UNIVERSITÉ BISHOP'S UNIVERSITY

CS 410/CS 560 - Software Engineerin

OCL - The Object Constraint Language in UML

Dr. Mohammed Ayoub Alaoui Mhamdi
Bishop's University
Sherbrooke, Qc, Canada
malaoui@ubishops.ca

Outline of the Lecture

- OCL
- Simple predicates
- Preconditions
- Postconditions
- Contracts
- Sets, Bags, and Sequences

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.

UML Diagrams are NOT Enough!

- We need a language to help with the spec.
- We look for some "add-on" instead of a brand 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.

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

OCL Basic Concepts

- OCL expressions
  - Return **True** or **False**
  - Are evaluated in a specified context, either a class or an operation
  - All constraints apply to all instances

OCL Simple Predicates

Example:

```
context Tournament inv:
  self.getMaxNumPlayers() > 0
```

In English:

"The maximum number of players in any tournament should be a postive number."

Notes:

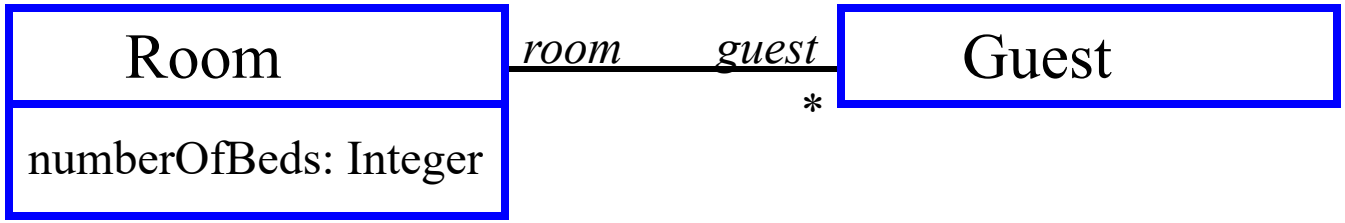
- "self" denotes all instances of "Tournament"
- OCL uses the same dot notation as Java.

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:
guests -> size <= numberOfBeds
```

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)

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)

b) The assessments for a module must total 100%

context *Module*

invariant:

$$set\_work.weight \rightarrow sum(\,) = 100$$

Constraint (c)

c) Students must register for 120 credits each year

context *Student*

invariant: $takes.credit \rightarrow sum(\ ) = 120$

Constraint (d)

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)

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

context *Module*

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

Constraint (f)

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

context *Student*

invariant:

takes→includesAll(submits.for_module)

OCL Preconditions

Example:

```
context Tournament::acceptPlayer(p) pre:
  not self.isPlayerAccepted(p)
```

In English:

"The acceptPlayer(p) operation can only be invoked if player p has not yet been accepted in the tournament."

Notes:

- The context of a precondtion is an operation
- isPlayerAccepted(p) is an operation defined by the class Tournament

OCL Postconditions

Example:

```
context Tournament::acceptPlayer(p)
  post:
  self.getNumPlayers() =
      self@pre.getNumPlayers() + 1
```

In English:

"The number of accepted player in a tournament increases by one after the completion of acceptPlayer()"

Notes:

- self@pre denotes the state of the tournament before the invocation of the operation.
- Self denotes the state of the tournament after the completion of the operation.

OCL Contract for acceptPlayer() in Tournament

context Tournament::acceptPlayer(p) **pre**:
    not isPlayerAccepted(p)

context Tournament::acceptPlayer(p) **pre**:
    getNumPlayers() < getMaxNumPlayers()

context Tournament::acceptPlayer(p) **post**:
    isPlayerAccepted(p)

context Tournament::acceptPlayer(p) **post**:
    getNumPlayers() = @pre.getNumPlayers() + 1

OCL Contract for removePlayer() in Tournament

context Tournament::removePlayer(p) **pre**:

isPlayerAccepted(p)

context Tournament::removePlayer(p) **post**:

not isPlayerAccepted(p)

context Tournament::removePlayer(p) **post**:

getNumPlayers() = @pre.getNumPlayers() - 1

Class (Contract as a set of JavaDoc comments)

```
public class Tournament {
/** The maximum number of players
 * is positive at all times.
 * @invariant maxNumPlayers > 0
 */
private int maxNumPlayers;

/** The players List contains
 *  references to Players who are
 *  are registered with the
 *  Tournament. */
private List players;

/** Returns the current number of
 * players in the tournament. */
public int getNumPlayers() {...}

/** Returns the maximum number of
 * players in the tournament. */
public int getMaxNumPlayers() {...}
```

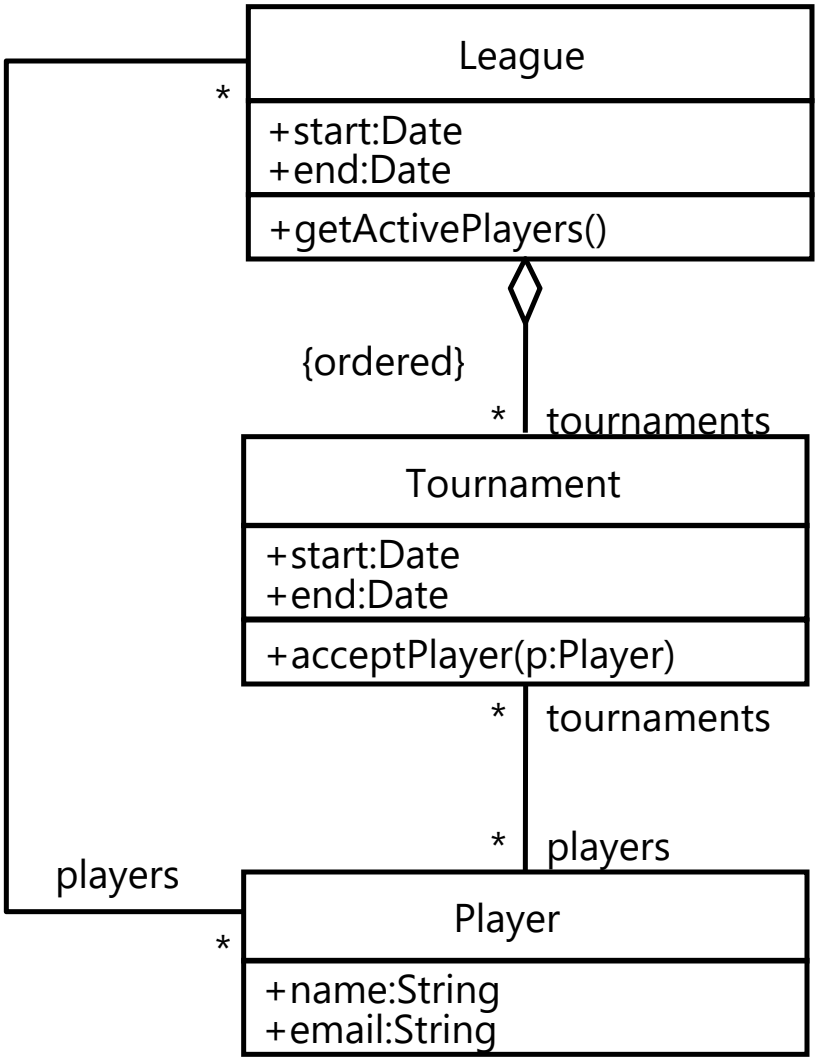
```
/** The acceptPlayer() operation
 * assumes that the specified
 * player has not been accepted
 * in the Tournament yet.
 * @pre !isPlayerAccepted(p)
 * @pre getNumPlayers()<maxNumPlayers
 * @post isPlayerAccepted(p)
 * @post getNumPlayers() =
 *     @pre.getNumPlayers() + 1
 */
public void acceptPlayer (Player p) {...}

