Classes

Jana Kostičová

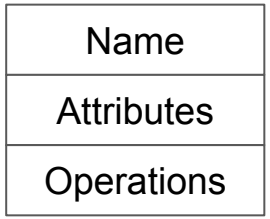
Class

= a template for creating objects

- It defines the **properties (attributes)** and **behaviors (methods, operations)** that objects (instances) of that class will have
- Classes are fundamental building blocks in software design, enabling modular and organized development. They help manage complexity and promote reusability across different parts of a system

Class in UML:

- A classifier whose features are attributes and operations
- Only class name is mandatory
    - Abstract class - the name is in italics or the textual annotation {abstract} is used after or below its name



UML Class diagram: modeling perspectives

**Three modeling perspectives according to Daniels[2002]*

Conceptual / domain model

- Represents the concepts in the domain
- Software is not yet modeled

Classes, possibly with attributes and high-level operations
Associations, multiplicities

"Specification" (logical application) model

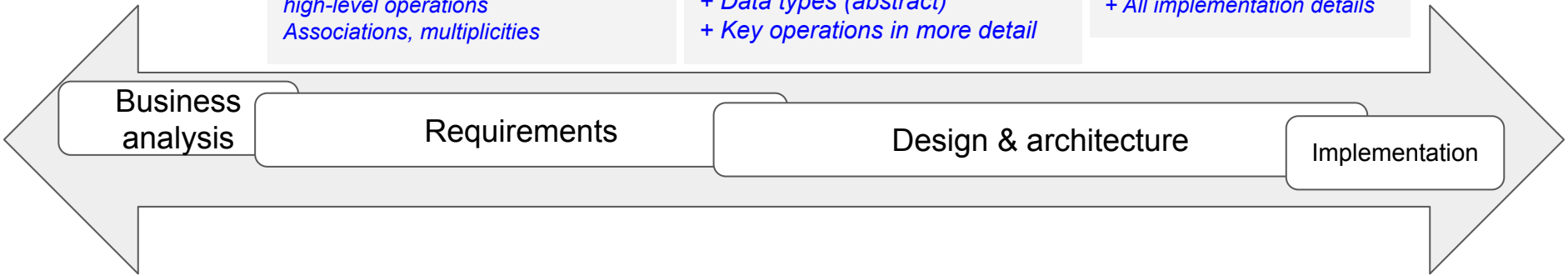
- Focused on software but still independent from implementation
- Captures data types (abstract)

+ Data types (abstract)
+ Key operations in more detail

Implementation model

- Represents specific implementation

+ All implementation details



- A perspective can be anything between the simplest form of conceptual model and the most detailed form of implementation model
- It is first necessary to determine which perspective will be used and to follow it consistently when modeling

Class attributes in UML (1)

[visibility] ['/'] name [':' type] ['[' multiplicity ']'] ['='default] ['{' prop-modifier [',' prop-modifier]* '}']

- Visibility
  - **'+'** public, **'-'** private, **'#'** protected, **'~'** package
- **'/'** marks derived property
  - = Property which can be computed from other properties
- Type
  - UML built-in primitive types: Boolean, Integer, UnlimitedNatural, String, Real
    - Their purpose is primarily to define the UML itself (they are used in metamodels)
  - User-defined data types (DataType)
    - Primitive types, structured types and enumerations
- Multiplicity
  - Positive number (0, 1, 2, …),
  - Interval: lower-bound '..' upper-bound; '*' for infinite upper bound
- Default
  - An expression for the default value(s) of the property

In UML, class attributes are represented by properties.

Commonly development teams have a convention that they can use standard data types from a particular programming language as data types in UML.

Class attributes in UML (2)

[visibility] [‘/’] name [‘:’ type] [‘[‘ multiplicity ‘]’] [‘=’default] [‘{‘ prop-modifier [‘,’ prop-modifier] ‘}’]*

- Property modifier

  ‘readOnly’: Read-only property
  ‘union’: Property is a derived union of its subsets
  ‘ordered’: Ordered property
  ‘unique’ / ‘nonunique’: Duplicates not allowed / allowed in a multi-valued property.

