# *Software Design, Modelling and Analysis in UML*

## *Lecture 09: Class Diagrams IV*

*2012-11-27*

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

Albert-Ludwigs-Universität Freiburg, Germany

# Contents & Goals

**Last Lectures:**

- Started to discuss "associations", the general case.

**This Lecture:**

- **Educational Objectives:** Capabilities for following tasks/questions.
  - Cont'd: Please explain this class diagram with associations.
  - When is a class diagram a good class diagram?
  - What are purposes of modelling guidelines? (Example?)
  - Discuss the style of this class diagram.
- **Content:**
  - Treat "the rest".
  - Where do we put OCL constraints?
  - Modelling guidelines, in particular for class diagrams (following [Ambler, 2005])

# *Associations: The Rest*

# *The Rest*

**Recapitulation**: Consider the following association:

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle$$

- **Association name** $r$ and **role names/types** $role_i/C_i$ induce extended system states $\lambda$.
- **Multiplicity** $\mu$ is considered in OCL syntax.
- **Visibility** $\xi$/**Navigability** $\nu$: well-typedness.

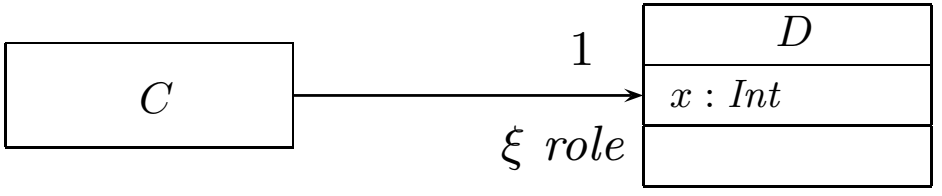
**Now the rest**:

- **Multiplicity** $\mu$: we propose to view them as constraints.
- **Properties** $P_i$: even more typing.
- **Ownership** $o$: getting closer to pointers/references.
- **Diamonds**: exercise.

# *Visibility*

Not so surprising: Visibility of role-names is treated completely similar to visibility of attributes, namely by **typing rules**.

**Question**: given



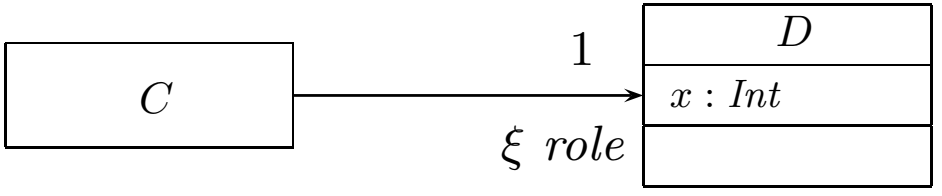

is the following OCL expression well-typed or not (wrt. visibility):

$$\text{context } C \text{ inv} : self.role.x > 0$$

# *Visibility*

Not so surprising: Visibility of role-names is treated completely similar to visibility of attributes, namely by **typing rules**.

**Question**: given




is the following OCL expression well-typed or not (wrt. visibility):

$$\text{context } C \text{ inv} : self.role.x > 0$$

Basically same rule as before: (analogously for other multiplicities)

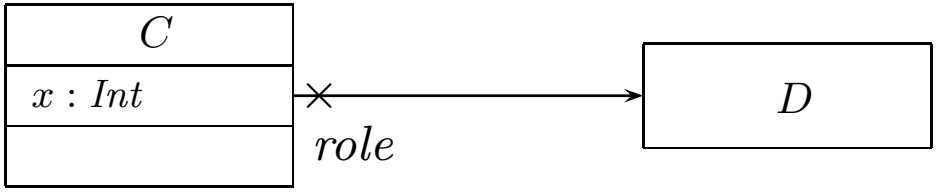
$$(Assoc_1) \quad \frac{A, B \vdash expr_1 : \tau_C}{A, B \vdash role(expr_1) : \tau_D}, \quad \begin{array}{l} \mu = 0..1 \text{ or } \mu = 1, \\ \xi = +, \text{ or } \xi = - \text{ and } C = B \end{array}$$

$$\langle r : \ldots \langle role : D, \mu, \_, \xi, \_, \_ \rangle, \ldots \langle role' : C, \_, \_, \_, \_, \_ \rangle, \ldots \rangle \in V$$

# *Navigability*

**Navigability** is similar to visibility: expressions over non-navigable association ends ($\nu = \times$) are **basically** type-correct, but **forbidden**.

**Question**: given



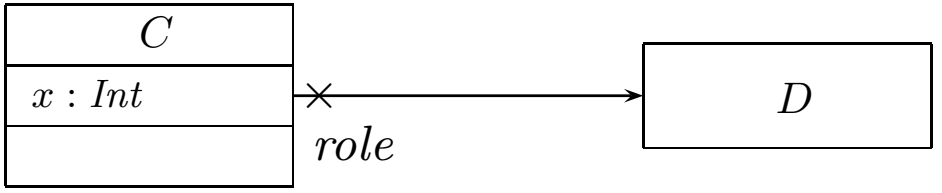

is the following OCL expression well-typed or not (wrt. navigability):

$$\text{context } D \text{ inv} : self.role.x > 0$$

# Navigability

**Navigability** is similar to visibility: expressions over non-navigable association ends ($\nu = \times$) are **basically** type-correct, but **forbidden**.

**Question**: given




is the following OCL expression well-typed or not (wrt. navigability):

$$\text{context } D \text{ inv} : self.role.x > 0$$

The standard says:

- '$-$': navigation is possible
- '$\times$': navigation is not possible
- '$>$': navigation is efficient

by context decide what "efficient" means to you and communicate this to the developers

**So**: In general, UML associations are different from pointers/references!

**But**: Pointers/references can faithfully be modelled by UML associations.

# *The Rest of the Rest*

**Recapitulation**: Consider the following association:

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle$$

- **Association name** $r$ and **role names/types** $role_i/C_i$ induce extended system states $\lambda$.
- **Multiplicity** $\mu$ is considered in OCL syntax.
- **Visibility** $\xi$/**Navigability** $\nu$: well-typedness.

**Now the rest**:

- **Multiplicity** $\mu$: we propose to view them as constraints.
- **Properties** $P_i$: even more typing.
- **Ownership** $o$: getting closer to pointers/references.
- **Diamonds**: exercise.

