Université de Paris

Génie Logiciel
UML to model the structure

Sylvain Lobry

12/11/2021

Resources: www.sylvainlobry.com/GenieLogiciel

Menu of the day

1 Intro to class diagrams

2 Representing a class

3 Association between classes

4 Hierarchy

5 Representing objects

UML to model the structure

Granularity of objects

- Objects can be represented at different granularity.
- Important to choose the appropriate granularity for the purpose of the diagram:
  - Use case diagrams: high-level diagram made for discussion: low granularity objects
  - Class diagrams made for designing the program: high granularity objects

- Example: a laptop can be seen as:
  - a… laptop object (low granularity)
  - the composition of a chassis, trackpad, keyboard, screen, motherboard, CPU, RAM, … (high granularity)

UML to model the structure

Discovering objects

- Possible to discover objects through:
  - dynamic: by looking at which object should receive the message
  - data: by analyzing the structure of the object

- Example: objects in a laptop can be found:
  - dynamically: I want to execute a program: first I will move my cursor with the **trackpad** to the search bar; then I type the name of the program with the **keyboard**; then the **CPU** runs the program…
  - through data: when I look at the specification of my laptop, I can see that it has a intel i7 **CPU**, with 16Go of **RAM**, an AZERTY **keyboard**, …

UML to model the structure

Types of diagrams

- UML defines 13 diagrams in 3 categories which can define a system according to different points of view
- Structure diagrams
  - Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram and Deployment Diagram
- Behavior diagrams
  - Use Case Diagram, Activity Diagram and State Machine Diagram
- Interaction diagrams
  - Sequence Diagram, Communication Diagram, Timing Diagram and Interaction Overview Diagram

UML to model the structure

Types of diagrams

- UML defines 13 diagrams in 3 categories which can define a system according to different points of view
- Structure diagrams
    - Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram and Deployment Diagram
- Behavior diagrams
    - Use Case Diagram, Activity Diagram and State Machine Diagram
- Interaction diagrams
    - Sequence Diagram, Communication Diagram, Timing Diagram and Interaction Overview Diagram

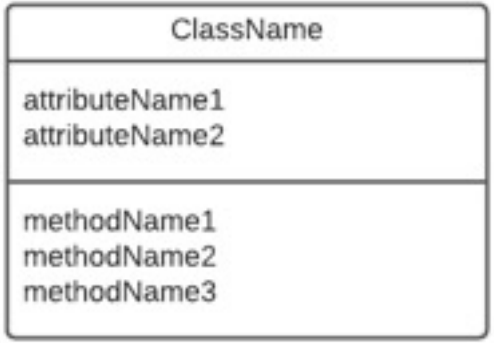
Menu of the day

1. Intro to class diagrams
2. Representing a class
3. Association between classes
4. Hierarchy
5. Representing objects

UML to model the structure

Representation of a class

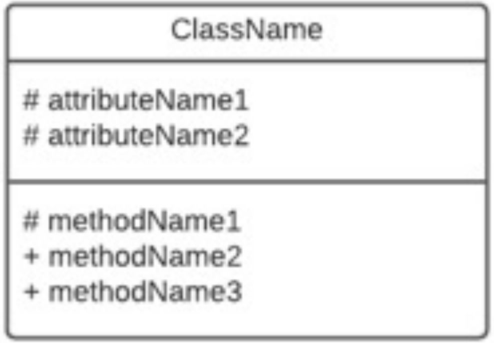
- In its simplest form, a class is represented as:
  - A name (always a noun, singular, represents the nature of the objects of this class)
  - A list of attributes
  - A list of methods
- Lacks information, but good for a first round of design



UML to model the structure

Encapsulation

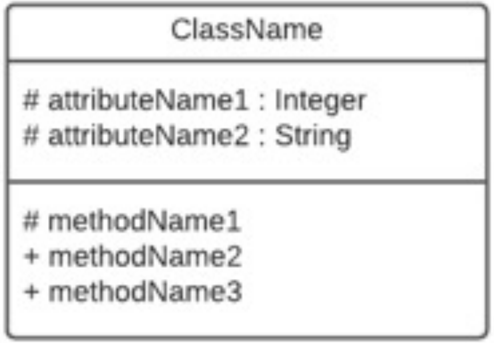
- Reminder: definition of encapsulation: to hide some attributes or methods to other objects
- In UML, elements can be qualified as:
    - public (sign: +): the element is not encapsulated; visible to everybody
    - protected (sign: #): the element is encapsulated and visible in specializations of this class
    - private (sign: -): the element is encapsulated (only visible inside the class)
    - package (sign: ~): the element is encapsulated and visible inside the package (we will not use that in this class)



