Softwaretechnik / Software-Engineering

Lecture 12: Structural Software Modelling

2015-06-25

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents of the Block "Design"

(i) **Introduction and Vocabulary**

(ii) **Principles of Design**

  a) modularity
  b) separation of concerns
  c) information hiding and data encapsulation
  d) abstract data types, object orientation

(iii) **Software Modelling**

  a) views and viewpoints, the 4+1 view
  b) model-driven/based software engineering
  c) Unified Modelling Language (UML)
  d) **modelling structure**
    1. (simplified) class diagrams
    2. (simplified) object diagrams
    3. (simplified) object constraint logic (OCL)
  e) **modelling behaviour**
    1. communicating finite automata
    2. Uppaal query language
    3. basic state-machines
    4. an outlook on hierarchical state-machines

(iv) **Design Patterns**

Introduction	L 1:	20.4., Mo
	T 1:	23.4., Do
Development Process, Metrics	L 2:	27.4., Mo
	L 3:	30.4., Do
	L 4:	4.5., Mo
	T 2:	7.5., Do
	L 5:	11.5., Mo
Requirements Engineering	-	14.5., Do
	L 6:	18.5., Mo
	L 7:	21.5., Do
	-	25.5., Mo
	-	28.5., Do
	T 3:	1.6., Mo
	-	4.6., Do
	L 8:	8.6., Mo
	L 9:	11.6., Do
	L 10:	15.6., Mo
	T 4:	18.6., Do
Architecture & Design, Software Modelling	L 11:	22.6., Mo
	L 12:	25.6., Do
	L 13:	29.6., Mo
	L 14:	2.7., Do
	T 5:	6.7., Mo
Quality Assurance	L 15:	9.7., Do
	L 16:	13.7., Mo
Invited Talks	L 17:	16.7., Do
	T 6:	20.7., Mo
Wrap-Up	L 18:	23.7., Do

Contents & Goals

Last Lecture:

- Design basics and vocabulary:
  modularity, separation of concerns, information hiding, data encapsulation, ADT, ...

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is the signature defined by this class diagram?
  - Give a system state corresponding to this class diagram.
  - Which system state is denoted by this object diagram?
  - To which value does this Proto-OCL formula evaluate on the given system state?
  - Give system states such that the given formula evaluates to true/false/$\perp$.
  - Why is Proto-OCL a 3-valued logic?

- **Content:**
  - Class Diagrams
  - Object Diagrams
  - Proto-OCL

Class Diagrams

Object System Signature

Definition. An **(Object System) Signature** is a 6-tuple

$$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, F, mth)$$

where

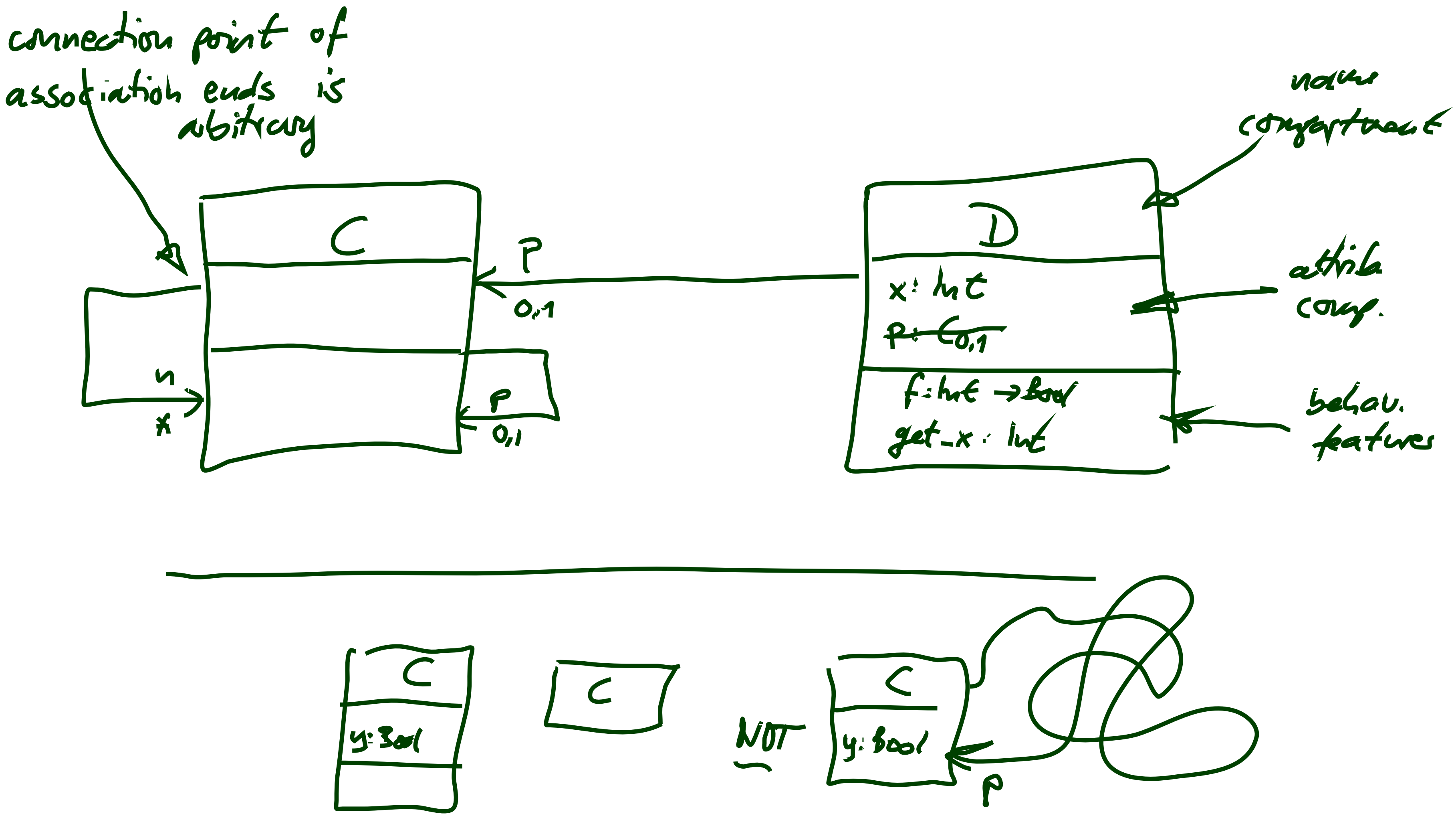
- $\mathscr{T}$ is a set of (basic) **types**,
- $\mathscr{C}$ is a finite set of **classes**,
- ~~V is a finite set of typed **attributes**, i.e., each $v \in V$ has type~~
- V is a finite set of typed **attributes** $v : T$, i.e., each $v \in V$ has type T,
- $atr : \mathscr{C} \to 2^V$ maps each class to its set of attributes.
- F is a finite set of typed **behavioural features** $f : T_1, \ldots, T_n \to T$,
- $mth : \mathscr{C} \to 2^F$ maps each class to its set of behavioural features.
- A type can be a basic type $\tau \in \mathscr{T}$, or $C_{0,1}$, or C_*, where $C \in \mathscr{C}$.

Note: Inspired by OCL 2.0 standard OMG (2006), Annex A.

Object System Signature Example

$$\mathscr{S}_0 = (\overbrace{\{Int\}}^{\text{Bool},}, \overset{\mathscr{T}}{\{C, D\}}, \overset{\mathscr{C}}{\{x : Int, p : C_{0,1}, n : C_*\}}, \overset{V}{\{C \mapsto \{p, n\}, D \mapsto \{p, x\}\}}, \overset{atr}{}$$

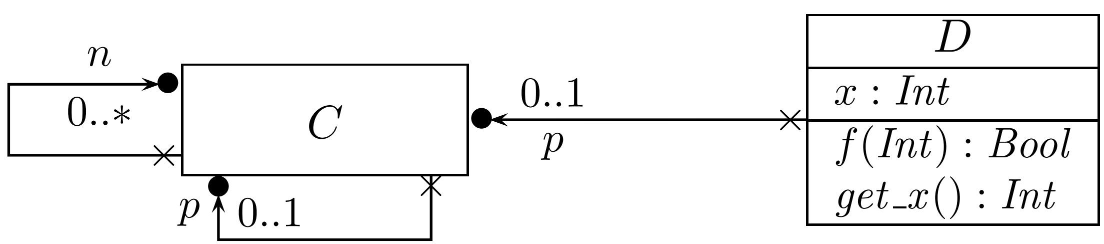
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})$$



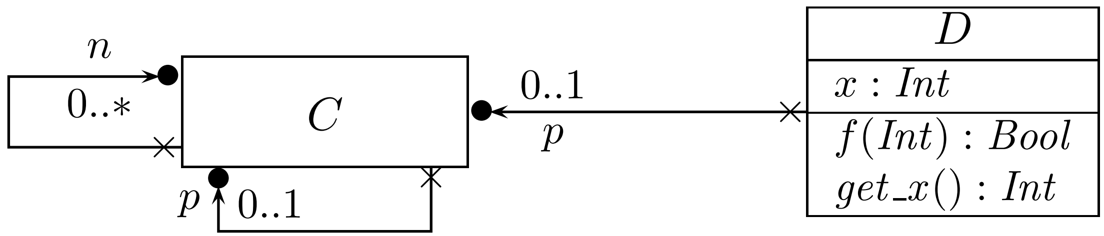
Object System Signature Example

$$\mathscr{S}_0 = (\overbrace{\{Int\}}^{Bool}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$