# *Multiplicities as Constraints*

**Recall**: The multiplicity of an association end is a term of the form:

$$\mu ::= * \mid N \mid N..M \mid N..* \mid \mu, \mu \qquad (N, M \in \mathbb{N})$$

**Proposal**: View multiplicities (except $0..1$, $1$) as additional invariants/constraints.

# *Multiplicities as Constraints*

**Recall**: The multiplicity of an association end is a term of the form:

$$\mu ::= * \mid N \mid N..M \mid N..* \mid \mu, \mu \qquad (N, M \in \mathbb{N})$$

**Proposal**: View multiplicities (except $0..1$, $1$) as additional invariants/constraints.

**Recall**: we can normalize each multiplicity $\mu$ to the form

$$N_1..N_2,\ \ldots,\ N_{2k-1}..N_{2k}$$

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $\quad N_1, \ldots, N_{2k-1} \in \mathbb{N}$, $\quad N_{2k} \in \mathbb{N} \cup \{*\}$.

e.g. 3, 10..12, 14 *
↓ normalize
3..3, 10..12, 14..14 0..*

# *Multiplicities as Constraints*

$$\mu = N_1..N_2,\ \ldots,\ N_{2k-1}..N_{2k}$$

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $\quad N_1, \ldots, N_{2k-1} \in \mathbb{N}$, $\quad N_{2k} \in \mathbb{N} \cup \{*\}$.

# *Multiplicities as Constraints*

$$\mu = N_1..N_2,\ \ldots,\ N_{2k-1}..N_{2k}$$

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $\quad N_1, \ldots, N_{2k-1} \in \mathbb{N}$, $\quad N_{2k} \in \mathbb{N} \cup \{*\}$.

**Define** $\mu^C_{\mathsf{OCL}}(role)$ := context $C$ inv :

$$(N_1 \leq role \text{ -> size()} \leq N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \leq role \text{ -> size()} \underbrace{\leq N_{2k}}_{\text{omit if } N_{2k} = *})$$

for each $\mu \neq 0..1$, $\mu \neq 1$,

$$\langle r : \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots \rangle \in V \text{ or}$$

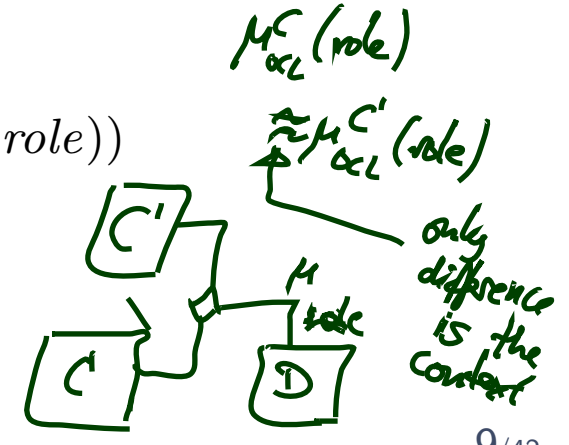
$$\langle r : \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots \rangle \in V, role \neq role'.$$

And **define**

$$\mu^C_{\mathsf{OCL}}(role) := \text{context } C \text{ inv : not(oclIsUndefined}(role))$$

for each $\mu = 1$.

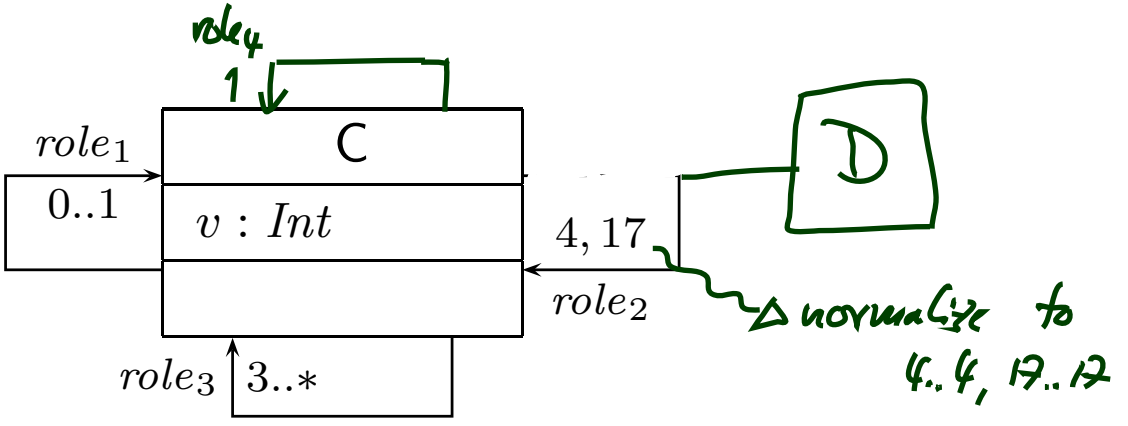
**Note**: in $n$-ary associations with $n > 2$, there is redundancy.



# Multiplicities as Constraints Example

$$\mu^{C}_{\mathsf{OCL}}(role) = \text{context } C \text{ inv :}$$
$$(N_1 \leq role \text{ -> size}() \leq N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \leq role \text{ -> size}() \leq N_{2k})$$

$\mathcal{CD}$ :




$Inv(\mathcal{CD}) =$

- { context D inv: $4 \leq role_2 \text{->size}() \leq 4$ or $17 \leq role_2 \text{->size}() \leq 17$ }
  equivalent to $role_2 \text{->size}() = 4$ or $role_2 \text{->size}() = 17$
- ∪ { context C inv: $3 \leq role_3 \text{->size}()$ }
- ∪ { context C inv: not oclIsUndefined($role_4$) }
- ~~∪ { context C inv~~

# *Why Multiplicities as Constraints?*

More precise, can't we just use **types**? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$:

# *Why Multiplicities as Constraints?*

More precise, can't we just use **types**? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$:
  many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$:

# *Why Multiplicities as Constraints?*

More precise, can't we just use **types**? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$:
  many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$:
  could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\mu_{\mathsf{OCL}} = \mathit{true}$ anyway.
- $\mu = 0..3$ :

# *Why Multiplicities as Constraints?*

More precise, can't we just use **types**? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$:
  many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$:
  could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\mu_{\mathsf{OCL}} = \mathit{true}$ anyway.
- $\mu = 0..3$ :
  use array of size $4$ — if model behaviour (or the implementation) adds 5th identity, we'll get a runtime error, and thereby see that the constraint is violated. **Principally acceptable**, but: checks for array bounds everywhere...?
- $\mu = 5..7$ :

# *Why Multiplicities as Constraints?*

More precise, can't we just use **types**? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$:
  many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$:
  could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\mu_{\mathsf{OCL}} = \mathit{true}$ anyway.
- $\mu = 0..4$:
  use array of size $4$ — if model behaviour (or the implementation) adds 5th identity, we'll get a runtime error, and thereby see that the constraint is violated. **Principally acceptable**, but: checks for array bounds everywhere...?
- $\mu = 5..7$ :
  could be represented by an array of size $7$ — but: few programming languages/data structure libraries allow lower bounds for arrays (other than $0$). If we have $5$ identities and the model behaviour removes one, this should be a violation of the constraints imposed by the **model**.
  The implementation which does this removal is **wrong**. How do we see this...?

