Université de Paris

Génie Logiciel
UML to model the structure

Sylvain Lobry

19/11/2021

Resources: www.sylvainlobry.com/GenieLogiciel

UML to model the structure

Before we start

Note:

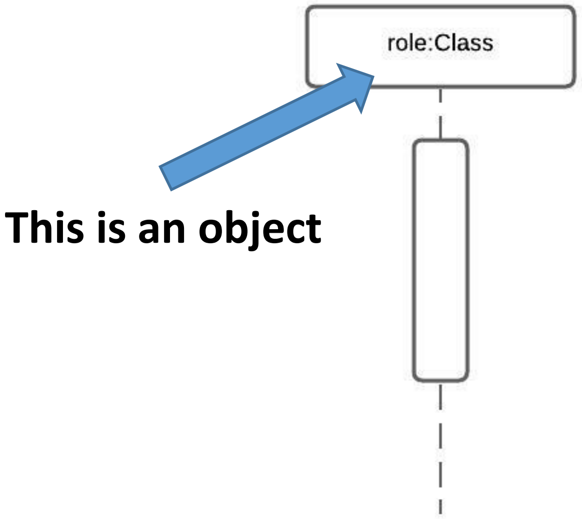
- 26/11: Amphi Weiss
- 10/12: Amphi Claude Bernard

Menu of the day

UML to model the structure

Previously… sequence diagrams

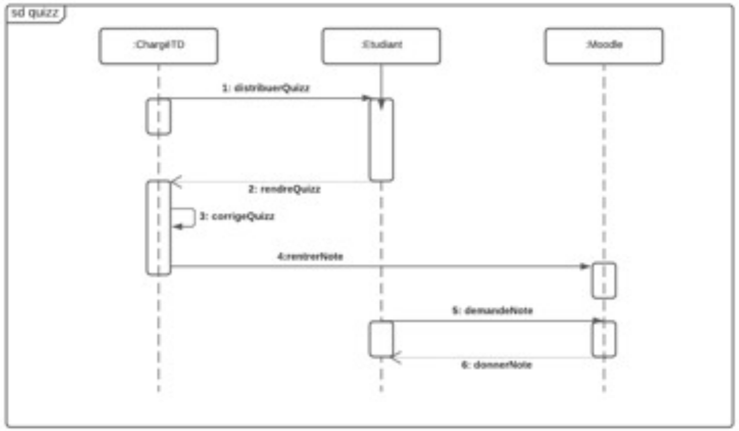
- Interactions between **objects**
- defined as **role:Class**
- Can be an actor (from the use case)
- Does not have to be an actor
- Most of the time, there are additional objects

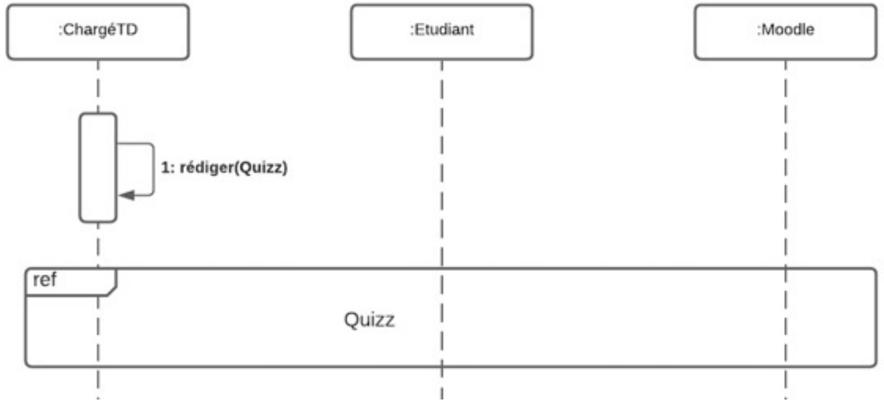


UML to model the structure

Previously… sequence diagrams

- sd = sequence diagram
- Can be used to name the sequence diagram, but optional
- Can be used to reference the sequence diagram in another sequence diagram