UML to model the structure

Types

- In a class diagram, it is possible to specify the type of a variable
- The type can be one of the standard types:
  - Boolean
  - Integer
  - Real
  - String
- Or it can be a class (from the system or not)
- Indicated with “:” after the variable



ClassName

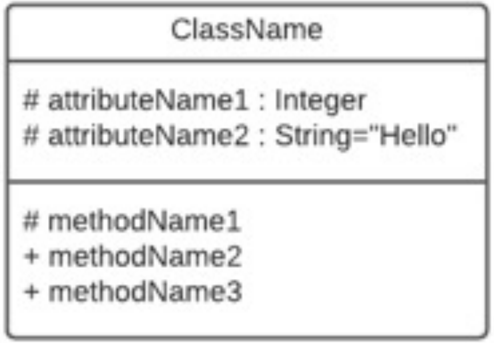
# attributeName1 : Integer
# attributeName2 : String

# methodName1
+ methodName2
+ methodName3

UML to model the structure

Default value

- An attribute of a class can have a default value. In this case, it will be indicated with “= default value”



ClassName

attributeName1 : Integer
attributeName2 : String="Hello"

methodName1
+ methodName2
+ methodName3

UML to model the structure

Cardinality

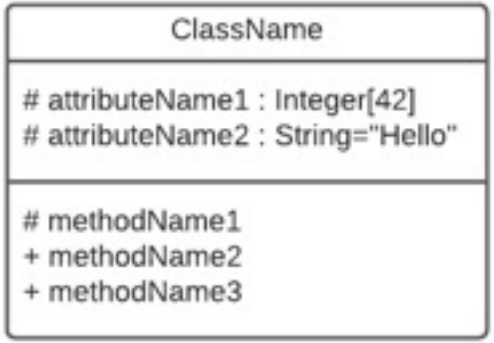
- A variable can have several values
- Indicated with cardinality
- Syntax: type[M..N]

Syntax	Cardinality
[1]	Exactly one time (default value, can be omitted)
[N]	Exactly N times
[*]	Any number (including 0) of times
[0..1]	0 or one time
[1..*]	One or more times
[M..N]	From M to N times

UML to model the structure

Cardinality

- A variable can have several values
- Indicated with cardinality
- Syntax: type[M..N]
- In a programming language, that would be done through an array, list, vector, …



ClassName

attributeName1 : Integer[42]
attributeName2 : String="Hello"

methodName1
+ methodName2
+ methodName3

UML to model the structure

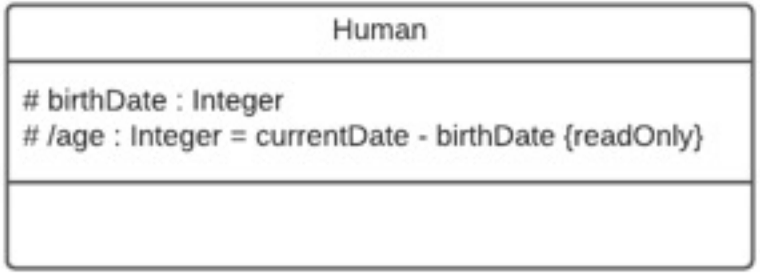
Modifiers

- A modifier can be used (it is optional) to give further information on a variable
- Syntax {modifier} or {m1, m2, …}
- Common ones:
  - id: the variable is part of the the identifier for the class
  - readOnly: variable cannot be modified
  - ordered (apply for cardinalities > 1): values of the variable should be ordered
  - unique (apply for cardinalities > 1): values of the variable should be unique (default!)
  - nonunique (apply for cardinalities > 1): values of the variable do not have to be unique
  - redefines “attribute name”: redefinition of “attribute name” from the superclass. If type is changed, the new type should be compatible with the old type.

