Software Engineering I: Software Technology

WS 2008/09

Object Design: Interface Specification and OCL

Bernd Bruegge
Applied Software Engineering
Technische Universitaet Muenchen

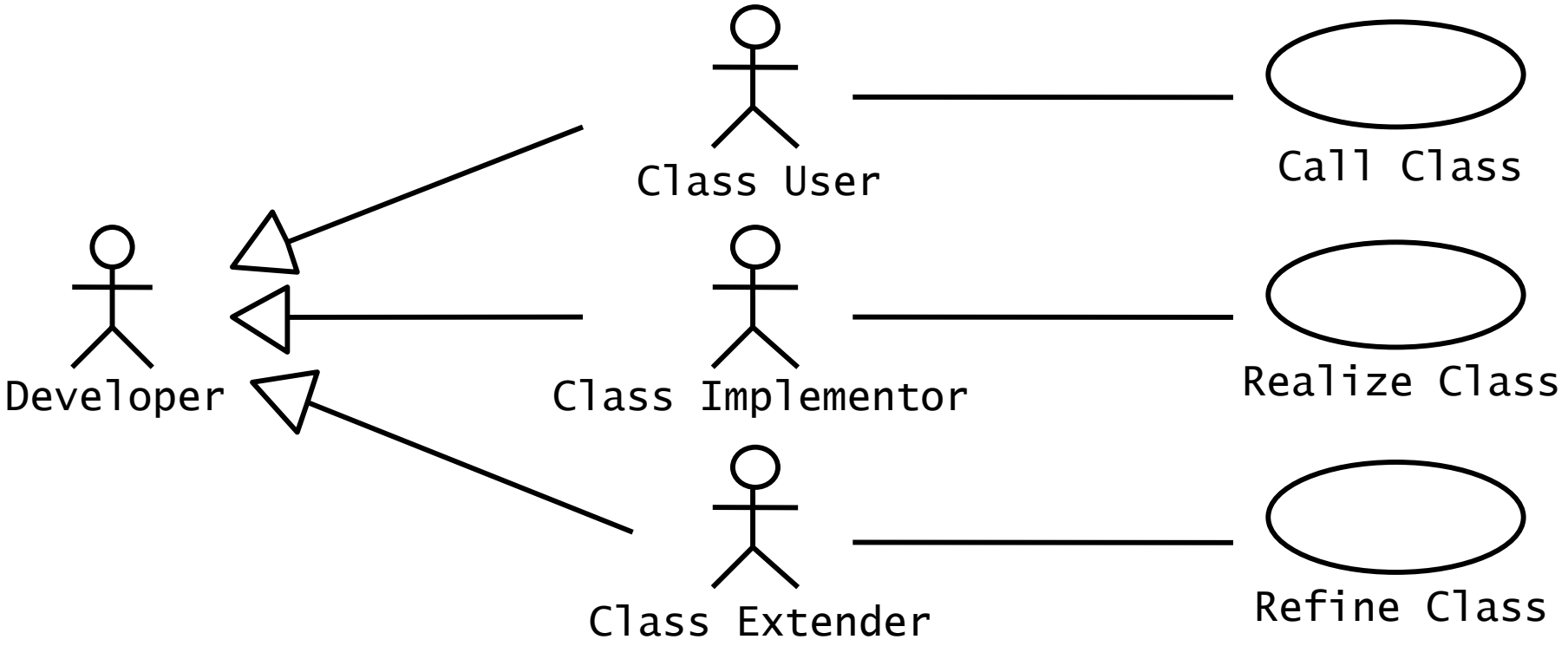
Outline of the Lecture

- Object Design: Interface Specification Activities
- Visibilities
- Information Hiding
- OO-Contracts
- OCL: A language for expressing OO-Contracts

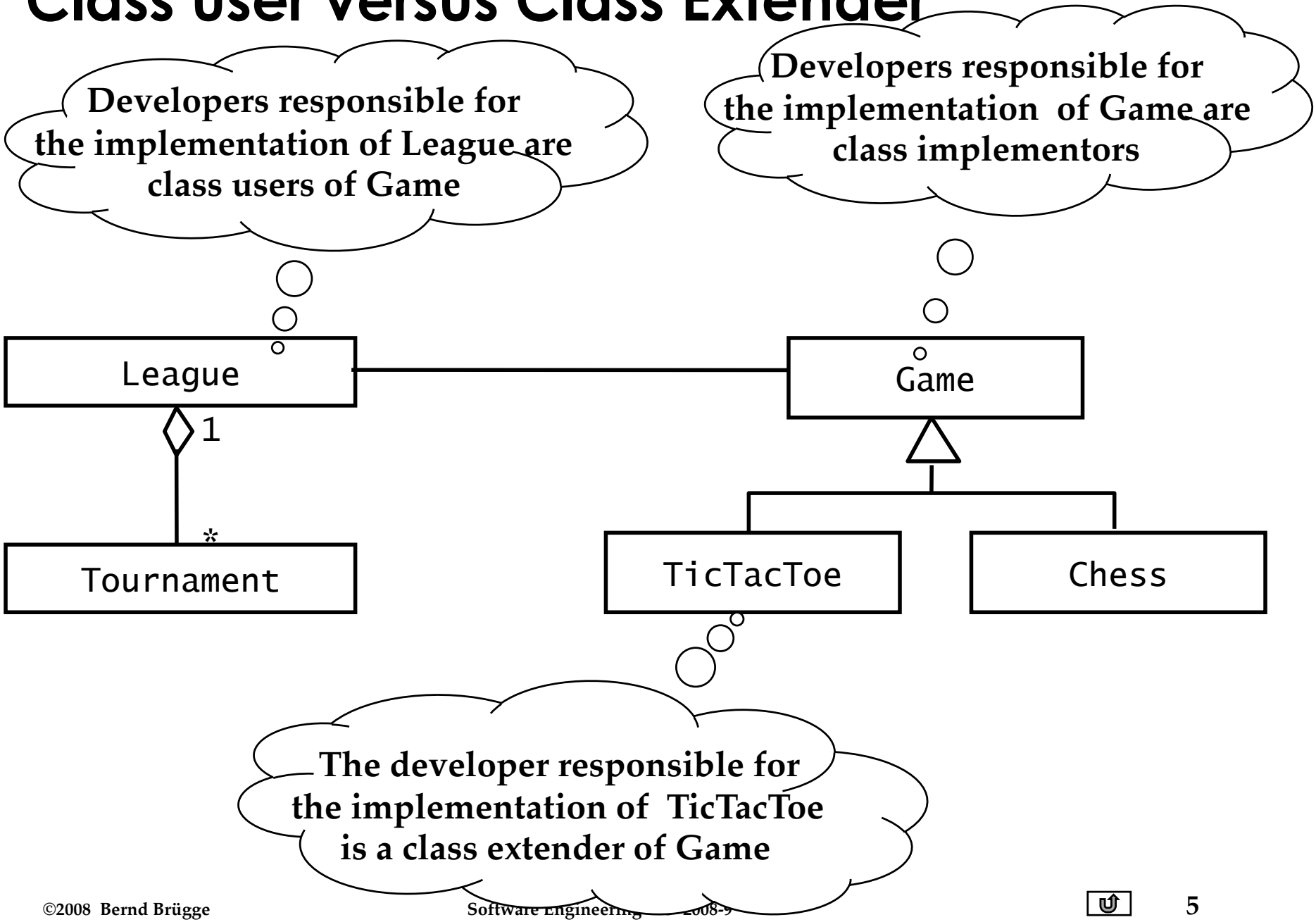
Object Design vs Requirements Analysis

- Requirements Analysis: The functional model and the dynamic model deliver operations for the object model
- Object Design:
  - The object designer decides where to put these operations in the object model
  - The object designer can choose among different ways to implement the analysis model
- Thus Object design is the process of
  - adding details to the requirements analysis
  - making implementation decisions
- Object design serves as the basis of implementation

Developers play different Roles during Object Design



Class user versus Class Extender



Object Design consists of 4 Activities

✓ 1. Reuse: Identification of existing solutions
- Use of inheritance
- Off-the-shelf components and additional solution objects
- Design patterns

➢ 2. Interface specification
- Describes precisely each class interface

3. Object model restructuring
- Transforms the object design model to improve its understandability and extensibility

4. Object model optimization
- Transforms the object design model to address performance criteria such as response time or memory utilization.

Interface Specification Activities

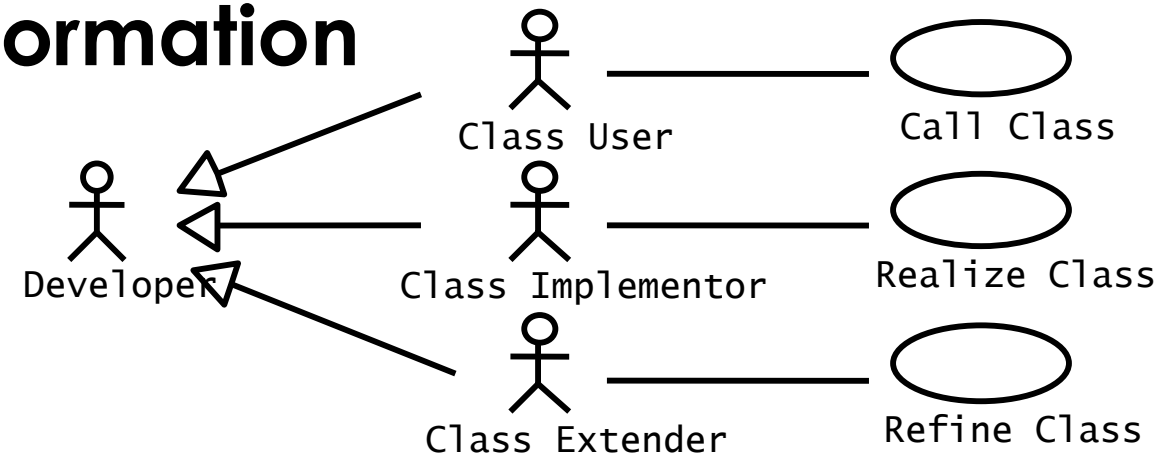
- Interface Specification during Object Design consists of 3 Activities:
  - Add visibility information
  - Add type signature information
  - Add contracts.