$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})$$

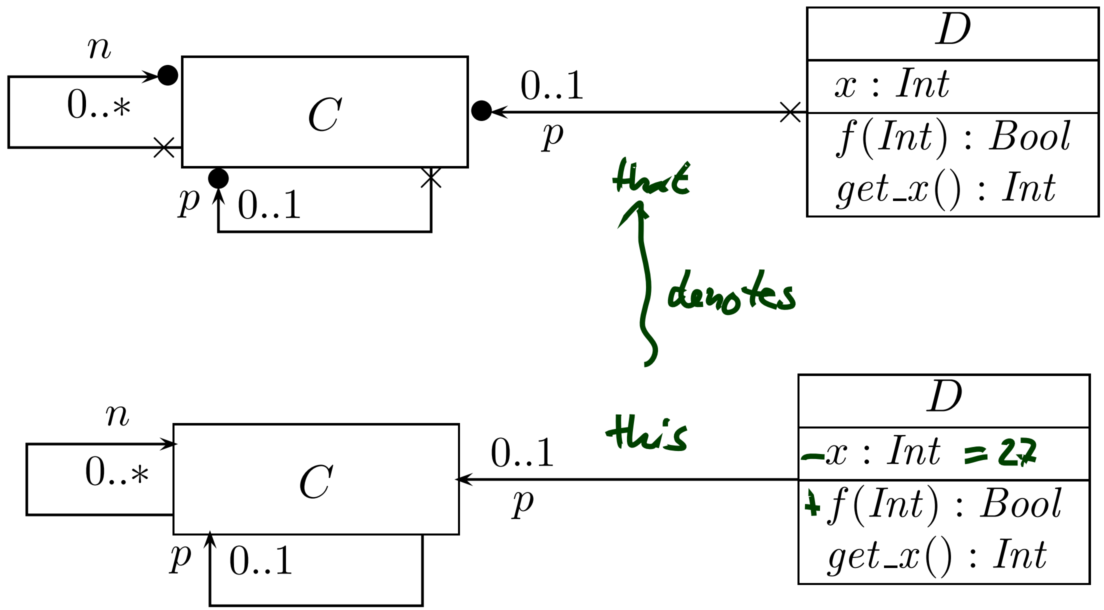


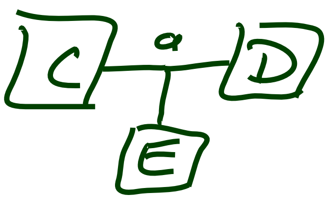
And The Other Way Round



$$\mathcal{S} = \Big(\{Int, Bool\}, \{C, D\}, \{p : C_{0,1}, n : C_{*}, x : Int\},$$
$$\{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\},$$
$$\{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\} \Big)$$

Shorthand Notation





In particular:

- **visibility** for attributes and association ends ($+$, $-$, $\#$, $\sim$): **later**
- **initial values, properties**: **not here**, cf. UML lecture
- **associations in general** (names, reading direction, ternary; visibility, navigability, etc. of association ends): **not here**, cf. UML lecture
- **inheritance**: **later** (maybe)
- **behavioural features**: **not here**, cf. UML lecture

Object System Structure

Definition. A Object System **Structure** of signature

$$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, F, mth)$$

is a domain function $\mathscr{D}$ which assigns to each type a domain, i.e.

- $\tau \in \mathscr{T}$ is mapped to $\mathscr{D}(\overset{\tau}{\cancel{T}})$,
- $C \in \mathscr{C}$ is mapped to an infinite set $\mathscr{D}(C)$ of **(object) identities**.
  - object identities of different classes are disjoint, i.e.
    $\forall\, C, D \in \mathscr{C} : C \neq D \rightarrow \mathscr{D}(C) \cap \mathscr{D}(D) = \emptyset$,
  - on object identities, (only) comparision for equality "=" is defined.
- C_* **and** $C_{0,1}$ for $C \in \mathscr{C}$ are mapped to $2^{\mathscr{D}(C)}$. ← powerset of $\mathscr{D}(C)$

We use $\mathscr{D}(\mathscr{C})$ to denote $\bigcup_{C \in \mathscr{C}} \mathscr{D}(C)$; analogously $\mathscr{D}(\mathscr{C}_*)$.

Note: We identify objects and object identities, because both uniquely determine each other (cf. OCL 2.0 standard).

Basic Object System Structure Example

Wanted: a structure for signature

$$\mathscr{S}_0 = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})$$

A structure $\mathscr{D}$ maps

- $\tau \in \mathscr{T}$ to **some** $\mathscr{D}(\tau)$, $C \in \mathscr{C}$ to **some** identities $\mathscr{D}(C)$ (infinite, pairwise disjoint),
- C_* and $C_{0,1}$ for $C \in \mathscr{C}$ to $\mathscr{D}(C_{0,1}) = \mathscr{D}(C_*) = 2^{\mathscr{D}(C)}$.

$$\begin{aligned}
\mathscr{D}(Int) &= \mathbb{Z} & &= \{-127, \dots, 127\} \\
\mathscr{D}(C) &= \mathbb{N}^+ \times \{C\} = \{1_C, 2_C, 3_C, \dots\} & &= \{1, 3, 5, \dots\} \\
\mathscr{D}(D) &= \mathbb{N}^+ \times \{D\} & &= \{2, 4, 6, \dots\} \\
\mathscr{D}(C_{0,1}) = \mathscr{D}(C_*) &= 2^{\mathscr{D}(C)} & &= 2^{\mathscr{D}(C)} \\
\mathscr{D}(D_{0,1}) = \mathscr{D}(D_*) &= 2^{\mathscr{D}(C)} & &= 2^{\mathscr{D}(D)}
\end{aligned}$$

System State

Definition. Let $\mathscr{D}$ be a structure of $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, F, mth)$.
A **system state** of $\mathscr{S}$ wrt. $\mathscr{D}$ is a **type-consistent** mapping

$$\sigma : \underset{\text{all obj. Ids}}{\mathscr{D}(\mathscr{C})} \overset{\text{partial function}}{\nrightarrow} (\underbrace{V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*))}_{\text{(each) attribute is assigned a value}}).$$

That is, for each $u \in \mathscr{D}(C)$, $C \in \mathscr{C}$, if $u \in \mathrm{dom}(\sigma)$

- $\mathrm{dom}(\underbrace{\sigma(u)}_{: V \nrightarrow \mathscr{D}(\cdot)}) = atr(C)$
- $\sigma(u)(v) \in \mathscr{D}(\tau)$ if $\underline{v : \tau}, \tau \in \mathscr{T}$
- $\sigma(u)(v) \in \mathscr{D}(D_*)$ if $v : D_{0,1}$ or $v : D_*$ with $D \in \mathscr{C}$

We call $u \in \mathscr{D}(\mathscr{C})$ **alive** in σ if and only if $u \in \mathrm{dom}(\sigma)$.

We use $\Sigma_{\mathscr{S}}^{\mathscr{D}}$ to denote the set of all system states of $\mathscr{S}$ wrt. $\mathscr{D}$.

System State Example

$$\mathscr{S}_0 = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})$$

$$\mathscr{D}(Int) = \mathbb{Z}, \quad \mathscr{D}(C) = \{1_C, 2_C, 3_C, ...\}, \quad \mathscr{D}(D) = \{1_D, 2_D, 3_D, ...\}$$

A system state is a partial function $\sigma : \mathscr{D}(\mathscr{C}) \nrightarrow (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$ such that

- $\mathrm{dom}(\sigma(u)) = atr(C)$,
- $\sigma(u)(v) \in \mathscr{D}(\tau)$ if $v : \tau, \tau \in \mathscr{T}$,
- $\sigma(u)(v) \in \mathscr{D}(C_*)$ if $v : D_*$ or $v : D_{0,1}$ with $D \in \mathscr{C}$.

?

$$\sigma = \{1_C \mapsto \{p \mapsto \{5_C\}, n \mapsto \{1_C, 5_C, 6_C\}\},$$
$$3_D \mapsto \{x \mapsto 27, p \mapsto \{1_C\}\},$$
$$5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}\}$$

$\mathrm{dom}(\sigma) = \{1_C, 3_D, 5_C\}$

alive in σ: $1_C, 3_D, 5_C$

System State Example

$$\mathscr{S}_0 = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\})$$

$$\mathscr{D}(Int) = \mathbb{Z}, \quad \mathscr{D}(C) = \{1_C, 2_C, 3_C, ...\}, \quad \mathscr{D}(D) = \{1_D, 2_D, 3_D, ...\}$$

A system state is a partial function $\sigma : \mathscr{D}(\mathscr{C}) \nrightarrow (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$ such that

- $\mathrm{dom}(\sigma(u)) = atr(C)$,
- $\sigma(u)(v) \in \mathscr{D}(\tau)$ if $v : \tau, \tau \in \mathscr{T}$,
- $\sigma(u)(v) \in \mathscr{D}(C_*)$ if $v : D_*$ or $v : D_{0,1}$ with $D \in \mathscr{C}$.

