Modeling Guidance (Part 1)

Software Design (40007) – 2024/2025

Justus Bogner
Ivano Malavolta

VU VRIJE UNIVERSITEIT AMSTERDAM

Software and Sustainability research group (S2)
Department of Computer Science, Faculty of Sciences

#1: Aim for simplicity

- UML has many complex constructs that you probably won’t benefit from that much
- Try to aim for simple, clear modeling that is easily understandable
- Examples:
  - Don’t use as many UML constructs as possible just because they exist (they should add a clear benefit for your model)
  - Avoid very complicated inheritance hierarchies
  - Avoid very complex n-ary associations
- **Note:** simplicity does **not** mean trivial, very abstract, vastly incomplete, etc.

#2: Consistency is key

- Try to be as consistent as possible in your modeling
- Use the same constructs in the same way
- Use consistent conventions for identifier naming styles, i.e., adhere to the default conventions for Java
  - Use `CapitalizedCamelCase` for class names
  - Use `camelCase` for attribute and operation names
  - Use `ALL_CAPS_SNAKE_CASE` for constants
  - Use `lowercasewithnoseparators` for package names
- Refer to the same constructs always by the same name
- Only use a single modeling tool and the same visual style

#3: Assume common data types

- UML officially only knows limited pre-defined data types
    - `Boolean`
    - `String`
    - `Integer`
    - `UnlimitedNatural`
- But: we design with a concrete programming language in mind (Java)
- For convenience: use common existing Java data types
- Examples: **`float`**, **`double`**, **`Date`**, **`List<?>`**, **`Map<?,?>`**, etc.
- Just be consistent in this usage!
- For more complex data types, it's best to model them explicitly

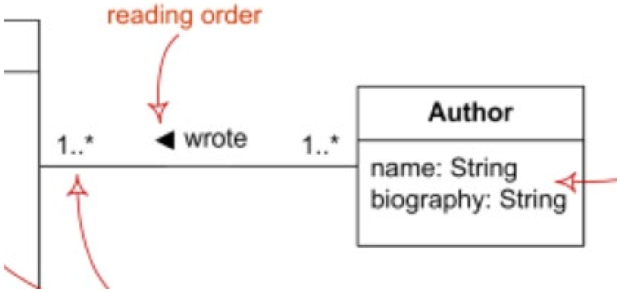
#4: Be precise in naming classes and attributes

- "There are 2 hard problems in computer science: cache invalidation, **naming things**, and off-by-1 errors."
  – Leon Bambrick [1]
- Try to be really precise with your names to make them understandable; is this the correct term in this domain?
- **But:** ensure a balance between conciseness and preciseness
- Make collections identifiable via their name, e.g., `List<Customer> customers`
- Pay special attention to numeric attributes with units
  - `float temperatureInCelsius`
  - `float priceInEuro` or `int priceInEuroCent`

[1] https://martinfowler.com/bliki/TwoHardThings.html

#5: Give relationships suitable names

- Make it easy to understand relationships by choosing suitable labels for each association in your class diagrams
- Phrasing should ideally align with the navigability arrow (“provides” vs. “is provided by”)
- Alternatively, use reading direction arrows with the label



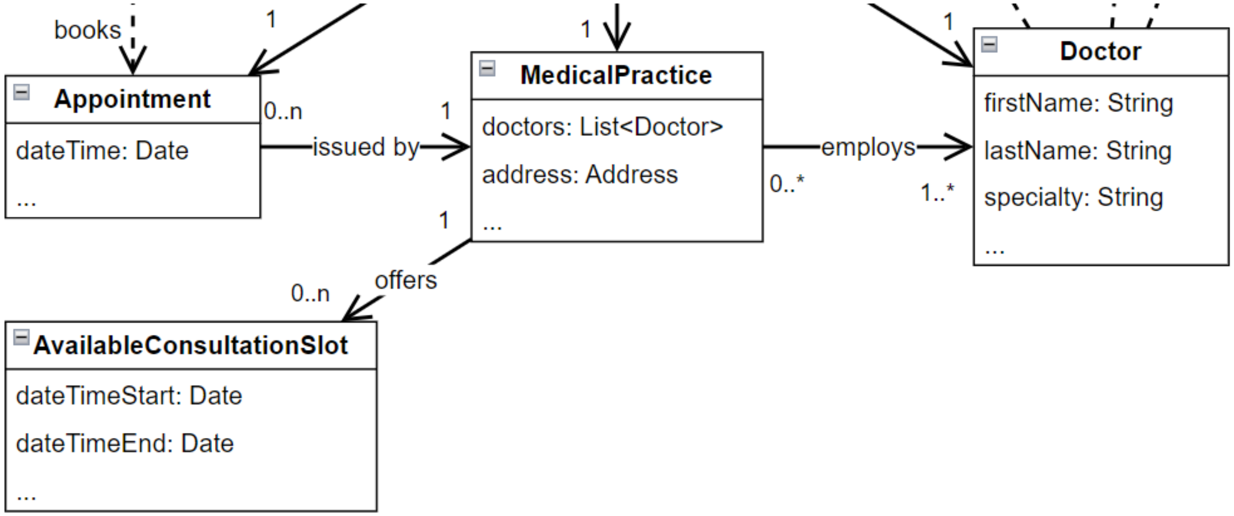
https://www.uml-diagrams.org/class-diagrams-overview.html

- Exceptions for omitting labels:
  - Aggregation and composition (“is a part of”)
  - Dependencies like `use` or `create` have standard names anyway [1]

[1] https://www.uml-diagrams.org/dependency.html?context=class-diagrams

#6: Make use of multiplicities

- UML knows no default values for multiplicities for class diagram relationships [1]
- If you don't specify them, it will usually be ambiguous
- Add them to all important relationships, especially associations, ideally everywhere



[1] https://www.uml-diagrams.org/multiplicity.html?context=class-diagrams

Conclusion

Key takeaways

- **UML** = general purpose modeling language, tailored to object-oriented software
- 1 UML model, many diagrams
- **Class diagrams** for describing main entities, data structures, and operations
- **Package diagrams** show the high-level module structure and dependencies between packages
- Remember the general **guidelines** for modeling!

Readings

- UML@Classroom: An Introduction to Object-Oriented Modeling", Chapters 2, 4, 9.1
- A Philosophy of Software Design, Chapters 4, 5, 6