Université de Paris

Génie Logiciel
UML to model the structure

Sylvain Lobry

02/12/2022

Menu of the day

1 Intro to class diagrams

2 Representing a class

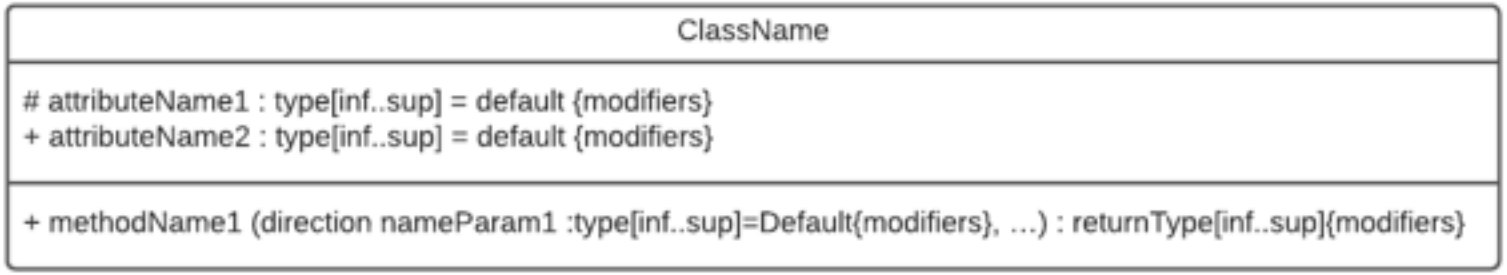
3 Association between classes

4 Hierarchy

5 Representing objects

UML to model the structure

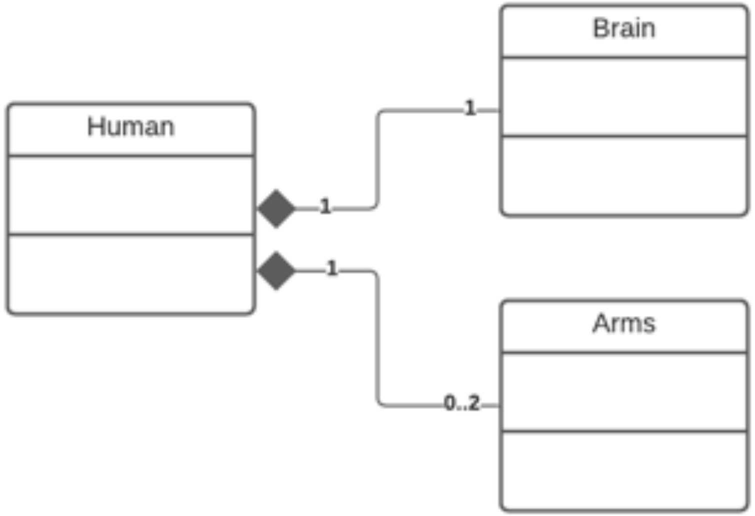
Representing a class - recap


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}

UML to model the structure

Strong composition (a.k.a. composition)

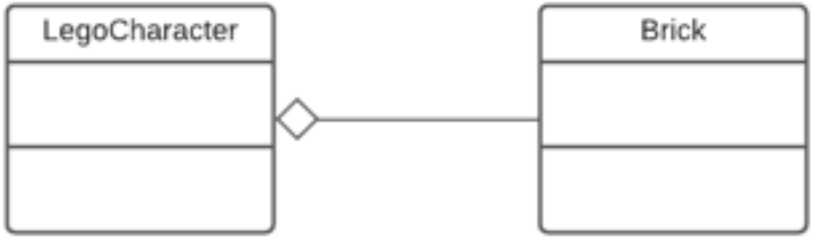
- Strong composition: components cannot be shared -> cardinality on the composing side is always 1
- Example: an instance of a human has one brain and at most 2 arms.



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

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.

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

Generalization/Specialization: reminder

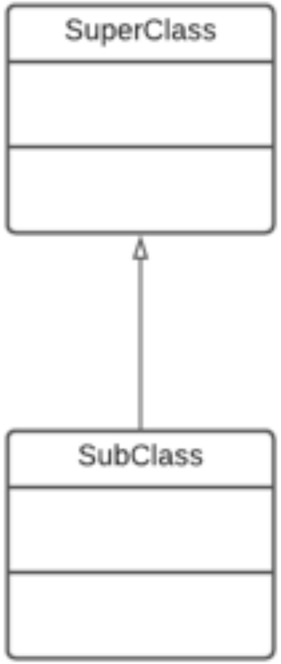
- **Specialization**: a new class A can be created as a subclass of another class B, in which case class A specializes the class B.
- Specialization is an "is a" relationship.
- **Generalization** is the opposite (superclass B is a generalization of subclass A).