/** The removePlayer() operation
 * assumes that the specified player
 * is currently in the Tournament.
 * @pre isPlayerAccepted(p)
 * @post !isPlayerAccepted(p)
 * @post getNumPlayers() =
 *     @pre.getNumPlayers() - 1
 */
public void removePlayer(Player p) {...}

}
```

Constraints can involve more than one class

How do we specify constraints on on a group of classes?

Starting from a specific class in the UML class diagram, we navigate the associations in the class diagram to refer to the other classes and their properties (attributes and Operations).

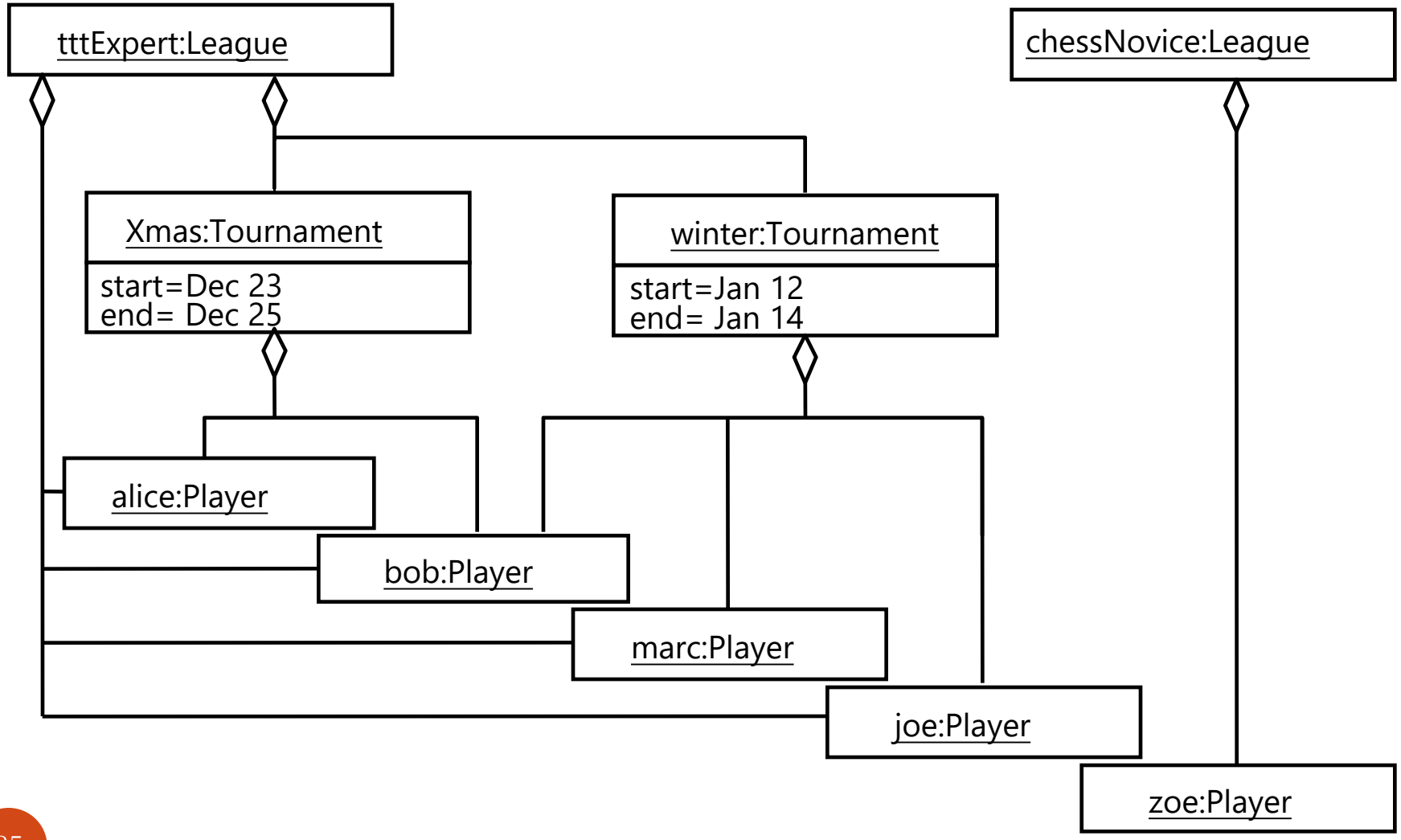
Example from ARENA: League, Tournament and Player



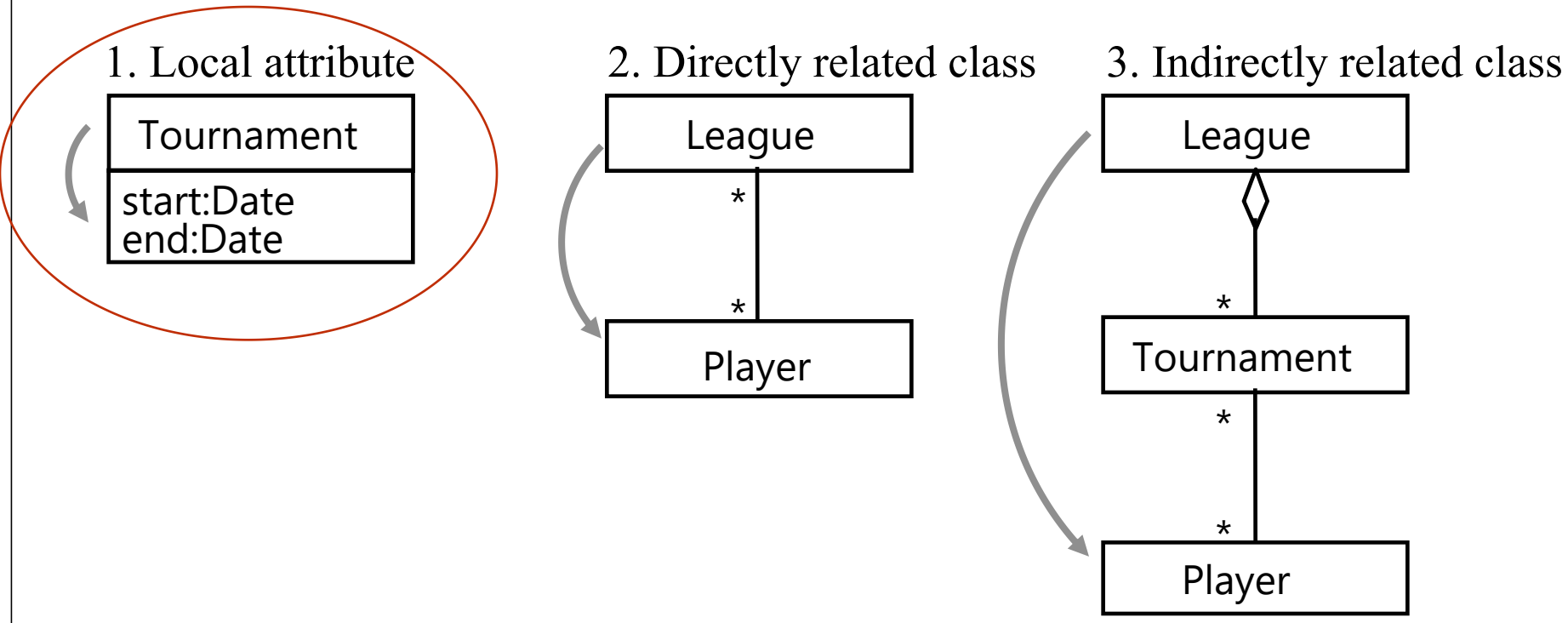
Constraints:

1. A Tournament's planned duration must be under one week.
2. Players can be accepted in a Tournament only if they are already registered with the corresponding League.
3. The number of active Players in a League are those that have taken part in at least one Tournament of the League.

Instance Diagram: 2 Leagues, 5 Players, 2 Tournaments



3 Types of Navigation through a Class Diagram



Any constraint for an arbitrary UML class diagram can be specified using only a combination of these 3 navigation types!

Specifying the Model Constraints in OCL

Local attribute navigation

context **Tournament** inv:

end - start <= 7

Directly related class navigation

context

Tournament::acceptPlayer(p)

pre:

league.players->includes(p)

OCL Sets, Bags and Sequences

- Sets, Bags and Sequences are predefined in OCL and subtypes of Collection. OCL offers a large number of predefined operations on collections. They are all of the form:

```
collection->operation(arguments)
```

Collection Operations

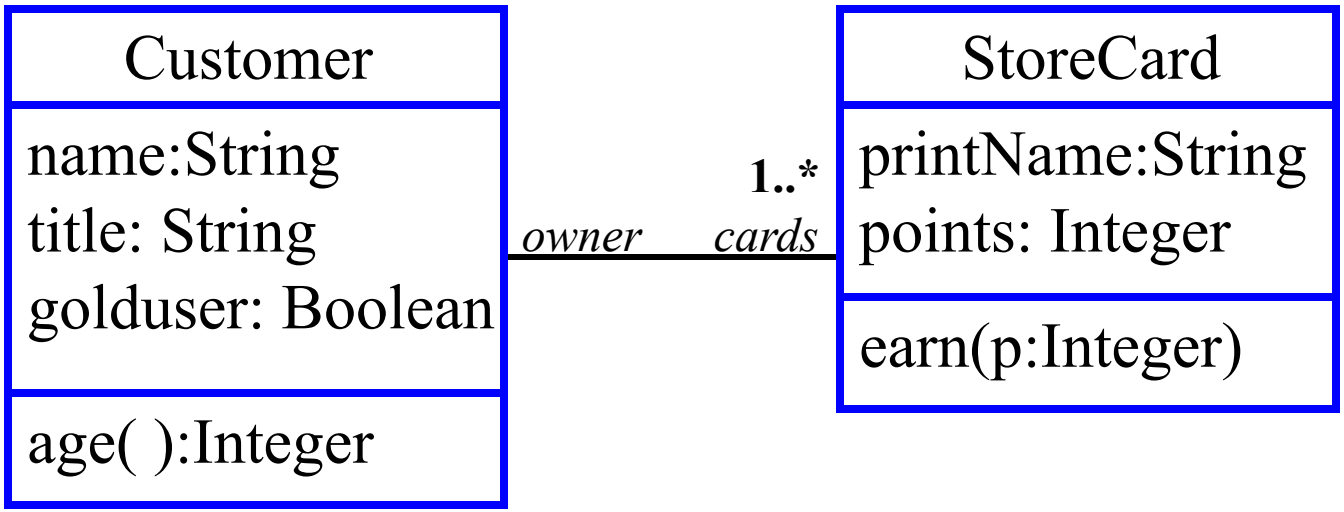
<collection> → size
→ isEmpty
→ notEmpty
→ sum ( )
→ count (object)
→ includes (object)
→ includesAll (collection)

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))

Note switch of context!

OCL-Collection

- The OCL-Type Collection is the generic superclass of a collection of objects of Type T
- Subclasses of Collection are
  - Set: Set in the mathematical sense. Every element can appear only once
  - Bag: A collection, in which elements can appear more than once (also called multiset)
  - Sequence: A multiset, in which the elements are ordered
- Example for Collections:
  - Set(Integer): a set of integer numbers
  - Bag(Person): a multiset of persons
  - Sequence(Customer): a sequence of customers

OCL Operations for OCL Collections (1)

size: Integer

Number of elements in the collection

includes(o:OclAny): Boolean

True, if the element **o** is in the collection

count(o:OclAny): Integer

Counts how many times an element is contained in the collection

isEmpty: Boolean

True, if the collection is empty

notEmpty: Boolean

True, if the collection is not empty

The OCL-Type **OclAny** is the most general OCL-Type

OCL-Operations for OCL-Collections(2)

union(c1:Collection)

Union with collection **c1**

intersection(c2:Collection)

Intersection with Collection **c2** (contains only elements, which appear in the collection as well as in collection **c2** auftreten)

including(o:OclAny)