- **Concrete**, **explicit** system state:

$$\sigma_1 = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{p \mapsto \{5_C\}, x \mapsto 23\}\}..$$

- **Alternative**: **symbolic** system state

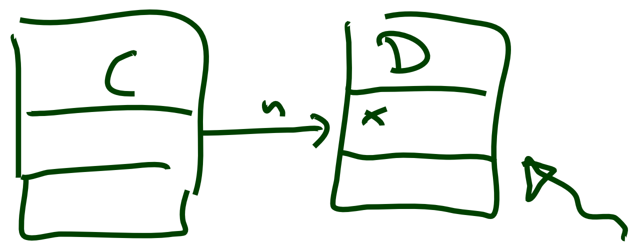
$$\sigma_2 = \{c_1 \mapsto \{p \mapsto \emptyset, n \mapsto \{c_2\}\}, c_2 \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, d \mapsto \{p \mapsto \{c_2\}, x \mapsto 23\}\}.$$

assuming $c_1, c_2 \in \mathscr{D}(C)$, $d \in \mathscr{D}(D)$, $c_1 \neq c_2$.
Can be seen as denoting **a set of** system states including σ_1 — how many?

Class Diagrams at Work

Visualisation of Implementation

- The class diagram syntax can be used to **visualise code**: **provide rules** which map (parts of) the code to class diagram elements.



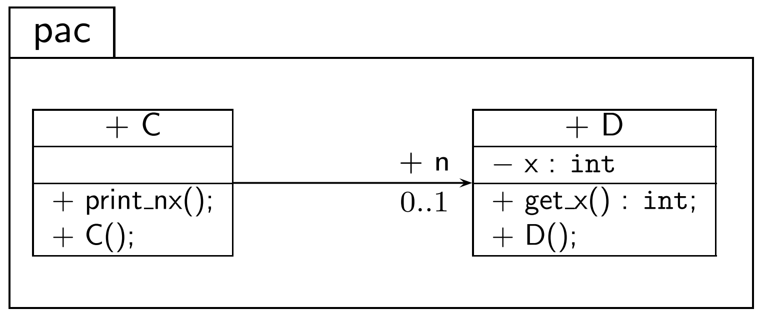
$(\mathcal{J}, \{C, D\}, \{x : int, \ldots\}, \{D \mapsto \{x, \ldots\}, \ldots)$

```java
package pac;

import pac.D;

public class C {

  public D n;

  public void print_nx() {
    System.out.printf(
      "%i\n", n.get_x() ); };

  public C() {};
}
```

```java
package pac;

import pac.C;

public class D {

  private int x;

  public int get_x()
    { return x; };

  public D() {};
}
```

Visualisation of Implementation

- The class diagram syntax can be used to **visualise code**: **provide rules** which map (parts of) the code to class diagram elements.

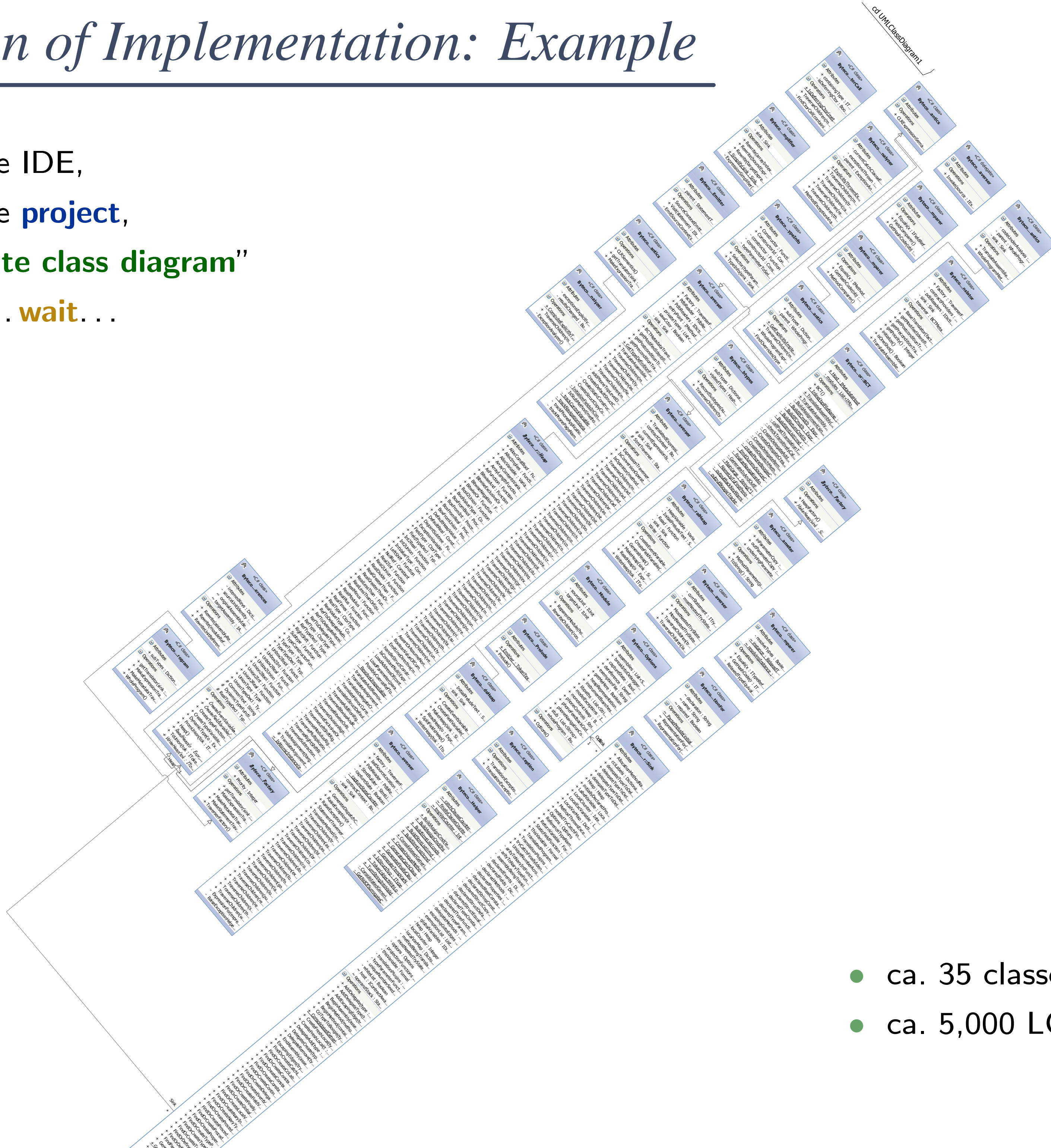



```java
package pac;

import pac.D;

public class C {

  public D n;

  public void print_nx() {
    System.out.printf(
      "%i\n", n.get_x() ); };

  public C() {};
}
```

```java
package pac;

import pac.C;

public class D {

  private int x;

  public int get_x()
    { return x; };

  public D() {};
}
```

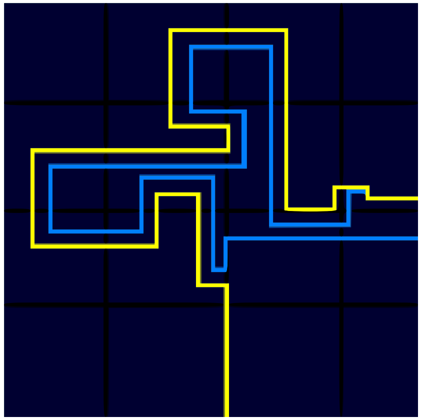
Visualisation of Implementation: Example