- **Inheritance**: the fact that a subclass gets the behaviour and the structure of the superclass
- This is a **consequence** of specialization

UML to model the structure

Generalization/Specialization

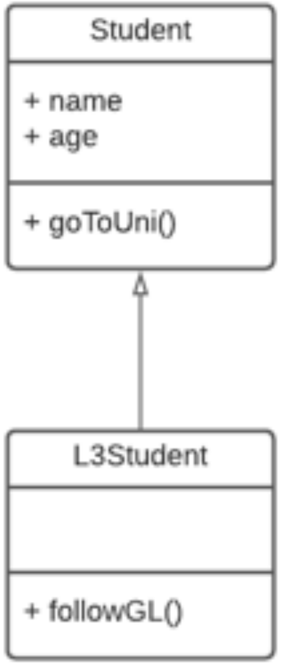
- Syntax:



UML to model the structure

Inheritance

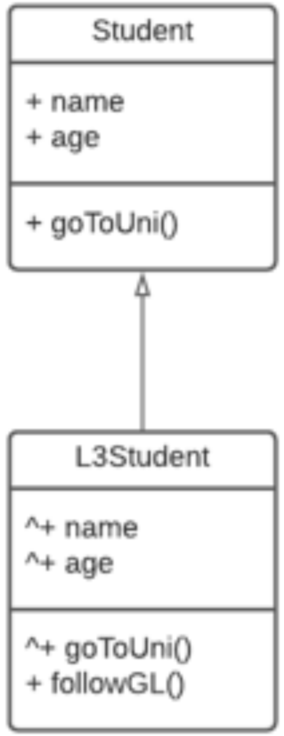
- Instances of a subclass are also instance of the superclass.
- Therefore, they inherit from methods defined in the superclass.
- Example:



UML to model the structure

Inheritance

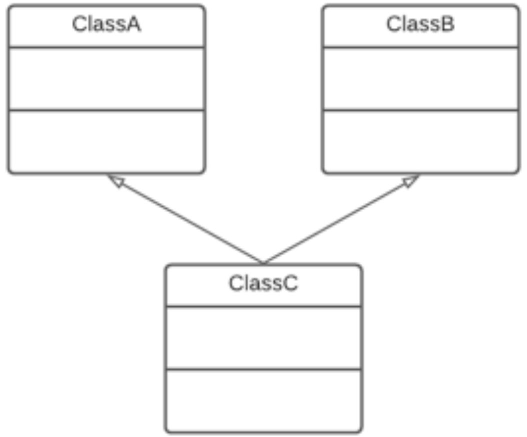
- Instances of a subclass are also instance of the superclass.
- Therefore, they inherit from methods defined in the superclass.
- Example:
- Note that you can explicitely show inherited elements by prefixing with “^”
- Finally, note that associations between a class and a superclass is inherited by its subclasses.



UML to model the structure

Multiple inheritance

- It is possible for a class to be a specialization of more than one class
- Example: ClassC is a specialization of both ClassA and ClassB.



UML to model the structure

Multiple inheritance

- It is possible for a class to be a specialization of more than one class
- Example: ClassC is a specialization of both ClassA and ClassB.
- Multiple inheritance can be problematic if an attribute with the same name/type or a mathod with the same signature is defined in more than one superclass.
- Not always possible in practice: no multiple inheritance in Java.