Collection containing all elements of the Collection and element **o**

select(expr:OclExpression)

Subset of all elements of the collection, for which the OCL-expression **expr** is true

How do we get OCL-Collections?

- A collection can be generated by explicitly enumerating the elements
- A collection can be generated by navigating along one or more 1-N associations
  - Navigation along a single 1:n association yields a Set
  - Navigation along a couple of 1:n associations yields a Bag (Multiset)
  - Navigation along a single 1:n association labeled with the constraint {ordered } yields a Sequence

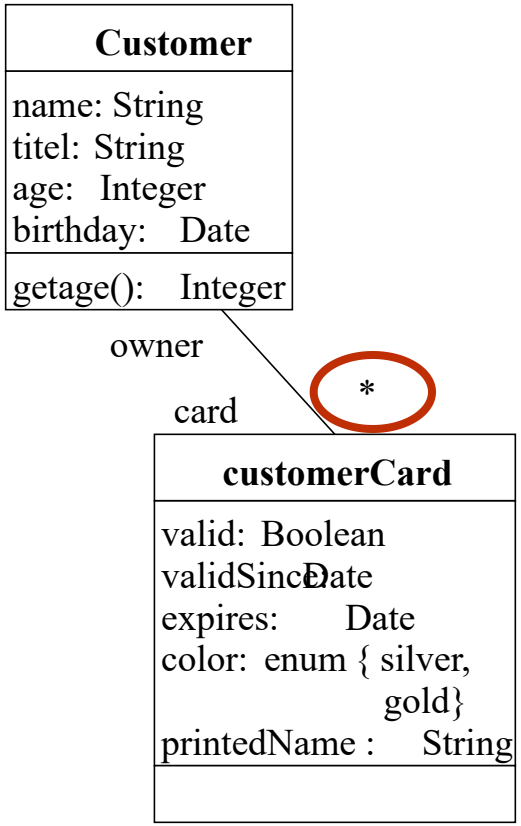
Navigation through a 1:n-Association

Example: A Customer should not have more than 4 cards

context Customer **inv**:
card->size <= 4

Alternative writing style
Customer
card->size <= 4

card denotes a set of **customercards**



Navigation through several 1:n-Associations

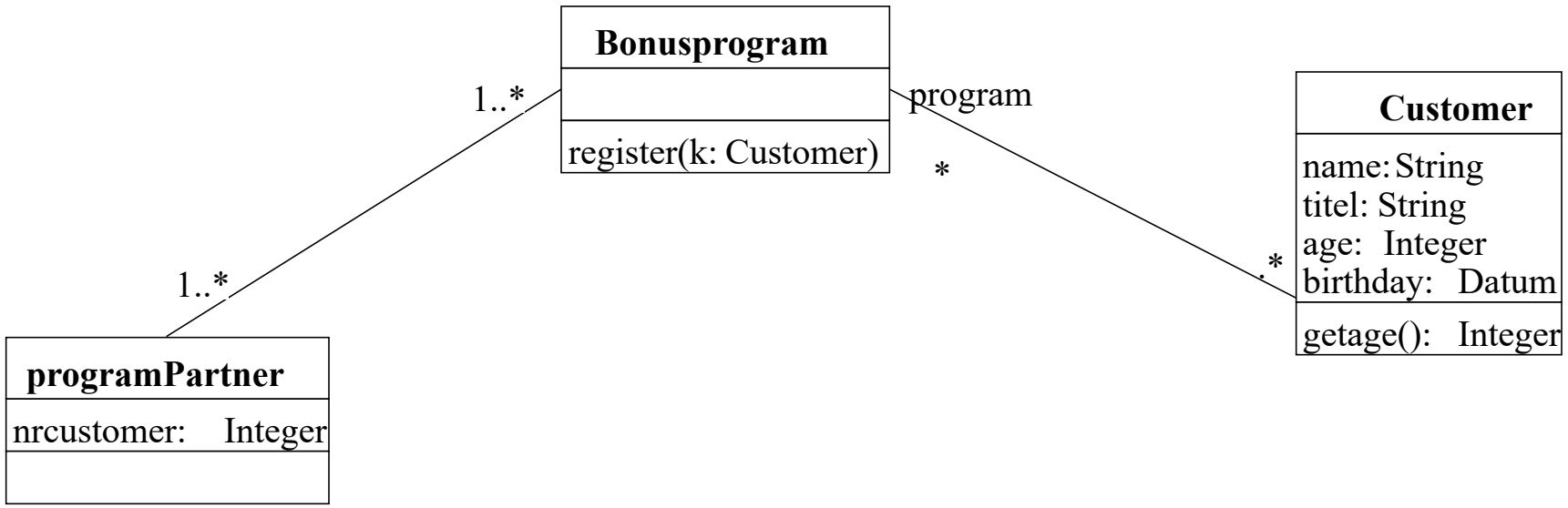
Example:

programPartner

nrcustomer = bonusprogram.customer->size

Customer denotes a multiset of **customer**

bonusprogram denotes a set of **Bonusprograms**



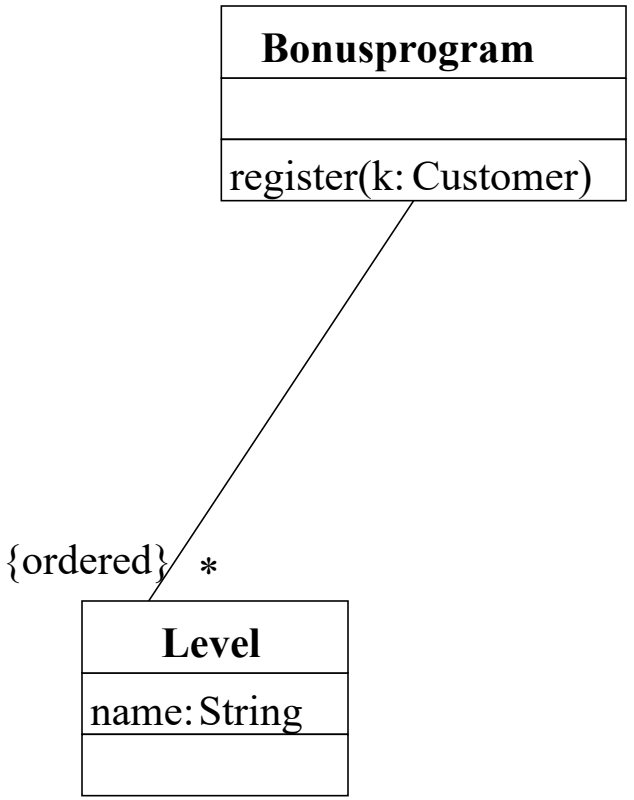
Navigation through a constrained Association

- Navigation through an association with the constraint **{ordered}** yields a *sequence*.
- Example:

Bonusprogram

level->size = 2

level denotes a sequence von **levels**



Conversion between OCL-Collections

- OCL offers operations to convert OCL-Collections:

asSet
  Transforms a multiset or sequence into a set