# *Multiplicities Never as Types...?*

Well, if the **target platform** is known and fixed,
**and** the target platform has, for instance,

- reference types,
- range-checked arrays with positions $0, \ldots, N$,
- set types,

then we could simply **restrict** the syntax of multiplicities to

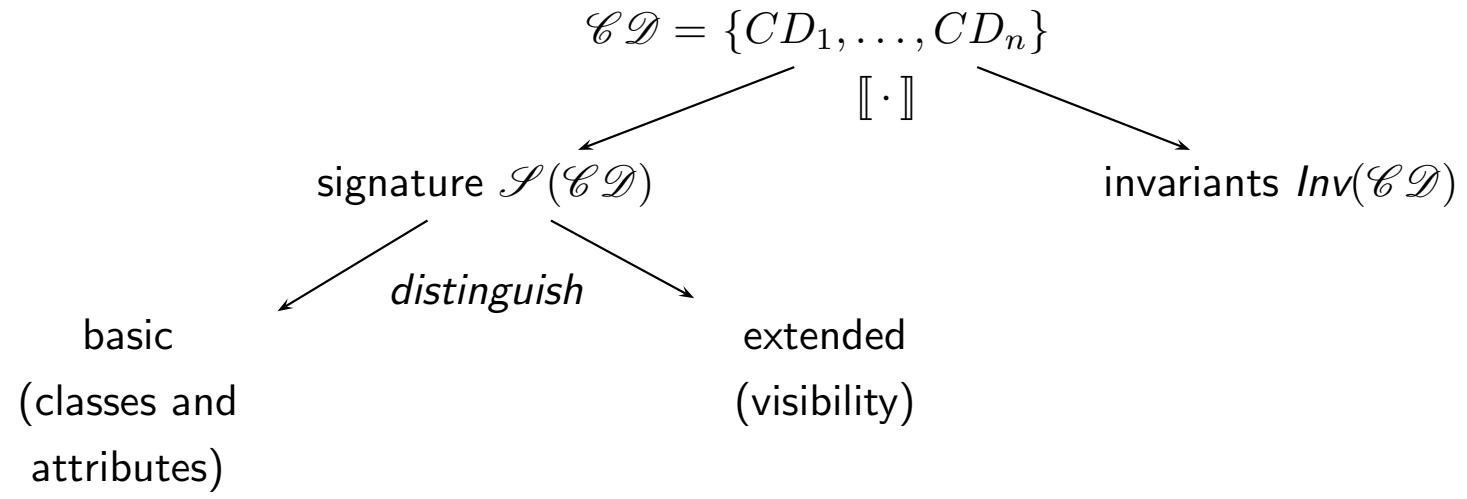
$$\mu ::= 1 \mid 0..N \mid *$$

and don't think about constraints
(but use the obvious 1-to-1 mapping to types)...

In general, **unfortunately**, we don't know.

# *Multiplicities as Constraints of Class Diagram*

**Recall/Later**:




**From now on**: *Inv*$(\mathscr{CD}) = \{$constraints occurring in notes$\} \cup \{\mu^C_{\mathsf{OCL}}(role) \mid$

$$\langle r : \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots\rangle \in V \text{ or}$$
$$\langle r : \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots\rangle \in V,$$
$$role \neq role', \mu \notin \{0..1\}\}.$$

# *Properties*

We don't want to cover association **properties** in detail, only some observations (assume binary associations):

| Property | Intuition | Semantical Effect |
|---|---|---|
| **unique** | one object has **at most one** $r$-link to a single other object | **current setting** |
| **bag** | one object may have **multiple** $r$-links to a single other object | have $\lambda(r)$ yield multi-sets |
| **ordered**, **sequence** | an $r$-link is a **sequence** of object identities (possibly including duplicates) | have $\lambda(r)$ yield sequences |

# *Properties*

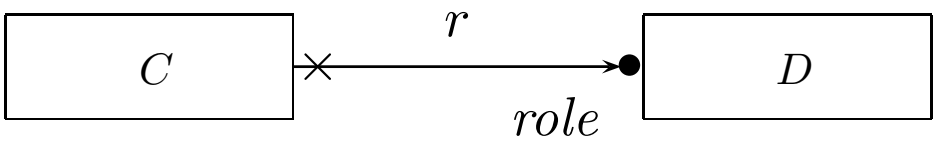
We don't want to cover association **properties** in detail, only some observations (assume binary associations):

| Property | Intuition | Semantical Effect |
|---|---|---|
| **unique** | one object has **at most one** $r$-link to a single other object | **current setting** |
| **bag** | one object may have **multiple** $r$-links to a single other object | have $\lambda(r)$ yield multi-sets |
| **ordered**, **sequence** | an $r$-link is a **sequence** of object identities (possibly including duplicates) | have $\lambda(r)$ yield sequences |

| Property | OCL Typing of expression $role(expr)$ |
|---|---|
| **unique** | $\tau_D \to Set(\tau_C)$ |
| **bag** | $\tau_D \to Bag(\tau_C)$ |
| **ordered**, **sequence** | $\tau_D \to Seq(\tau_C)$ |

For **subsets**, **redefines**, **union**, etc. see [OMG, 2007a, 127].

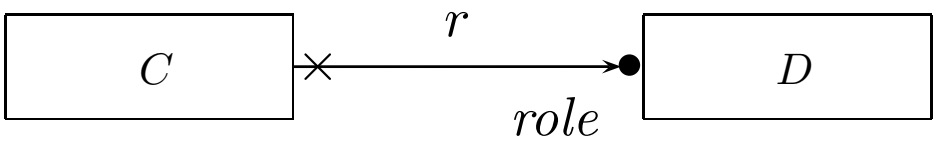
# *Ownership*




Intuitively it says:

> Association $r$ is **not a "thing on its own"** (i.e. provided by $\lambda$),
> but association end '$role$' is **owned** by $C$ (!).
> (That is, it's stored inside $C$ object and provided by $\sigma$).

# *Ownership*



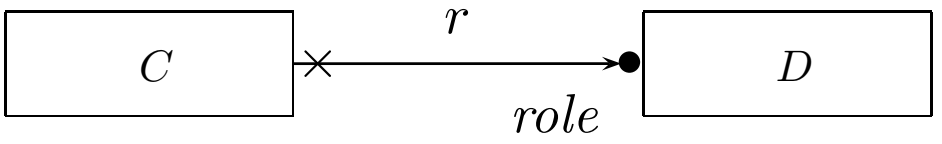

Intuitively it says:

> Association $r$ is **not a "thing on its own"** (i.e. provided by $\lambda$),
> but association end '$role$' is **owned** by $C$ (!).
> (That is, it's stored inside $C$ object and provided by $\sigma$).

**So**: if multiplicity of $role$ is $0..1$ or $1$, then the picture above is very close to concepts of pointers/references.

Actually, ownership is seldom seen in UML diagrams. Again: if target platform is clear, one may well live without (cf. [OMG, 2007b, 42] for more details).