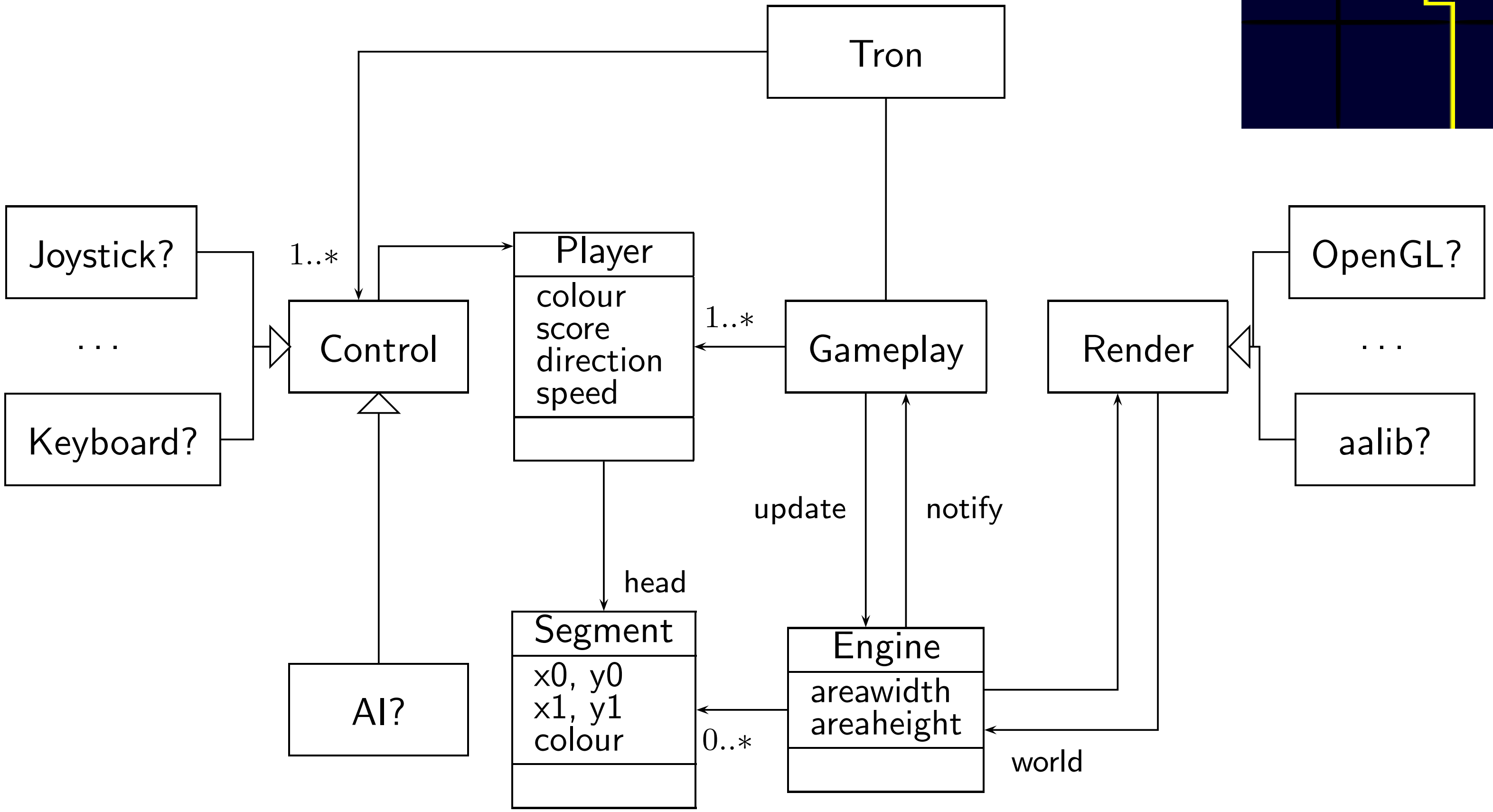
- open favourite IDE,
- open favourite **project**,
- press "**generate class diagram**"
- **wait**. . . **wait**. . . **wait**. . .



- ca. 35 classes,
- ca. 5,000 LOC C#

Documentation of Implementation





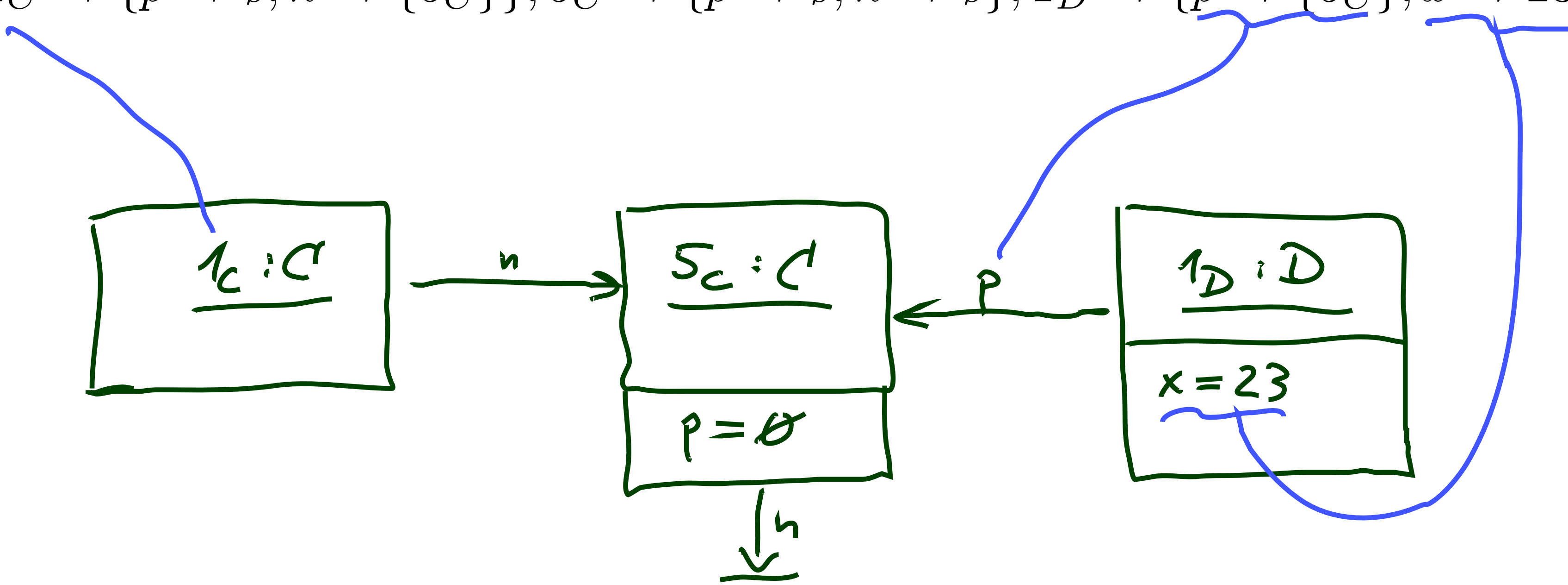
- **Note**: a class **diagram** may be partial, i.e. show only certain aspects of a signature.
- **Note**: a signature can be defined by **a set of** class diagrams.

Object Diagrams

Object Diagram

$$\mathscr{S}_0 = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

$$\sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{p \mapsto \{5_C\}, x \mapsto 23\}\}.$$

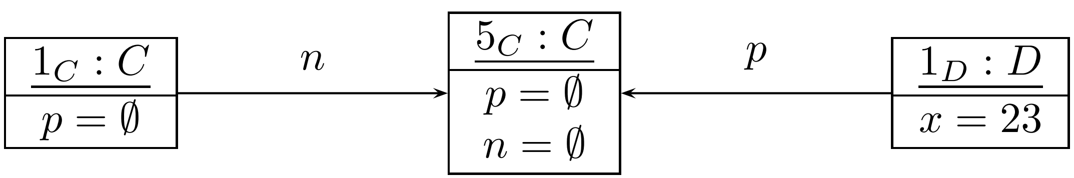


Object Diagram

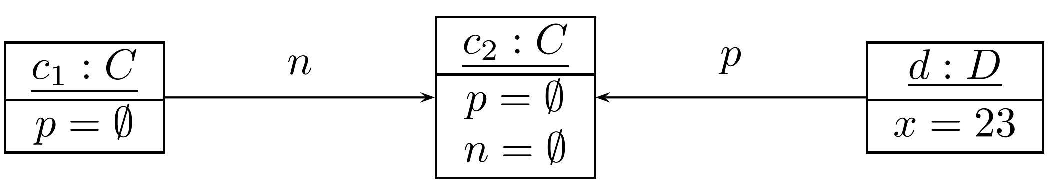
$$\mathscr{S}_0 = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

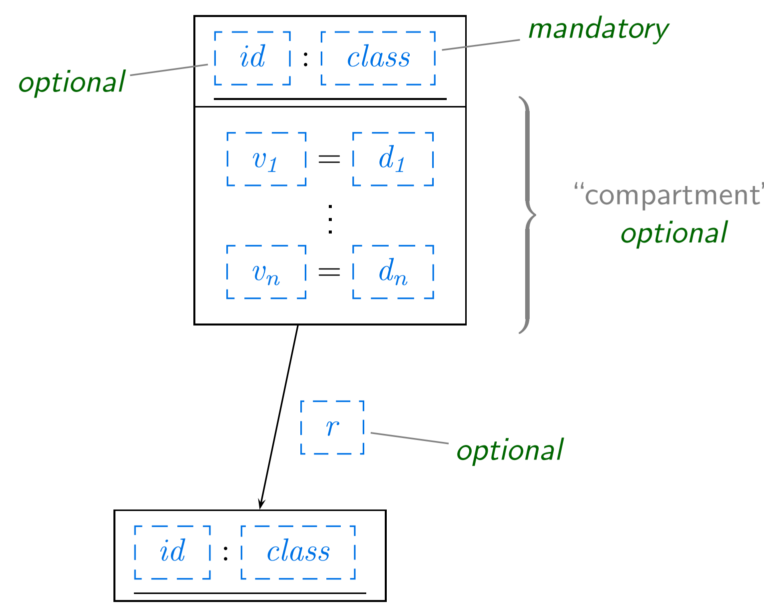
$$\sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{p \mapsto \{5_C\}, x \mapsto 23\}\}.$$

- We may **represent** σ graphically as follows:



or (symbolic identities)