asBag
  transforms a set or sequence into a multiset

asSequence
  transforms a set or multiset into a sequence.

Example of a Conversion

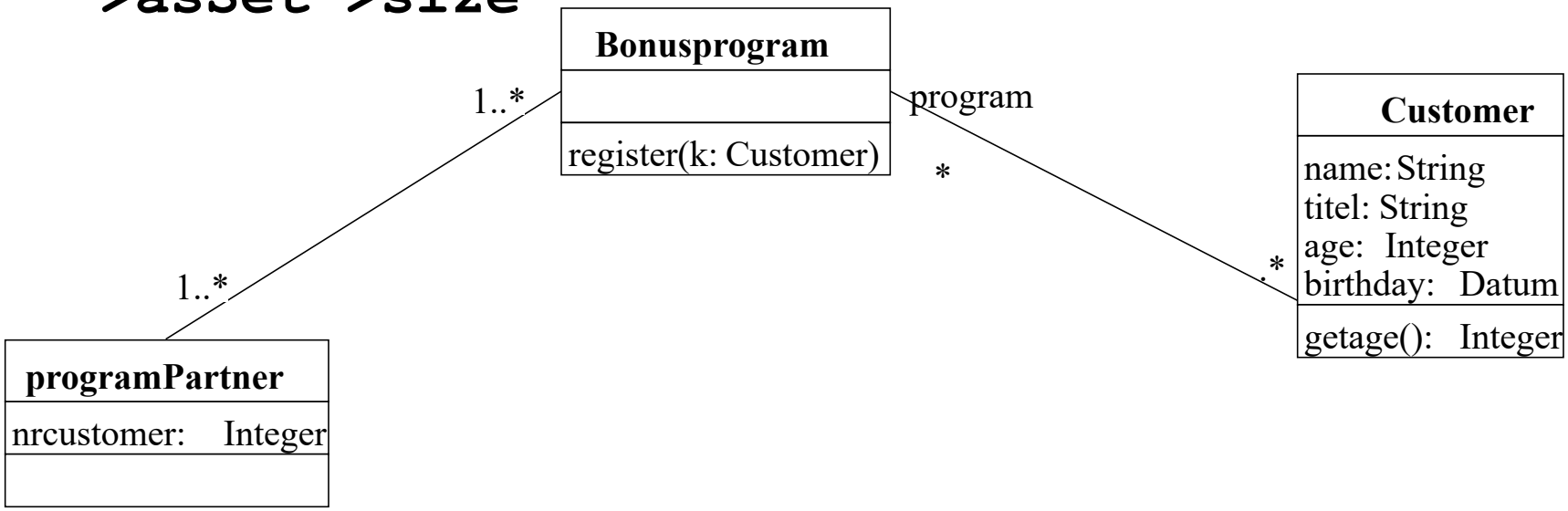
programPartner

nrcustomer = bonusprogram.Customer->size

This expression may contain customer multiple times, we can get the number of unique customers as follows:

programPartner

nrcustomer = bonusprogram.Customer->asSet->size



Operations on OCL Type Sequence

first: T

The first element of a sequence

last: T

The last element of a sequence

at(index:Integer): T

The element with index **index** in the sequence

Example: „*The first Level, you can reach in the bonusprogram has the name 'Silber'.*"

OCL-Invariant:

Bonusprogram:

level->first.name = "Silber"

Specifying the Model Constraints: Using asSet

Local attribute navigation

context **Tournament** inv:

end - start <= Calendar.WEEK

Directly related class navigation

context **Tournament::acceptPlayer(p)**

pre:

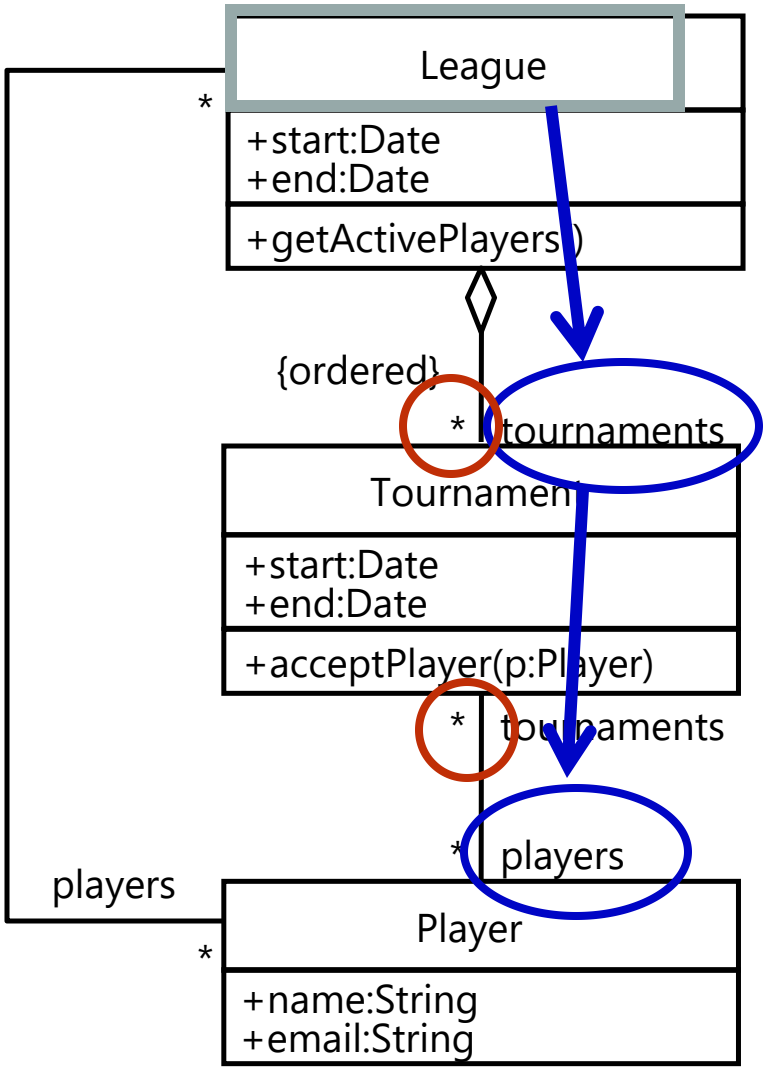
league.players->includes(p)

Indirectly related class navigation

context **League::getActivePlayers**

post:

result=tournaments.players->asSet



Evaluating OCL Expressions

The value of an OCL expression is an object or a collection of objects.

- Multiplicity of the association-end is 1
  - The value of the OCL expression is a single object
- Multiplicity is 0..1
  - The result is an empty set if there is no object, otherwise a single object
- Multiplicity of the association-end is *
  - The result is a collection of objects