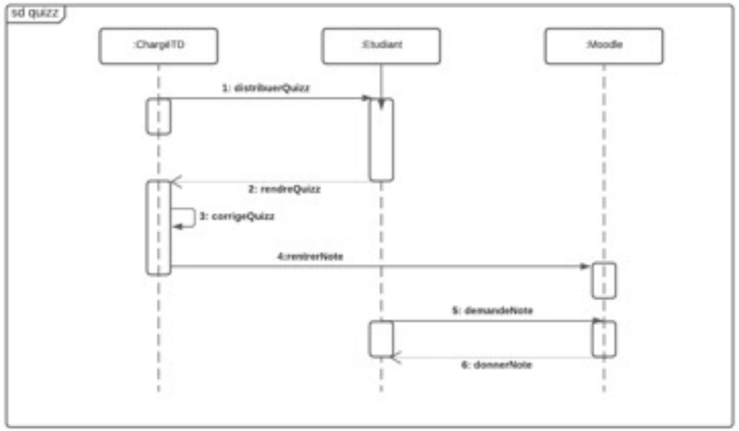


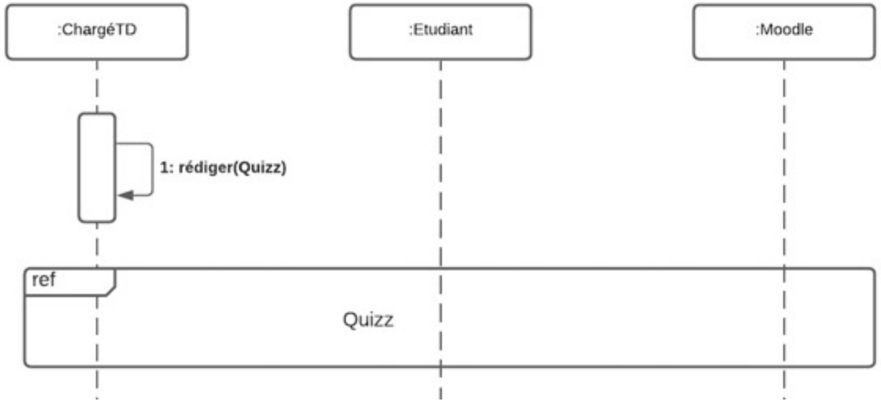


UML to model the structure

Previously… sequence diagrams

- Diagram on the left: sequence for a quiz during a lab
- Diagram on the right: “lifecycle” of a quiz uses the diagram on the left:
  - more concise
  - avoid duplication

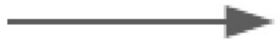




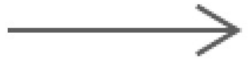
UML to model the structure

Notations in UML – Sequence diagram

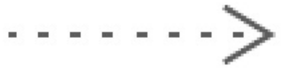
- Most important thing!
- Messages in sequence diagrams:
  - Synchronous: the sender stops its activity while the recipient is working on the message