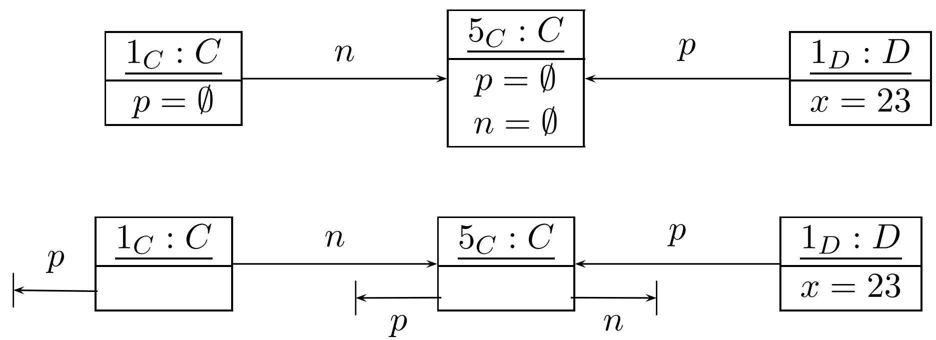




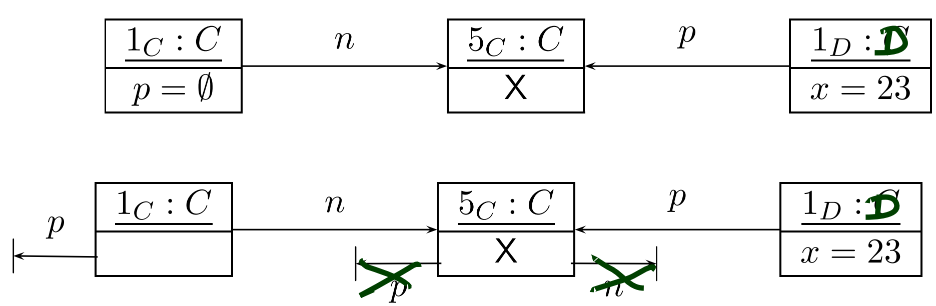
Alternative Presentation, Dangling References

$$\mathscr{S}_0 = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

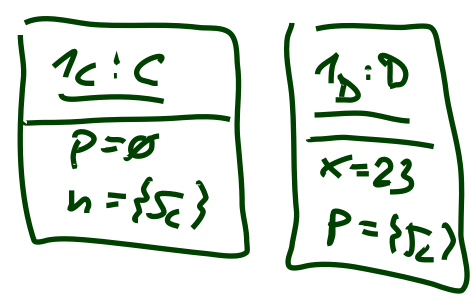
- $\sigma_1 = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{p \mapsto \{5_C\}, x \mapsto 23\}\}.$



- $\sigma_2 = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 1_D \mapsto \{p \mapsto \{5_C\}, x \mapsto 23\}\}.$



alternative:



"**dangling reference**" $(\exists\, u \in \mathrm{dom}(\sigma)\ \exists\, r : T, T \notin \mathscr{T} \bullet \sigma(u)(r) \not\subseteq \mathrm{dom}(\sigma))$

Partial vs. Complete Object Diagrams

$$\mathscr{S}_0 = (\{Int\}, \{C, D\}, \{x : Int, p : C_{0,1}, n : C_*\}, \{C \mapsto \{p, n\}, D \mapsto \{p, x\}\},$$
$$\{f : Int \to Bool, get\_x : Int\}, \{C \mapsto \emptyset, D \mapsto \{f, get\_x\}\}), \qquad \mathscr{D}(Int) = \mathbb{Z}$$

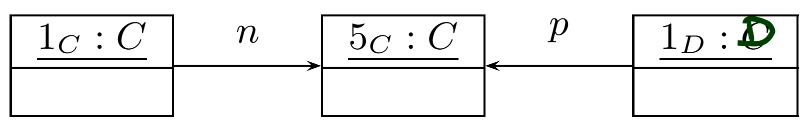
- $\sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{p \mapsto \{5_C\}, x \mapsto 23\}\}.$

  Recall definition **system state**:

  - **Each** attribute of an object **alive** in σ obtains a value by σ.
  - IOW: **Each** σ assigns to **each** attribute of **each** of its **alive** objects a value from $\mathscr{D}(V)$.

  May hinder readability of object diagrams of system states with **many** alive objects...

- So: **partial object diagrams**



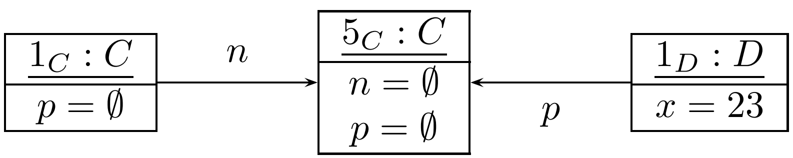
"It is (should be, must not, ...) be possible that a C-object and a D-object have a link to one C-object"

- An object diagram is
  - **partial** if it is a projection of a proper system state, and
  - **complete** if **we say** that it is complete and it uniquely defines a system state.

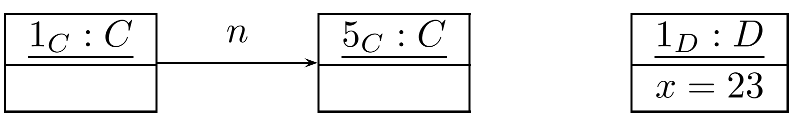
Complete vs. Partial Examples

$$\sigma = \{1_C \mapsto \{p \mapsto \emptyset, n \mapsto \{5_C\}\}, 5_C \mapsto \{p \mapsto \emptyset, n \mapsto \emptyset\}, 1_D \mapsto \{p \mapsto \{5_C\}, x \mapsto 23\}\}.$$

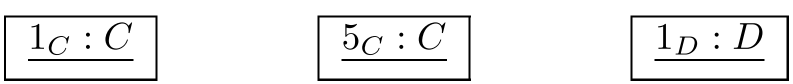
Complete or partial?

- 

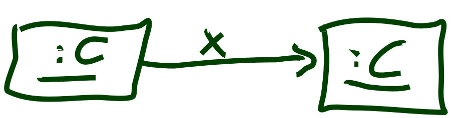
– complete wrt. σ
– without σ above: not clear, not stated whether complete