Implementation of UML Visibility in Java


```
public class Tournament {
        private int maxNumPlayers;

    public Tournament(League l, int maxNumPlayers)
    public int getMaxNumPlayers() {…};
    public List getPlayers() {…};
    public void acceptPlayer(Player p) {…};
    public void removePlayer(Player p) {…};
    public boolean isPlayerAccepted(Player p) {…};
```

Add Visibility Information



+ Public => Class user
  - Public attributes/operation can be accessed by any class.
- Private => Class implementor
  - Private attributes and operations can be accessed only by the class in which they are defined.
  - They cannot be accessed by subclasses or other classes.
Protected => Class extender
  - Protected attributes/operations can be accessed by the class in which they are defined and by any descendent of the class.

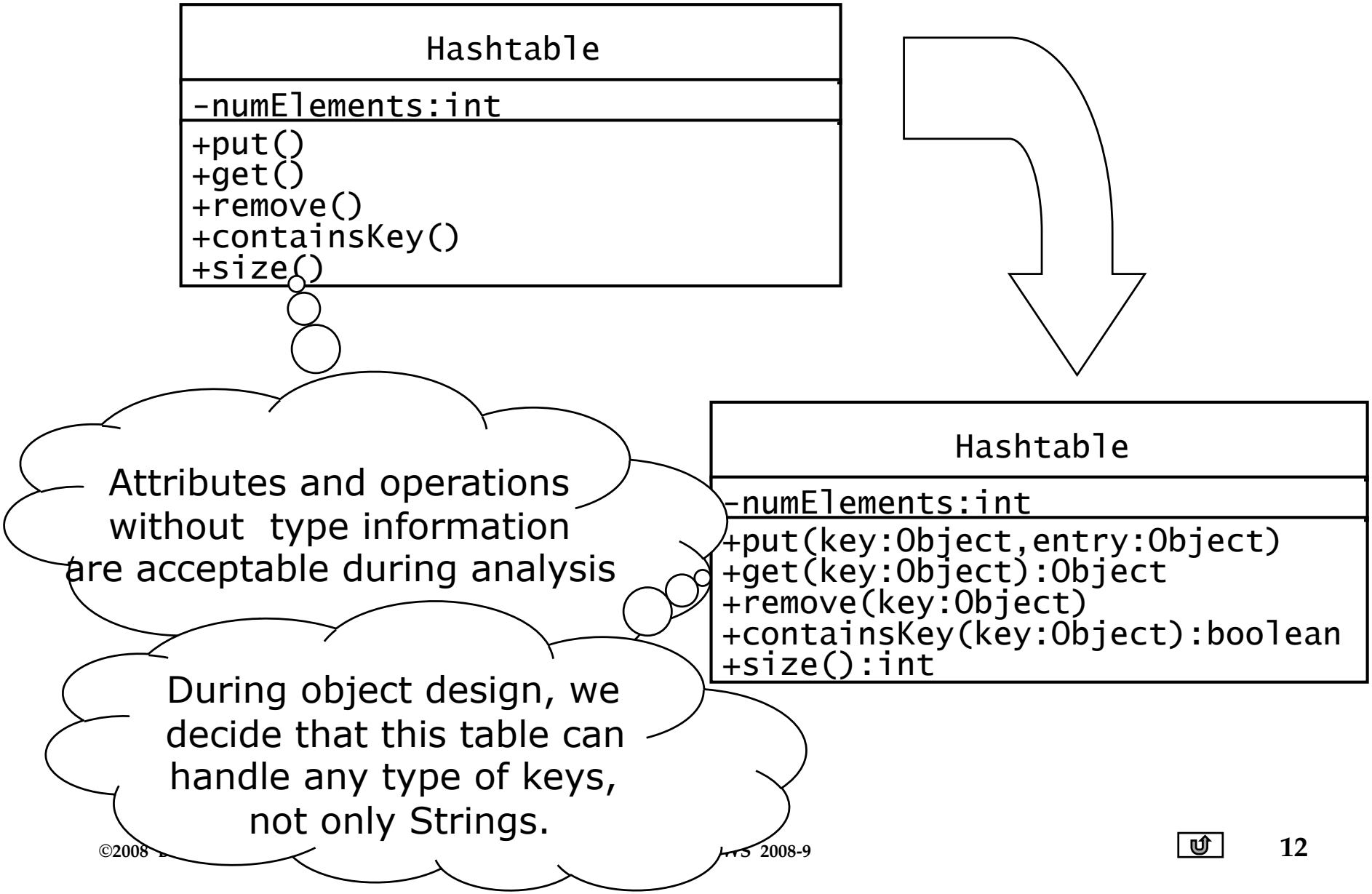
Information Hiding Heuristics

- Carefully define the public interface for classes as well as subsystems
  - For subsystems use a façade if possible
- Always apply the "Need to know" principle
  - Only if somebody needs to access the information, make it publicly possible,
    - But only through well defined channels, so you always know the access
- The fewer details class users need to know
  - the less likely they will be affected by any changes
  - the easier the class can be changed
- Trade-off: Information hiding vs efficiency
  - Accessing a private attribute might be too slow.

Information Hiding Design Principles

- Only the operations of a class are allowed to manipulate its attributes
  - Access attributes only via operations
- Hide external objects at subsystem boundary
  - Define abstract class interfaces which mediate between the system and the external world as well as between subsystems
- Do not apply an operation to the result of another operation.
  - Write a new operation that combines the two operations.

Add Type Signature Information



Add Contracts

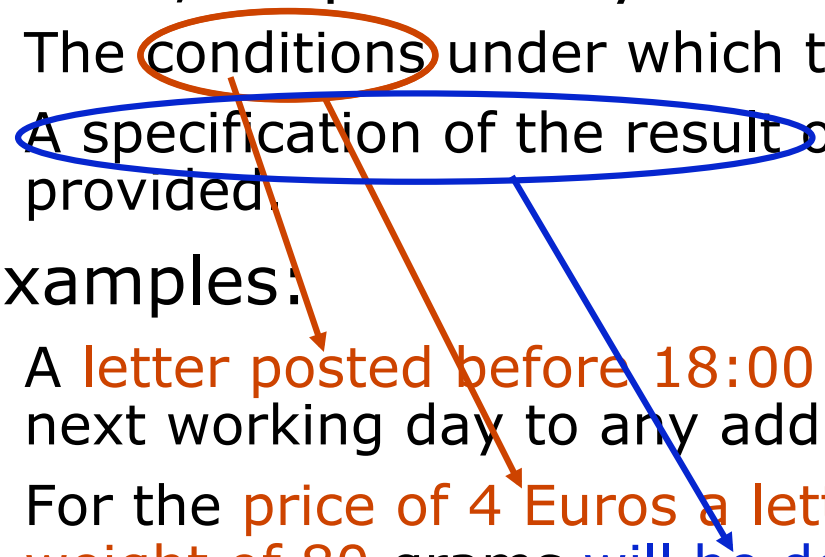
- Example of constraints (taken from Arena, see Bruegge&Dutoit, Chapter 9):
  - An already registered player cannot be registered again
  - The number of players in a tournament should not be more than maxNumPlayers
  - One can only remove players that have been registered
- These constraints cannot be modeled in UML
- We model them with contracts in OCL.

Contract

- Contract: A lawful agreement between two parties in which both parties accept obligations and on which both parties can found their rights
  - The remedy for breach of contract is usually an award of money to the injured party
- Object-oriented Contract: Describes the services that are provided by an object if certain conditions are fulfilled
  - Services = "Obligations"
  - Conditions = "Rights"
  - The remedy for breach of OO-contracts is the generation of an exception.

Object-Oriented Contract

- An object-oriented contract describes the services that are provided by an object. For each service, it specifically describes two things:
  - The conditions under which the service will be provided
  - A specification of the result of the service that is provided.
- Examples:
  - A letter posted before 18:00 will be delivered on the next working day to any address in Germany.
  - For the price of 4 Euros a letter with a maximum weight of 80 grams will be delivered anywhere in Germany within 4 hours of pickup.



Modeling OO-Contracts

- Natural Language:
  - Advantage: Contract partners already know the language
  - Disadvantage: When using natural language, one often makes implicit assumptions about the rights and obligations of the contract partners
- Mathematical Notation:
  - Advantage: Contract can be precisely and uniquely specified
  - Disadvantage: Normal customers are not mathematicians
- Models and contracts:
  - A language for the formulation of constraints with the formal strength of the mathematical notation and the easiness of natural language:

    $\Rightarrow$ UML + OCL (Object Constraint Language)
  - Uses the abstractions of the UML model
  - OCL is based on predicate calculus.