UML to model the structure

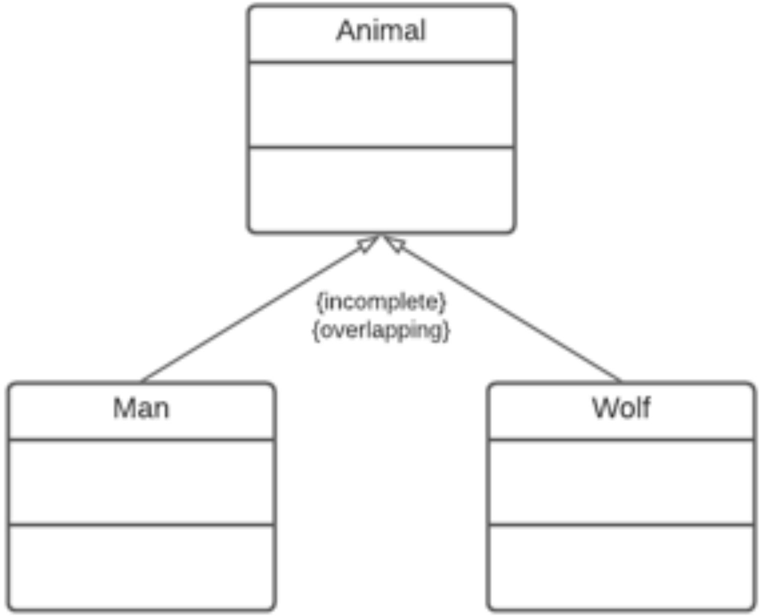
Constraints

- It is possible to add constraints on the relation, either on:
  - completeness: the specialization can be *complete* or *incomplete*. If it is complete, it indicates that the set of domains of the subclasses cover the domain of the superclass.
  - superimposition: the specialization can either be *disjoint* (they have no common instances) or *overlapping* (they can have common instances)
- Syntax: {constraint}

UML to model the structure

Constraints

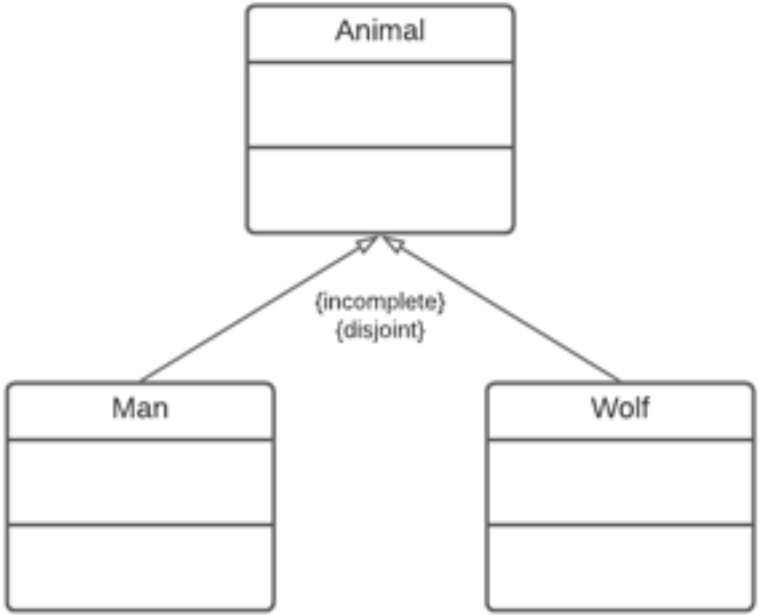
- Example:
- there are other animals than men and wolves, so the relation is **incomplete**.
- if you believe in werewolves, an instance can be both a man and a wolf, hence it is **overlapping**.



UML to model the structure

Constraints

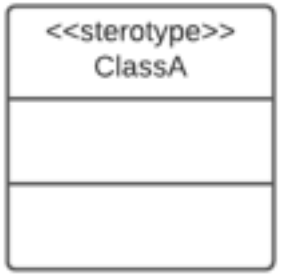
- Example:
- there are other animals than men and wolves, so the relation is **incomplete**.
- *Probably*, a man cannot be a wolf. So the relation is actually disjoint.



UML to model the structure

Stereotypes

- Stereotypes can be used to specialize an element in UML.
- Syntax: <<stereotype>> above the class name.



UML to model the structure

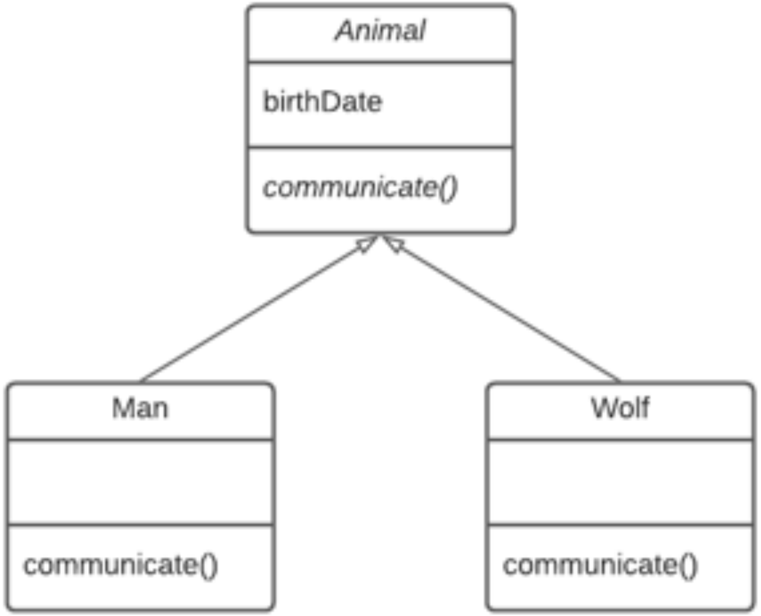
Stereotypes

- Stereotypes can be used to specialize an element in UML.
- Syntax: <<stereotype>> above the class name.
- Possible stereotypes:
  - enumeration: class introducing a type with a list of constant values
  - auxiliary: to indicate a secondary class
  - abstract
  - interface