# Ownership



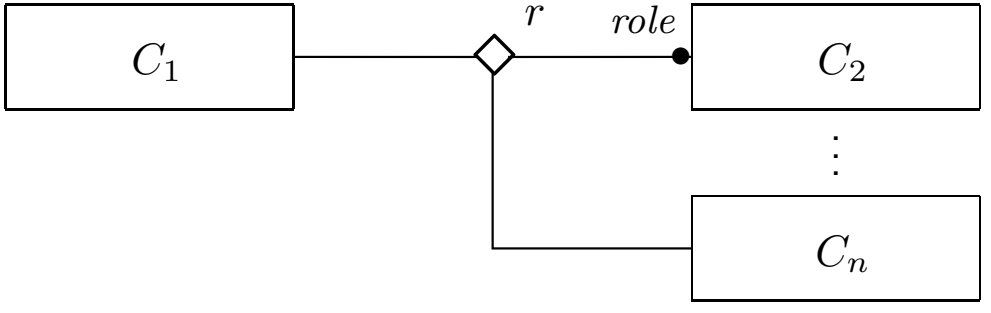

Intuitively it says:

> Association $r$ is **not a "thing on its own"** (i.e. provided by $\lambda$),
> but association end '$role$' is **owned** by $C$ (!).
> (That is, it's stored inside $C$ object and provided by $\sigma$).

**So**: if multiplicity of $role$ is $0..1$ or $1$, then the picture above is very close to concepts of pointers/references.

Actually, ownership is seldom seen in UML diagrams. Again: if target platform is clear, one may well live without (cf. [OMG, 2007b, 42] for more details).

**Not clear to me**:



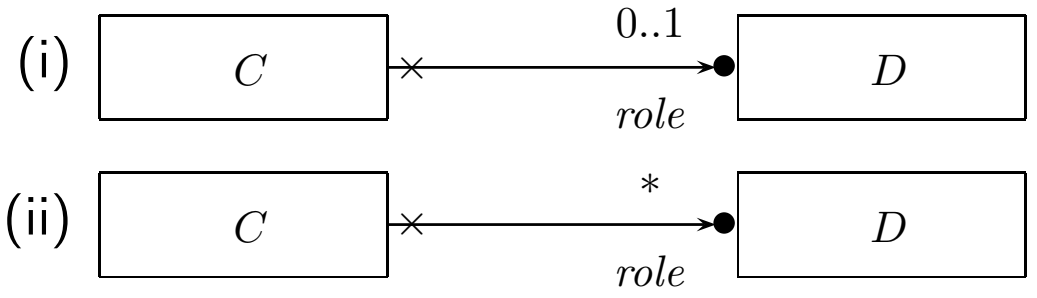
*Back to the Main Track*

# *Back to the main track:*

**Recall**: on some earlier slides we said, the extension of the signature is **only** to study associations in "full beauty".
For the remainder of the course, we should look for something simpler...

**Proposal**:

- **from now on**, we only use associations of the form




(And we may omit the non-navigability and ownership symbols.)

- Form (i) introduces $role : C_{0,1}$, and form (ii) introduces $role : C_*$ in $V$.
- In both cases, $role \in atr(C)$.
- We drop $\lambda$ and go back to our nice $\sigma$ with $\sigma(u)(role) \subseteq \mathscr{D}(D)$.

# *OCL Constraints in (Class) Diagrams*

# *Where Shall We Put OCL Constraints?*

**Numerous options**:

(i) Additional documents.

(ii) Notes.

(iii) Particular dedicated places.

# *Where Shall We Put OCL Constraints?*

**Numerous options**:

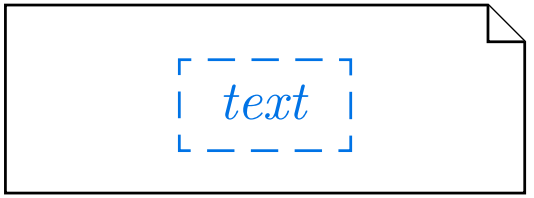
(i) Additional documents.

(ii) Notes.

(iii) Particular dedicated places.

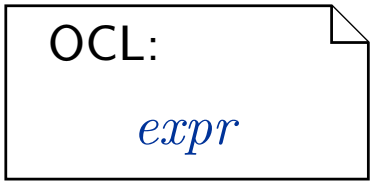
(i) **Notes**:

A UML **note** is a picture of the form

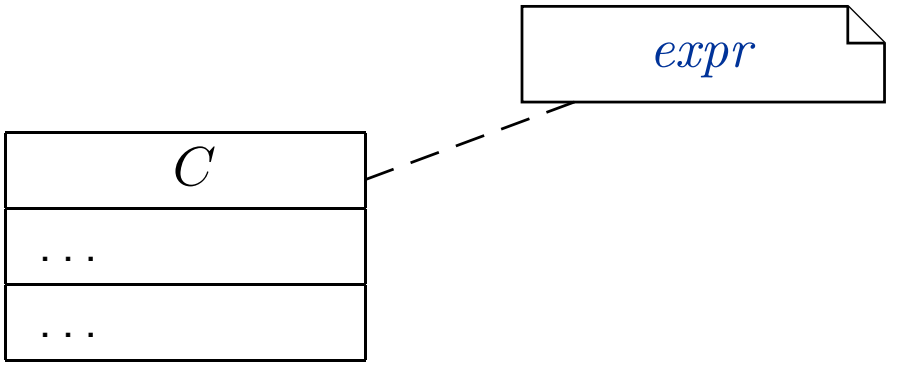



*text* can principally be **everything**, in particular **comments** and **constraints**.

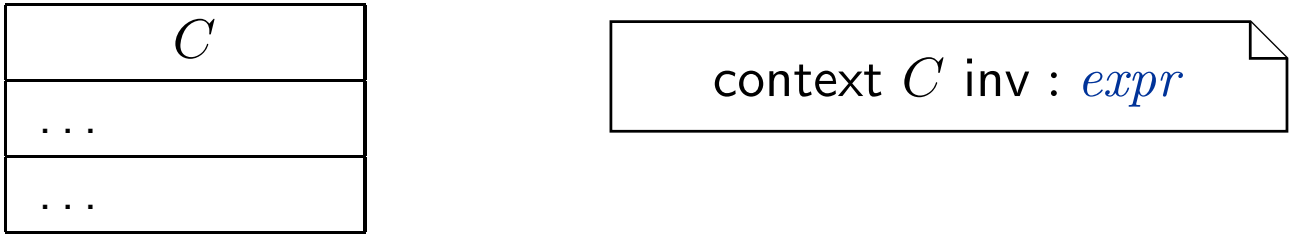
**Sometimes**, content is **explicitly classified** for clarity:



# *OCL in Notes: Conventions*




**stands for**



# *Where Shall We Put OCL Constraints?*

(ii) **Particular dedicated places** in class diagrams: (behav. feature: later)