Contracts and Formal Specification

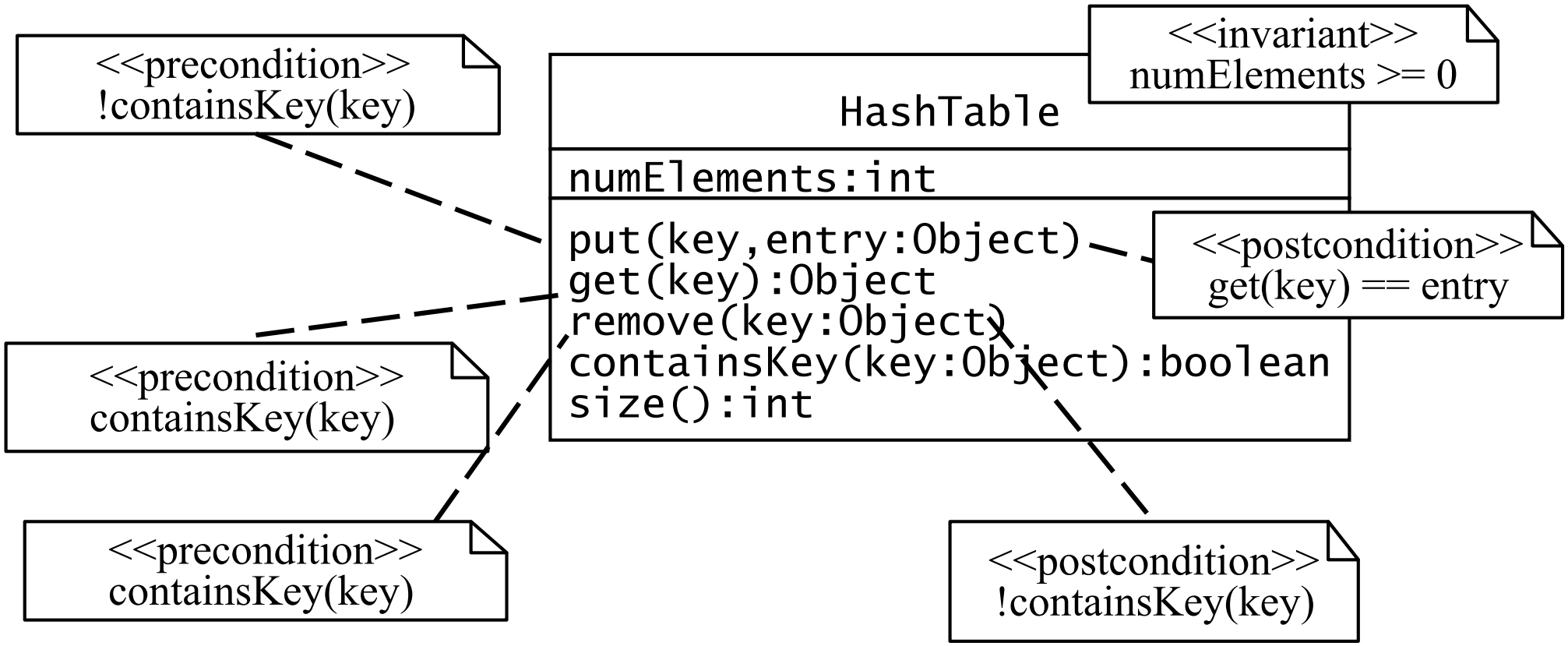
- Contracts enable the caller and the provider to share the same assumptions about the class
- A contract is an exact specification of the interface of an object
- A contract includes three types of constraints:
  - Invariant:
    - A predicate that is always true for all instances of a class
  - Precondition ("rights"):
    - Must be true before an operation is invoked
  - Postcondition ("obligation"):
    - Must be true after an operation is invoked.

Formal Specification

- Definition: A contract is called a formal specification, if the invariants, rights and obligations in the contract are unambigious.

Expressing Constraints in UML Models

- A constraint can also be depicted as a note attached to the constrained UML element by a dependency relationship.



Why not use Contracts already in Requirements Analysis?

- Many constraints represent domain level information
- So, why not use them in requirements analysis?
  - Constraints increase the precision of requirements
  - Constraints can yield more questions for the end user
  - Constraints can clarify the relationships among several objects
- Constraints are sometimes used during requirements analysis, however there are trade offs

Requirements vs. Object Design Trade-offs

- Communication among stakeholders
  - Can the client understand formal constraints?
- Level of detail vs. rate of requirements change
  - Is it worth precisely specifying a concept that will change several times?
- Level of detail vs. elicitation effort
  - Is it worth the time interviewing the end user
  - Will these constraints be discovered during object design anyway?
- Testing constraints
  - If tests are generated early, do theyrequire this level of precision?

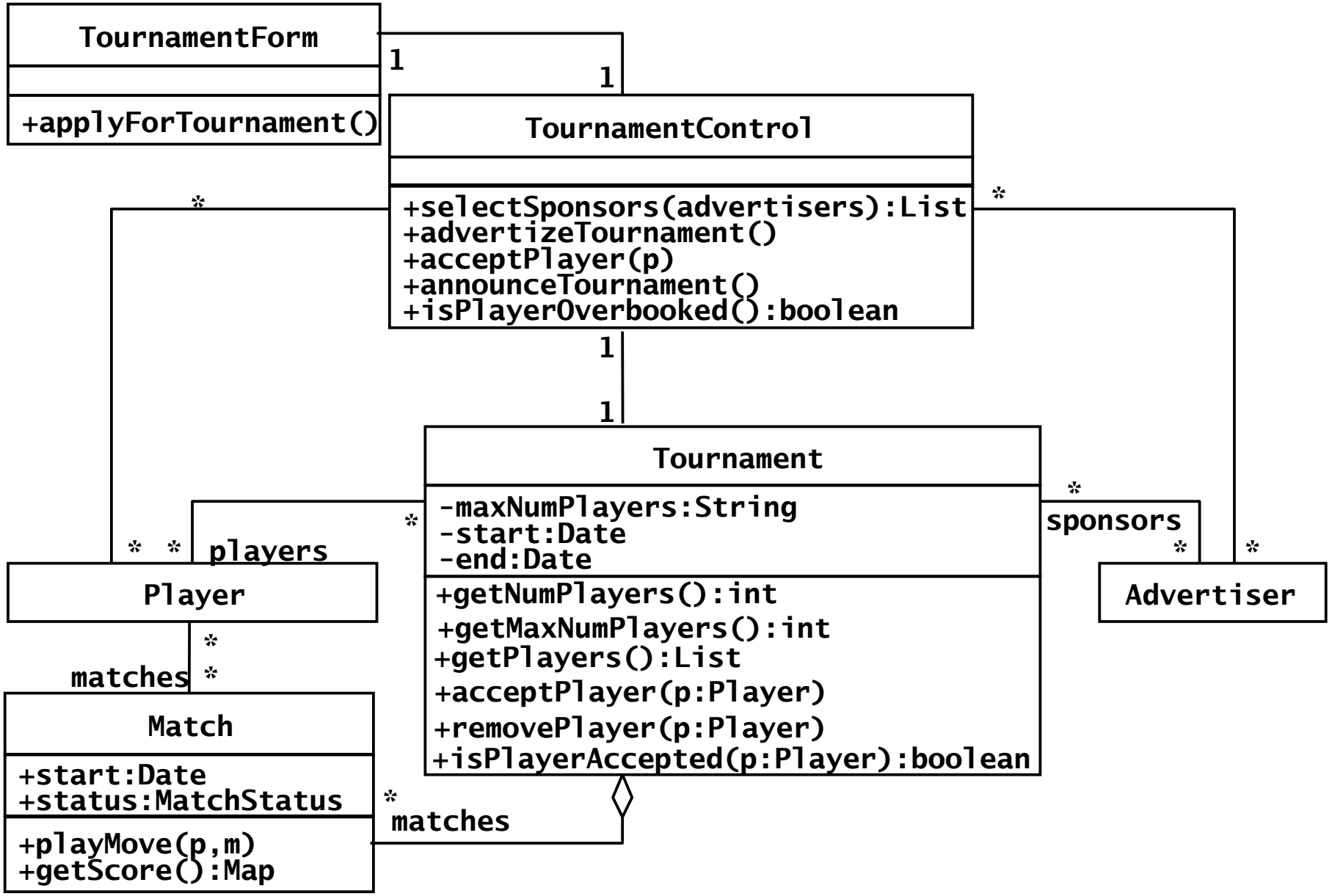
Outline of the Lecture

- ✓ Object Design Activities
- ✓ Visibilities
- ✓ Information Hiding
- ✓ OO-Contracts
- ➢ OCL: A language for expressing OO-Contracts