  - Asynchronous: the sender does not stop its activity



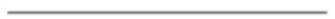
  - Reply



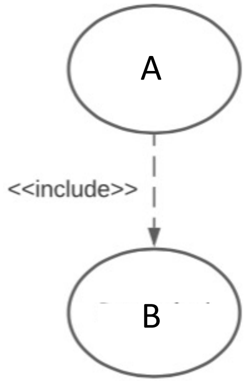
- Can be named, must be numbered

UML to model the structure

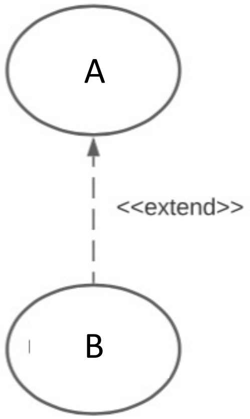
Notations in UML – Use cases



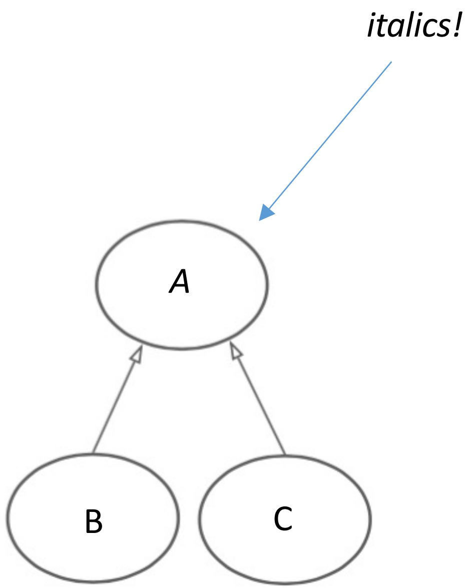
Association link (between actor and use case)



Use case A includes use case B



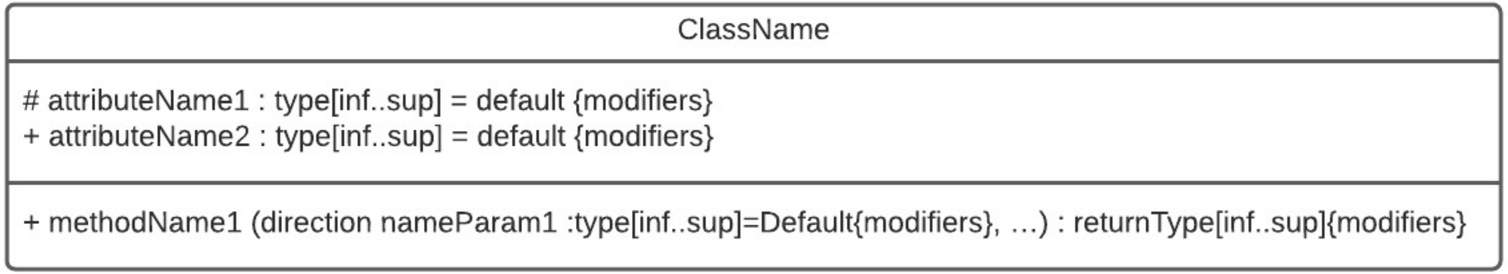
Use case B extends use case A



Use case B and C are specializations of use case A

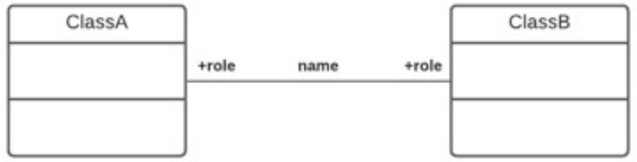
UML to model the structure

Notations in UML – Class diagram

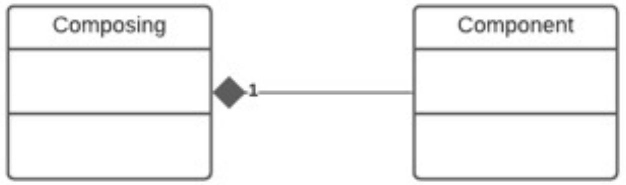

ClassName
# attributeName1 : type[inf..sup] = default {modifiers}
+ attributeName2 : type[inf..sup] = default {modifiers}
+ methodName1 (direction nameParam1 :type[inf..sup]=Default{modifiers}, …) : returnType[inf..sup]{modifiers}

