# *Software Design, Modelling and Analysis in UML*

# *Lecture 03: Object Constraint Language*

*2014-10-28*

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

# *Contents & Goals*

$(\{Int\}, \{C, D\}, \{x : Int\}, \{C \mapsto \{x\}, D \mapsto \emptyset\})$

**Last Lecture:**

- Basic Object System Signature $\mathscr{S}$ and Structure $\mathscr{D}$, System State $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$

*(Seems like they're related to class/object diagrams, officially we don't know yet...)*

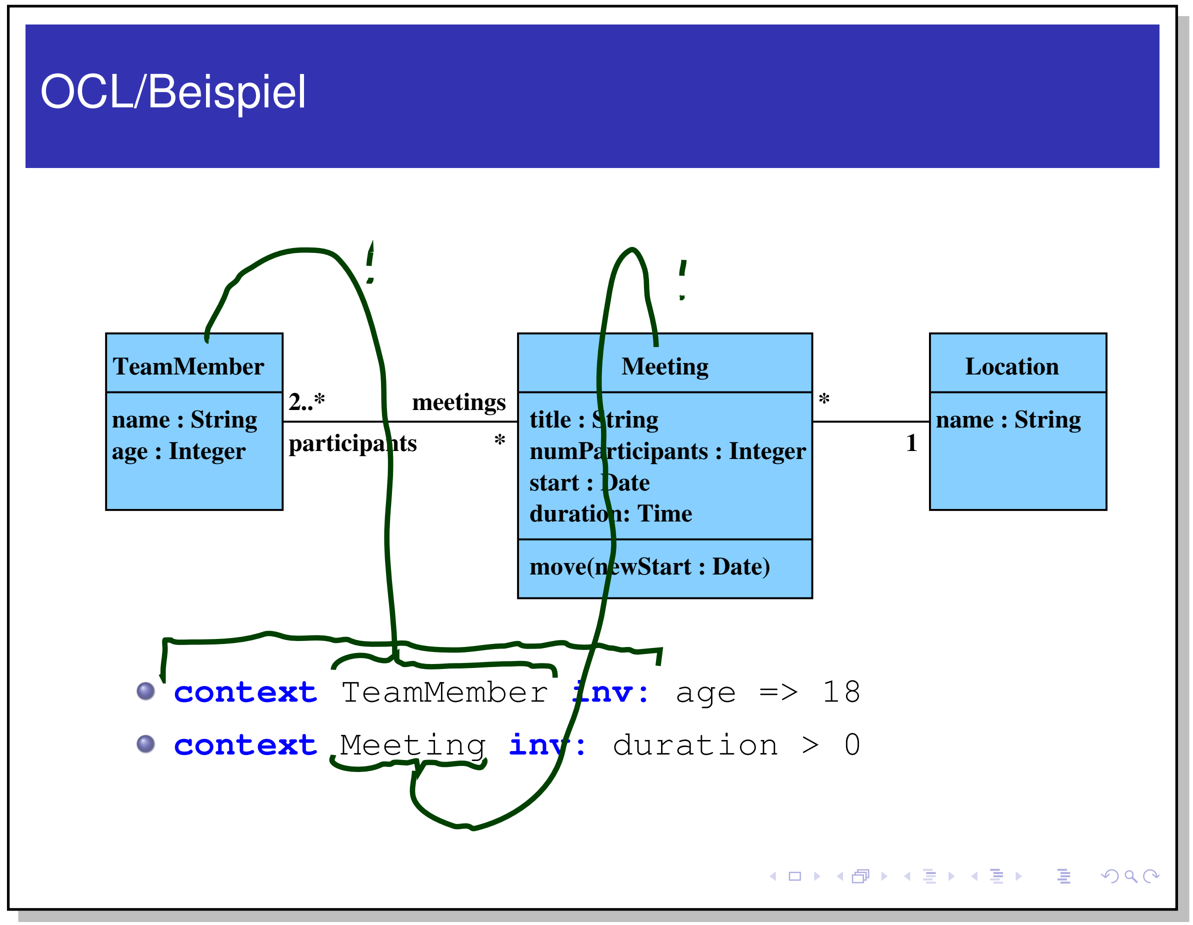
**This Lecture:**

- **Educational Objectives:** Capabilities for these tasks/questions:
  - Please explain this OCL constraint.
  - Please formalise this constraint in OCL.
  - Does this OCL constraint hold in this system state?
  - Can you think of a system state satisfying this constraint?