UML to model the structure

Abstract classes

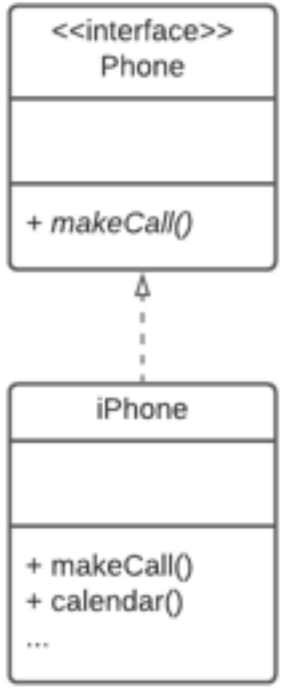
- Reminder: **Abstract** and **concrete** classes: abstract classes are classes that do not have instances (e.g. Mammal). Concrete classes do (e.g. Human).
- Abstract classes allow for class hierarchies and to group attributes and methods. They should have subclasses.
- Example: the method communicate of class Animal is abstract (indicated in *italic*). It is not defined for an animal, but it is for concrete classes
- Note: can also be indicated by italic class name.



UML to model the structure

Interface

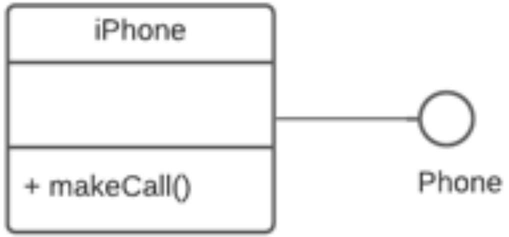
- Definition: an interface is a fully abstract class: it does not have any attribute and its methods are all public and abstract
- Syntax: stereotype + dashed empty arrow



UML to model the structure

Interface

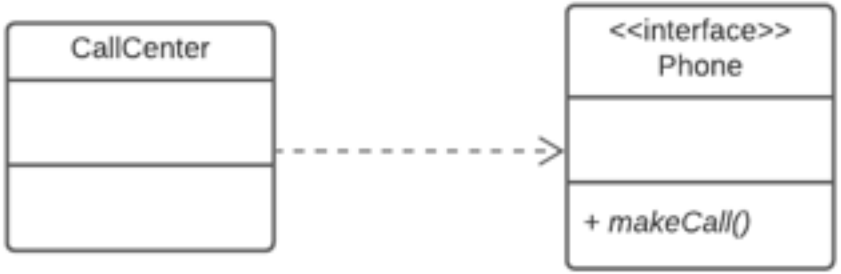
- Definition: an interface is a fully abstract class: it does not have any attribute and its methods are all public and abstract
- Syntax: stereotype + dashed empty arrow
- Alternative: lollipop



UML to model the structure

Interface

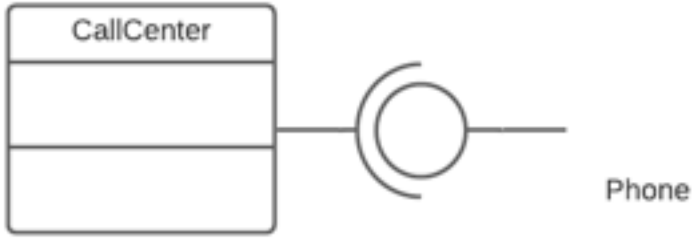
- When there is a dependency on an interface it can be noted “classically”



UML to model the structure

Interface

- When there is a dependency on an interface it can be noted “classically”
- Or through a lollipop



