- Associations: The Rest
  - Visibility, Navigability, Properties,
  - Ownership, “Diamonds”,
  - Multiplicity

- Back to the Main Track

- OCL in (Class) Diagrams
  - What makes a class diagram a good class diagram?
    - Web-Shop Examples
    - The Elements of UML 2.0 Style
    - Example: Game Architecture
Associations: The Rest
\textit{CD}:

\begin{align*}
\text{role}_1 & \quad 0..1
\text{v} : \text{Int} \\
\text{role}_3 & \quad 3..*
\end{align*}

context \( C' \) \( \text{inv} : \text{role}_3 \rightarrow \text{size} \geq 3 \)
Recall: Multiplicity is a term of the form $N_1..N_2, \ldots, N_{2k-1}..N_{2k}$ where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $N_1, \ldots, N_{2k-1} \in \mathbb{N}$, $N_{2k} \in \mathbb{N} \cup \{\ast\}$.

Define $\mu^C_{OCL}(\text{role}) :=$

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

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

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

with $\text{role} \neq \text{role}'$, if $\mu \neq 0..1, \mu \neq 1..1$, and

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

if $\mu = 1..1$.

Note: in $n$-ary associations with $n > 2$, there is redundancy.
Multiplicities as Constraints Example

\[ \mu^{C}_{OCL}(role) = \text{context } C \text{ inv :} \]
\[ (N_1 \leq role \rightarrow size() \leq N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \leq role \rightarrow size() \leq N_{2k}) \]
\[ \mu^{C}_{OCL}(role) = \text{context } C \text{ inv : not(oclIsUndefined(role))} \]

\[ CD : \]

- \{ \text{context } C \text{ inv : } 3 \leq role_3 \rightarrow size() \leq 4 \text{ or } 17, 1 \leq role_2 \rightarrow size() \leq 17 \}
- \{ \text{context } C \text{ inv : role}_2 \rightarrow size() = 4 \text{ or } role_2 \rightarrow size() = 17 \}
- \{ \text{context } C \text{ inv : } 4 \leq role_2 \rightarrow size() \leq 17 \}

\[ \begin{array}{c}
\text{role}_1 \quad 0..1 \\
\text{C} \\
v : \text{Int} \\
\text{role}_2 \quad 4, 17 \\
\text{role}_3 \quad 3..\ast \\
\end{array} \]

\[ \{ \text{context } C \text{ inv : } 3 \leq role_3 \rightarrow size() \leq 4 \text{ or } 17 \leq role_2 \rightarrow size() \leq 17 \} \]
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

\[
\begin{align*}
(i) & \quad C & \times & 0..1 & \longrightarrow & \bullet & D \\
(ii) & \quad C & \times & ^* & \longrightarrow & \bullet & D \\
\end{align*}
\]

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

- Form (i) introduces \( role \in \mathcal{D}_{0,1} \), and form (ii) introduces \( role \in \mathcal{D}_* \) in the set of attributes \( V \).

- In both cases, \( role \in \text{atr}(C) \).

- **We drop \( \lambda \) and go back to our nice \( \sigma \) with \( \sigma(u)(role) \subseteq \mathcal{D}(D) \).**
OCL Constraints in (Class) Diagrams
Where Shall We Put OCL Constraints?

Three options:

(o) Separate document.

(i) Notes.

(ii) Particular dedicated places.

(i) Notes:

A UML note is a picture of the form

```
[ text ]
```

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

Sometimes, content is explicitly classified for clarity:

```
OCL:
  expr
```
stands for

context $C$ inv : $expr$
(ii) **Particular dedicated places** in class diagrams:  

\[
\begin{align*}
C & \quad \xi v : T \{ p_1, \ldots, p_n \} \{ expr \} \\
& \quad \xi f(v_1 : T, \ldots, v_n : T_n) : T \{ p_1, \ldots, p_n \} \{ \text{pre} : expr_1 \\
& \qquad \text{post} : expr_2 \}
\end{align*}
\]

For simplicity, we view the above as an abbreviation for

\[
C \quad \xi v : T \{ p_1, \ldots, p_n \} \\
\text{context } f \text{ pre} : expr_1 \text{ post} : expr_2
\]
Let $\mathcal{CD}$ be a class diagram.

We are (now) able to recognise OCL constraints when we see them, so define

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

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

As usual: consider all invariants in all notes in any class diagram – plus implicit multiplicity-induced invariants.

$$\text{Inv}(\mathcal{CD}) = \bigcup_{\mathcal{CD} \in \mathcal{CD}} \text{Inv}(\mathcal{CD}) \cup \{ \mu^C_{\text{OCL}}(\text{role}) \mid \langle r : \ldots, \langle \text{role} : D, \mu, _, _ ,_ ,_ \rangle, \ldots, \langle \text{role}' : C, _ ,_ ,_ ,_ \rangle, \ldots \rangle \in V \text{ or } \langle r : \ldots, \langle \text{role}' : C, _ ,_ ,_ ,_ \rangle, \ldots, \langle \text{role} : D, \mu, _ ,_ ,_ \rangle, \ldots \rangle \in V \}.$$ 

Analogously: $\text{Inv}(\cdot)$ for any kind of diagram (like state machine diagrams).
**Definition.** Let $\mathcal{CD}$ be a set of class diagrams. We say, the **semantics** of $\mathcal{CD}$ is the signature it induces and the set of OCL constraints occurring in $\mathcal{CD}$, denoted

$$[[\mathcal{CD}]] := \langle \mathcal{I}(\mathcal{CD}), \text{inv}(\mathcal{CD}) \rangle.$$

Given a structure $\mathcal{D}$ of $\mathcal{I}$ (and thus of $\mathcal{CD}$), the class diagrams **describe** the system states $\Sigma^\mathcal{D}$, of which **some** may satisfy $\text{inv}(\mathcal{CD})$.