  - Please un-abbreviate all abbreviations in this OCL expression.
  - In what sense is OCL a three-valued logic? For what purpose?
  - How are $\mathscr{D}(C)$ and $\tau_C$ related?
- **Content:**
  - OCL Syntax, OCL Semantics over system states

*What is OCL? And What is It Good For?*

# What is OCL? How Does it Look Like?

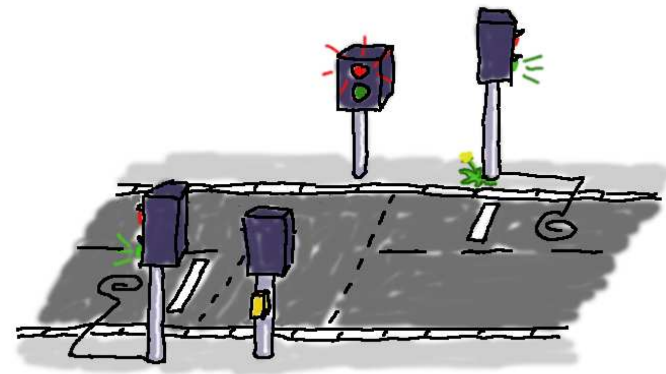
- **OCL**: Object Constraint Logic.



# *What's It Good For?*

- **Most prominent**:
  write down **requirements** supposed to be satisfied by all system states.

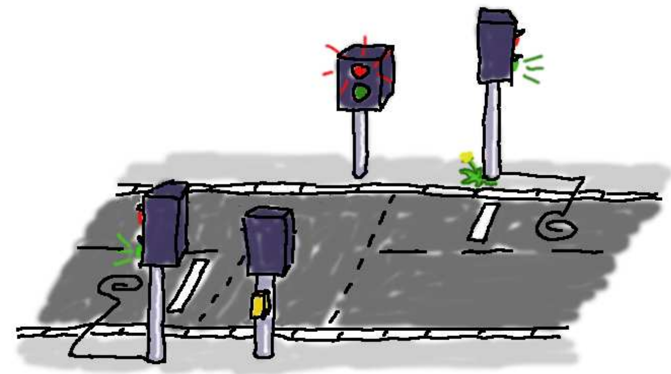
  Often targeting all alive objects of a certain class.



| TLC |
| --- |
| redP: Bool<br>greenP: Bool |
| |

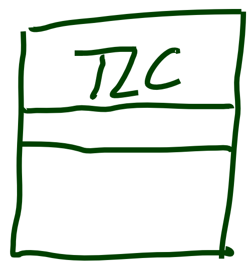
context TLC inv:
    not (redP and greenP)

# What's It Good For?



- **Most prominent**:
  write down **requirements** supposed to be satisfied by all system states.