- 

– partial: attributes missing

- 

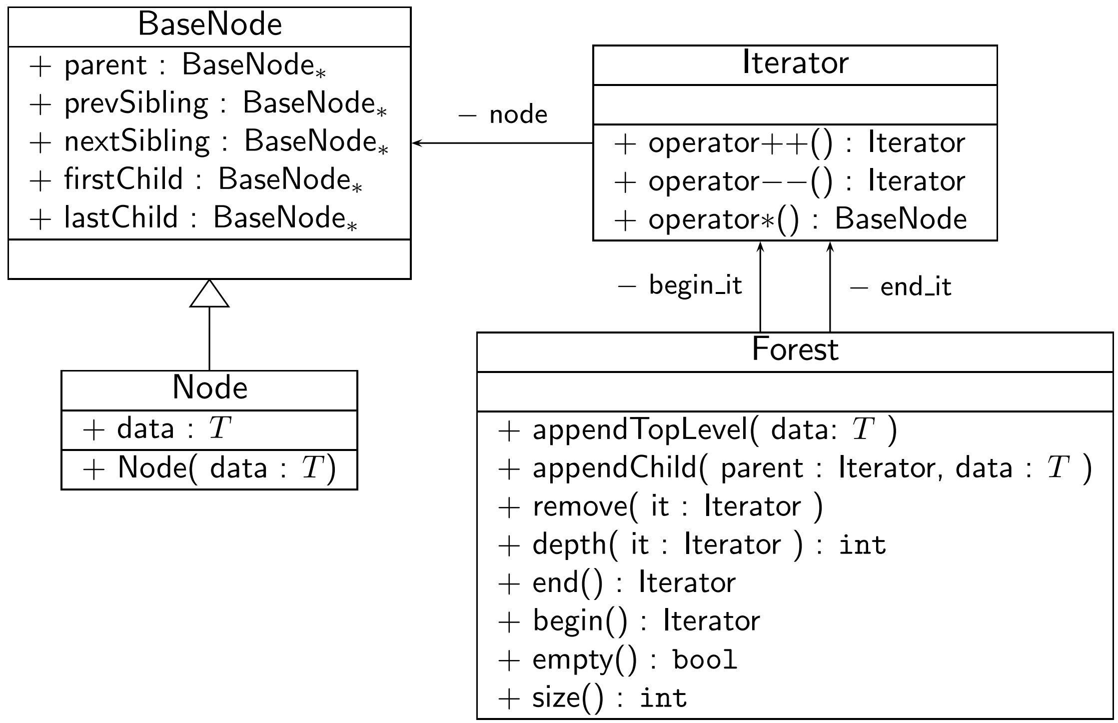
– ——— " ———

- 

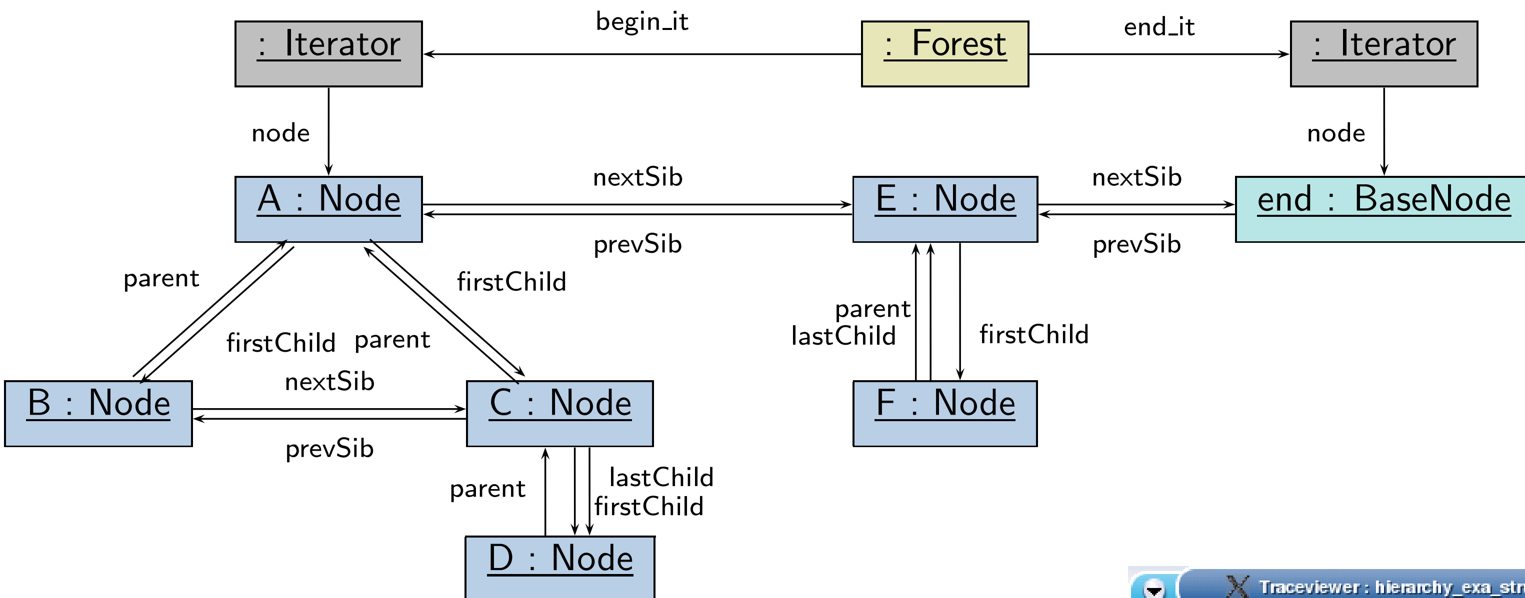
– for $\mathcal{S}$, not even an object diagram
(no $\sigma \in \Sigma_{\mathcal{S}}^{\mathcal{D}}$ for diagram)

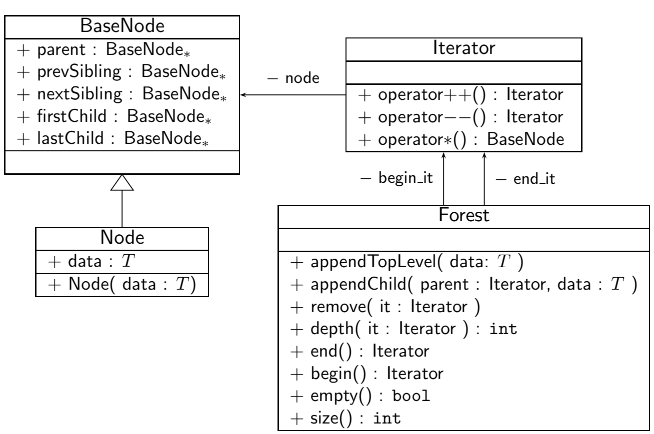
Object Diagrams at Work

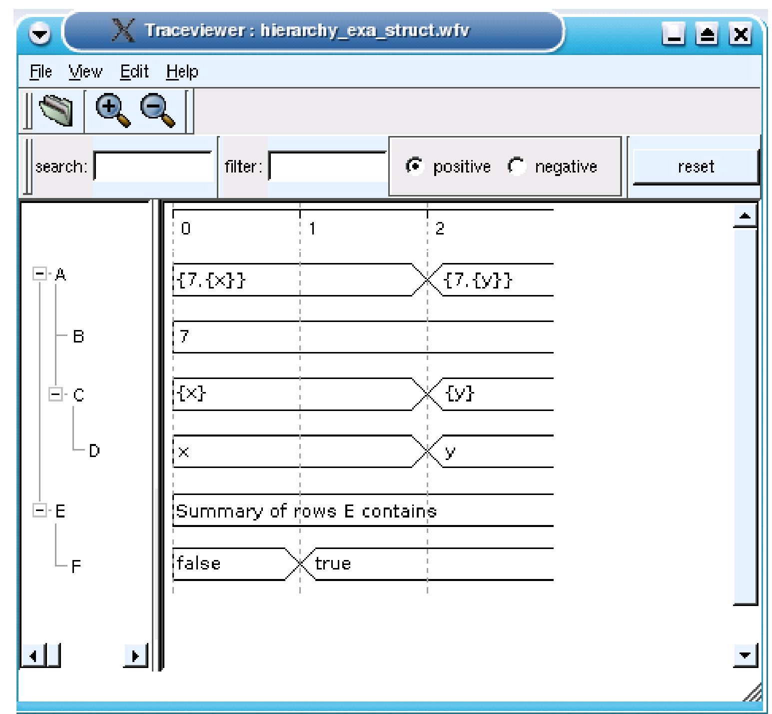
Example: Data Structure (Schumann et al., 2008)



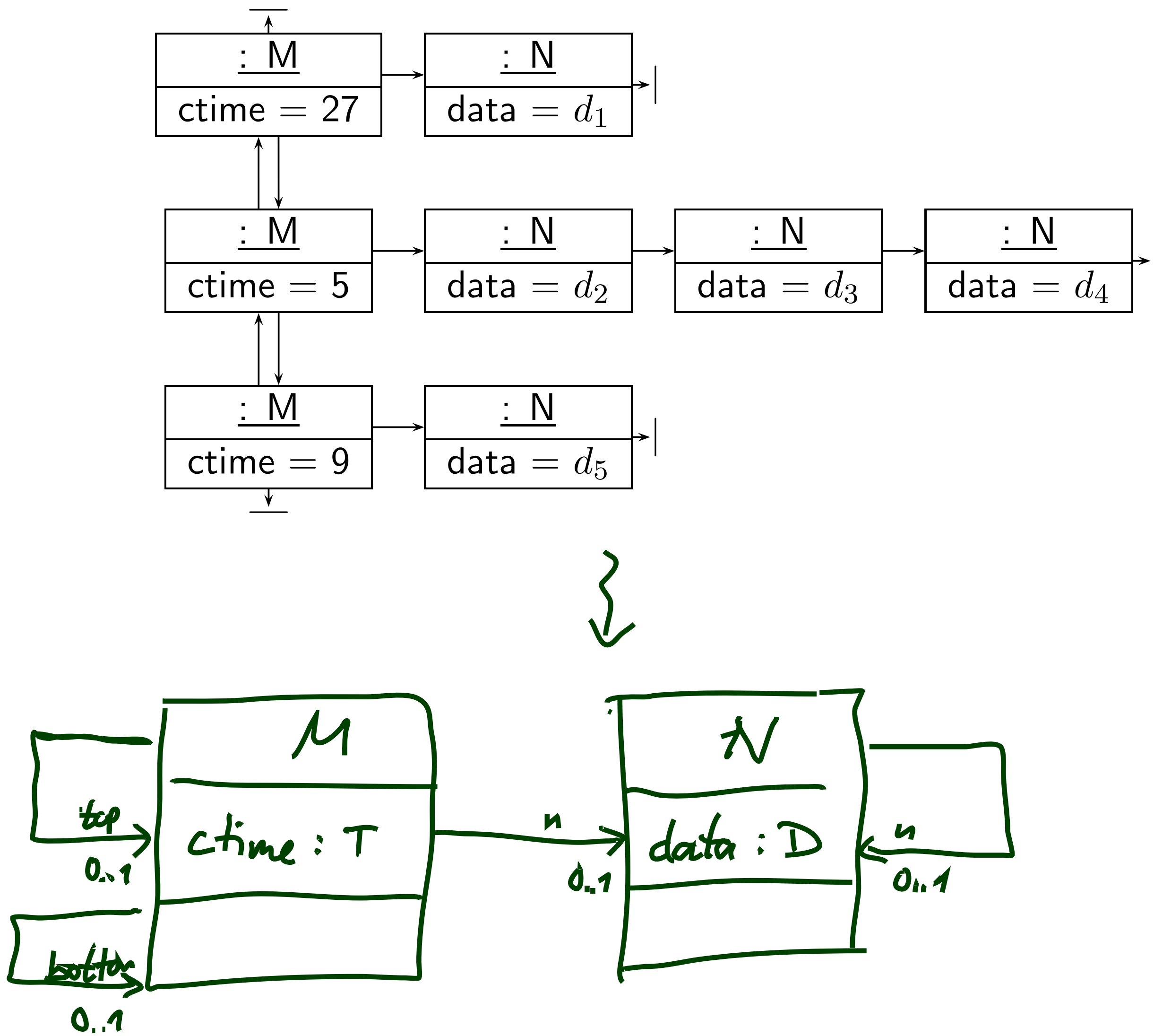
Example: Illustrative Object Diagram (Schumann et al., 2008)