UML to model the structure

Derivated attributes

- An attribute can be derived from other information (often: other attribute(s))
- indicated with “/” before the name
- Can be followed by an expression explaining how to compute the value
- Often “readOnly”



Human

# birthDate : Integer
# /age : Integer = currentDate - birthDate {readOnly}

UML to model the structure

Types 2

- To type a method, we will talk about its “signature”:
  - Name of the method
  - Name of the parameters, their types, their cardinalities and properties, default value
  - Result of the method: type, cardinality, property
- Syntax:

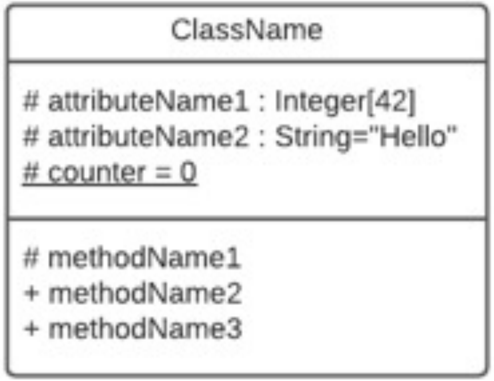
name (direction nameParam :type[inf..sup]=Default{modifiers}, …) : returnType[inf..sup]{modifiers}

- Direction can be:
  - in: value of the parameter is transmitted at call time
  - out: value of the parameter is transmitted at end of the execution of the method
  - inout: both

UML to model the structure

Class attributes

- One instance of a class = one instance of each attribute
- It is possible to have attributes that are linked to the class rather than an instance. In this case, we will talk about “class attributes”
- A class attribute is the same (name, value, type) for each instance. In general, it should have a default value.
- A class attribute is never inherited
- Is accessed through the class directly (not an instance)
- Syntax: underline the attribute.



ClassName

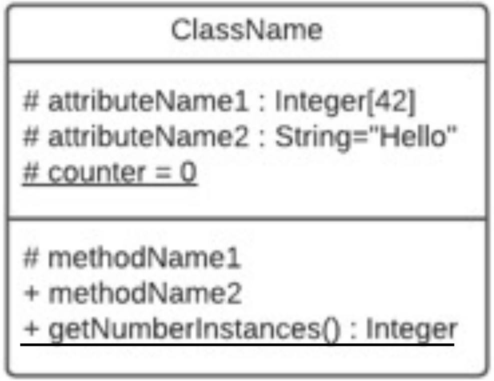
attributeName1 : Integer[42]
attributeName2 : String="Hello"
counter = 0

methodName1
+ methodName2
+ methodName3

UML to model the structure

Class methods

- Similarly, a method can be linked to a class, in which case we will talk about a “class method”
- Can only be accessed through the class itself, and can only use class attributes.



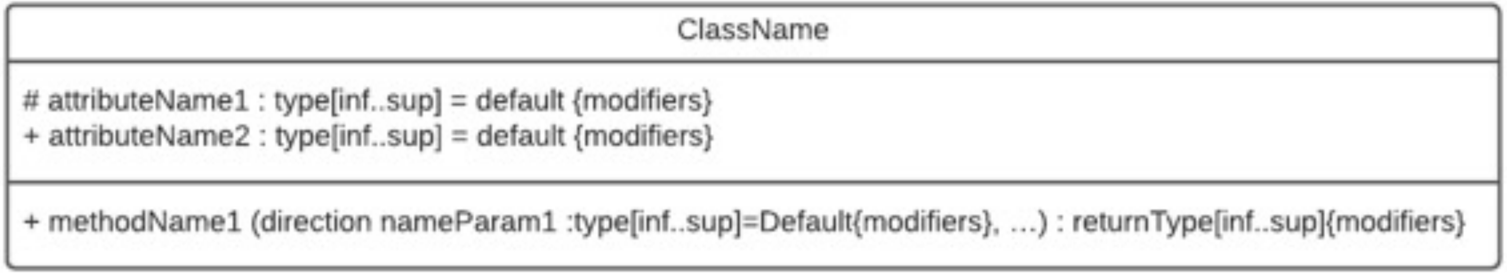
ClassName

attributeName1 : Integer[42]
attributeName2 : String="Hello"
counter = 0

methodName1
+ methodName2
+ getNumberInstances() : Integer

UML to model the structure

Representing a class - recap



ClassName

# attributeName1 : type[inf..sup] = default {modifiers}
+ attributeName2 : type[inf..sup] = default {modifiers}

+ methodName1 (direction nameParam1 :type[inf..sup]=Default{modifiers}, ...) : returnType[inf..sup]{modifiers}