OCL: A language for expressing OO-Contracts

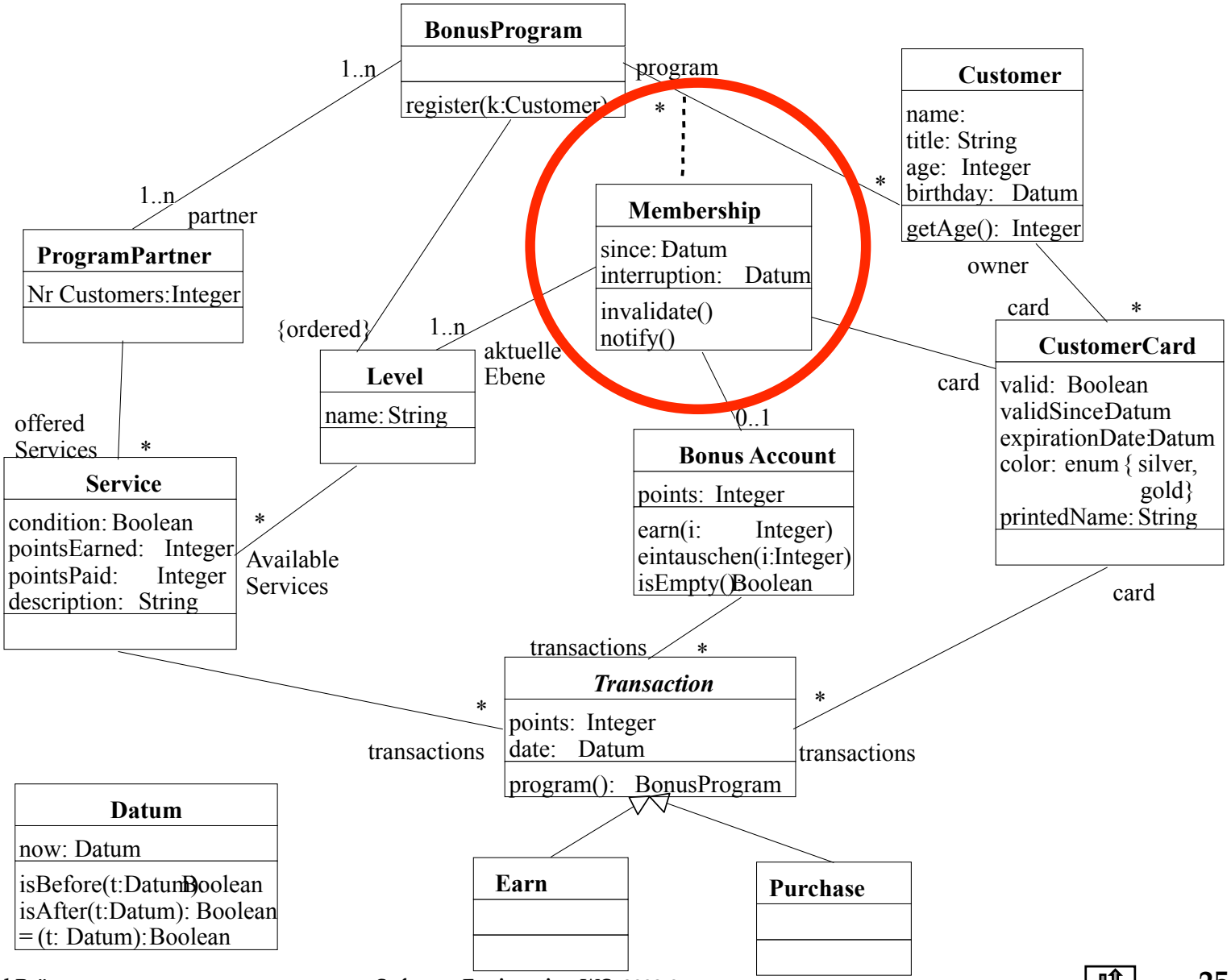
- Basic Concepts
- Simple predicates
- Preconditions
- Postconditions
- Contracts
- Sets, Bags, and Sequences

UML Models used in this Lecture: ARENA (Chapter 9, Bruegge & Dutoit, 2003)



[Warmer], pp. 12

Bonus Program (Warmer and Kleppe, 2003)



OCL: Object Constraint Language

- Formal language for expressing constraints over a set of objects and their attributes
- Part of the UML standard
- Used to write constraints that cannot otherwise be expressed in a diagram
- Declarative
  - No side effects
  - No control flow
- Based on Sets and Multi Sets

OCL Expressions and OCL Data Types

- Constraints are written as OCL expressions
- Operands in OCL expressions are objects and properties (attributed and operations).
- Each OCL object has an OCL type, which defines the operations that can be called on this object
- OCL Types:
  - **Predefined Types:**
    - Base types: **`Integer`**, **`Real`**, **`String`** and **`Boolean`**
    - Collection types: **`Collection`**, **`Set`**, **`Bag`** and **`Sequence`**
  - **User-defined OCL Types:**
    - *All Classes* in the system model, that is all classes in the UML diagrams are automatically OCL types.

OCL Basic Concepts

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

OCL Type `Boolean`

- The truth values are written in OCL as **`true`** and **`false`**. All the standard operators known from boolean algebra are defined in OCL:

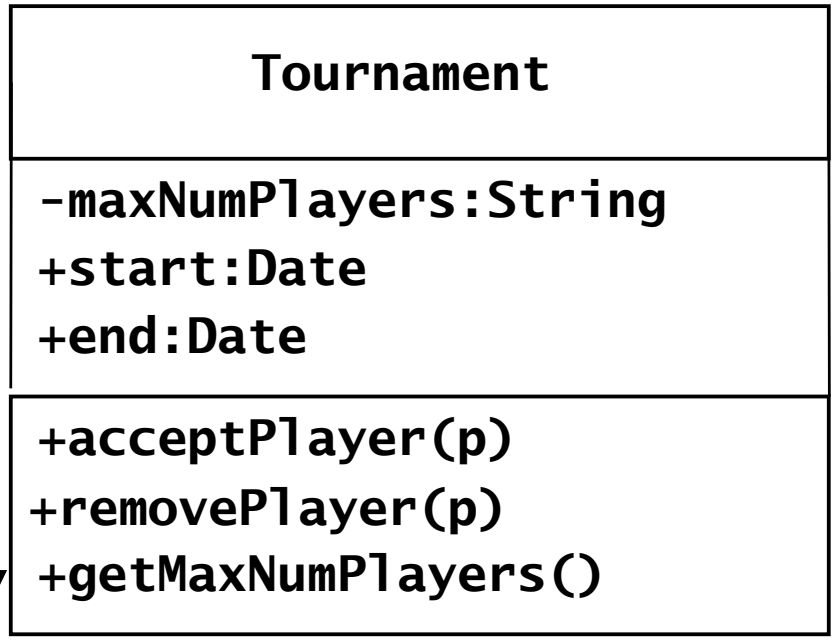
Boolean Operator		OCL Operator
Not:	¬	`not`
And:	∧	`and`
Or:	∨	`or`
Implication:	⇒	`implies`
Equality:	=	`equals`
Unequality:	≠	`xor`

OCL Expression of Type `Boolean`

- **Example:**
  - Problem statement: *"Customers can earn points on a service only, if they have not obtained this service by purchasing it with points"*
  - Alternative formulation: *„A customer cannot earn points with services, if these services have been paid with points"*
- **OCL Expression:**

```
context Service inv:
(pointsEarned > 0) implies (pointsPaid = 0)
```

An OCL Invariant



English (Problem Statement):

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

OCL:

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

Notes:

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

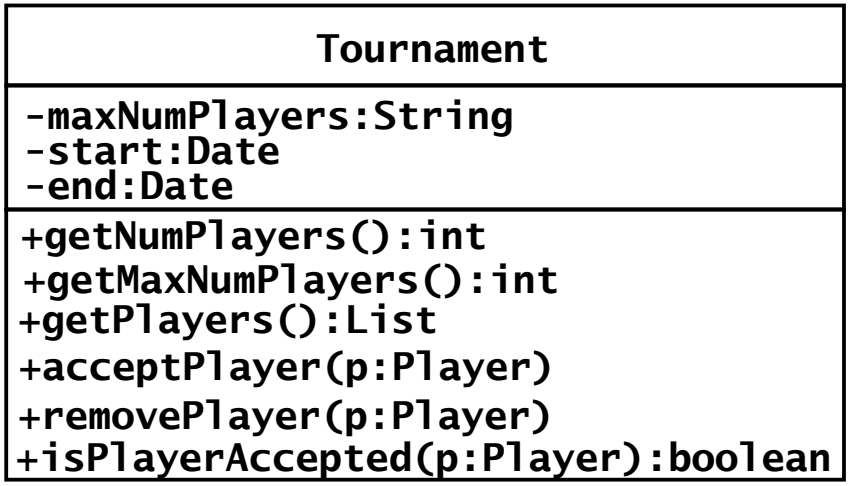
Pre and Post Conditions

- The context of a pre/post condition is the UML operation of a class (called the context operation)
- Generic Form of a Pre- and Post condition:

```
context Complete signature of the context
  operation
pre: OCL Expression
post: OCL Expression
```

- **`pre`** and **`post`** are OCL keywords
- The signature is taken from the class operation in object design model
- Formal parameters in the signature can be used in the formulation of the OCL expressions.

Example of a Precondition in OCL



English:

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

OCL:

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

Note:

- The context of a precondition is always an operation.