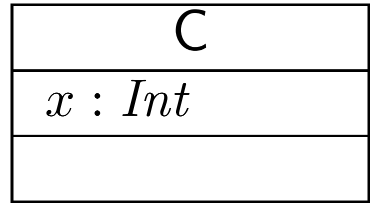


Object Diagrams for Analysis



Towards Object Constraint Logic (OCL)
— “Proto-OCL” —

Constraints on System States



- **Example**: for all C-instance, x should never have the value 27.

$$\forall\, c : C \bullet x(c) \neq 27$$

- **Syntax** (wrt. signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, F, mth)$), c a **logical variable**:

$$\begin{array}{rlll}
F ::= & c & : \tau_C & \\
\mid & v(F) & : \tau_C \to \mathscr{D}(\tau)_\perp, & \text{if } v : \tau \in atr(C) \\
\mid & v(F) & : \tau_C \to \tau_D, & \text{if } v : D_{0,1} \in atr(C) \\
\mid & v(F) & : \tau_C \to 2^{\tau_D}, & \text{if } v : D_* \in atr(C) \\
\mid & f(F_1, \dots, F_n) & : \tau_1 \times \dots \times \tau_n \to \tau, & \text{if } f : \tau_1 \times \dots \times \tau_n \to \tau \\
\mid & \forall\, c : C \bullet F & : \tau_C \times \mathbb{B}_\perp \to \mathbb{B}_\perp &
\end{array}$$

Semantics

- **Syntax**: $F ::= c \mid v(F) \mid f(F_1, \dots, F_n) \mid \forall\, c : C \bullet F$

- **Proto-OCL Types:**
  - values of τ_C: $\mathscr{D}(C) \mathbin{\dot\cup} \{\bot\}$
  - values of $\mathscr{D}(\tau)_\bot$: $\mathscr{D}(\tau) \mathbin{\dot\cup} \{\bot\}$
  - values of 2^{τ_C}: $\mathscr{D}(C_*) \mathbin{\dot\cup} \{\bot\}$ disjoint union
  - values of $\mathbb{B}_\bot$: $\{\mathit{true}, \mathit{false}\} \mathbin{\dot\cup} \{\bot\}$
  - plus: integer, strings, whatever you like (need not be in $\mathscr{T}$), values including $\bot$.

- **Semantics**: mapping logical variables to $\mathscr{D}(C)$
  - $\mathcal{I}[\![c]\!](\sigma, \beta) = \beta(c)$,
  - $\mathcal{I}[\![v(F)]\!](\sigma, \beta) = \sigma\,(\mathcal{I}[\![F]\!](\sigma, \beta))\,(v)$ if $\mathcal{I}[\![F]\!](\sigma, \beta) \neq \bot$, and $\bot$ otherwise,
  - $\mathcal{I}[\![f(F_1, \dots, F_n)]\!](\sigma, \beta) = f_{\mathcal{I}}(\mathcal{I}[\![F_1]\!](\sigma, \beta), \dots, \mathcal{I}[\![F_n]\!](\sigma, \beta))$,

    alive object of class C
  - $\mathcal{I}[\![\forall\, c : C \bullet F]\!](\sigma) = \begin{cases} \mathit{true} & \text{, if } \mathcal{I}[\![F]\!](\sigma, \beta[c := u]) = \mathit{true} \text{ for all } u \in \mathrm{dom}(\sigma) \cap \mathscr{D}(C) \\ \mathit{false} & \text{, if } \mathcal{I}[\![F]\!](\sigma, \beta[c := u]) = \mathit{false} \text{ for some } u \in \mathrm{dom}(\sigma) \cap \mathscr{D}(C) \\ \bot & \text{, otherwise} \end{cases}$

Semantics Cont'd

- Proto-OCL is a **three-valued** logic: a formula evaluates to *true*, *false*, or $\bot$.
- **Example**: $\wedge_\mathcal{I}(\cdot,\cdot) : \{\textit{true}, \textit{false}, \bot\}^2 \to \{\textit{true}, \textit{false}, \bot\}$ is defined as follows:

x_1	*true*	*true*	*true*	*false*	*false*	*false*	$\bot$	$\bot$	$\bot$
x_2	*true*	*false*	$\bot$	*true*	*false*	$\bot$	*true*	*false*	$\bot$
$\wedge_\mathcal{I}(x_1, x_2)$	*true*	*false*	$\bot$	*false*	*false*	*false*	$\bot$	*false*	$\bot$

  We assume common logical connectives $\neg, \wedge, \vee, \dots$ with canonical 3-valued interpretation.

- **Example**: $+_\mathcal{I}(\cdot,\cdot) : (\mathbb{Z} \dot\cup \{\bot\})^2 \to \mathbb{Z} \dot\cup \{\bot\}$

$$+_\mathcal{I}(x_1, x_2) = \begin{cases} x_1 + x_2 & \text{, if } x_1 \neq \bot \text{ and } x_2 \neq \bot \\ \bot & \text{, otherwise} \end{cases}$$

  We assume common arithmetic operations $-, /, *, \dots$ and relation symbols $>, <, \leq, \dots$ with monotone 3-valued interpretation.

- And we assume the special unary function symbol $\mathit{isUndefined}$:

$$\mathit{isUndefined}_\mathcal{I}(x) = \begin{cases} \textit{true} & \text{, if } x = \bot, \\ \textit{false} & \text{, otherwise} \end{cases}$$

  $\mathit{isUndefined}_\mathcal{I}$ is **definite**: it never yields $\bot$.

Semantics Cont'd

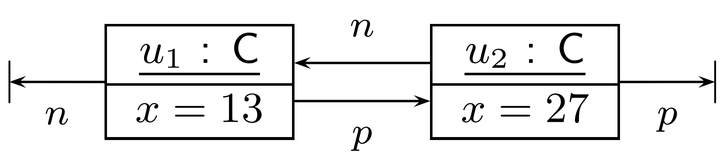
- Lift σ to a **total** function which yields $\perp$ for non-existing objects or attributes:

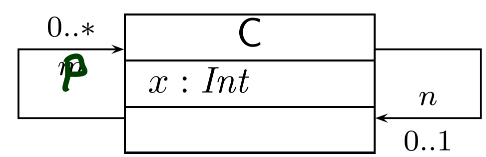
$$\sigma_{\mathcal{I}}(u)(v) = \begin{cases} \perp & \text{, if } u \notin \mathrm{dom}(\sigma) \text{ or } v \notin \mathrm{dom}(\sigma(u)) \quad 1 \\ u' & \text{, if } \sigma(u)(v) = \{u'\} \text{ and } v : C_{0,1} \text{ for some } C \quad 2 \\ \perp & \text{, if } \sigma(u)(v) = \emptyset \text{ and } v : C_{0,1} \text{ for some } C \quad 3 \\ \sigma(u)(v) & \text{, otherwise} \quad 4 \end{cases}$$

In the following, we use σ and $\sigma_{\mathcal{I}}$ interchangeably; which one is meant should be clear from context.

Example:

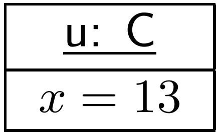
σ :

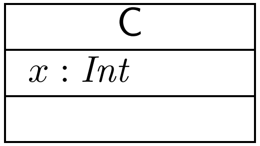




- $\sigma_{\mathcal{I}}(u_1)(x) = 13$ 4
- $\sigma_{\mathcal{I}}(u_1)(y) = \perp$ 1
- $\sigma_{\mathcal{I}}(u_3)(x) = \perp$ 1
- $\sigma_{\mathcal{I}}(u_3)(y) = \perp$ 1
- $\sigma_{\mathcal{I}}(u_2)(n) = u_1$ 2
- $\sigma_{\mathcal{I}}(u_1)(n) = \perp$ 3
- $\sigma_{\mathcal{I}}(u_1)(p) = \{u_2\}$ 4
- $\sigma_{\mathcal{I}}(u_2)(p) = \emptyset$ 4

Example: Evaluate Formula for System State

σ : 



- **infix notation**: $\forall\, c : C \bullet x(c) \neq 27$
- **prefix notation**: $\forall\, c : C \bullet \neq(x(c), 27)$

  Note: $\neq$ as a binary function symbol, 27 as a 0-ary function symbol.

- **Example**:

  $\mathcal{I}[\![\forall\, c : C \bullet \neq(x(c), 27)]\!](\sigma, \emptyset) =$ *true*, because...

  $\mathcal{I}[\![\neq(x(c), 27)]\!](\sigma, \beta), \quad \beta = \{c \mapsto u\}$

  $=$

Example: Evaluate Formula for System State

σ : **u: C** | $x = 13$

C | $x : Int$

- **infix notation**: $\forall\, c : C \bullet x(c) \neq 27$
- **prefix notation**: $\forall\, c : C \bullet \neq(x(c), 27)$

  Note: $\neq$ as a binary function symbol, 27 as a 0-ary function symbol.

- **Example**:

  $\mathcal{I}[\![\forall\, c : C \bullet \neq(x(c), 27)]\!](\sigma, \emptyset) =$ *true*, because...