  Often targeting all alive objects of a certain class.




context off
pre: (true)
post: (not redP
and not greenP)

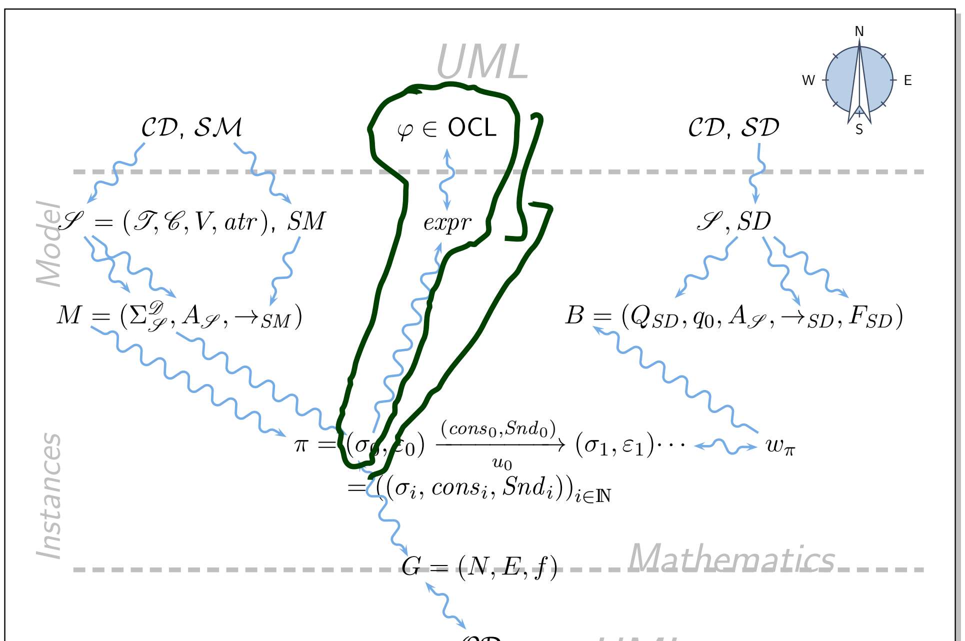
- **Not unknown**:
  write down **pre/post-conditions** of methods (*Behavioural Features*).
  Then evaluated over **two** system states.
- **Common with State Machines**:
  **guards** in transitions.

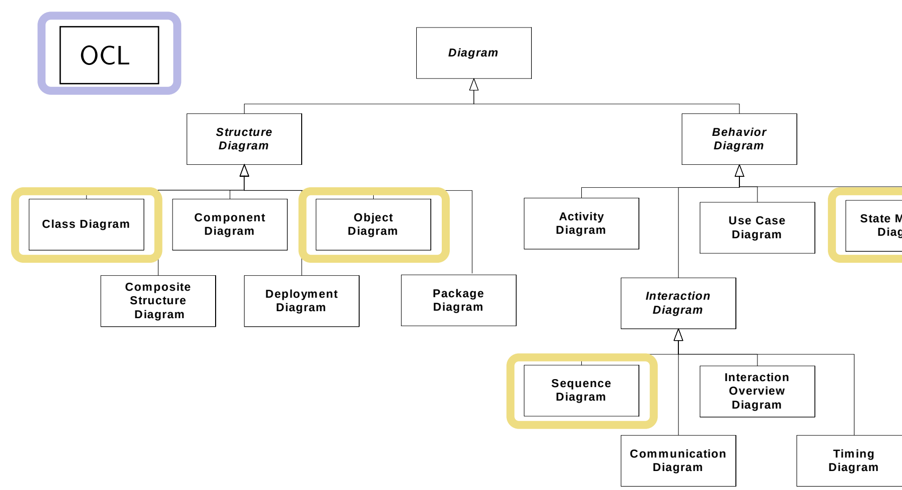

- **Lesser known**:
  provide **operation bodies**.
- **Metamodeling**: the UML standard is a MOF-Model of UML.
  OCL expressions define well-formedness of UML models (cf. Lecture $\sim$ 21).