    - By default, the navigation result is a Set
    - When the association is {ordered}, the navigation results in a Sequence
    - Multiple "1-Many" associations result in a Bag

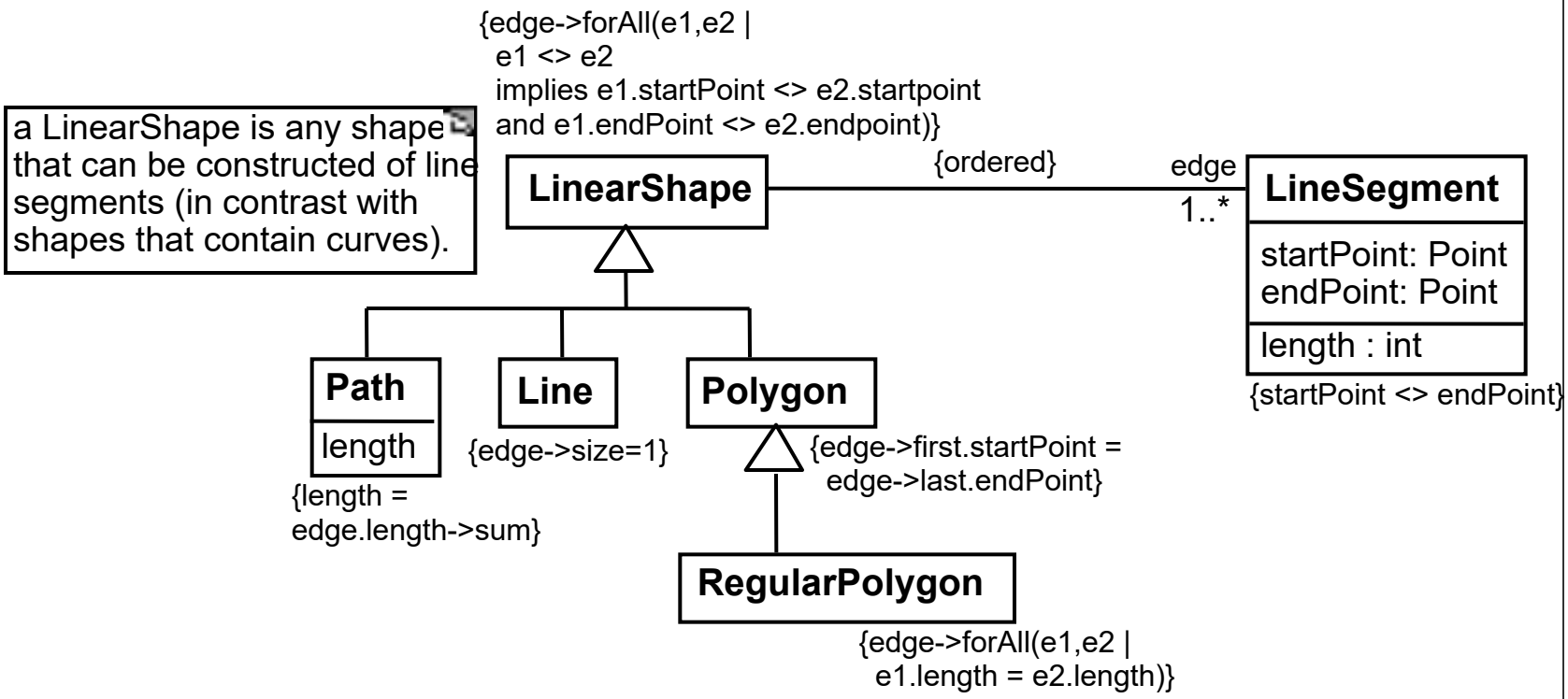
Additional Readings

- J.B. Warmer, A.G. Kleppe
  The Object Constraint Language: Getting your Models ready for MDA, Addison-Wesley, 2nd edition, 2003
- B. Meyer
  Object-Oriented Software Construction, 2nd edition, Prentice Hall, 1997.
- B. Meyer,
  Design by Contract: The Lesson of Ariane, Computer, IEEE, Vol. 30, No. 2, pp. 129-130, January 1997. http://archive.eiffel.com/doc/manuals/technology/contract/ariane/page.html
- C. A. R. Hoare,
  An axiomatic basis for computer programming. Communications of the ACM, 12(10):576-585, October 1969. (Good starting point for Hoare logic: http://en.wikipedia.org/wiki/Hoare_logic)

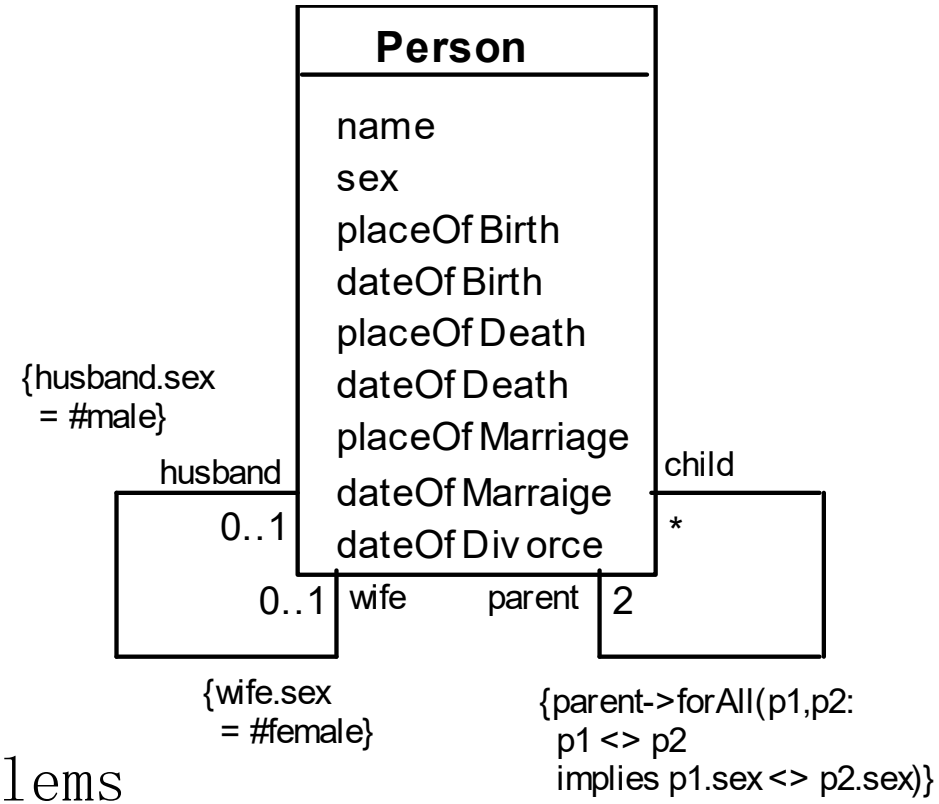
Summary

- Constraints are predicates (often boolean expressions) on UML model elements
- Contracts are constraints on a class that enable class users, implementors and extenders to share the same assumption about the class (“Design by contract”)
- OCL is the example of a formal language that allows us to express constraints on UML models
- Complicated constrains involving more than one class, attribute or operation can be expressed with 3 basic navigation types.

An example: constraints on Polygons

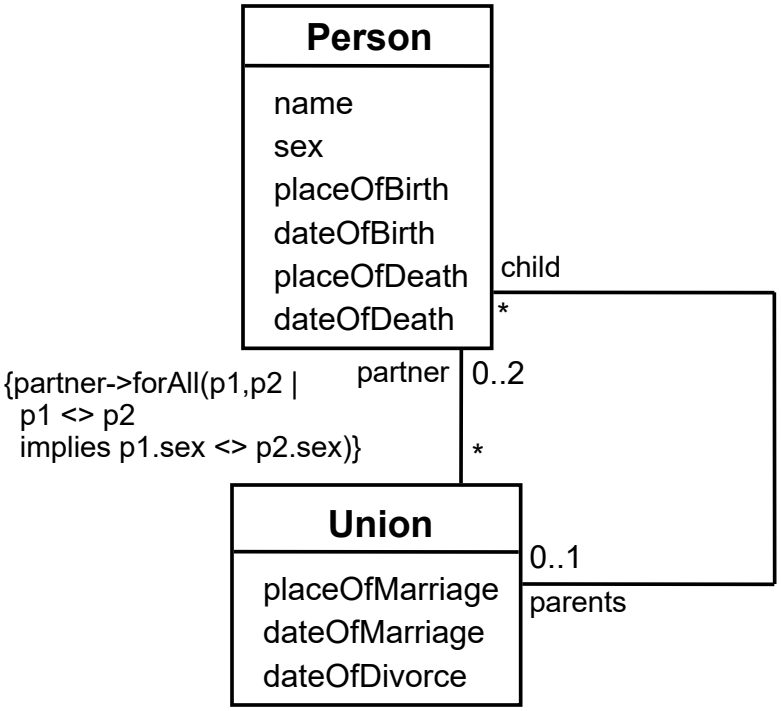


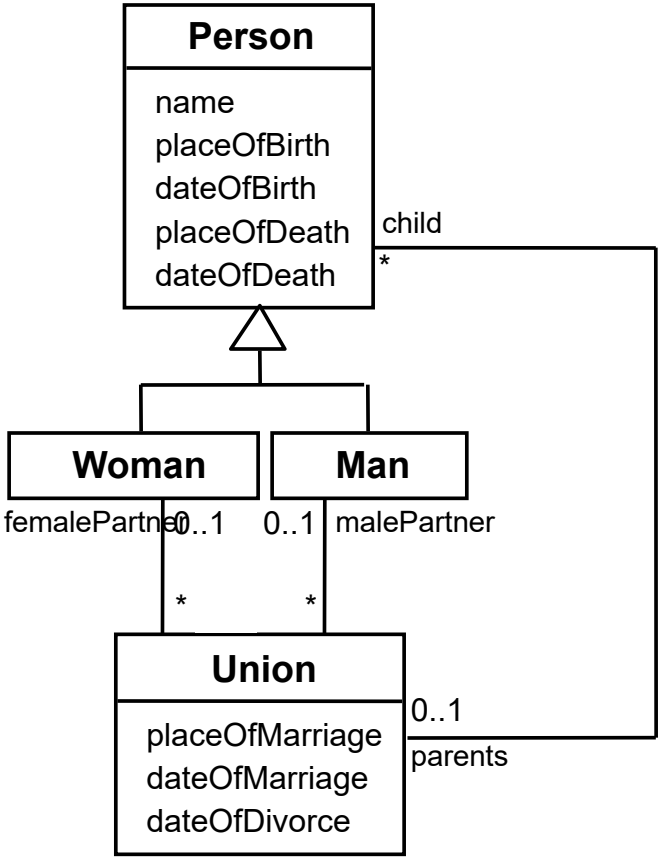
Detailed Example: A Class Diagram for Genealogy