Menu of the day

1 Intro to class diagrams

2 Representing a class

3 Association between classes

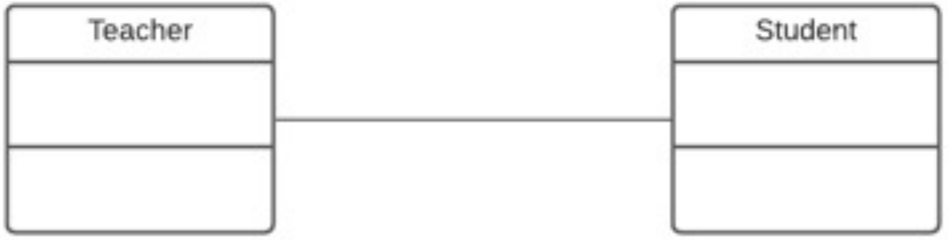
4 Hierarchy

5 Representing objects

UML to model the structure

Association between classes

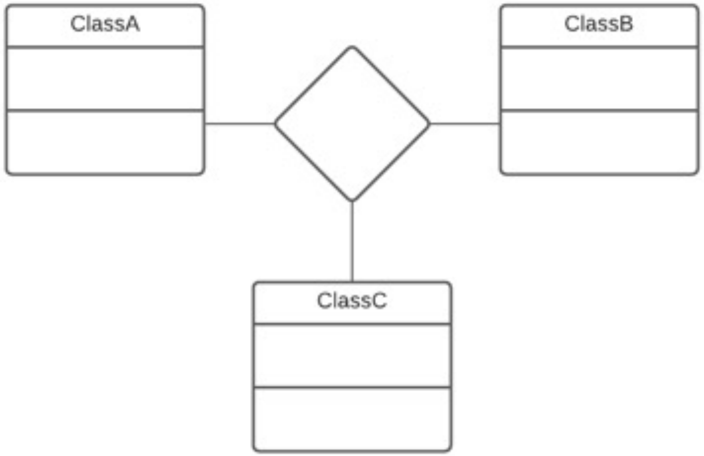
- An association can be established between two (or more) classes that are connected
- Shows how classes are linked through the system
- Syntax solid line (for binary relations)



UML to model the structure

Association between classes

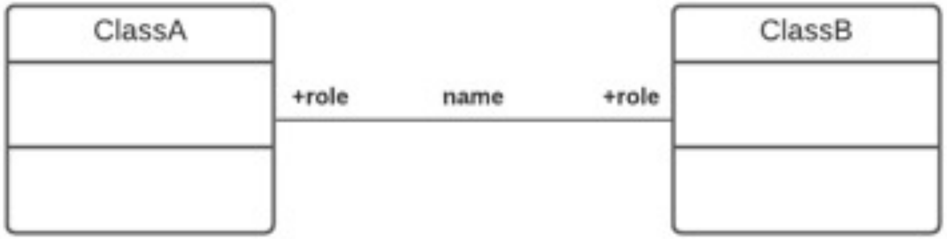
- Ternary (or N-ary) relations: with a diamond



UML to model the structure

Association between classes

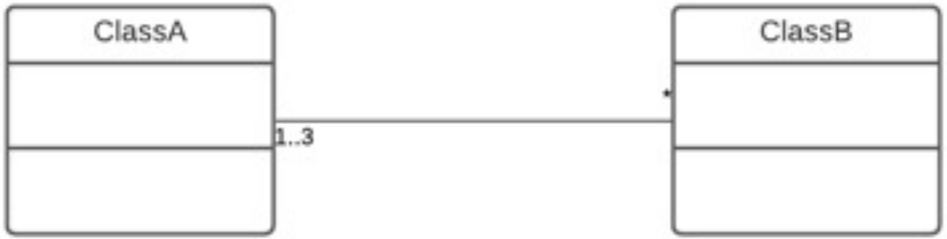
- Association can be named
- Role of each instance can be indicated with the syntax “+role”