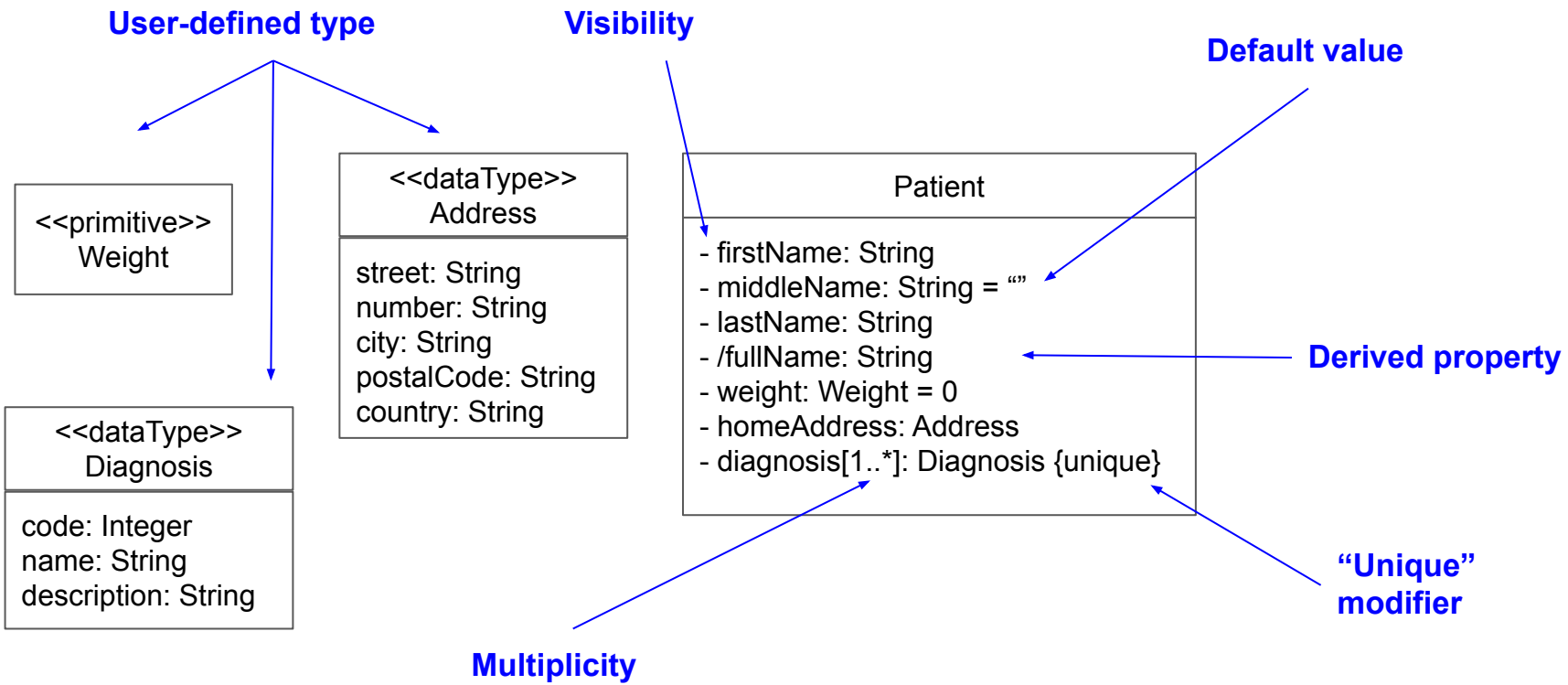
  ‘subsets’ property-name
  - Property is a proper subset of the property identified by property-name

  ‘redefines’ property-name
  - Property redefines an inherited property identified by property-name

  prop-constraint
  - An expression that specifies a constraint that applies to the property.

- Static attributes are <u>underlined</u>

Examples


User-defined type
Visibility
Default value
<<primitive>>
Weight
<<dataType>>
Address
street: String
number: String
city: String
postalCode: String
country: String
Patient
- firstName: String
- middleName: String = “”
- lastName: String
- /fullName: String
- weight: Weight = 0
- homeAddress: Address
- diagnosis[1..*]: Diagnosis {unique}
Derived property
<<dataType>>
Diagnosis
code: Integer
name: String
description: String
“Unique”
modifier
Multiplicity

Class operations in UML (1)

[visibility] name '(' [parameter-list] ')' *[':' return-spec]*

parameter-list ::= parameter[',' parameter]*

parameter ::= [direction] parm-name ':' type-expression['['multiplicity']'] ['=' default] ['{' parm-property[',' parm-property]* '}']