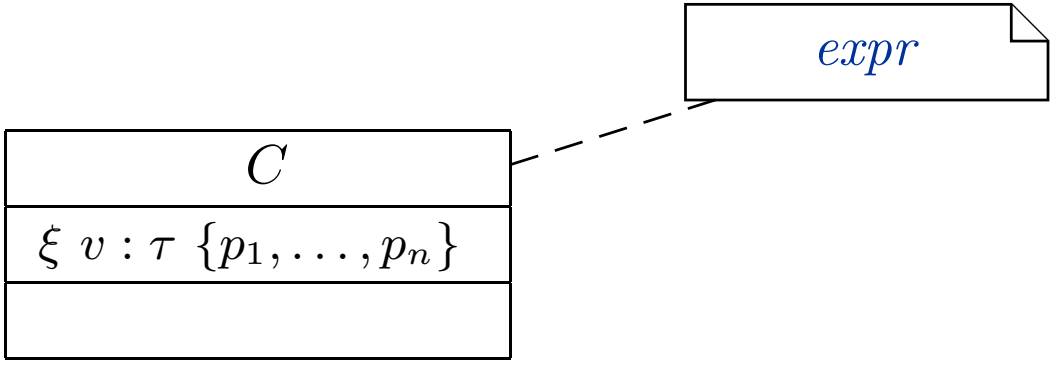
| $C$ |
| --- |
| $\xi\ v : \tau\ \{p_1, \ldots, p_n\}\ \{expr\}$ |
| $\xi\ f(v_1 : \tau, \ldots, v_n : \tau_n) : \tau\ \{p_1, \ldots, p_n\}\ \{\text{pre} : expr_1$ <br> $\text{post} : expr_2\}$ |

# *Where Shall We Put OCL Constraints?*

(ii) **Particular dedicated places** in class diagrams: (behav. feature: later)

| $C$ |
| --- |
| $\xi\ v : \tau\ \{p_1, \ldots, p_n\}\ \{expr\}$ |
| $\xi\ f(v_1 : \tau, \ldots, v_n : \tau_n) : \tau\ \{p_1, \ldots, p_n\}\ \{\text{pre} : expr_1$ <br> $\text{post} : expr_2\}$ |

For simplicity, we view the above as an abbreviation for




context $f$ pre : $expr_1$ post : $expr_2$

# *Invariants of a Class Diagram*

- Let $\mathcal{CD}$ be a class diagram.
- As we (now) are able to recognise OCL constraints when we see them, we can define

$$\mathit{Inv}(\mathcal{CD})$$

as the set $\{\varphi_1, \ldots, \varphi_n\}$ of OCL constraints **occurring** in notes in $\mathcal{CD}$ — after **unfolding** all abbreviations (cf. next slides).

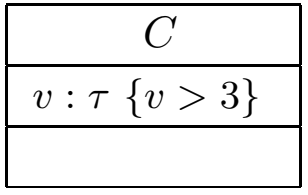
# *Invariants of a Class Diagram*

- Let $\mathcal{CD}$ be a class diagram.
- As we (now) are able to recognise OCL constraints when we see them, we can define

  $$\mathit{Inv}(\mathcal{CD})$$

  as the set $\{\varphi_1, \ldots, \varphi_n\}$ of OCL constraints **occurring** in notes in $\mathcal{CD}$ — after **unfolding** all abbreviations (cf. next slides).
- As usual: $\mathit{Inv}(\mathscr{CD}) := \bigcup_{\mathcal{CD} \in \mathscr{CD}} \mathit{Inv}(\mathcal{CD})$.
- **Principally clear**: $\mathit{Inv}(\,\cdot\,)$ for any kind of diagram.

# *Invariant in Class Diagram Example*




If $\mathscr{CD}$ consists of only $\mathcal{CD}$ with the single class $C$, then

- $\mathit{Inv}(\mathscr{CD}) = \mathit{Inv}(\mathcal{CD}) =$

# *Semantics of a Class Diagram*

**Definition.** Let $\mathscr{CD}$ be a set of class diagrams.

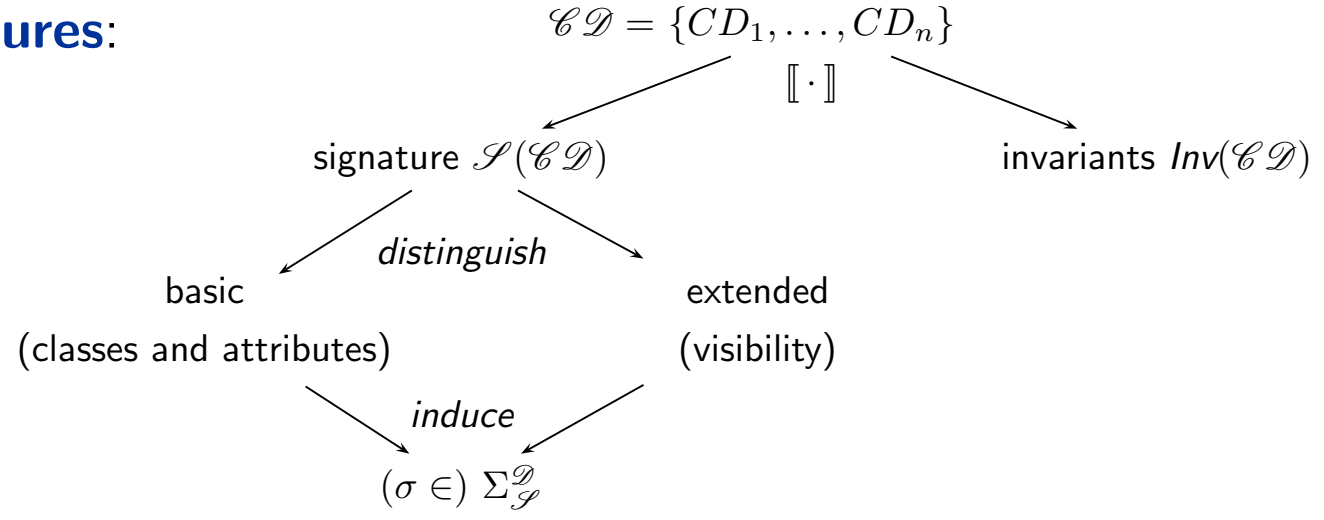
We say, the semantics of $\mathscr{CD}$ is the signature it induces and the set of OCL constraints occurring in $\mathscr{CD}$, denoted

$$[\![\mathscr{CD}]\!] := \langle \mathscr{S}(\mathscr{CD}), \mathit{Inv}(\mathscr{CD}) \rangle.$$

Given a structure $\mathscr{D}$ of $\mathscr{S}$ (and thus of $\mathscr{CD}$), the class diagrams describe the system states $\Sigma^{\mathscr{D}}_{\mathscr{S}}$. Of those, **some** satisfy $\mathit{Inv}(\mathscr{CD})$ and some don't.

We call a system state $\sigma \in \Sigma^{\mathscr{D}}_{\mathscr{S}}$ consistent if and only if $\sigma \models \mathit{Inv}(\mathscr{CD})$.

**In pictures**:



# *Pragmatics*

**Recall**: a UML **model** is an image or pre-image of a software system.

A set of class diagrams $\mathscr{CD}$ with invariants $Inv(\mathscr{CD})$ describes the **structure** of system states.