# Plan.

- **Today**:
  - The set $OCLExpressions(\mathscr{S})$ of OCL expressions over $\mathscr{S}$.
- **Next time**:
  - Given an OCL expression $expr$, a system state $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$, and a valuation of logical variables $\beta$, define the **interpretation function**

$$I[\![expr]\!](\sigma, \beta) \in \{\mathit{true}, \mathit{false}, \bot\}.$$



# *(Core) OCL Syntax [OMG, 2006]*

# *OCL Syntax 1/4: Expressions*

$expr ::=$

| | |
|---|---|
| $w$ | $: \tau(w)$ |
| $\mid\ expr_1 =_\tau expr_2$ | $: \tau \times \tau \to Bool$ |
| $\mid$ oclIsUndefined$_\tau(expr_1)$ | $: \tau \to Bool$ |
| $\mid\ \{expr_1, \dots, expr_n\}$ | $: \tau \times \dots \times \tau \to Set(\tau)$ |
| $\mid$ isEmpty$(expr_1)$ | $: Set(\tau) \to Bool$ |
| $\mid$ size$(expr_1)$ | $: Set(\tau) \to Int$ |
| $\mid$ allInstances$_C$ | $: Set(\tau_C)$ |
| $\mid\ v(expr_1)$ | $: \tau_C \to \tau(v)$ |
| $\mid\ r_1(expr_1)$ | $: \tau_C \to \tau_D$ |
| $\mid\ r_2(expr_1)$ | $: \tau_C \to Set(\tau_D)$ |

Where, given $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$,

- $W \supseteq \{self_C \mid C \in \mathscr{C}\}$ is a set of typed logical variables, $w$ has type $\tau(w)$, $\tau(self_C) = \tau_C$
- $\tau$ is any type from $\mathscr{T} \cup T_B \cup T_\mathscr{C}$ $\cup \{Set(\tau_0) \mid \tau_0 \in T_B \cup T_\mathscr{C}$ $\cup \mathscr{T}$$\}$ from $\mathscr{S}$
  - $T_B$ is a set of basic types, in the following we use $T_B = \{Bool, Int, String\}$
  - $T_\mathscr{C} = \{\tau_C \mid C \in \mathscr{C}\}$ is the set of object types,
  - $Set(\tau_0)$ denotes the set-of-$\tau_0$ type for $\tau_0 \in T_B \cup T_\mathscr{C}$ (sufficient because of "flattening" (cf. standard))
- $\underbrace{v : \tau(v)} \in atr(C)$, $\tau(v) \in \mathscr{T}$,
- $\underbrace{r_1 : D_{0,1}} \in atr(C)$,
- $\underbrace{r_2 : D_*} \in atr(C)$,
- $C, D \in \mathscr{C}$.

# Expression Examples

$expr ::=$

| | | | |
|---|---|---|---|
| $w$ | $: \tau(w)$ | $\mid$ size$(expr_1)$ | $: Set(\tau) \to Int$ |
| | | $\mid$ allInstances$_C$ | $: Set(\tau_C)$ |
| $\mid expr_1 =_\tau expr_2$ | $: \tau \times \tau \to Bool$ | | |
| $\mid$ oclIsUndefined$_\tau(expr_1)$ | $: \tau \to Bool$ | $\mid v(expr_1)$ | $: \tau_C \to \tau(v)$ |
| $\mid \{expr_1, \dots, expr_n\}$ | $: \tau \times \dots \times \tau \to Set(\tau)$ | $\mid r_1(expr_1)$ | $: \tau_C \to \tau_D$ |
| $\mid$ isEmpty$(expr_1)$ | $: Set(\tau) \to Bool$ | $\mid r_2(expr_1)$ | $: \tau_C \to Set(\tau_D)$ |

$\mathscr{S} = (\{Int\}, \{TeamMember, Meeting\}, \{age : Int, meeting : M_{0,1}, partic : TM_*\}, \{TM \mapsto \{age, meeting\}, M \mapsto \{partic\}\})$

(short: TM) (short M)