Post Conditions can describe Temporal Aspects

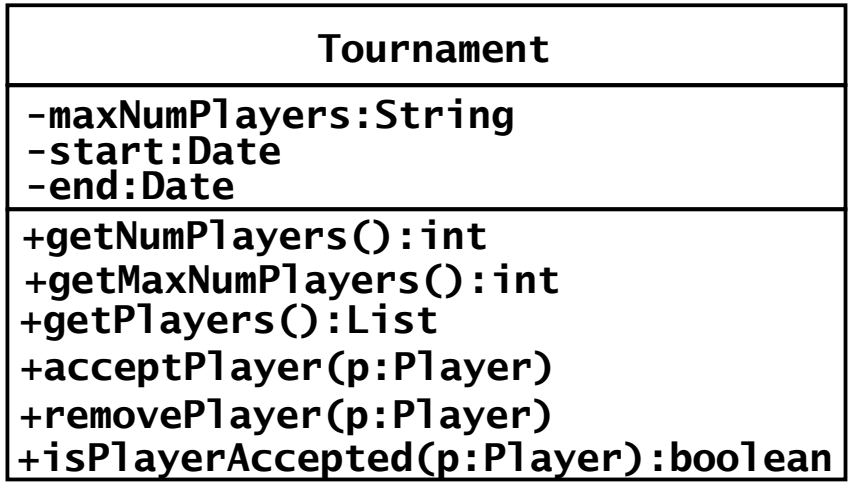
- We use two OCL keywords to model temporal aspects in post conditions

`id@pre`
Represents the value of an attribute id before the execution of the operation

Example: **`customer@pre`** denotes the set of all customers before the execution of the operation

`result`
The result of the operation immediately after the execution.

Example of a OCL Postcondition



English:

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

OCL:

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

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.

Contract4J: Design by Contract ® for Java

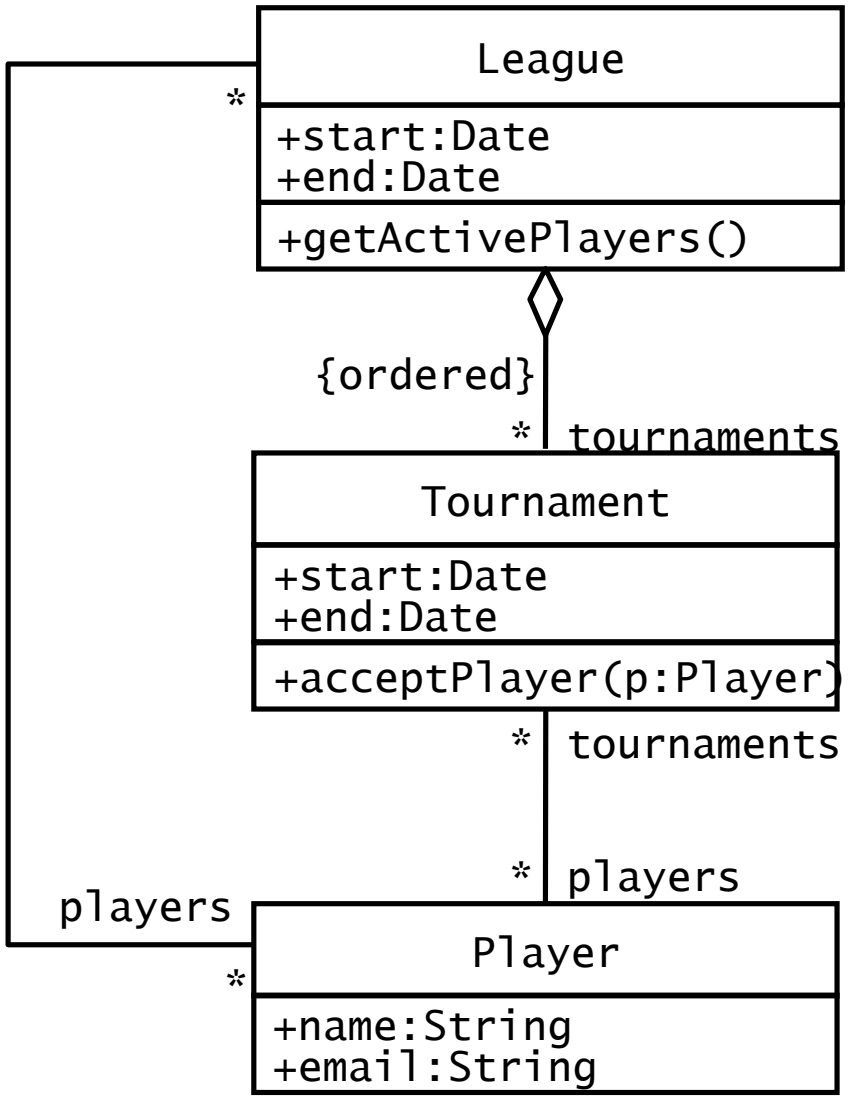
- Contract4J is a tool that supports Design by Contract
- Contract4J uses Java 5 annotations to define OCL expressions
- Annotations have several advantages over JavaDoc-style tags
  - the JVM can be made aware of the annotations at runtime
  - In this case the OCL expressions can be evaluated at runtime and handle failures
  - They can be included in the generated JavaDocs.
- Contract4J Tutorial-Example:
  - http://www.contract4j.org/contract4j/example

Constraints can involve more than one class

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

1. Start from a specific class in the UML class diagram (i.e. select the context)
2. Follow a direct association from that class (context) to to another class (target class)
3. If the association end has a name, use it in the OCL expression, otherwise use the lower-case name of the target class
4. Refer to its attributes and operations or follow an association from that class to another class.

