Software Design, Modelling and Analysis in UML

Lecture 07: Class Diagrams II

2014-11-13

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Representing class diagrams as (extended) signatures — for the moment without associations (see Lecture 08).

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What is a class diagram?
  - For what purposes are class diagrams useful?
  - Could you please map this class diagram to a signature?
  - Could you please map this signature to a class diagram?
  - What is visibility good for?

- **Content:**
  - Map class diagram to (extended) signature cont'd.
  - Stereotypes – for documentation.
  - Visibility as an extension of well-typedness.

Mapping UML CDs to Extended Signatures

From Class Boxes to Extended Signatures

A class box n **induces** an (extended) signature class as follows:

n:

$\langle\langle S_1, \ldots, S_k \rangle\rangle$ C
$\xi_1\ v_1 : \tau_1 = v_{0,1}\ \{P_{1,1}, \ldots, P_{1,m_1}\}$ $\vdots$ $\xi_\ell\ v_\ell : \tau_\ell = v_{0,\ell}\ \{P_{\ell,1}, \ldots, P_{\ell,m_\ell}\}$

$$\Downarrow$$

$$\mathscr{C}(n) := \langle C, \{S_1, \ldots, S_k\}, a(n), t(n)\rangle$$

$$V(n) := \{\langle v_1 : \tau_1, \xi_1, v_{0,1}, \{P_{1,1}, \ldots, P_{1,m_1}\}\rangle, \ldots, \langle v_\ell : \tau_\ell, \xi_\ell, v_{0,\ell}, \{P_{\ell,1}, \ldots, P_{\ell,m_\ell}\}\rangle\}$$

$$atr(n) := \{C \mapsto \{v_1, \ldots, v_\ell\}\}$$

where

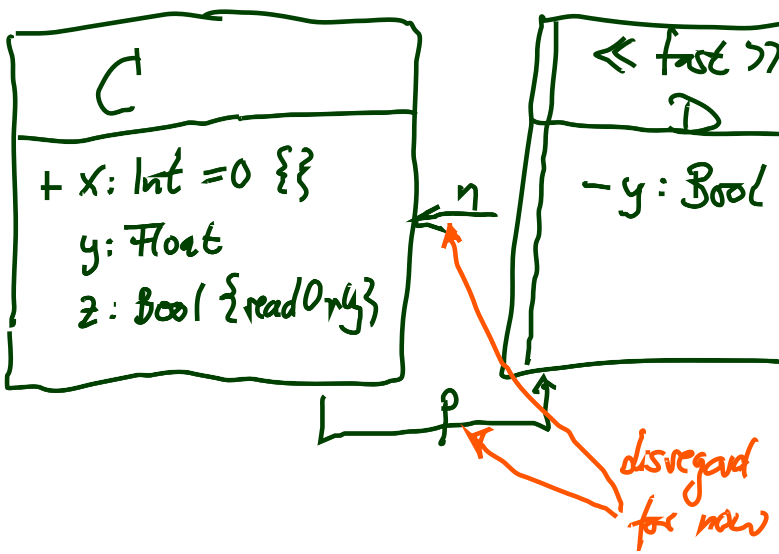
- "abstract" is determined by the font:

$$a(n) = \begin{cases} \textit{true} & \text{, if } n = \boxed{C} \text{ or } n = \boxed{\mathit{C}\ \{A\}} \\ \textit{false} & \text{, otherwise} \end{cases}$$

- "active" is determined by the frame:

$$t(n) = \begin{cases} \textit{true} & \text{, if } n = \boxed{C} \text{ or } n = \boxed{\| C \|} \\ \textit{false} & \text{, otherwise} \end{cases}$$

Recall: Example



"same name, different type"

- prefix names

  C::y ✓

  D::y

- require unique names
- "formalist approach"