- $self_{TM} : \tau_{TM}$
- $allInstances_M : Set(\tau_M)$
- $size(allInstances_M) : \xrightarrow{Set(\tau_M)} Int$
- $age(self_{TM}) : \tau_{TM} \to Int$
- $age(self_M)$ NO, because $age \notin atr(M)$
- $meeting(self_{TM}) : \tau_{TM} \to \tau_M$
- $partic(self_M) : \tau_M \to Set(\tau_{TM}$

# *Notational Conventions for Expressions*

- Each expression

$$\omega(expr_1, expr_2, \dots, expr_n) : \tau_1 \times \dots \times \tau_n \to \tau$$

may alternatively be written ("abbreviated as")

- $expr_1 \;.\; \omega(expr_2, \dots, expr_n)$ if $\tau_1$ is an **object type**, i.e. if $\tau_1 \in T_{\mathscr{C}}$.
- $expr_1$ -> $\omega(expr_2, \dots, expr_n)$ if $\tau_1$ is a **collection type** (here: only sets), i.e. if $\tau_1 = Set(\tau_0)$ for some $\tau_0 \in T_B \cup T_{\mathscr{C}}$.

$\mathscr{C} = \{C, D\}$, $atr(C) = \{v, r_2, r_1\}$, $atr(D) = \{w\}$

- **Examples:** $(self : \tau_C \in W; \quad v, w : Int \in V; \quad r_1 : D_{0,1}, r_2 : D_* \in V)$
  - $\underbrace{self}_{expr : \tau_C} \;.\; \underbrace{v}_{\omega} \quad\longleftrightarrow\quad \underbrace{v}_{\omega}(\underbrace{self}_{expr_1})$
  - $\underbrace{self \;.\; r_1}_{expr_1} \;.\; \underbrace{w}_{\omega} \quad\longleftrightarrow\quad w(\underbrace{self}_{expr_1}.\underbrace{r_1}_{\omega}) \quad\longleftrightarrow\quad w(r_1(self))$
  - $\underbrace{self \;.\; r_2}_{expr_1 : Set(\tau_0)}$ -> isEmpty $\quad\longleftrightarrow\quad isEmpty(r_2(self))$

# OCL Syntax 2/4: Constants & Arithmetics

**For example**:

$$
\begin{array}{rll}
expr ::= & \dots & \\
& \mid \mathsf{true} \mid \mathsf{false} & : Bool \\
& \mid expr_1 \ \{\mathsf{and}, \mathsf{or}, \mathsf{implies}\} \ expr_2 & : Bool \times Bool \to Bool \\
& \mid \mathsf{not} \ expr_1 & : Bool \to Bool \\
& \mid 0 \mid -1 \mid 1 \mid -2 \mid 2 \mid \dots & : Int \\
& \mid \mathsf{OclUndefined}_\tau & : \tau \\
& \mid expr_1 \ \{+, -, \dots\} \ expr_2 & : Int \times Int \to Int \\
& \mid expr_1 \ \{<, \leq, \dots\} \ expr_2 & : Int \times Int \to Bool
\end{array}
$$

Generalised notation:

$$expr ::= \omega(expr_1, \dots, expr_n) \qquad : \tau_1 \times \dots \times \tau_n \to \tau$$

with $\omega \in \{+, -, \dots\}$

e.g. $+(expr_1, expr_2)$
for
$expr_1 + expr_2$

# OCL Syntax 3/4: Iterate

$$expr ::= \cdots \mid expr_1\text{->iterate}(w_1 : \tau_1 ; w_2 : \tau_2 = expr_2 \mid expr_3)$$

or, with a little renaming,

$$expr ::= \cdots \mid expr_1\text{->iterate}(iter : \tau_1 ; result : \tau_2 = expr_2 \mid expr_3)$$

where

- $expr_1$ is of a **collection type** (here: a set $Set(\tau_0)$ for some $\tau_0$),
- $iter \in W$ is called **iterator**, gets type $\tau_1$
  (if $\tau_1$ is omitted, $\tau_0$ is assumed as type of $iter$)
- $result \in W$ is called **result variable**, gets type $\tau_2$,
- $expr_2$ in an expression of type $\tau_2$ giving the **initial value** for $result$,
  ('OclUndefined' if omitted)
- $expr_3$ is an expression of type $\tau_2$
  in which in particular $iter$ and $result$ may appear.