UML to model the structure

Cardinality

- Possible to add cardinality to associations
- The cardinality on the side of a class indicates how many instances of this class are linked to one instance of the class at the other end
- Example: 1 to 3 instances of ClassA are linked to each instance of ClassB. 0 or more instance of ClassB are linked to each instance of ClassA



UML to model the structure

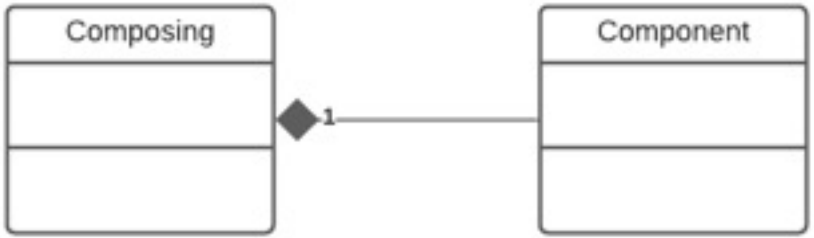
Composition

- Reminder: **Composition**: complex objects can be composed of other objects.
- It is defined at the class level, but we only compose actual instances
- It can be:
  - a strong relationship: components cannot be shared; destruction of the composed object implies destruction of the components
  - a weak relationship (a.k.a. aggregation): components can be shared
- A composition is an association!
- “Has a” relationship

UML to model the structure

Strong composition (a.k.a. composition)

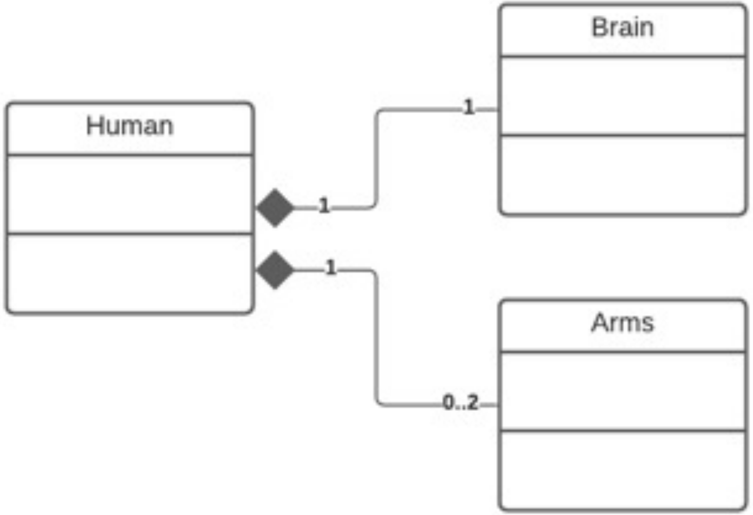
- Strong composition: components cannot be shared -> cardinality on the composing side is always 1



UML to model the structure

Strong composition (a.k.a. composition)

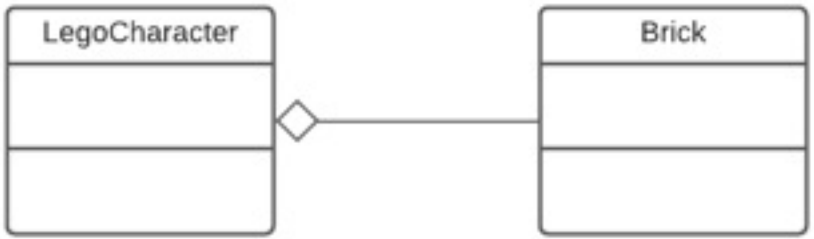
- Strong composition: components cannot be shared -> cardinality on the composing side is always 1
- Example: an instance of a human has one brain and at most 2 arms.