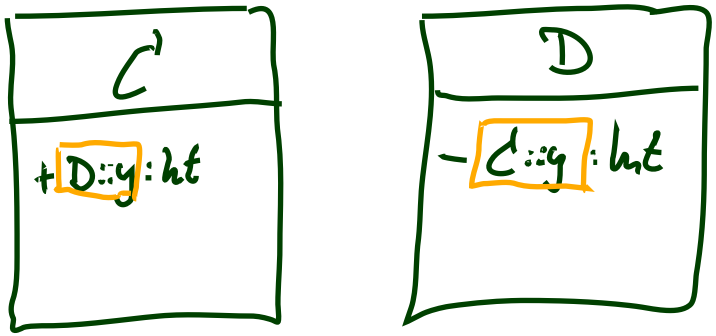
$$
\begin{aligned}
\mathscr{S} = (&\{Int, Float, Bool\},\\
&\{\langle C, \emptyset, false, false\rangle,\\
&\ \ \langle D, \{fast\}, false, true\rangle\},\\
&\{\langle x : Int, +, 0, \emptyset\rangle,\\
&\ \ \langle C{::}y : Float, +, \star, \emptyset\rangle,\\
&\ \ \langle z : Bool, +, \star, \{readOnly\}\rangle,\\
&\ \ \langle D{::}y : Bool, -, \star, \emptyset\rangle\},\\
&\{C \mapsto \{x, C{::}y, z\},\\
&\ \ D \mapsto \{D{::}y\}\})
\end{aligned}
$$

? $\neq$

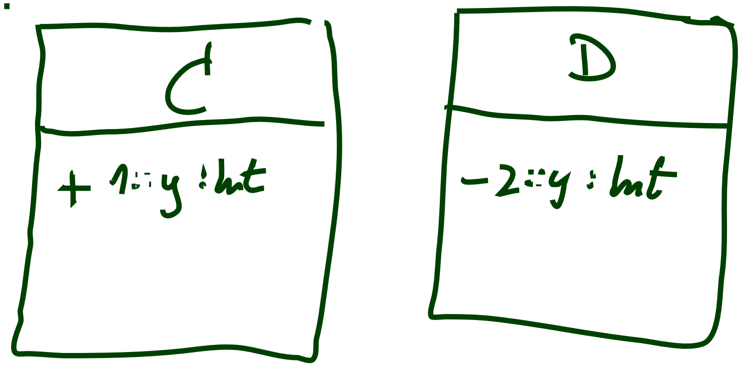
default:
visibility +
stereotypes $\emptyset$
properties $\emptyset$

put a note to explicate the used "defaults"

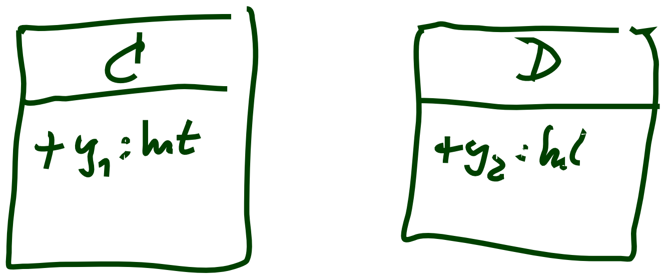
$$\mathcal{S} = (\{Int\}, \{C, D\}, \{\langle C::y : Int, -, \star, \emptyset\rangle, \langle D::y : Int, +, \star, \emptyset\rangle\},$$
$$, \{C \mapsto \{D::y\}, D \mapsto \{C::y\}\})$$



or:



or:



What If Things Are Missing?

C
$v : Int$

- For instance, what about the box above?
- v has **no visibility**, **no initial value**, and (strictly speaking) **no properties**.

It depends.

- What does the standard say? [OMG, 2007a, 121]

> “**Presentation Options.**
> *The type, visibility, default, multiplicity, property string may be suppressed from being displayed, even if there are values in the model.”*

- **Visibility**: There is no “no visibility” — an attribute **has** a visibility in the (extended) signature.

  Some (and we) assume **public** as default, but conventions may vary.

- **Initial value**: some assume it **given by domain** (such as “leftmost value”, but what is “leftmost” of $\mathbb{Z}$?).
  Some (and we) understand **non-deterministic initialisation**.

- **Properties**: probably safe to assume $\emptyset$ if not given at all.