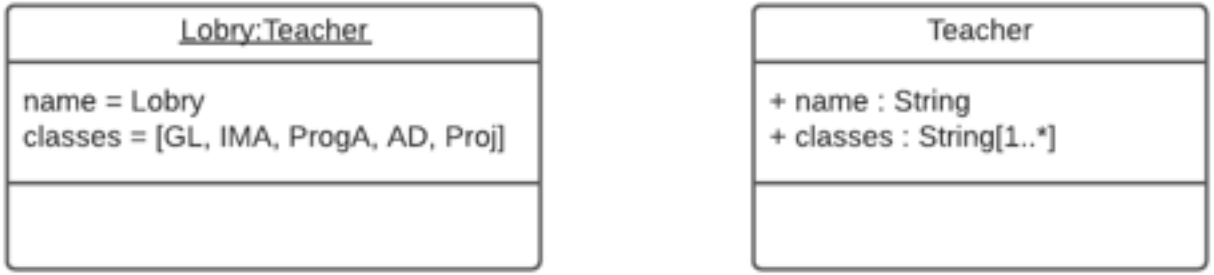
Menu of the day

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

UML to model the structure

Representing an object

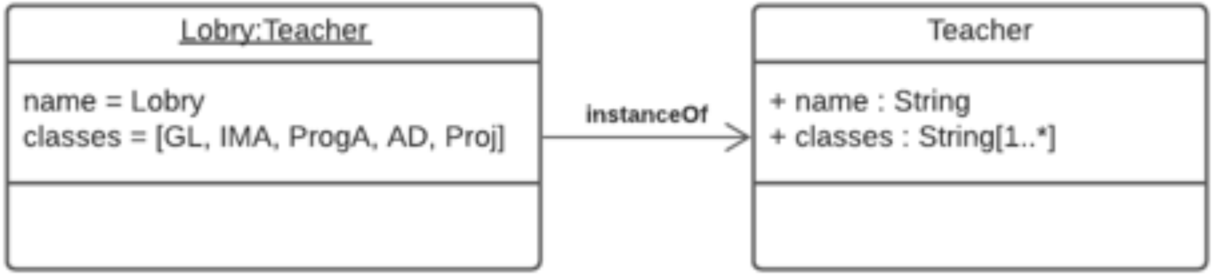
- Class diagram represent a static view of the structure
- Object diagram can show a snapshot of the system:
- Object diagram shows instances and values of their attributes
- Syntax: <u>name_of_the_instance:ClassName</u>



UML to model the structure

Representing an object

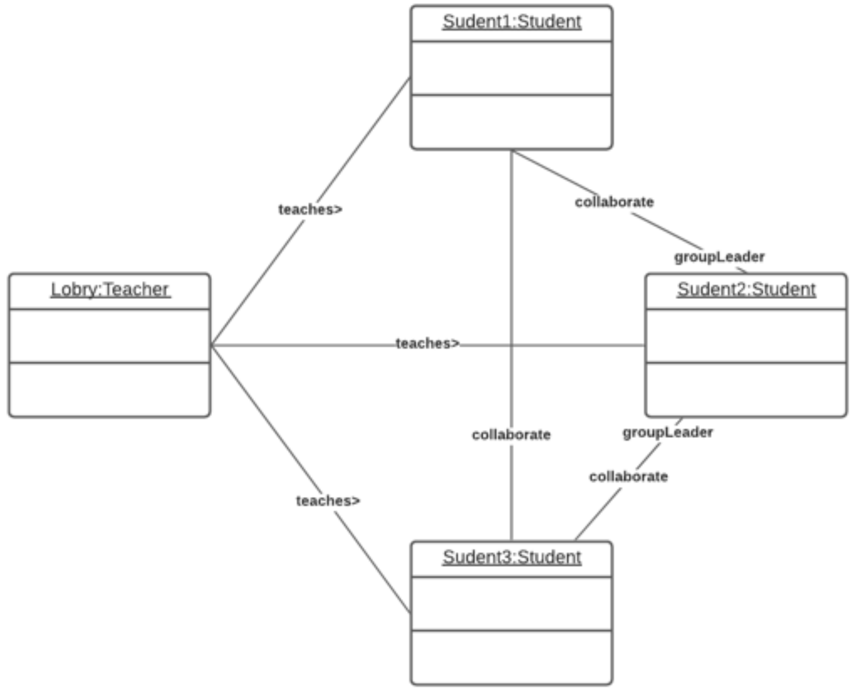
- Class diagram represent a static view of the structure
- Object diagram can show a snapshot of the system:
- Object diagram shows instances and values of their attributes
- Syntax: <u>name_of_the_instance:ClassName</u>
- Optionnally “instanceOf” link



UML to model the structure

Relation between instances

- Finally it is possible to represent interactions between instances with a solid line
- Optional: name of the relation and roles



UML to model the structure

Conclusion

- Class diagrams allow to add information on the structure of our model
- Adding the right links between classes enhance the semantics and makes the diagram lighter
- As always with modeling:
  - Pay attention to the target of the model: what do they need to know?
  - Not just a diagram, should come with documentation (in particular: your choices)
  - Not a unique good solution
- Requires practice