Together with the invariants it can be used to state:

- **Pre-image**: Dear programmer, please provide an implementation which uses only system states that satisfy $Inv(\mathscr{CD})$.
- **Post-image**: Dear user/maintainer, in the existing system, only system states which satisfy $Inv(\mathscr{CD})$ are used.

(The exact meaning of "use" will become clear when we study behaviour — intuitively: the system states that are reachable from the initial system state(s) by calling methods or firing transitions in state-machines.)

# *Pragmatics*

**Recall**: a UML **model** is an image or pre-image of a software system.

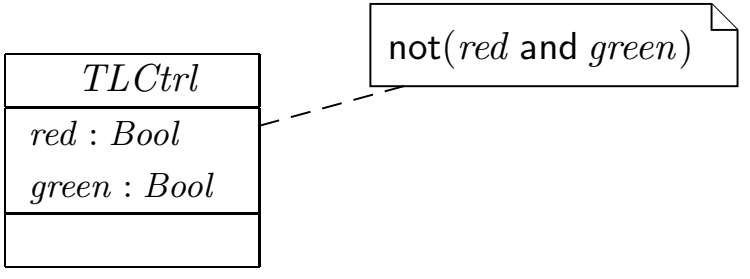
A set of class diagrams $\mathscr{CD}$ with invariants $\mathit{Inv}(\mathscr{CD})$ describes the **structure** of system states.

Together with the invariants it can be used to state:

- **Pre-image**: Dear programmer, please provide an implementation which uses only system states that satisfy $\mathit{Inv}(\mathscr{CD})$.
- **Post-image**: Dear user/maintainer, in the existing system, only system states which satisfy $\mathit{Inv}(\mathscr{CD})$ are used.

(The exact meaning of "use" will become clear when we study behaviour — intuitively: the system states that are reachable from the initial system state(s) by calling methods or firing transitions in state-machines.)

**Example**: highly abstract model of traffic lights controller.



# *Constraints vs. Types*

**Find the 10 differences**:

| $C$ |
| --- |
| $x : Int\ \{x = 3 \vee x > 17\}$ |
| |

| $C$ |
| --- |
| $x : T$ |
| |

$$\mathscr{D}(T) = \{3\} \cup \{n \in \mathbb{N} \mid n > 17\}$$

- $x = 4$ is well-typed in the left context,
  a system state satisfying $x = 4$ violates the constraints of the diagram.
- $x = 4$ is not even well-typed in the right context,
  there cannot be a system state with $\sigma(u)(x) = 4$ because $\sigma(u)(x)$ is supposed to be in $\mathscr{D}(T)$ (by definition of system state).

# *Constraints vs. Types*

**Find the 10 differences**:

| $C$ |
| --- |
| $x : Int\ \{x = 3 \vee x > 17\}$ |
| |

| $C$ |
| --- |
| $x : T$ |
| |

$$\mathscr{D}(T) = \{3\} \cup \{n \in \mathbb{N} \mid n > 17\}$$

- $x = 4$ is well-typed in the left context,
  a system state satisfying $x = 4$ violates the constraints of the diagram.
- $x = 4$ is not even well-typed in the right context,
  there cannot be a system state with $\sigma(u)(x) = 4$ because $\sigma(u)(x)$ is supposed to be in $\mathscr{D}(T)$ (by definition of system state).

**Rule-of-thumb**:

- If something **"feels like" a type** (one criterion: has a natural correspondence in the application domain), then make it a type.
- If something is a **requirement** or restriction of an otherwise useful type, then make it a constraint.

# *Design Guidelines for (Class) Diagram*

*(partly following [Ambler, 2005])*

*Be careful whose advice you buy, but, be patient with those who supply it.*

*Baz Luhrmann/Mary Schmich*

# *Main and General Modelling Guideline* *(admittedly: trivial and obvious)*

Be good to your audience.

# *Main and General Modelling Guideline* *(admittedly: trivial and obvious)*

Be good to your audience.

"Imagine you're given **your** diagram $\mathcal{D}$ and asked to conduct task $\mathcal{T}$.

- Can you do $\mathcal{T}$ with $\mathcal{D}$?

  (semantics sufficiently clear? all necessary information available? ...)

- Does doing $\mathcal{T}$ with $\mathcal{D}$ cost you more nerves/time/money/... than it should?"

  (syntactical well-formedness? readability? intention of deviations from standard syntax clear? reasonable selection of information? layout? ...)

# *Main and General Modelling Guideline* *(admittedly: trivial and obvious)*

Be good to your audience.

"Imagine you're given **your** diagram $\mathcal{D}$ and asked to conduct task $\mathcal{T}$.

- Can you do $\mathcal{T}$ with $\mathcal{D}$?
  (semantics sufficiently clear? all necessary information available? ...)
- Does doing $\mathcal{T}$ with $\mathcal{D}$ cost you more nerves/time/money/... than it should?"
  (syntactical well-formedness? readability? intention of deviations from standard syntax clear? reasonable selection of information? layout? ...)

In other words:

- the things **most relevant** for $\mathcal{T}$, do they **stand out** in $\mathcal{D}$?
- the things **less relevant** for $\mathcal{T}$, do they **disturb** in $\mathcal{D}$?

# *Main and General Quality Criterion* *(again: trivial and obvious)*

- **Q:** When is a (class) diagram a good diagram?

# *Main and General Quality Criterion* *(again: trivial and obvious)*

- **Q:** When is a (class) diagram a good diagram?
- **A:** If it serves its purpose/makes its point.

# *Main and General Quality Criterion* *(again: trivial and obvious)*

- **Q:** When is a (class) diagram a good diagram?
- **A:** If it serves its purpose/makes its point.

**Examples** for purposes and points and rules-of-thumb:

- **Analysis/Design**

# *Main and General Quality Criterion* *(again: trivial and obvious)*

- **Q:** When is a (class) diagram a good diagram?
- **A:** If it serves its purpose/makes its point.

**Examples** for purposes and points and rules-of-thumb:

- **Analysis/Design**
  - realizable, no contradictions
  - abstract, focused, admitting degrees of freedom for (more detailed) design
  - platform independent – as far as possible but not (artificially) farer
- **Implementation/A**

# *Main and General Quality Criterion* *(again: trivial and obvious)*

- **Q:** When is a (class) diagram a good diagram?
- **A:** If it serves its purpose/makes its point.

**Examples** for purposes and points and rules-of-thumb:

- **Analysis/Design**
  - realizable, no contradictions
  - abstract, focused, admitting degrees of freedom for (more detailed) design
  - platform independent – as far as possible but not (artificially) farer
- **Implementation/A**
  - close to target platform
    ($C_{0,1}$ is easy for Java, $C_*$ comes at a cost — other way round for RDB)
- **Implementation/B**

# *Main and General Quality Criterion* *(again: trivial and obvious)*

- **Q:** When is a (class) diagram a good diagram?
- **A:** If it serves its purpose/makes its point.