# Iterate: Intuitive Semantics (Formally: later)

$$expr ::= expr_1\text{->iterate}(iter : \tau_1;$$
$$result : \tau_2 = expr_2 \mid expr_3)$$

pseudocode

$Set(\tau_0)\ hlp = \langle expr_1 \rangle;$

$\tau_1\ iter;$

$\tau_2\ result = \langle expr_2 \rangle;$

$while\ (!hlp.empty())\ do$

$\quad iter = hlp.pop();$

$\quad result = \langle expr_3 \rangle;$

$od$

pick one element and remove

allInstances$_{TM}$ ->iterate( iter : $\tau_{TM}$ ; result : Bool = true | result and iter.age ≥ 18)

# Iterate: Intuitive Semantics (Formally: later)

$$expr ::= expr_1\text{->iterate}(iter : \tau_1; \quad result : \tau_2 = expr_2 \mid expr_3)$$

$$
\begin{array}{l}
Set(\tau_0)\ hlp = \langle expr_1 \rangle; \\
\tau_1\ iter; \\
\tau_2\ result = \langle expr_2 \rangle; \\
while\ (!hlp.empty())\ do \\
\quad iter = hlp.pop(); \\
\quad result = \langle expr_3 \rangle; \\
od
\end{array}
$$

**Note**: In our (simplified) setting, we always have $expr_1 : Set(\tau_1)$ and $\tau_0 = \tau_1$.
In the type hierarchy of full OCL with inheritance and `oclAny`, they may be different and still type consistent.

# Abbreviations on Top of Iterate

$$expr ::= expr_1\text{->iterate}(w_1 : \tau_1;\ w_2 : \tau_2 = expr_2 \mid expr_3)$$

- $$expr_1\text{->forAll}(w[: \tau_1] \mid expr_3)$$ is an abbreviation for

  e.g. allInstances$_{TA}$ ->forAll( i | i.age ≥ 18 )

  $$expr_1\text{->iterate}(w{:}\ \tau_1;\ w_1 : Bool = true \mid w_1\, and\, expr_3).$$

  (To ensure confusion, we may again omit all kinds of things, cf. [OMG, 2006]).

- Similar: $$expr_1\text{->Exists}(w : \tau_1 \mid expr_3)$$

# OCL Syntax 4/4: Context

$$context ::= \text{context } w_1 : \tau_1, \dots, w_n : \tau_n \text{ inv : } expr$$

where $w \in W$ and $\tau_i \in T_{\mathscr{C}}$, $1 \leq i \leq n$, $n \geq 0$.

$$\text{context } w_1 : C_1, \dots, w_n : C_n \text{ inv : } expr$$

is an **abbreviation** for

$$
\begin{array}{l}
\text{allInstances}_{C_1} \text{ -> forAll}(w_1 : C_1 \mid \\
\quad \dots \\
\qquad \text{allInstances}_{C_n} \text{ -> forAll}(w_n : C_n \mid \\
\qquad\quad expr \\
\qquad ) \\
\quad \dots \\
)
\end{array}
$$

context TM, M inv:
$self_M$ -> partic
-> contains ($self_{TM}$)
implies
$self_{TM}$ . meeting
= $self_M$

context TM inv:
age >= 18

$\Updownarrow$

allInstances$_{TM}$
-> forAll ( $self_{TM}$ |
$self_{TM}$ . age >= 18)