UML to model the structure

Weak composition (a.k.a. aggregation)

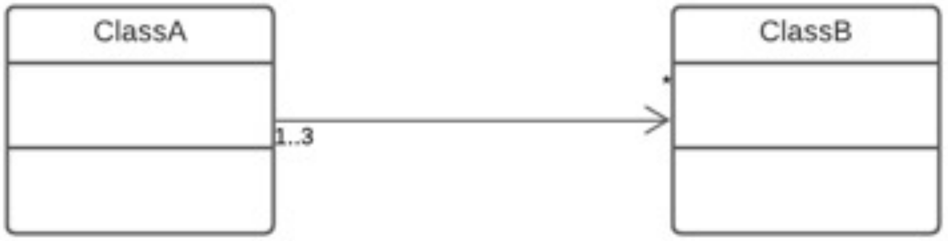
- Weak composition: components can be shared and destroying the composing object does not destroy the component
- More frequent than strong composition



UML to model the structure

Directionality

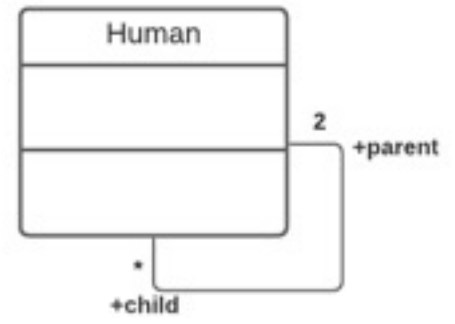
- By default, associations are bi-directional: from an instance of ClassA, I should be able to find the link to ClassB and from an instance of ClassB, I should be able to find the link to ClassA.
- In general, not good in practice
- Possible to add an arrow to indicate the direction of the association.
- Example: ClassA knows it links to ClassB, but ClassB does not.

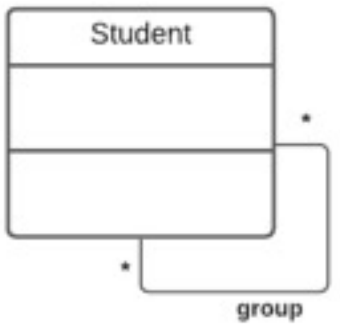


UML to model the structure

Reflexive association

- It is possible to have an association from a class to itself.
- Can be used to create
  - a hierarchy
  - a group

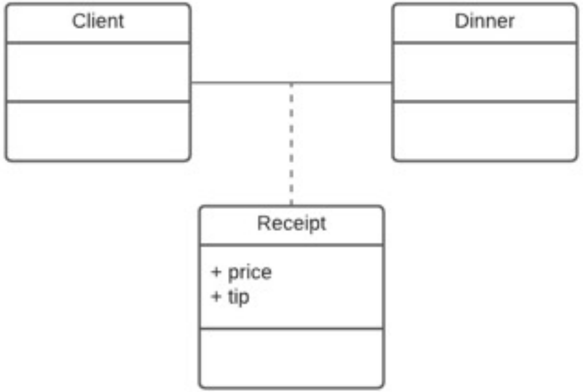




UML to model the structure

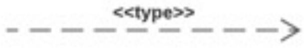
Association classes

- In some cases, the association can be complex
- In the case, the association can be modeled as a class
- Syntax: dotted line between the association class and the association
- Example:



UML to model the structure

Dependency association

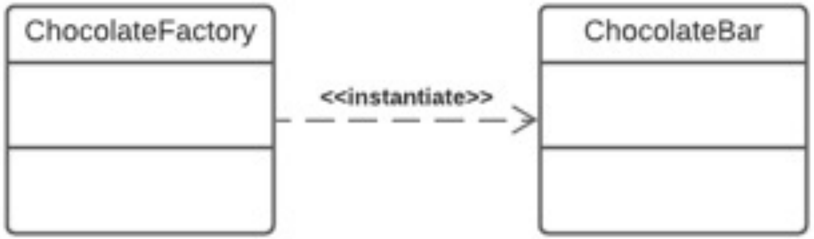
- Directed association
- Indicates that a class needs another class for its specification or implementation
- Syntax: 

- Type indicate the dependency and can be:
  - call: uses a method
  - create: create an instance
  - derive: indicate a redundancy
  - instantiate: factory class that needs this other class for its specification.
  - permit: allows for access to private elements
  - use: general case

UML to model the structure

Dependency association

- Example: the ChocolateFactory class is a factory that has the only goal of producing (instantiating) instances of ChocolateBar. ChocolateFactory hence has a need to know the specification of ChocolateBar to produce it.



UML to model the structure

Associations - recap

- An association between two classes represents a link between them.
- In the general case, the association is qualified by names and roles and indicates a simple link between classes (solid line)
- Composition: indicates a component that **cannot** be shared
- Aggregation: indicates a component that **can** be shared
- Dashed arrow: dependency between classes.