**Examples** for purposes and points and rules-of-thumb:

- **Analysis/Design**
  - realizable, no contradictions
  - abstract, focused, admitting degrees of freedom for (more detailed) design
  - platform independent – as far as possible but not (artificially) farer
- **Implementation/A**
  - close to target platform
    ($C_{0,1}$ is easy for Java, $C_*$ comes at a cost — other way round for RDB)
- **Implementation/B**
  - complete, executable
- **Documentation**

# *Main and General Quality Criterion* *(again: trivial and obvious)*

- **Q:** When is a (class) diagram a good diagram?
- **A:** If it serves its purpose/makes its point.

**Examples** for purposes and points and rules-of-thumb:

- **Analysis/Design**
  - realizable, no contradictions
  - abstract, focused, admitting degrees of freedom for (more detailed) design
  - platform independent – as far as possible but not (artificially) farer
- **Implementation/A**
  - close to target platform
    ($C_{0,1}$ is easy for Java, $C_*$ comes at a cost — other way round for RDB)
- **Implementation/B**
  - complete, executable
- **Documentation**
  - Right level of abstraction: "if you've only one diagram to spend, illustrate the concepts, the architecture, the difficult part"
  - The more detailed the documentation, the higher the probability for regression
    "outdated/wrong documentation is worse than none"

# *General Diagramming Guidelines [Ambler, 2005]*

(Note: “Exceptions prove the rule.”)

- **2.1 Readability**
  - 1.–3. Support Readability of Lines

# *General Diagramming Guidelines [Ambler, 2005]*

(Note: “Exceptions prove the rule.”)

- **2.1 Readability**
  - 1.–3. Support Readability of Lines
  - 4. Apply Consistently Sized Symbols

# *General Diagramming Guidelines [Ambler, 2005]*

(Note: “Exceptions prove the rule.”)

- **2.1 Readability**
  - 1.–3. Support Readability of Lines
  - 4. Apply Consistently Sized Symbols
  - 9. Minimize the Number of Bubbles

# *General Diagramming Guidelines [Ambler, 2005]*

(Note: “Exceptions prove the rule.”)

- **2.1 Readability**
  - 1.–3. Support Readability of Lines
  - 4. Apply Consistently Sized Symbols
  - 9. Minimize the Number of Bubbles
  - 10. Include White-Space in Diagrams

# *General Diagramming Guidelines [Ambler, 2005]*

(Note: "Exceptions prove the rule.")

- **2.1 Readability**
  - 1.–3. Support Readability of Lines
  - 4. Apply Consistently Sized Symbols
  - 9. Minimize the Number of Bubbles
  - 10. Include White-Space in Diagrams
  - 13. Provide a Notational Legend

# *General Diagramming Guidelines [Ambler, 2005]*

- **2.2 Simplicity**
  - 14. Show Only What You Have to Show
  - 15. Prefer Well-Known Notation over Exotic Notation
  - 16. Large vs. Small Diagrams
  - 18. Content First, Appearance Second

# *General Diagramming Guidelines [Ambler, 2005]*

- **2.2 Simplicity**
  - 14. Show Only What You Have to Show
  - 15. Prefer Well-Known Notation over Exotic Notation
  - 16. Large vs. Small Diagrams
  - 18. Content First, Appearance Second
- **2.3 Naming**
  - 20. Set and (23. Consistently) Follow Effective Naming Conventions

# *General Diagramming Guidelines [Ambler, 2005]*

- **2.2 Simplicity**
  - 14. Show Only What You Have to Show
  - 15. Prefer Well-Known Notation over Exotic Notation
  - 16. Large vs. Small Diagrams
  - 18. Content First, Appearance Second
- **2.3 Naming**
  - 20. Set and (23. Consistently) Follow Effective Naming Conventions
- **2.4 General**
  - 24. Indicate Unknowns with Question-Marks
  - 25. Consider Applying Color to Your Diagram
  - 26. Apply Color Sparingly

# *Class Diagram Guidelines [Ambler, 2005]*

- **5.1 General Guidelines**
  - 88. Indicate Visibility Only on Design Models **(in contrast to analysis models)**

# *Class Diagram Guidelines [Ambler, 2005]*

- **5.1 General Guidelines**
  - 88. Indicate Visibility Only on Design Models **(in contrast to analysis models)**
- **5.2 Class Style Guidelines**
  - 96. Prefer Complete Singular Nouns for Class Names
  - 97. Name Operations with Strong Verbs
  - 99. Do Not Model Scaffolding Code **[Except for Exceptions]**

# *Class Diagram Guidelines [Ambler, 2005]*

- **5.2 Class Style Guidelines**
  - 103. Never Show Classes with Just Two Compartments
  - 104. Label Uncommon Class Compartments
  - 105. Include an Ellipsis (...) at the End of an Incomplete List
  - 107. List Operations/Attributes in Order of Decreasing Visibility

# *Class Diagram Guidelines [Ambler, 2005]*

- **5.3 Relationships**
  - 112. Model Relationships Horizontally
  - 115. Model a Dependency When the Relationship is Transitory
  - 117. Always Indicate the Multiplicity
  - 118. Avoid Multiplicity "$*$"
  - 119. Replace Relationship Lines with Attribute Types

# *Class Diagram Guidelines [Ambler, 2005]*

- **5.4 Associations**
  - 127. Indicate Role Names When Multiple Associations Between Two Classes Exist
  - 129. Make Associations Bidirectional Only When Collaboration Occurs in Both Directions
  - **131. Avoid Indicating Non-Navigability**
  - 133. Question Multiplicities Involving Minimums and Maximums

# *Class Diagram Guidelines [Ambler, 2005]*

- **5.4 Associations**
  - 127. Indicate Role Names When Multiple Associations Between Two Classes Exist
  - 129. Make Associations Bidirectional Only When Collaboration Occurs in Both Directions
  - **131. Avoid Indicating Non-Navigability**
  - 133. Question Multiplicities Involving Minimums and Maximums
- **5.6 Aggregation and Composition**
  - $\rightarrow$ exercises

*[...] But trust me on the sunscreen.*

*Baz Luhrmann/Mary Schmich*

# *Example: Modelling Games*

# *Task: Game Development*

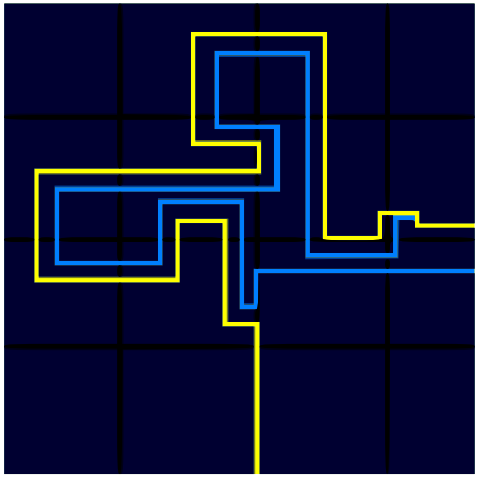
**Task**: **develop** a **video game**. **Genre**: **Racing**. **Rest**: **open**, i.e.

