1
1..*
*
0..1
0..*

- עובד יכול להזמין עד 5 פריטים במהלך שנה.
- סך כל הפריטים שעובד יכול להזמין צריך להיות קטן מ- 100.
- לכל בקשה יש רק הזמנה אחת.
- לא יכולה להיות הזמנה לבקשה שאינה מאושרת ע"י מנהל המחלקה של העובד.
- הבקשות של עובדים מאותה מחלקה צריכים להיות לכלול עד 10 פריטים מכל קטגוריה.

עובד יכול להזמין עד 5 פריטים במהלך שנה.

Context: Employee

Inv: orderRequest->select (oclIsTypeOf(orderRequest)).date.year()->forall(y|self.orderRequest->select (oclIsTypeOf(orderRequest))->select(date.year()=y)<6))

סך כל הפריטים שעובד יכול להזמין צריך להיות קטן מ- 100.

Context: Employee

Inv: orderRequest.product->size<100.

לכל בקשה יש רק הזמנה אחת.

אין צורך באילוץ.

לא יכולה להיות הזמנה לבקשה שאינה מאושרת ע"י מנהל המחלקה של העובד.

Context: PurchaseOrder

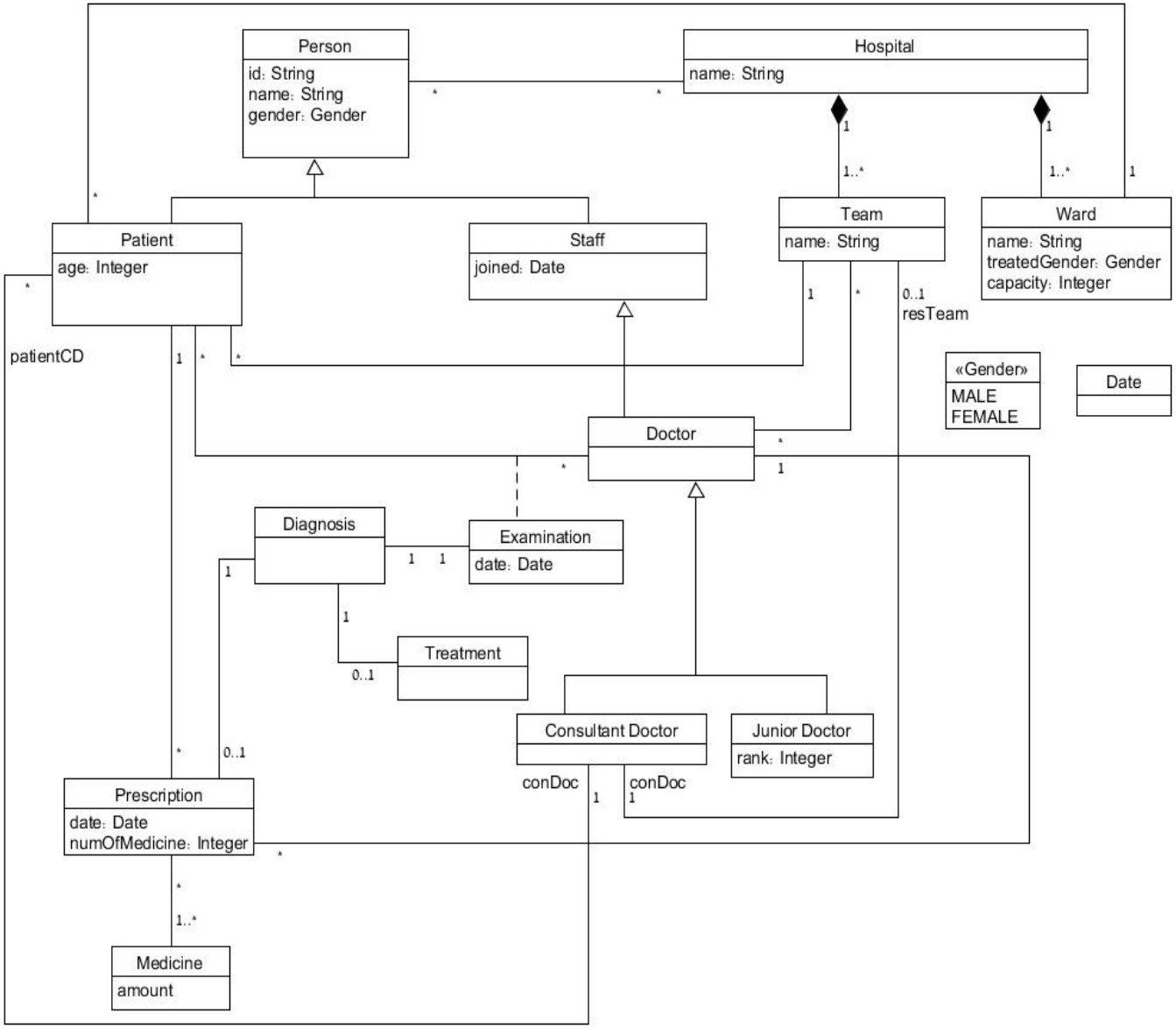
Inv: orderRequest.employee.department.manager=approvedBy

הבקשות של עובדים מאותה מחלקה צריכים להיות לכלול עד 10 פריטים מכל קטגוריה.