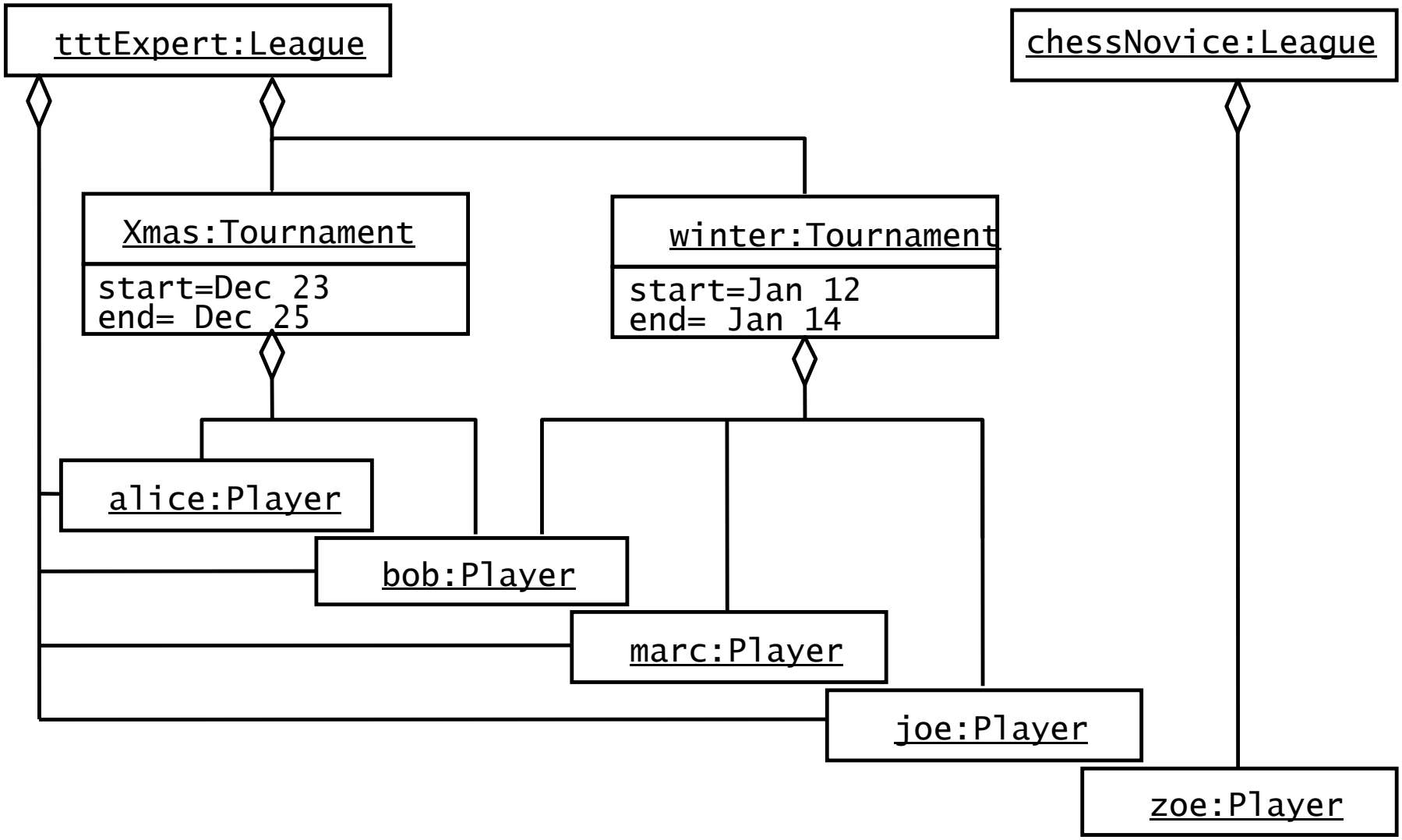
ARENA: League, Tournament and Player



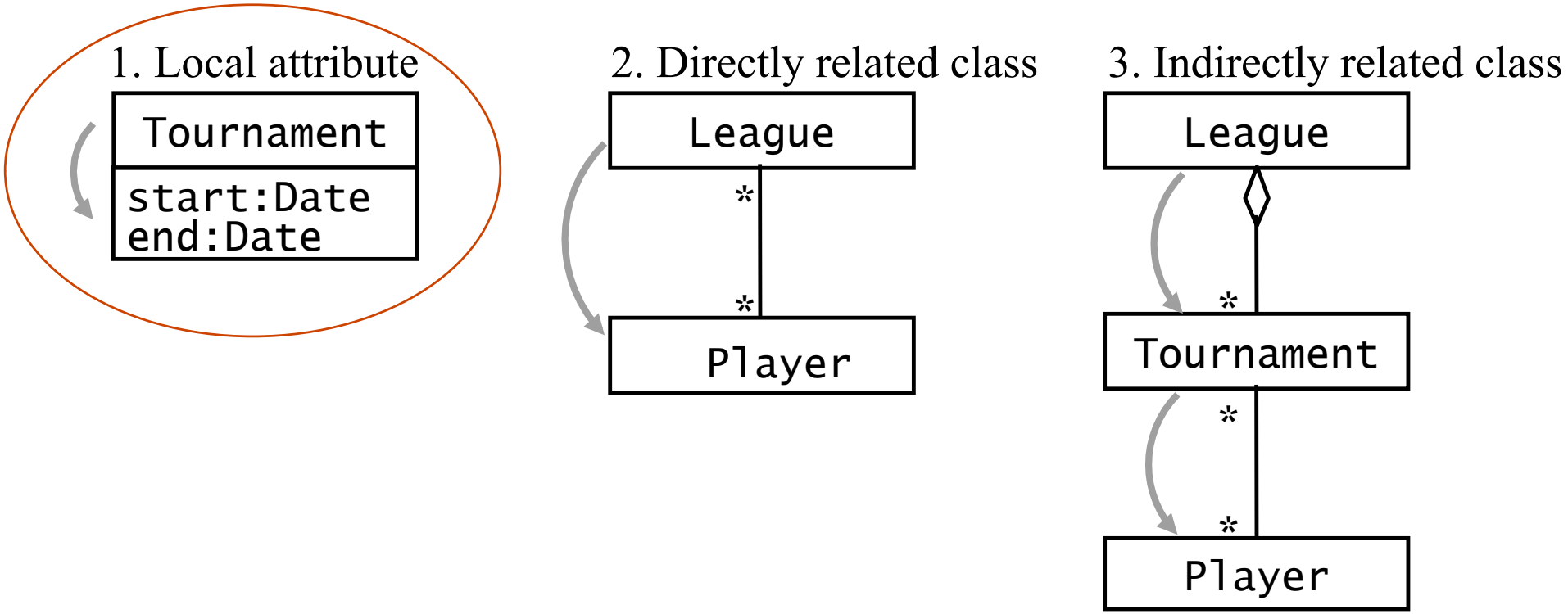
Constraints:

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!

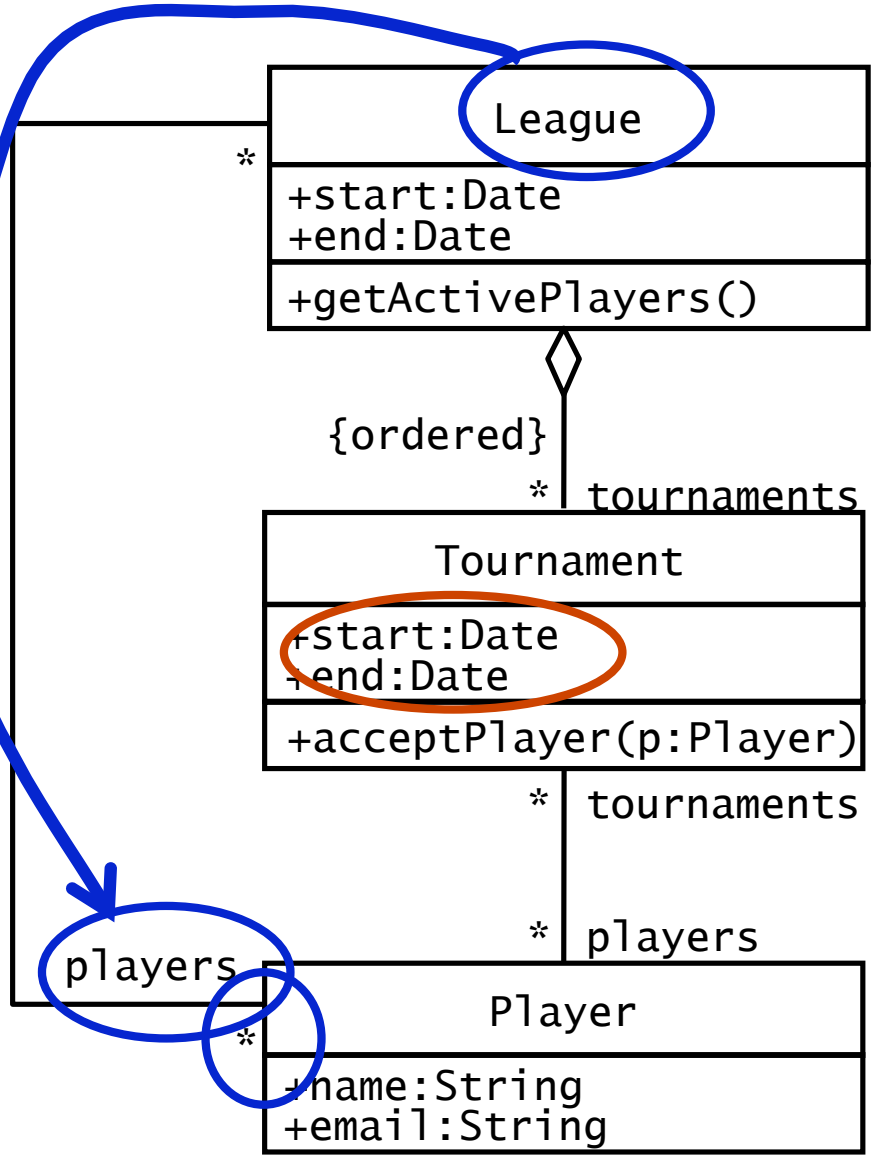
Specifying the Model Constraints in OCL

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)
```

The OCL-Type Collection is the generic superclass of a collection of objects of Type T

OCL-Collection

- 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

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
    - Navigation through multiple "1-Many" associations results in a Bag.

OCL-Operations for 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)

OCT operations for the generation of new collections:

`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)

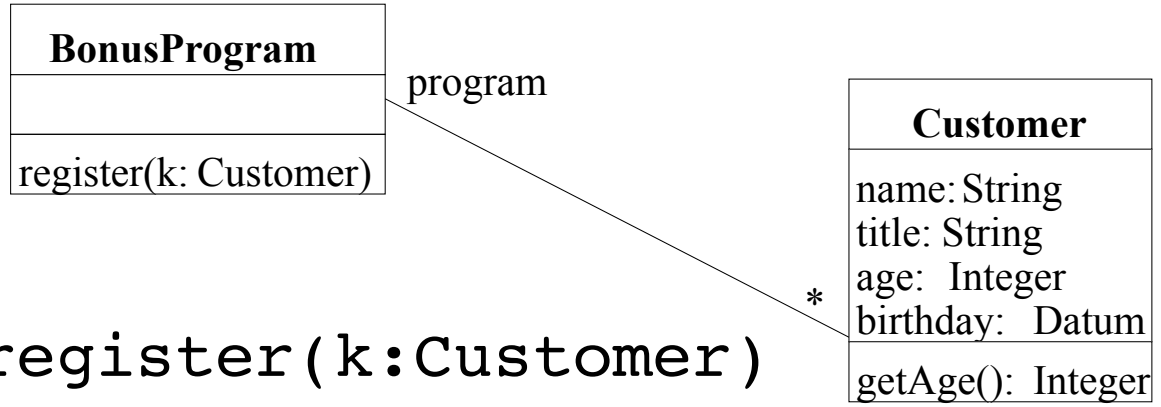
`including(o:OclAny)`
Collection containing all elements of the Collection and element **`o`**

`select(expr:OclExpression)`
Subset of all elements of the collection, for which the OCL-expression **`expr`** is true

Modeling with „includes“ and „including“

Problem statement: *„A new customer who is registering for the bonus program, is not allowed to be a member of this program. After the registration, the customer is a member of the bonus program."*

This constraint can be formulated as a pre and post condition for the context operation **`register()`** in the class **`BonusProgram`**




```
context
  BonusProgram::register(k:Customer)
pre:  not (customer->includes(k))
post: customer = customer@pre->including(k)
```

How do we get OCL Collections?

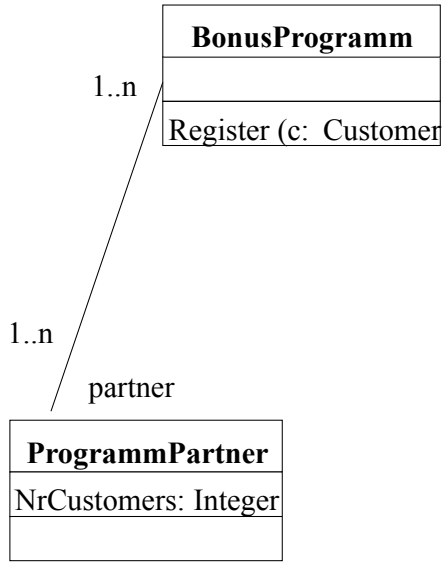
1. A collection can be generated by explicitly enumerating the elements
2. A collection can be generated by navigating along one or more 1 to many associations
   - Navigation along a single 1 to many association yields a Set
   - Navigation along a couple of 1 to many associations yields a Bag (Multiset)
   - Navigation along a single 1 to many association labeled with the constraint {ordered} yields a Sequence
3. By calling an OCL collection operation, which results in another OCL collection.

Calling an OCL Operation: „Arrow-Notation“

- The call of an collection operation consists of the concatenation of
  - the collection identifier (e.g. **`partner`**)
  - the arrow "**`->`**"
  - the name of the operation (e.g. **`size`**)
- **Example:** *„A bonus program must always have exactly 4 program partners*“

OCL Invariant (the role name **partner** is the collection identifier**):**