From Class Diagrams to Extended Signatures

- We view a **class diagram** $\mathcal{CD}$ as a graph with nodes $\{n_1, \dots, n_N\}$ (each "class rectangle" is a node).
  - $\mathscr{C}(\mathcal{CD}) := \cancel{\boxed{\bigcup_{i=1}^{N} \mathscr{C}(n_i)}}$ $\{ \mathscr{C}(n_i) \mid 1 \leq n_i \leq N \}$
  - $V(\mathcal{CD}) := \bigcup_{i=1}^{N} V(n_i)$
  - $atr(\mathcal{CD}) := \bigcup_{i=1}^{N} atr(n_i)$
- In a **UML model**, we can have **finitely many** class diagrams,

$$\mathscr{C}\mathscr{D} = \{\mathcal{CD}_1, \dots, \mathcal{CD}_k\},$$

which **induce** the following signature:

$$\mathscr{S}(\mathscr{C}\mathscr{D}) = \left(\mathscr{T}, \bigcup_{i=1}^{k} \mathscr{C}(\mathcal{CD}_i), \bigcup_{i=1}^{k} V(\mathcal{CD}_i), \bigcup_{i=1}^{k} atr(\mathcal{CD}_i) \right).$$

(Assuming $\mathscr{T}$ given. In "reality" (i.e. in full UML), we can introduce types in class diagrams, the class diagram then contributes to $\mathscr{T}$. Example: enumeration types.)

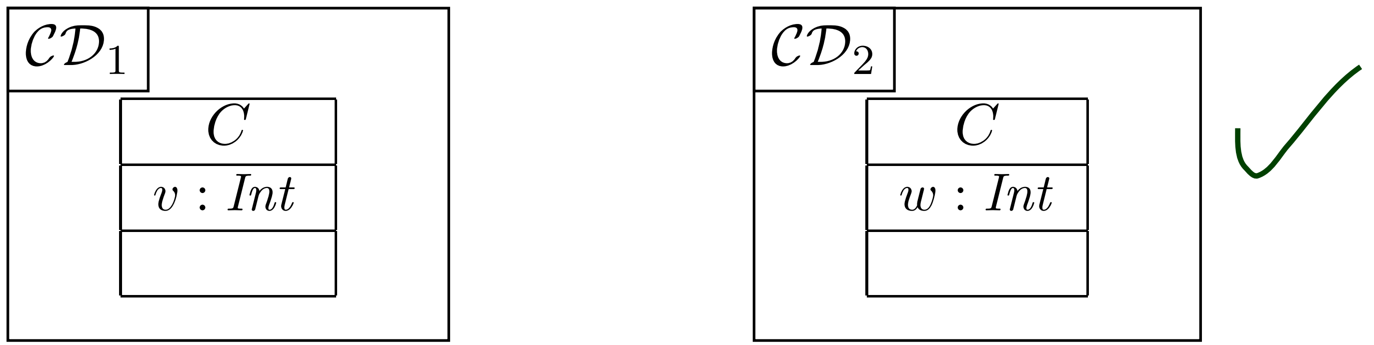
Is the Mapping a Function?

- Is $\mathscr{S}(\mathscr{CD})$ **well-defined**?

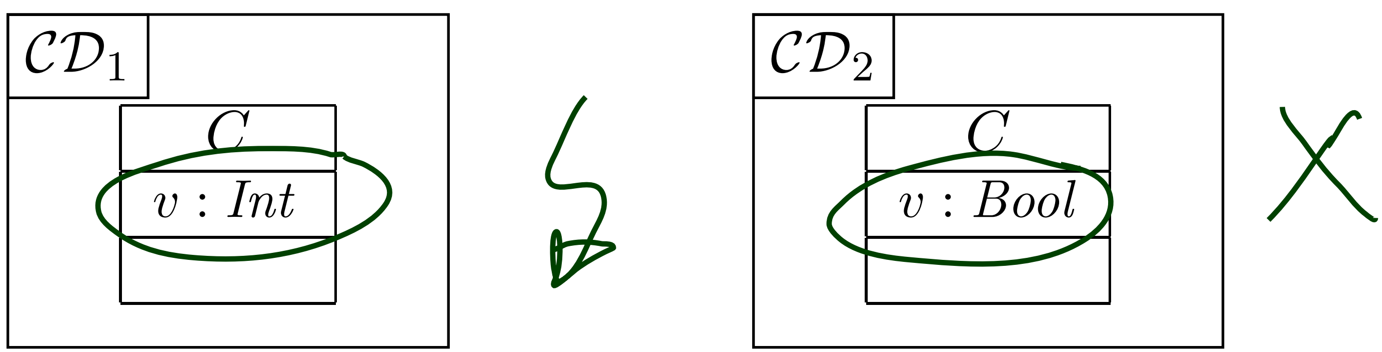
Two possible **sources for problems**:

(1) A **class** C may appear in **multiple** class **diagrams**:

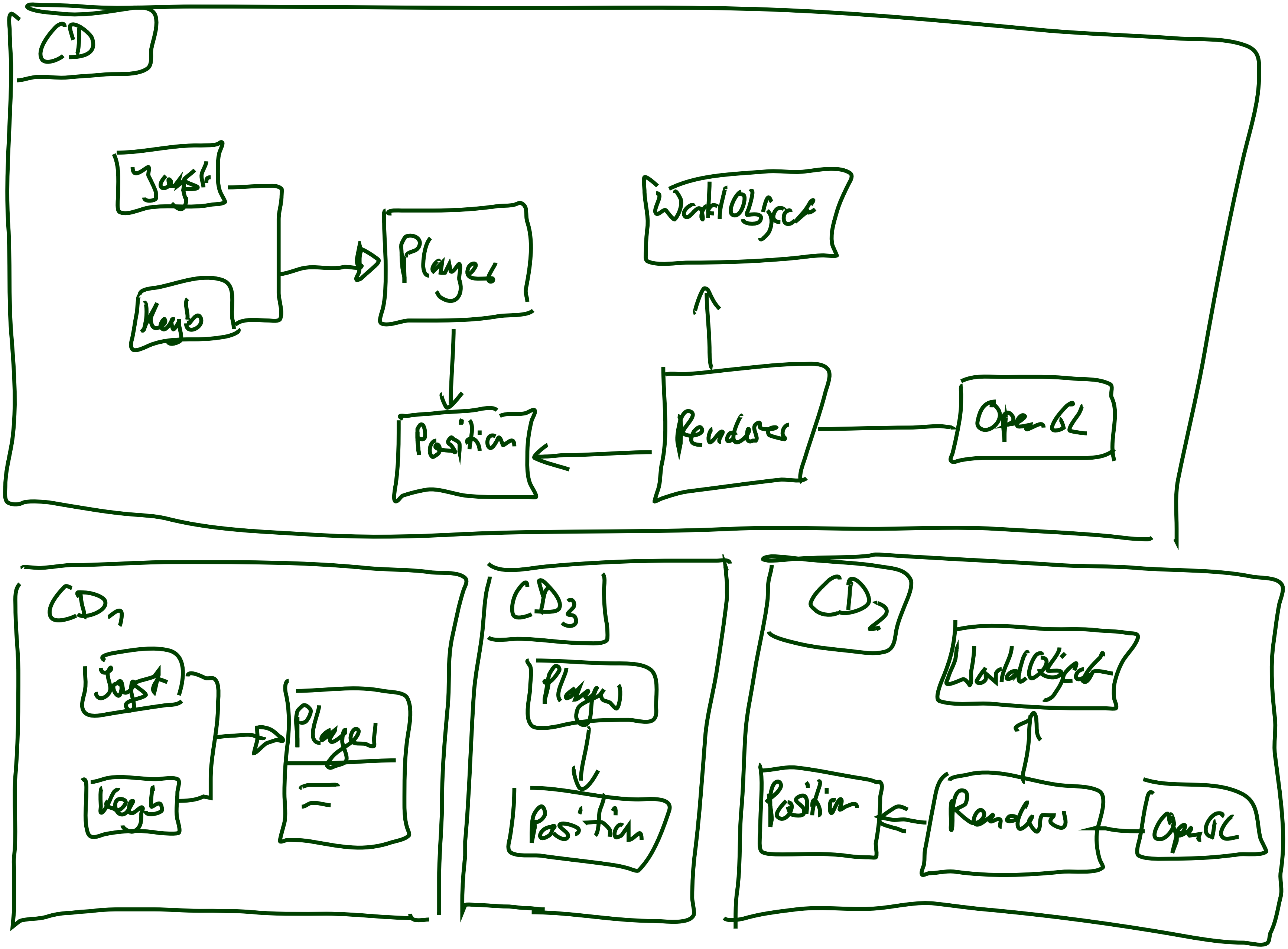
(i)



(ii)



Simply **forbid** the case (ii) — easy syntactical check on diagram.


CD
Joyst
Keyb
Player
WorldObject
Position
Renderer
OpenGL
CD1
Joyst
Keyb
Player
CD3
Player
Position
CD2
WorldObject
Position
Renderer
OpenGL

Is the Mapping a Function?