- Visibility
  - As for attributes:
  - **'+'** public, **'-'** private, **'#'** protected, **'~'** package
- Parameters
  - Direction: 'in', 'out', 'inout' (default = 'in')
  - Type-expression: specifies the type of the parameter
  - Multiplicity, default, parm-property: as for attributes

Class operations in UML (2)

[visibility] name '(' [parameter-list] ')' [':' return-spec]

return-spec ::= [return-type] ['[' multiplicity ']'] ['{' oper-property [',' oper-property]* '}']

- Return type
  - UML built-in primitive type or user-defined DataType / Class / Interface
- Operation properties (modifiers):

  'query' The operation does not change the state of the system

  'ordered' The values of the return parameter are ordered

  'unique' The values returned by parameters have no duplicates

  'redefines' oper-name:
  - The operation redefines an inherited operation identified by oper-name

  oper-constraint:
  - A constraint that applies to the operation.
- Static operations are <u>underlined</u>

Examples

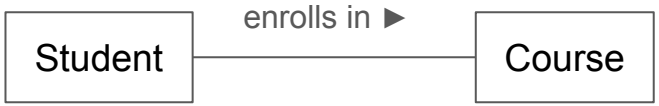
- display ()
- -hide ()
- +createWindow (location: Coordinates, container: Container [0..1]): Window
- +toString (): String

UML association

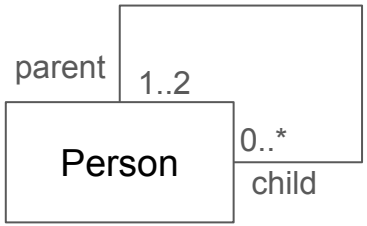
= a relationship between classes that indicates that there can be links between objects of given classes

- Association name (optional)
  - A solid triangle next to the association name indicates the order of association ends as well as the order of reading (used especially in conceptual modeling)
- Association ends
  - Association has at least two ends
  - Self-associations are allowed
  - Each association end is represented by a property - I.e., properties represent both class attributes and association ends
- Association vs. attribute
  - “A useful convention for general modeling scenarios is that a Property whose type is a kind of Class is an Association end, while a property whose type is a kind of DataType is not. This convention is not enforced by UML.” (*UML Specification 2.5.1)*
- Association and its ends may be derived; marked by ‘/’ before their names.

Examples

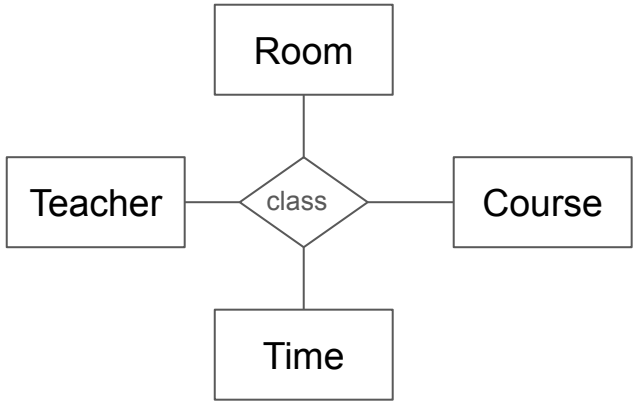


Association name with “►” symbol indicating the order of association ends and reading