**In pictures:**

$$\mathcal{CD} = \{CD_1, \ldots, CD_n\}$$

- **Signature**: $\mathcal{I}(\mathcal{CD})$
- **Invariants**: $\text{inv}(\mathcal{CD})$
- **Basic**: (classes and attributes)
- **Extended**: (visibility, etc.)
- **Distinguish**
Recall: a UML model is an image or pre-image of a software system.

A set of class diagrams $\mathcal{CD}$ describes the structure of system states. Together with the invariants $\text{Inv}(\mathcal{CD})$ it can be used to state:

- **Pre-image**: Dear programmer, please provide an implementation which uses only system states that satisfy $\text{Inv}(\mathcal{CD})$.

- **Post-image**: Dear user/maintainer, in the existing system, only system states which satisfy $\text{Inv}(\mathcal{CD})$ 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.

```
<table>
<thead>
<tr>
<th>TLC</th>
<th>not(red and green)</th>
</tr>
</thead>
<tbody>
<tr>
<td>red  : Bool</td>
<td></td>
</tr>
<tr>
<td>green : Bool</td>
<td></td>
</tr>
</tbody>
</table>
```
CD, SM

\[ \mathcal{S} = (T, \mathcal{C}, V, \text{atr}), SM \]

\[ M = (\Sigma_{\mathcal{S}}, A_{\mathcal{S}}, \rightarrow_{SM}) \]

\[ \pi = (\sigma_0, \varepsilon_0) \xrightarrow{(\text{cons}_0, \text{Snd}_0)} u_0 (\sigma_1, \varepsilon_1) \cdots \]

\[ w_{\pi} = ((\sigma_i, \text{cons}_i, \text{Snd}_i))_{i \in \mathbb{N}} \]

\[ \varphi \in \text{OCL} \]

\[ G = (N, E, f) \]

\[ \mathcal{O} \]

\[ CD, SD \]

\[ B = (Q_{SD}, q_0, A_{\mathcal{S}}, \rightarrow_{SD}, F_{SD}) \]

\[ \mathcal{G} = (N, E, f) \]

\[ \mathcal{D} \]
Design Guidelines for (Class) Diagram

(partly following Ambler (2005))
Some Web-Shop Class Diagrams
A Closer Look

\[ \exists n : \langle \text{itsBackEnd : BackEnd}, +, \ldots \rangle, \langle \ldots \rangle \]
A Closer Look
A Closer Look

Diagram:
- FrontEnd
- BackEnd
- Session
  - id: String
- Item
  - name: String
  - price: float
  - in_stock: int
  - reserved: int

Relationships:
- FrontEnd to Session: 0..* (s)
- BackEnd to Session: 0..* (r)
- BackEnd to FrontEnd: 0..* (m)
- Item to Session: 0..* (r)
- Item to Session: 0..* (k)
A Closer Look

![UML Diagram]

1. **FrontEnd**
   - id: char* {readOnly}

2. **Session**
   - ss *
   - itemsSS *
   - itemsBE 1..*

3. **Item**
   - name: char* {readOnly}
   - price: float {readOnly}
   - in_stock: int {readOnly}
   - reserved: int {readOnly}

Note: The text "write as default" is circling a connection point on the diagram.
A Closer Look
A Closer Look

```
FrontEnd

Session
  id:OMString

BackEnd
  itsBackEnd, 0,1

Item
  name:OMString
  price:double
  in_stock:int
  reserved:int

itsSessions, *

itsBackEnd, 0,1

itsItems
```
Some Web-Shop Class Diagrams
So: what makes a class diagram a good class diagram?
Be good to your audience.

“Imagine you’re given your diagram $D$ and asked to conduct task $T$.

- Can you do $T$ with $D$?
  (semantics sufficiently clear? all necessary information available? ...)

- Does doing $T$ with $D$ cost you more nerves/time/money/... than it should?”

In other words:

- the things most relevant for task $T$, do they stand out in $D$?
- the things less relevant for task $T$, do they disturb in $D$?
Main and General Quality Criterion

- **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) farther

- **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”
2.1 Readability

1.–3. Support Readability of Lines

4. Apply Consistently Sized Symbols

9. Minimize the Number of “Bubbles” / Things

10. Include White-Space in Diagrams

13. Provide a Notational Legend

(Note: “Exceptions prove the rule.”)
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
2.2 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
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]

\[\text{e.g. get/set methods}\]
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

(from + to −)
5.3 Relationships

112. Model Relationships Horizontally

115. Model a Dependency When the Relationship is Transitory

117. Always Indicate the Multiplicity
    (or have good defaults)

118. Avoid Multiplicity “∗”

119. Replace Relationship Lines with Attribute Types
    (to have fewer lines)
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

→ exercises
Tell Them What You’ve Told Them...

- **Associations:**
  - view multiplicities as shorthand for constraints,

- **OCL constraints** can be added to a class diagram in notes or at dedicated places.

- The semantics of a class diagram is its (extended) signature, and a set of (explicit and implicit) OCL constraints.

- **Class Diagrams** can be “drawn” well or not so well.

- A diagram is a good diagram if it serves its purpose.

- Purposes (for class diagrams):
  - Documentation of the top-level architecture.
  - Documentation of the structural design decisions.
  - Details can go into comments in the code.

- **Ambler (2005): The Elements of UML 2.0 Style.**
References
References