# Context: More Notational Conventions

- For

$$\text{context } \mathit{self} : \tau_C \ \text{ inv} : \mathit{expr}$$

we may alternatively write ("abbreviate as")

$$\text{context } \tau_C \ \text{ inv} : \mathit{expr}$$

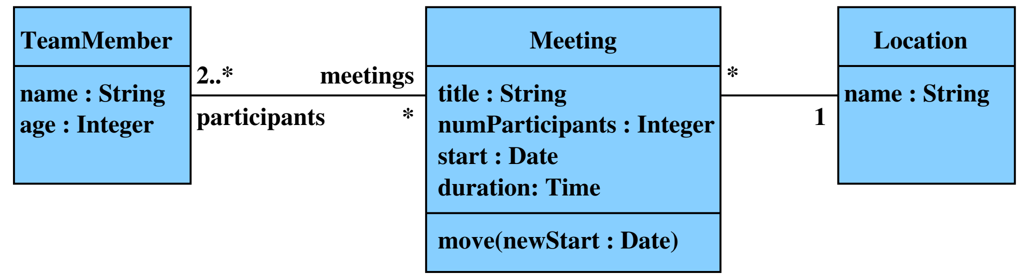
- **Within** the latter abbreviation, we may omit the "$\mathit{self}$" in $\mathit{expr}$, i.e. for

$$\mathit{self}.v \quad \text{and} \quad \mathit{self}.r$$

we may alternatively write ("abbreviate as")

$$v \quad \text{and} \quad r$$

# Examples (from lecture...




- **context** TeamMember **inv:** age => 18
- **context** Meeting **inv:** duration > 0

Unabbreviate

context $self_{TM}$ : TeamMember inv: $self_{TM}$ . age >= 18

allInstances$_{TM}$ -> forAll( $self_{TM}$ : TM | $self_{TM}$ . age ≥ 18)

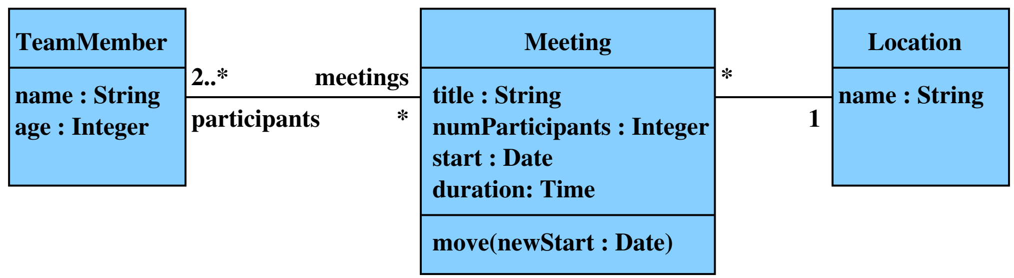
allInstances$_{TM}$ -> iterate( $self_{TM}$ : TM; res : Bool = true | res and $self_{TM}$ . age ≥ 18)

normalize

allInstances$_{TM}$ -> iterate( $self_{TM}$ : TM, res : Bool = true |
and ( res, ≥ ( age ( $self_{TM}$ ), 18 ))