Context department inv:

worker.order.product.category->forAll (c | self.worker.order.product->
select(category = c)-> size() <= 10)

The Hospital Case



Constraint # 1

- בצוות יש רופא זוטר אחד לפחות עם דרגה 1

```
self.doctor->select(oclAsType(JuniorDoctor).rank=1)->size()>0
```

```
public boolean juniorDoctor(Team team){
    boolean result=false;
    for(Doctor doc:  team.doctor){
        if((doc instanceof JuniorDoctor) && ((JuniorDoctor)
doc).rank==1){
            result=true;
        }
    }
    return result;
}
```

Constraint # 2

- אם במחלקת אשפוז יש לפחות שלושה חולים, אזי מספר הרופאים בצוותים שהחולים משויכים אליהם גדול או שווה לשלוש.

```
context Ward
self.patient->size()>=3 implies self.patient.team.doctor->asSet()->size()>=3
```

```
public boolean doctor(Ward ward) {
    boolean result=true;
    Collection<Doctor> allDocs = new HashSet<Doctor>();
    if (ward.patient.size() >= 3) {
        for (Patient p : ward.patient) {
            allDocs.addAll(p.team.doctor);
        }
        result=allDocs.size() >= 3;
    }
    return result;
}
```

Constraint # 3

- הרופא שכתב את המרשם הוא אותו רופא שעשה את הבדיקה

context Examination
self.diagnosis.prescription->size()>0 implies
self.diagnosis.prescription.doctor=doctor

```
public boolean doctor(Examination examination){
    boolean result=true;
    if(examination.diagnosis.prescription!=null)
        result=
    examination.diagnosis.prescription.doctor ==
                examination.doctor;
    return result;
}
```

Constraint # 4

- במחלקת אשפוז ולחולים יש אותו מגדר

Context Ward

self.patient->forAll(gender=self.treatedGender)

```
public boolean gender(Ward ward){
    boolean sameGender=true;
    for(Patient patient: ward.patient){
      sameGender=patient.gender==ward.treatedGender
                && sameGender;
    }
    return sameGender;
}
```

Constraint # 5

- מספר התרופות שמקבלים ממרשם הוא כמספר התרופות שמצוין במרשם

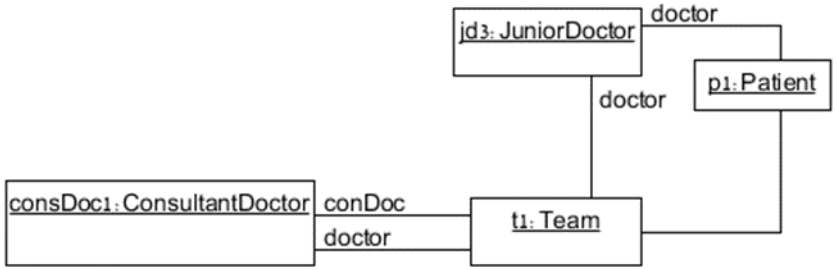
Context Prescription

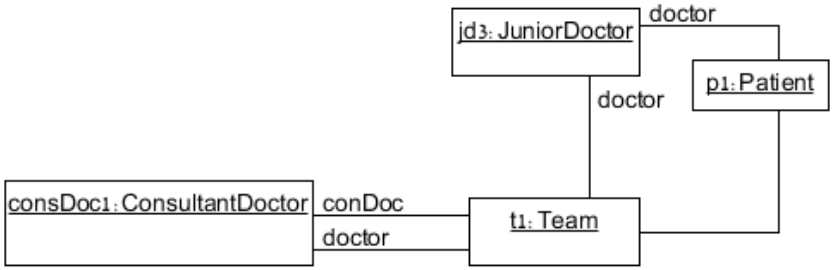
self.numberOfMedicine=self.medicine->size()

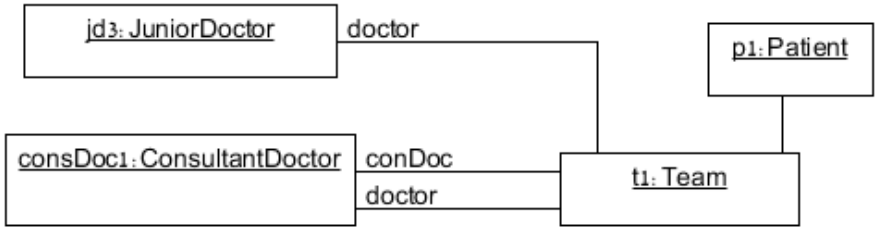
```
public boolean medicine(Prescription pres){
    return pres.numberOfMedicine ==pres.medicine.size();
}
```

Context Patient

self.team.doctor->includesAll(self.doctor)







```
public boolean capacity(Ward ward){
   return ward.capacity>=ward.patient.size();
}
```