(2) An **attribute** v may appear in **multiple classes**:

C
$v : Bool$

D
$v : Int$

Two approaches:

- Require **unique** attribute names.
  This requirement can easily be established (implicitly, behind the scenes) by viewing v as an abbreviation for

$$C{::}v \quad \text{or} \quad D{::}v$$

  depending on the context. ($C{::}v : Bool$ and $D{::}v : Int$ are unique.)

- Subtle, formalist's approach: observe that

$$\langle v : Bool, \dots \rangle \quad \text{and} \quad \langle v : Int, \dots \rangle$$

  are **different things** in V. But we don't follow that path...

Class Diagram Semantics

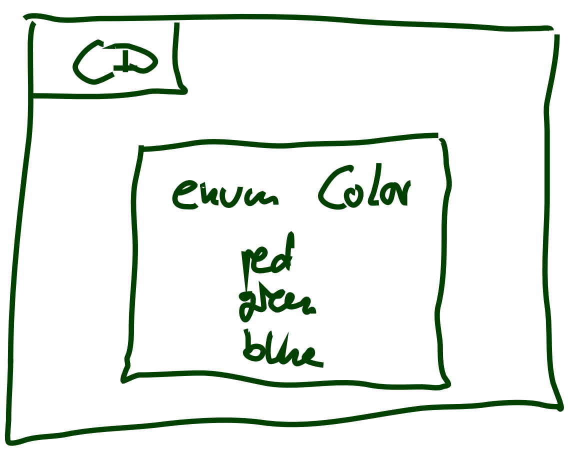
Semantics

The semantics of a set of **class diagrams** $\mathscr{CD}$ first of all is the induced (extended) **signature** $\mathscr{S}(\mathscr{CD})$.

The **signature** gives rise to a set of **system states** given a **structure** $\mathscr{D}$.

- Do we need to redefine/extend $\mathscr{D}$? **No.**

  (Would be different if we considered the definition of enumeration types in class diagrams. Then the domain of an enumeration type τ, i.e. the set $\mathscr{D}(\tau)$, would be determined by the class diagram, and not free for choice.)



$\mathscr{S} = (\{Color, \ldots\}, \ldots)$

$\rightsquigarrow \mathscr{D}(Color) = \{red, green, blue\}$

Semantics

The semantics of a set of **class diagrams** $\mathscr{CD}$ first of all is the induced (extended) **signature** $\mathscr{S}(\mathscr{CD})$.
The **signature** gives rise to a set of **system states** given a **structure** $\mathscr{D}$.

- Do we need to redefine/extend $\mathscr{D}$? **No.**

  (Would be different if we considered the definition of enumeration types in class diagrams. Then the domain of an enumeration type τ, i.e. the set $\mathscr{D}(\tau)$, would be determined by the class diagram, and not free for choice.)

- What is the effect on $\Sigma_{\mathscr{S}}^{\mathscr{D}}$? **Little.**

  For now, we only **remove** abstract class instances, i.e.

  $$\sigma : \mathscr{D}(\mathscr{C}) \nrightarrow (V \nrightarrow (\mathscr{D}(\mathscr{T}) \cup \mathscr{D}(\mathscr{C}_*)))$$

  is now **only** called **system state** if and only if, for all $\langle C, S_C, 1, t\rangle \in \mathscr{C}$,

  $$\mathrm{dom}(\sigma) \cap \mathscr{D}(C) = \emptyset.$$

  With $a = 0$ as default "abstractness", the earlier definitions apply directly. We'll revisit this when discussing inheritance.

What About The Rest?

- **Classes**:
  - **Active**: not represented in σ.
    Later: relevant for behaviour, i.e., how system states evolve over time.
  - **Stereotypes**: in a minute.
- **Attributes**:
  - **Initial value**: not represented in σ.
    Later: provides an initial value as effect of "creation action".
  - **Visibility**: not represented in σ.
    Later: viewed as additional **typing information** for well-formedness of system transformers; and with inheritance.
  - **Properties**: such as `readOnly`, `ordered`, `composite`
    (**Deprecated** in the standard.)
    - `readOnly` — **later** treated similar to visibility.
    - `ordered` — not considered in our UML fragment ($\rightarrow$ sets vs. sequences).
    - `composite` — cf. lecture on associations.