```
context BonusProgram inv:
partner->size = 4
```



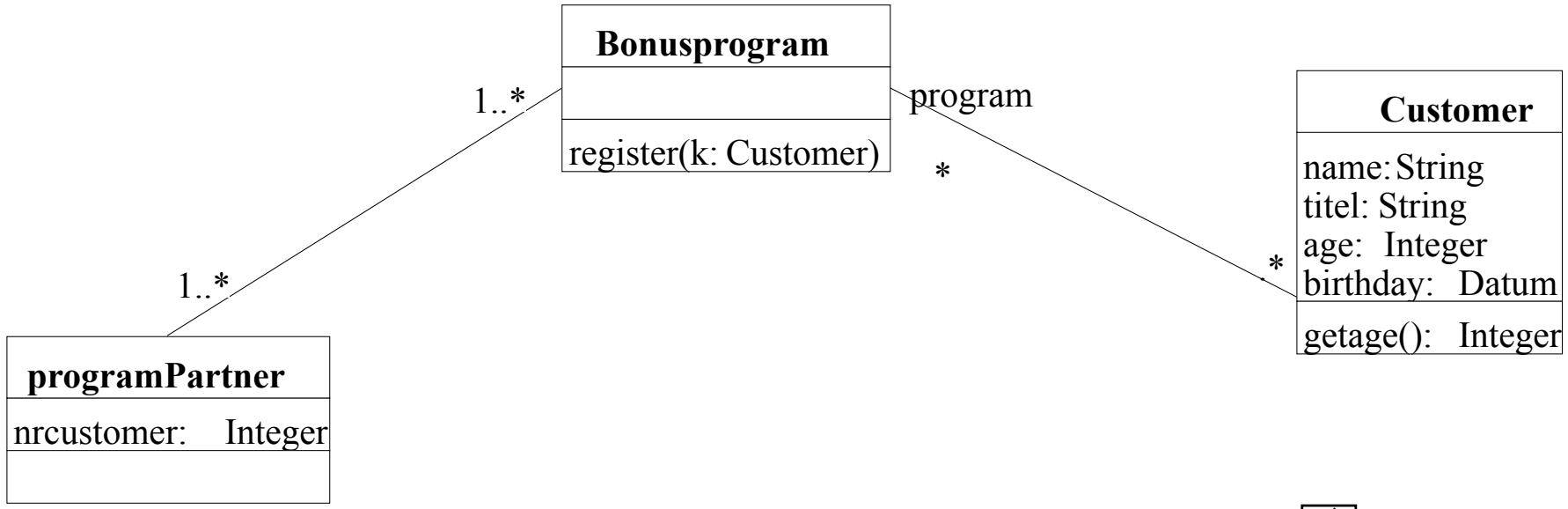
Navigation through several 1 to Many Associations

Example:

```
context programPartner inv:
nrcustomer = bonusprogram.customer->size
```

customer denotes a multiset of **Customer**

bonusprogram denotes a set of **Bonusprograms**



Conversion between OCL-Collections

- OCL offers operations to convert OCL-Collections:

`asSet`
Transforms a multiset into a set

`asBag`
transforms a set into a multiset

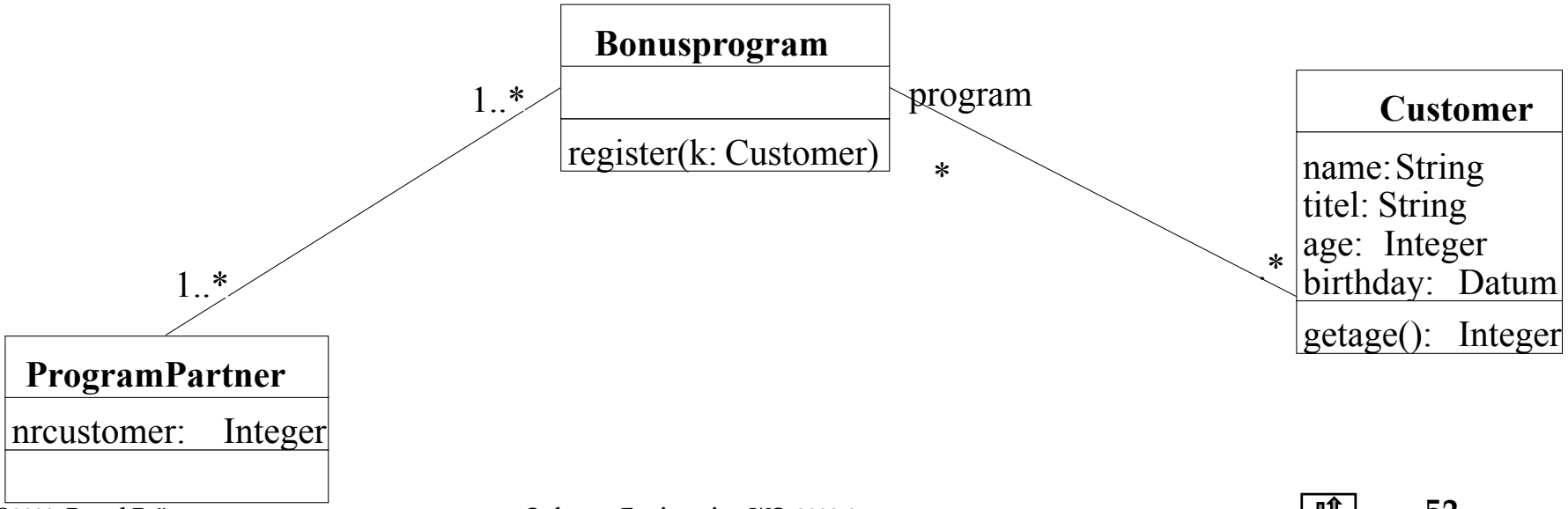
`asSequence`
transforms a set or multiset into a sequence.

Example of a Conversion

```
context ProgramPartner inv:
    nrcustomer = bonusprogram.customer->size
```

This expression may contain **Customer** multiple times, we can get the number of unique instances of **Customer** as follows:

```
context ProgramPartner inv:
    nrcustomer = bonusprogram.customer->asSet->size
```



Turning Bags into Sets

Local attribute navigation

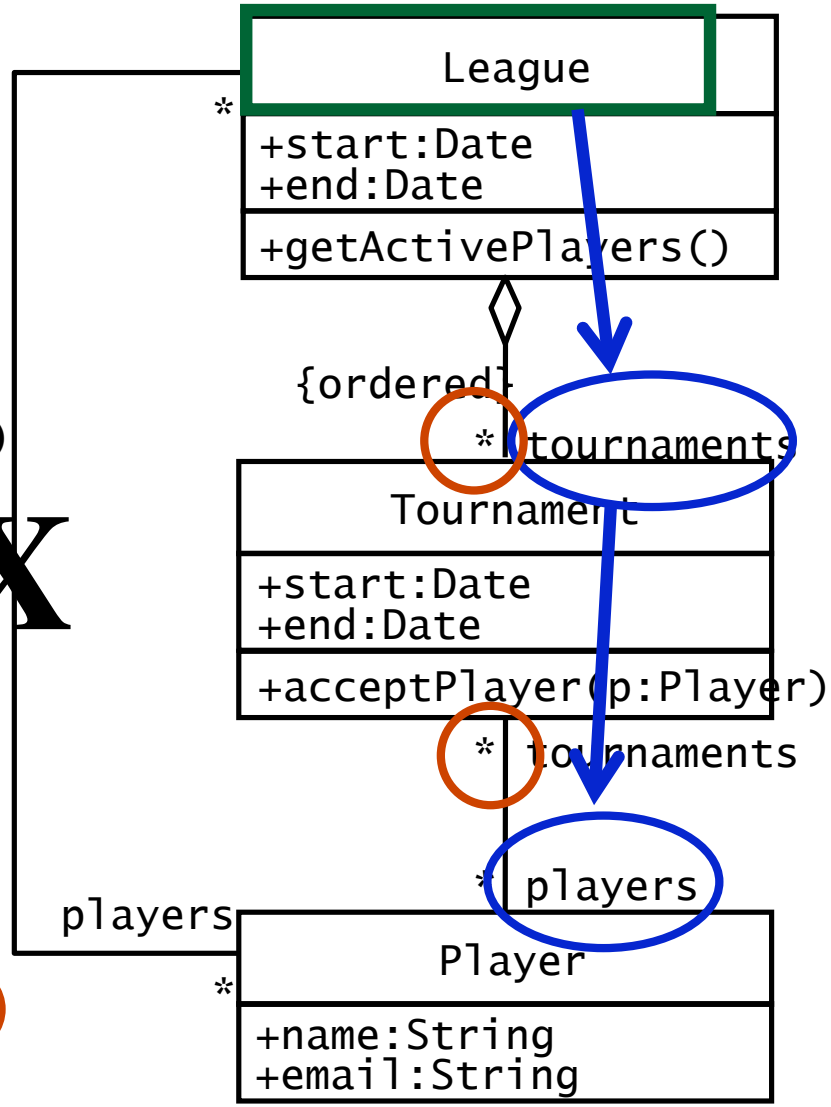
```
context Tournament inv:
 end - start <= Calendar.WEEK
```

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
```