  $\mathcal{I}[\![\neq(x(c), 27)]\!](\sigma, \beta), \quad \beta = \{c \mapsto u\}$

  $= \neq_{\mathcal{I}}(\underbrace{\mathcal{I}[\![x(c)]\!](\sigma, \beta)}_{\sigma(\mathcal{I}[\![c]\!](\sigma,\beta))(x)}, \underbrace{\mathcal{I}[\![27]\!](\sigma, \beta)}_{27_{\mathcal{I}}})$

  $=$

Example: Evaluate Formula for System State

σ :

<u>u: C</u>
$x = 13$

C
$x : Int$

- **infix notation**: $\forall\, c : C \bullet x(c) \neq 27$
- **prefix notation**: $\forall\, c : C \bullet \neq(x(c), 27)$

  Note: $\neq$ as a binary function symbol, 27 as a 0-ary function symbol.

- **Example**:

  $\mathcal{I}[\![\forall\, c : C \bullet \neq(x(c), 27)]\!](\sigma, \emptyset) =$ *true*, because...

  $\mathcal{I}[\![\neq(x(c), 27)]\!](\sigma, \beta), \quad \beta = \{\cancel{x}\, c \mapsto u\}$

  $= \neq_{\mathcal{I}}(\ \mathcal{I}[\![x(c)]\!](\sigma, \beta),\ \mathcal{I}[\![27]\!](\sigma, \beta)\ )$

  $= \neq_{\mathcal{I}}(\ \sigma(\ \underbrace{\mathcal{I}[\![c]\!](\sigma, \beta)}_{\beta(c) = u}\ )(x),\ 27_{\mathcal{I}}\ )$

  $=$

Example: Evaluate Formula for System State

σ :

$\underline{\text{u: C}}$
$x = 13$

C
$x : Int$

- **infix notation**: $\forall\, c : C \bullet x(c) \neq 27$
- **prefix notation**: $\forall\, c : C \bullet \neq(x(c), 27)$

  Note: $\neq$ as a binary function symbol, 27 as a 0-ary function symbol.

- **Example**:

  $\mathcal{I}[\![\forall\, c : C \bullet \neq(x(c), 27)]\!](\sigma, \emptyset) = \textit{true}$, because...

  $\mathcal{I}[\![\neq(x(c), 27)]\!](\sigma, \beta), \quad \beta = \{c \mapsto u\}$

  $= \neq_{\mathcal{I}}(\ \mathcal{I}[\![x(c)]\!](\sigma, \beta),\ \mathcal{I}[\![27]\!](\sigma, \beta)\ )$

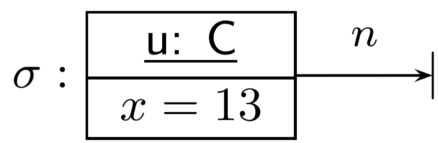
  $= \neq_{\mathcal{I}}(\ \sigma(\ \mathcal{I}[\![c]\!](\sigma, \beta)\ )(x),\ 27_{\mathcal{I}}\ )$

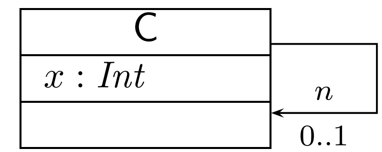
  $= \neq_{\mathcal{I}}(\ \sigma(\ \beta(c)\ )(x),\ 27_{\mathcal{I}}\ )$

  $= \neq_{\mathcal{I}}(\ \sigma(\ u\ )(x),\ 27_{\mathcal{I}}\ )$

  $= \neq_{\mathcal{I}}(\ 13,\ 27\ ) = \textit{true}$ ... and u is the only C-object in σ.

More Interesting Example





$$\underbrace{\forall c : C \bullet x(n(c)) \neq 27}_{=: F}$$

- Similar to the previous slide, we need the value of

$$\sigma(\, \underbrace{\sigma(\, \overbrace{\mathcal{I}[\![c]\!](\sigma,\beta)}^{=u} \,)(n)}_{\perp} \,)(x) = \perp$$

- $\mathcal{I}[\![c]\!](\sigma,\beta) = \beta(c) = u$
- $\sigma(\, \mathcal{I}[\![c]\!](\sigma,\beta) \,)(n) = \sigma_{\mathcal{I}}(u)(n) = \perp$
- $\sigma(\, \sigma(\, \mathcal{I}[\![c]\!](\sigma,\beta) \,)(n) \,)(x) = \sigma_{\mathcal{I}}(\perp)(x) = \perp$

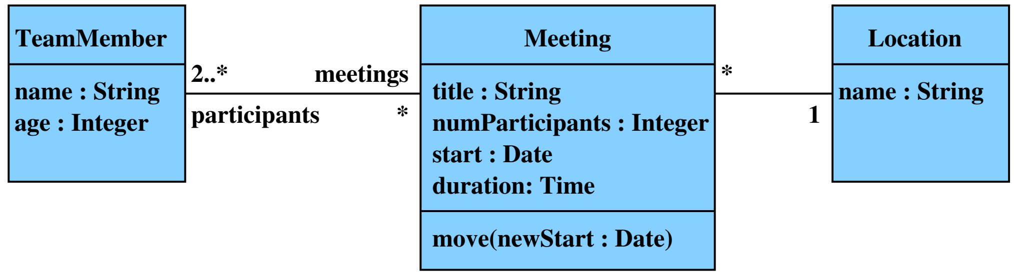
↳ $\mathcal{I}[\![F]\!](\sigma,\beta) = \perp$

Object Constraint Language (OCL)

OCL is the same — just with less readable (?) syntax.

Literature: (OMG, 2006; Warmer and Kleppe, 1999).

Examples (from lecture “Softwaretechnik 2008”)



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

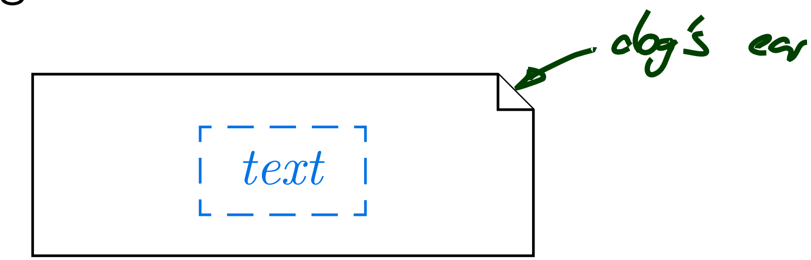
Prof. Dr. P. Thiemann, http://proglang.informatik.uni-freiburg.de/teaching/swt/2008/

$\forall\, self : Meeting \bullet size(participants(self)) = numParticipants(self)$

$\forall\, self : Location \bullet name(self) = \text{"Lobby"} \text{ implies } isEmpty(meeting(self))$

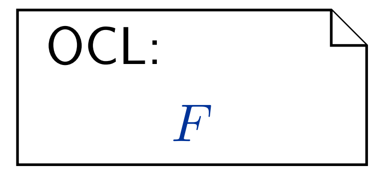
Where To Put OCL Constraints?

- **Notes**: A UML **note** is a diagram element of the form

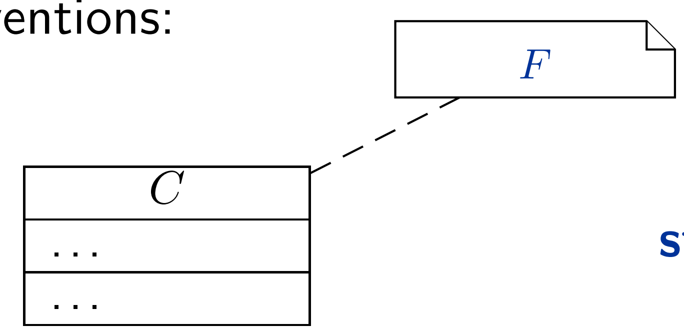


text can principally be **everything**, in particular **comments** and **constraints**.

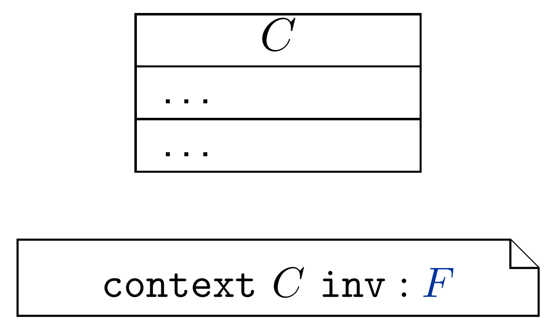
Sometimes, content is **explicitly classified** for clarity:



- Conventions:



stands for



$\forall c : C \bullet F$

References

References

Kopetz, H. (2011). What I learned from Brian. In Jones, C. B. et al., editors, *Dependable and Historic Computing*, volume 6875 of *LNCS*. Springer.

Lovins, A. B. and Lovins, L. H. (2001). *Brittle Power - Energy Strategy for National Security*. Rocky Mountain Institute.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

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

Schumann, M., Steinke, J., Deck, A., and Westphal, B. (2008). Traceviewer technical documentation, version 1.0. Technical report, Carl von Ossietzky Universität Oldenburg und OFFIS.

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