- Problems
  - A person must have two parents
  - Marriages not properly accounted for

Genealogy example: Possible solutions





Additional Constraints on this Model

1. A Tournament’s planned duration must be under one week.
2. Players can be accepted in a Tournament only if they are already registered with the corresponding League.
3. The number of active Players in a League are those that have taken part in at least one Tournament of the League.

Additional Constraints on this Model

1. A Tournament's planned duration must be under one week.
2. Players can be accepted in a Tournament only if they are already registered with the corresponding League.
3. The number of active Players in a League are those that have taken part in at least one Tournament of the League.

Additional Constraints on this Model

1. A Tournament's planned duration must be under one week.
2. Players can be accepted in a Tournament only if they are already registered with the corresponding League.
3. The number of active Players in a League are those that have taken part in at least one Tournament of the League.

OCL supports Quantification

OCL forall quantifier

/* All Matches in a Tournament occur within the Tournament's time frame */

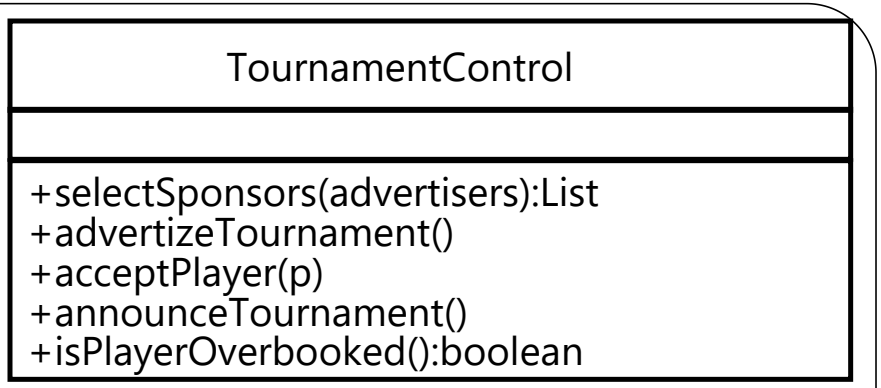
```
context Tournament inv:
    matches->forAll(m:Match |
            m.start.after(t.start) and m.end.before(t.end))
```

OCL exists quantifier

/* Each Tournament conducts at least one Match on the first day of the Tournament */