Stereotypes

Stereotypes as Labels or Tags

- So, a class is $\langle C, S_C, a, t \rangle$ with
  the abstractness flag a, activeness flag t, and a set of **stereotypes** S_C.

- What are Stereotypes?

  - **Not** represented in system states.

  - **Not** contributing to typing rules.
    (cf. **later** ~~lecture on type theory for UML~~) ; visibility)

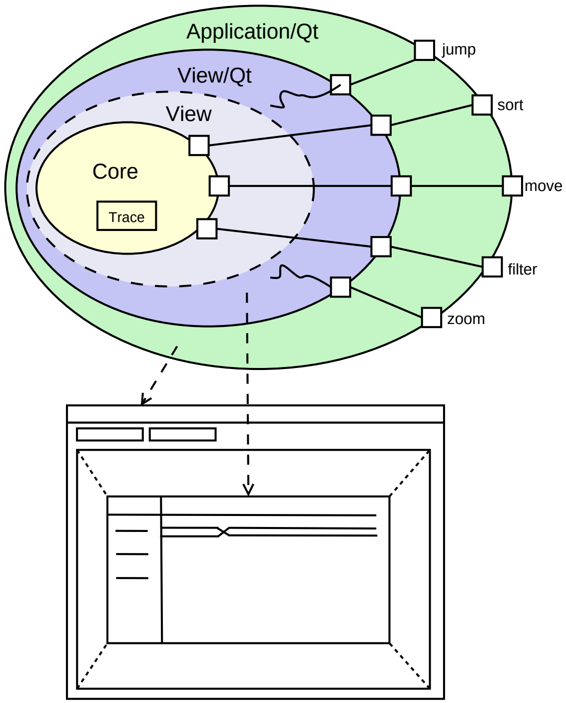
- [Oestereich, 2006]:
  View stereotypes as (additional) "**labelling**" ("tags") or as "**grouping**".

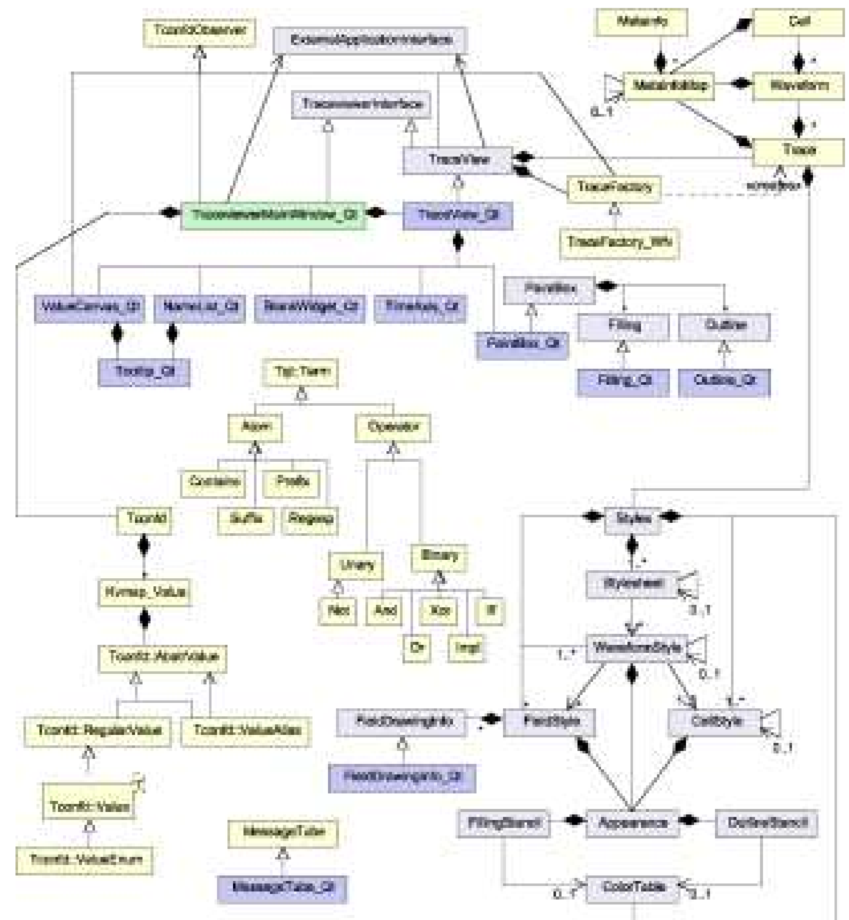
  Useful for documentation and MDA.