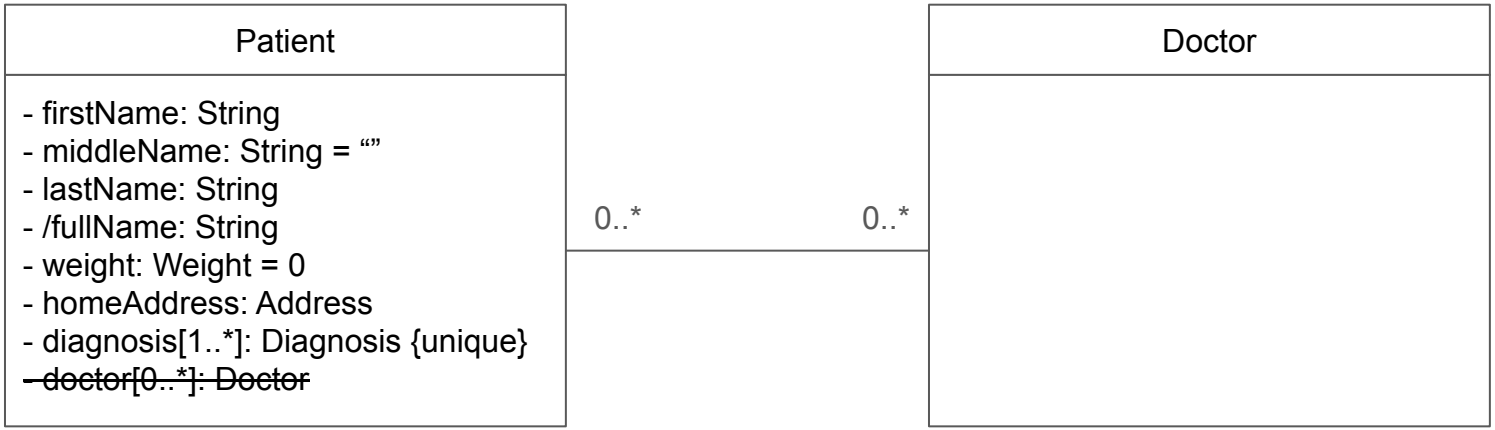


Self-association

4-ary association



Example - attribute vs association



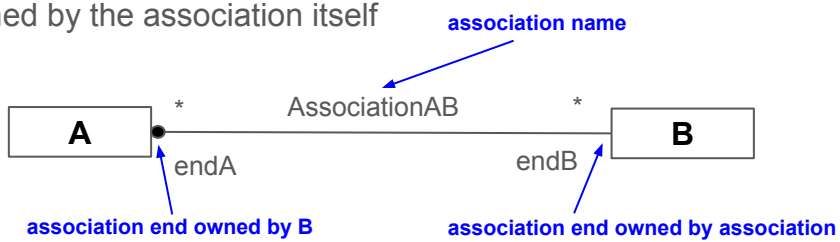
- String, Weight, Address and Diagnosis are dataTypes → attributes
- Doctor is class → association

Association end

- Association role name
  - It indicates what role the object of the given class plays within the relationship
- Multiplicity, visibility
  - As for attributes
- Aggregation kind
  - Aggregation vs composition, see later
- Property string (in curly braces)
  - As *property-modifier* for attributes

Association end - ownership and navigability

- **Ownership**
  - Association end can be owned by one of the connected classes or by the association itself
  - Indicated by a small circle (“dot” notation)
  - If not shown, the end is owned by the association itself



- **Navigability**

  - At runtime, objects of class B can be accessed efficiently from objects of class A
  - Association ends owned by classes are always navigable
  - Association ends owned by associations may be navigable or not

UML aggregation and composition

- Special types of associations
- They have an “aggregation kind” set:
  - Shared → aggregation
  - Composite → composition