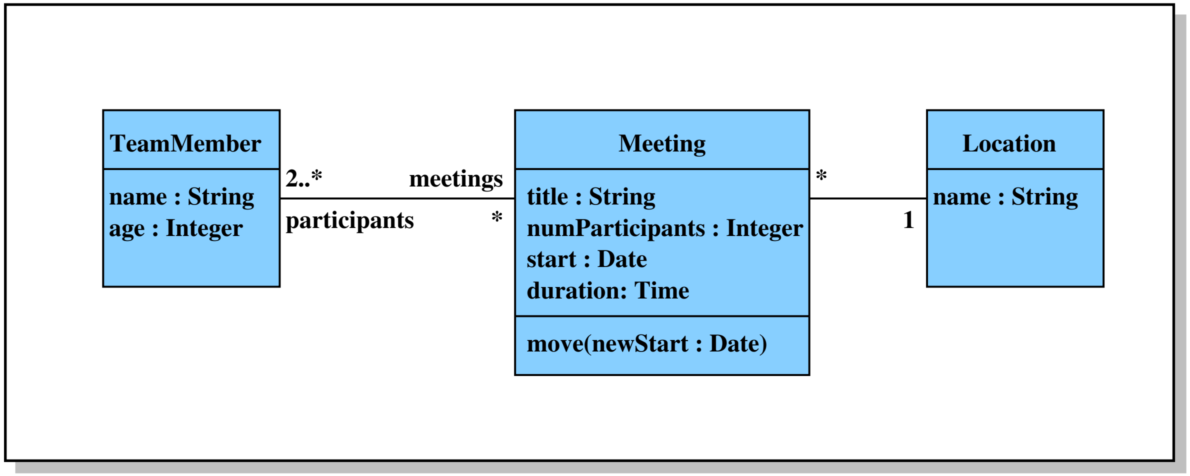
# Examples (from lecture “Softwaretechnik 2008”)

## OCL/Mehr Navigation/Beispiele




- **context** Meeting
  - **inv:** `self.participants->size() = numParticipants`
- **context** Location
  - **inv:** `name="Lobby"` **implies** `meeting->isEmpty()`

# Example (from lecture “Softwaretechnik 2008”)




- context $Meeting$ inv :

$$participants \text{ -> } \mathsf{iterate}(i : TeamMember; n : Int = 0 \mid n + i\,.\,age)$$

$$/ participants \text{ -> } \mathsf{size}() > 25$$

# “Not Interesting”

Among others:

- Enumeration types
- Type hierarchy
- Complete list of arithmetical operators
- The two other collection types Bag and Sequence
- Casting
- Runtime type information
- Pre/post conditions
  (maybe later, when we officially know what an operation is)
- ...

# OCL Semantics: The Task

$$
\begin{array}{lll|lll}
\multicolumn{3}{l}{expr ::=} & & & \\
 & w & : \tau(w) & & \mathsf{size}(expr_1) & : Set(\tau) \to Int \\
 & & & & \mathsf{allInstances}_C & : Set(\tau_C) \\
| & expr_1 =_\tau expr_2 & : \tau \times \tau \to Bool & & & \\
| & \mathsf{oclIsUndefined}_\tau(expr_1) & : \tau \to Bool & & v(expr_1) & : \tau_C \to \tau(v) \\
 & & & & r_1(expr_1) & : \tau_C \to \tau_D \\
| & \{expr_1, \dots, expr_n\} & : \tau \times \dots \times \tau \to Set(\tau) & & r_2(expr_1) & : \tau_C \to Set(\tau_D) \\
| & \mathsf{isEmpty}(expr_1) & : Set(\tau) \to Bool & & &
\end{array}
$$

- Given an OCL expression $expr$, a system state $\sigma \in \Sigma_{\mathscr{S}}^{\mathscr{D}}$, and a valuation of logical variables $\beta$, define

$$I[\![\cdot]\!](\cdot\,,\cdot) : OCLExpressions(\mathscr{S}) \times \Sigma_{\mathscr{S}}^{\mathscr{D}} \times (W \to I(\mathscr{T} \cup T_B \cup T_{\mathscr{C}})) \to I(Bool)$$

i.e.

$\sigma = \{1_{TM} \mapsto \{age = 27,$
$meeting = 5_M\}\}$

$$I[\![expr]\!](\sigma, \beta) \in \{\mathit{true}, \mathit{false}, \bot_{Bool}\}.$$

$\models^?$ self.age $\geq 18$, $\beta$ : self $\mapsto 1_{TM}$

# *References*

[OMG, 2006] OMG (2006). Object Constraint Language, version 2.0. Technical Report formal/06-05-01.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.

[Warmer and Kleppe, 1999] Warmer, J. and Kleppe, A. (1999). *The Object Constraint Language*. Addison-Wesley.