| Degrees of freedom: | |
|---|---|
| • simulation vs. arcade<br>• platform (SDK or not, open or proprietary, hardware capabilities...)<br>• graphics (3D, 2D, ...)<br>• number of players, AI<br>• controller<br>• game experience | |

# *Task: Game Development*

**Task**: **develop** a **video game**. **Genre**: **Racing**. **Rest**: **open**, i.e.

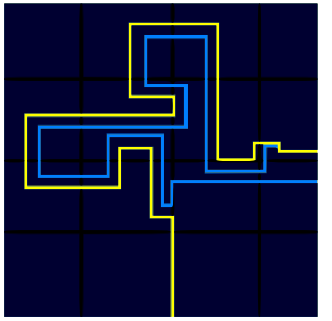
| Degrees of freedom: | Exemplary choice: 2D-Tron |
|---|---|
| • simulation vs. arcade | arcade |
| • platform (SDK or not, open or proprietary, hardware capabilities...) | open |
| • graphics (3D, 2D, ...) | 2D |
| • number of players, AI | min. 2, AI open |
| • controller | open (later determined by platform) |
| • game experience | minimal: main menu and game |



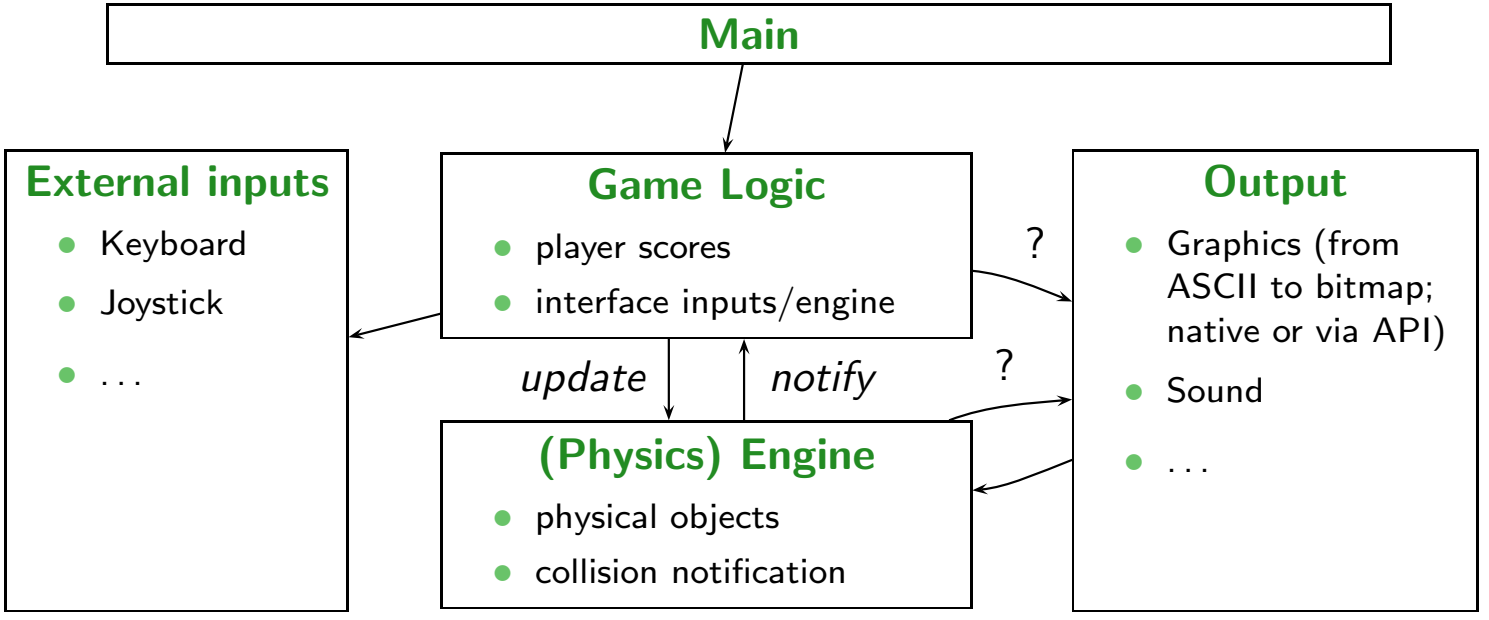
# *Modelling Structure: 2D-Tron*

2D-Tron

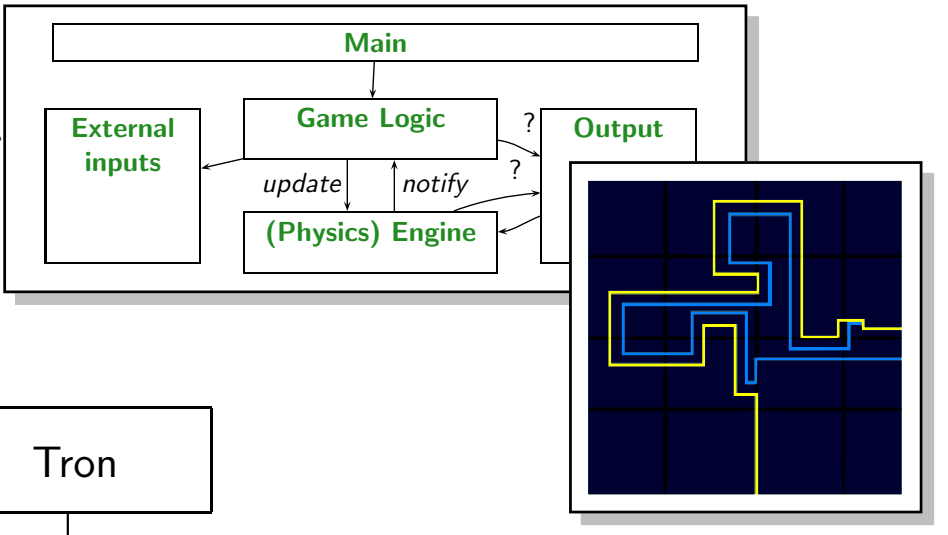
- arcade
- platform open
- 2D
- min. 2, AI open
- controller open
- only game, no menues



- In many domains, there are canonical architectures – and adept readers try to see/find/match this!
- For games:



# Modelling Structure: 2D-Tron




Tron

Joystick?

. . .

Keyboard?

1..*

Control

Player

colour
score
direction
speed

1..*

Gameplay

update

notify

Render

OpenGL?

. . .

aalib?

head

Segment

x0, y0
x1, y1
colour

0..*

Engine

areawidth
areaheight

world

AI?

Conventions:

- default $\xi$ is $1$

# *References*

# References

[Ambler, 2005] Ambler, S. W. (2005). *The Elements of UML 2.0 Style*. Cambridge University Press.

[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.