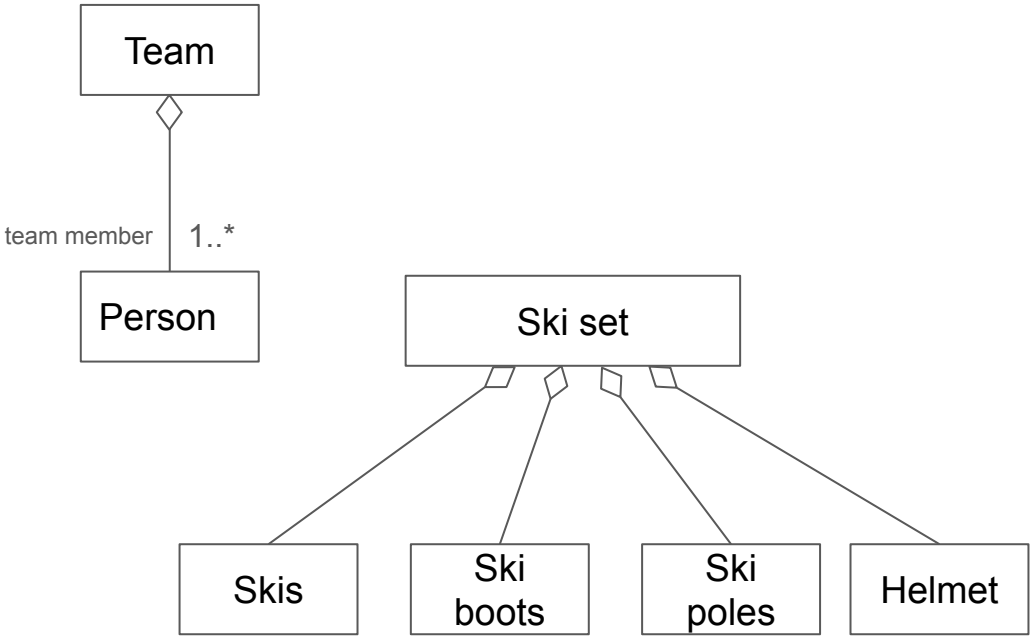
Aggregation

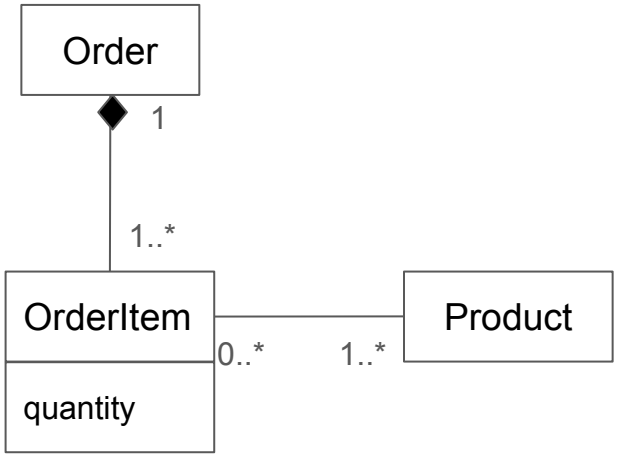
- Weak relationship between the whole and its parts - part can exist without the whole

Composition

- Strong relationship between the whole and its parts - part cannot exist without the whole
- If a composite is deleted, its parts are typically deleted as well

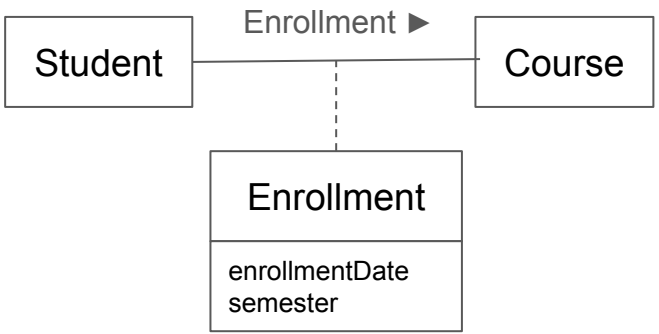
Examples


Team
team member
1..*
Person
Ski set
Skis
Ski boots
Ski poles
Helmet


Order
1
1..*
OrderItem
quantity
0..*
1..*
Product

Association class

- A class representing an association, allows adding attributes and operations to the association
- It has the same name as the association

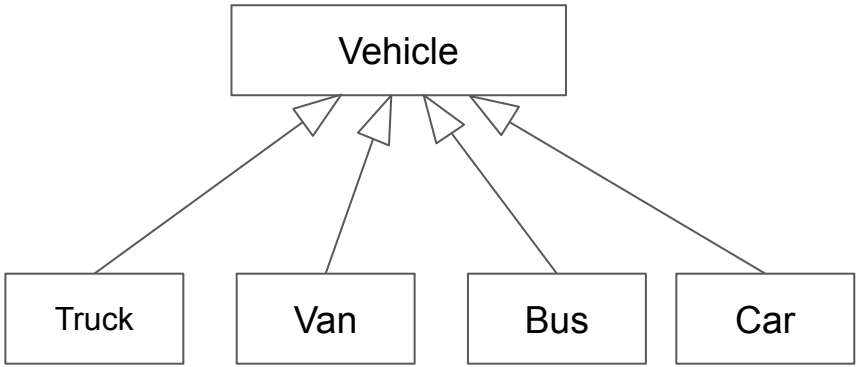


UML generalization

= the taxonomic relationship between a more general class and a more specific class

- The specific class inherits the features of the more general class
- A class can inherit from multiple general classes

Example:



Further reading

- R. Lukoťka: Domain analysis, modeling (PTS1)
- S. Ambler: UML Class Diagrams: An Agile Introduction
- M. Fowler: UML Distilled: A Brief Guide to the Standard Object Modeling Language 3rd Edition, 2003

References

- OMG. OMG Unified Modeling Language. Version 2.5.1, December 2017
- K. Fakhroutdinov: UML Class and Object Diagrams Overview
- R. Červenka, UML Classes
- J. Daniels: Modeling with a Sense of Purpose, 2002
- B. Selic: Getting It Right on the Dot, 2013