Where place OCL Models

- **OCL model:** The set of all OCL expressions connected with a UML model
- 2 possibilities to connect a OCL model with its UML model

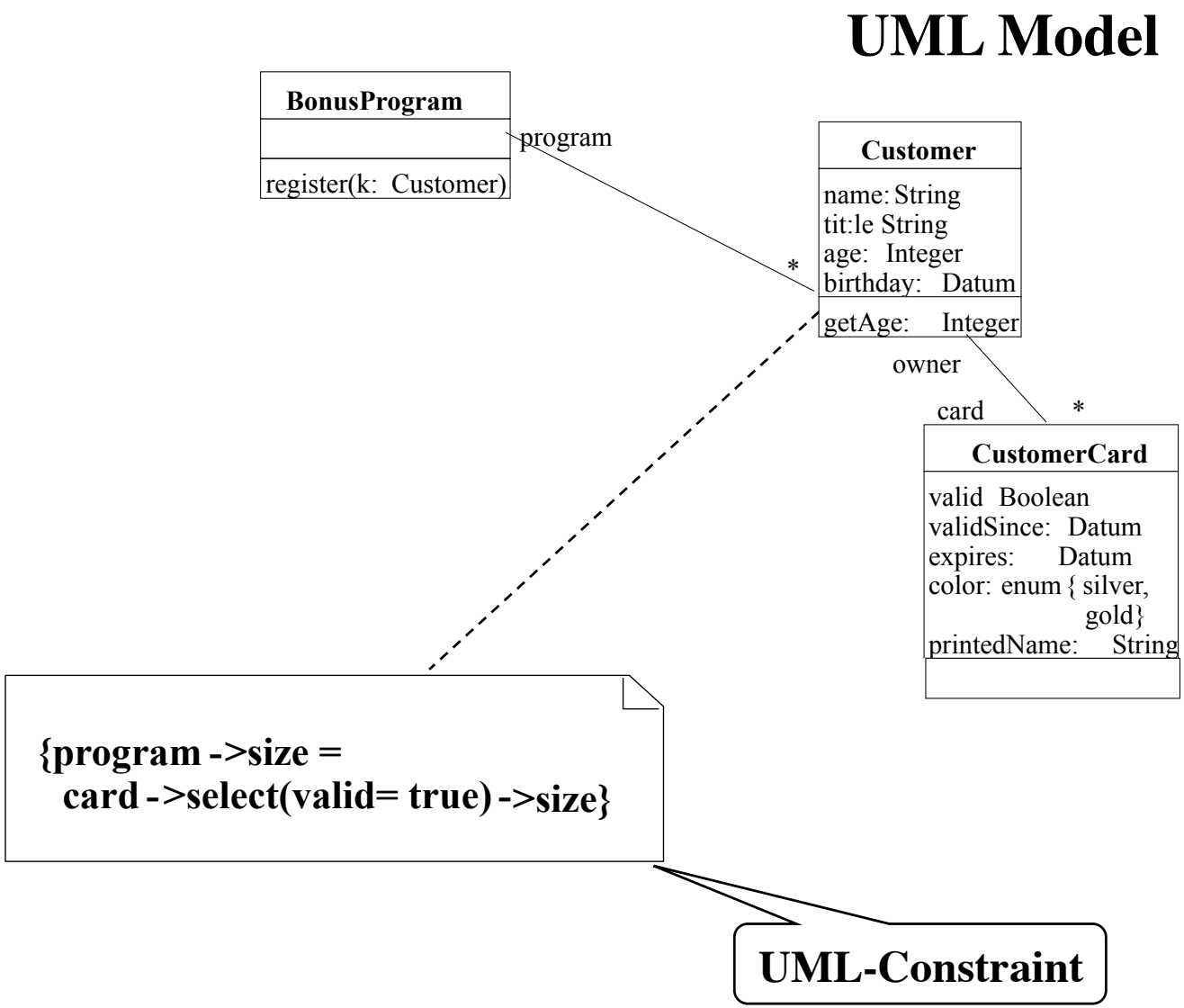
1. All OCL expressions are notes in the UML model

- *Advantage*: Everything in one model
- *Disadvantage*: The UML model becomes unreadable

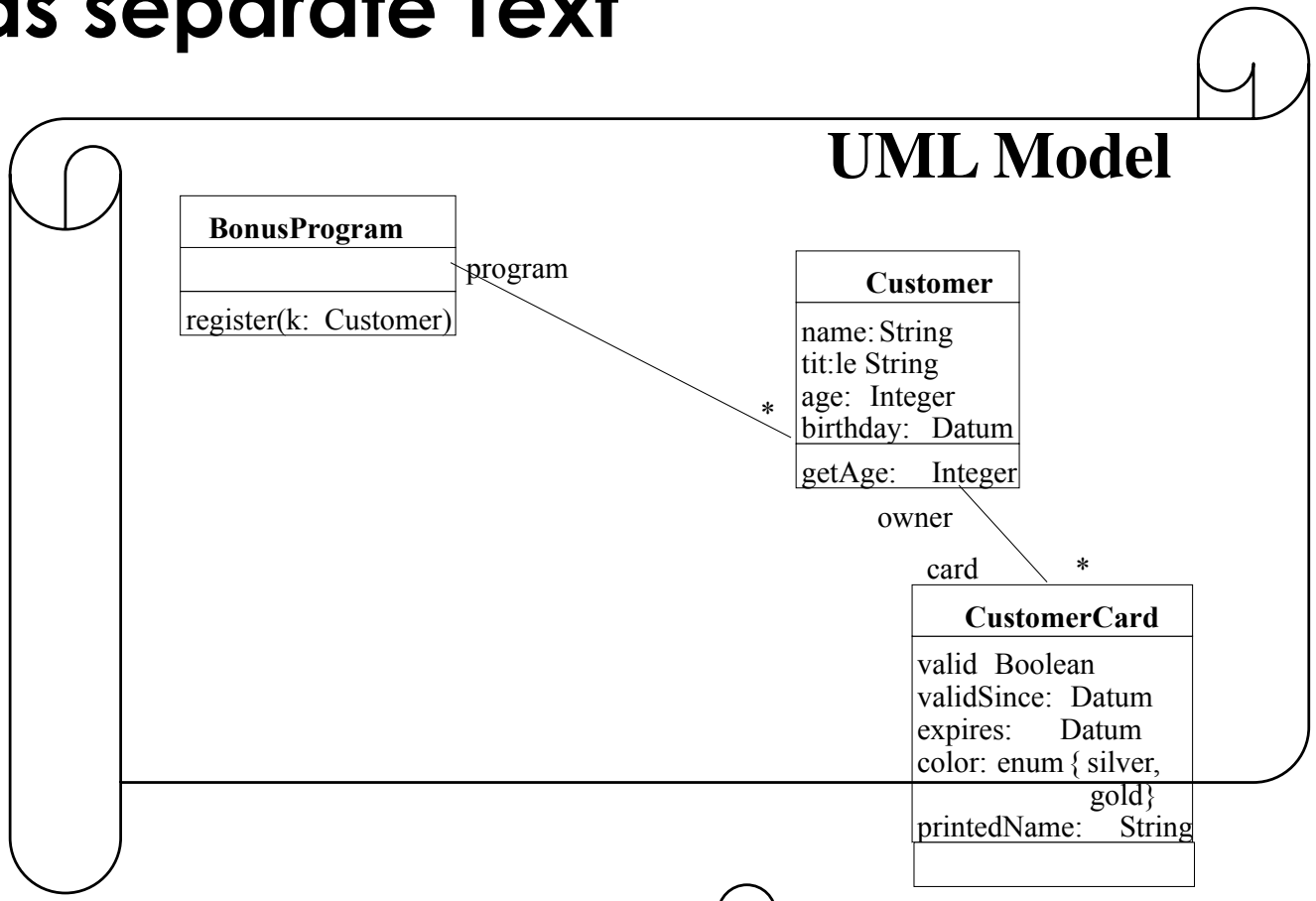
2. All OCL expressions are stored in a separate text file

- *Advantage*: The UML model stays readable
- *Disadvantage*: UML model and OCL model are stored in two different files
    - Can lead to consistency problems (Name changes in the UML model must explicitly be changed in the OCL model and vice versa).

OCL Model as part of the UML-Model



OCL Model as separate Text



OCL Model:

```
context Customer inv:
  program->size =
   card->select(valid= true)->size
```

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.
  - The explosion of Ariane was due to an error in a piece of the software that was *not needed during the crash.*
  - http://archive.eiffel.com/doc/manuals/technology/contract/ariane/page.html
- **C. A. R. Hoare**, An axiomatic basis for computer programming CACM, 12(10), pp 576-585, October 1969.
  - Refresher for Hoare logic: http://en.wikipedia.org/wiki/Hoare_logic
- **Contract4J: Design by Contract ® for Java**
  - http://www.contract4j.org/contract4j

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 constraints involving more than one class, attribute or operation can be expressed with 3 basic navigation types.