Chapter 8, Object Design: Object Constraint Language
Outline of the Lecture

- OCL
- Simple predicates
- Preconditions
- Postconditions
- Contracts
- Sets, Bags, and Sequences
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 positive number.”

Notes:
• “self” denotes all instances of “Tournament”
• OCL uses the same dot notation as Java.
**OCL Preconditions**

Example:

```plaintext
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 precondition is an operation
- `isPlayerAccepted(p)` is an operation defined by the class Tournament.
OCL Postconditions

Example:

```java
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
JavaDoc

- Add documentation comments to the source code.
- A doc comment consists of characters between /** and */
- When JavaDoc parses a doc comment, leading * characters on each line are discarded. First, blanks and tabs preceding the initial * characters are also discarded.
- Doc comments may include HTML tags
- Example of a doc comment:
  
  /**
   * This is a <b> doc </b> comment
   */
More on Java Doc

• Doc comments are only recognized when placed immediately before class, interface, constructor, method or field declarations.

• When you embed HTML tags within a doc comment, **you should not use heading tags such as `<h1>` and `<h2>`,** because JavaDoc creates an entire structured document and these structural tags interfere with the formatting of the generated document.
Java Implementation of Tournament class
(Contract as a set of JavaDoc comments)

```java
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 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

- tttExpert:League
- chessNovice:League
- Xmas:Tournament
  - start=Dec 23
  - end=Dec 25
- winter:Tournament
  - start=Jan 12
  - end=Jan 14
- alice:Player
- bob:Player
- marc:Player
- joe:Player
- zoe:Player

5 Players, 2 Tournaments
3 Types of Navigation through a Class Diagram

1. Local attribute

   | Tournament |
   | start: Date |
   | end: Date |

2. Directly related class

   | League |
   | * |
   | * |
   | * |

3. Indirectly related class

   | League |
   | * |
   | * |
   | * |

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-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 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(operands)
  ```
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 \( c_1 \)

**intersection(c2:Collection)**
- Intersection with Collection \( c_2 \) (contains only elements, which appear in the collection as well as in collection \( c_2 \) 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 \( \text{expr} \) is true.
Other examples of OCL

(optional)
OCL supports Quantification

• OCL **forall** quantifier

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

```ocm
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 */

```ocm
context Tournament inv:
matches->exists(m:Match | m.start.equals(start))
```
Backup and Additional Slides
How do we get OCL-Collections?

• A collection can be generated by explicitly enumerating the elements from the UML model
• A collection can be generated by navigating along one or more 1-N associations in the UML model
  • 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**