```
context Tournament inv:
            matches->exists(m:Match | m.start.equals(start))
```

Pre- and post-conditions for ordering operations on TournamentControl



context TournamentControl::selectSponsors(advertisers) **pre**:
    interestedSponsors->notEmpty and
        tournament.sponsors->isEmpty
context TournamentControl::selectSponsors(advertisers) **post**:
    tournament.sponsors.equals(advertisers)
context TournamentControl::advertiseTournament() **pre**:
    tournament.sponsors->isEmpty and
        not tournament.advertised
context TournamentControl::advertiseTournament() **post**:
    tournament.advertised
context TournamentControl::acceptPlayer(p) **pre**:
    tournament.advertised and
        interestedPlayers->includes(p) and
            not isPlayerOverbooked(p)
context TournamentControl::acceptPlayer(p) **post**:
    tournament.players->includes(p)

Specifying invariants on Tournament and Tournament Control

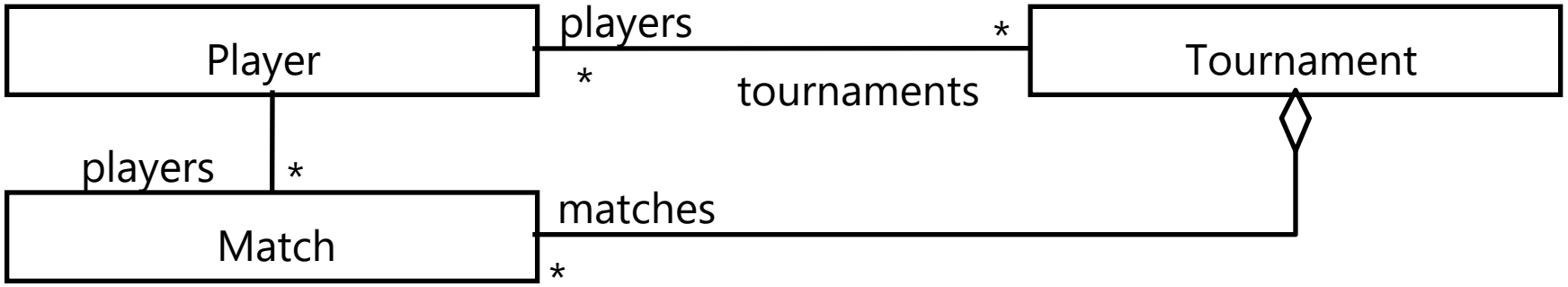
English: “All Matches of in a Tournament must occur within the time frame of the Tournament”

```
context Tournament inv:
    matches->forAll(m|
        m.start.after(start) and m.start.before(end))
```

English: “No Player can take part in two or more Tournaments that overlap”

```
context TournamentControl inv:
    tournament.players->forAll(p|
        p.tournaments->forAll(t|
            t <> tournament implies
                not t.overlap(tournament)))
```

Specifying invariants on Match



English: "A match can only involve players who are accepted in the tournament"

```
context Match inv:
        players->forAll(p|
                p.tournaments->exists(t|
                        t.matches->includes(self)))
context Match inv:
        players.tournaments.matches.includes(self)
```