  - **Documentation**: e.g. layers of an architecture.
    Sometimes, packages (cf. the standard) are sufficient and "right".

  - **Model Driven Architecture** (MDA): **later**.

Example: Stereotypes for Documentation





- Example: Timing Diagram Viewer [Schumann et al., 2008]
- Architecture of four layers:
  - core, data layer
  - abstract view layer
  - toolkit-specific view layer/widget
  - application using widget
- Stereotype "=" layer "=" colour

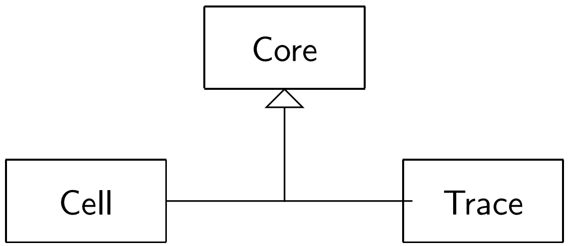
Stereotypes as Inheritance

- Another view (due to whom?): distinguish
  - **Technical Inheritance**

    If the **target platform**, such as the programming language for the implementation of the blueprint, is object-oriented, assume a 1-to-1 relation between inheritance in the model and on the target platform.
  - **Conceptual Inheritance**

    Only meaningful with a **common idea** of what stereotypes stand for. For instance, one could label each class with the team that is responsible for realising it. Or with licensing information (e.g., LGPL and proprietary).
    Or one could have labels understood by code generators (cf. lecture on MDSE).

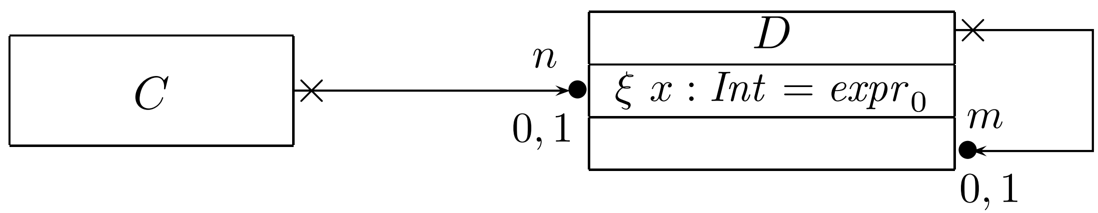
- **Confusing**:
  - Inheritance is often referred to as the "is a"-relation.
    Sharing a stereotype also expresses "being something".
  - We can always (ab-)use UML-inheritance for the conceptual case, e.g.

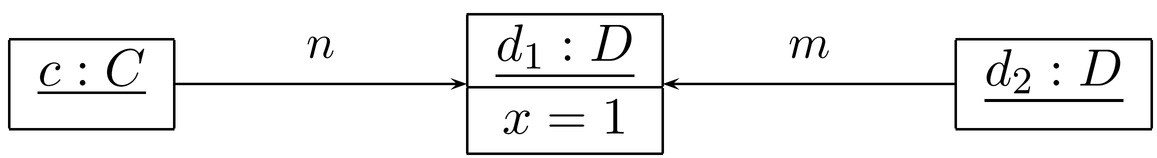


Visibility

The Intuition by Example

$$\mathscr{S} = (\{Int\}, \{C, D\}, \{n : D_{0,1},\ m : D_{0,1}, \langle x : Int, \xi, expr_0, \emptyset\rangle\},\ \{C \mapsto \{n\}, D \mapsto \{x, m\}\}$$





Assume $\boxed{w_1 : \tau_C}$ and $\boxed{w_2 : \tau_D}$ are logical variables. **Which** of the following **syntactically correct** (?) OCL expressions **shall** we consider to be **well-typed**?

ξ of x:	public	private	protected	package
$w_1 . n . x = 0$	✔ 卌 I	✔ 0	later	not
$x(n(w_1))$	✘	✘ 卌 III		
	? II	?		
$w_2 . m . x = 0$	✔ 卌 IIII	✔ III	later	not
$x(m(w_2))$	✘ 0	✘ III		
	?	?		

References

[Oestereich, 2006] Oestereich, B. (2006). *Analyse und Design mit UML 2.1, 8. Auflage*. Oldenbourg, 8. edition.

[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.

[Schumann et al., 2008] 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.