UML to model the structure

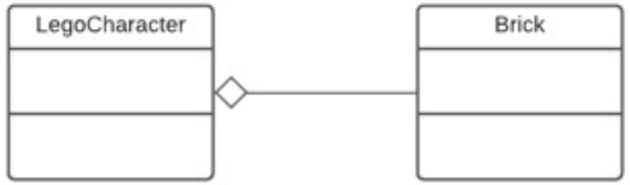
Notations in UML – Class diagram



Association between ClassA and ClassB



Composing is a composition of Component



LegoCharacter is an aggregation of Brick

Menu of the day

UML to model the structure

Generalization/Specialization: reminder

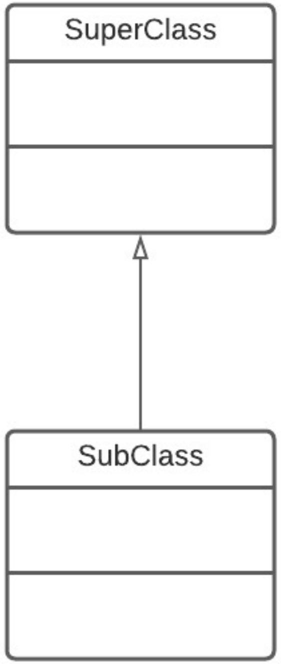
- **Specialization**: a new class A can be created as a subclass of another class B, in which case class A specializes the class B.
- Specialization is an “is a” relationship.
- **Generalization** is the opposite (superclass B is a generalization of subclass A).

- **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

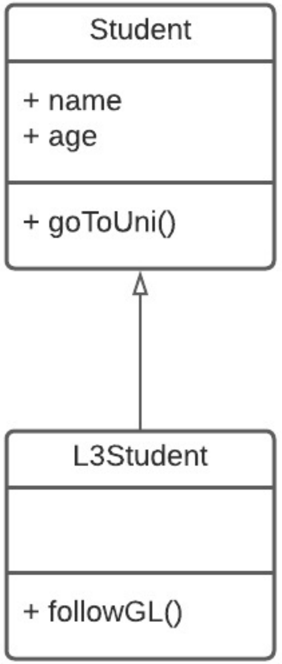
- Syntax:



UML to model the structure

Inheritance

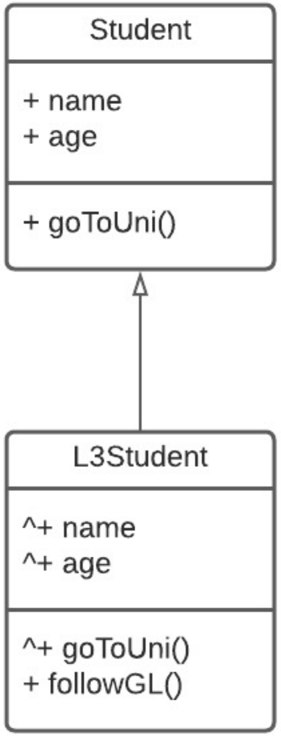
- Instances of a subclass are also instance of the superclass.
- Therefore, they inherit from methods defined in the superclass.
- Example:



UML to model the structure

Inheritance

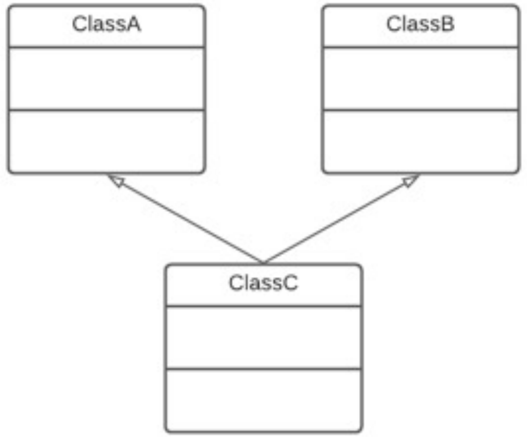
- Instances of a subclass are also instance of the superclass.
- Therefore, they inherit from methods defined in the superclass.
- Example:
- Note that you can explicitely show inherited elements by prefixing with “^”
- Finally, note that associations between a class and a superclass is inherited by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

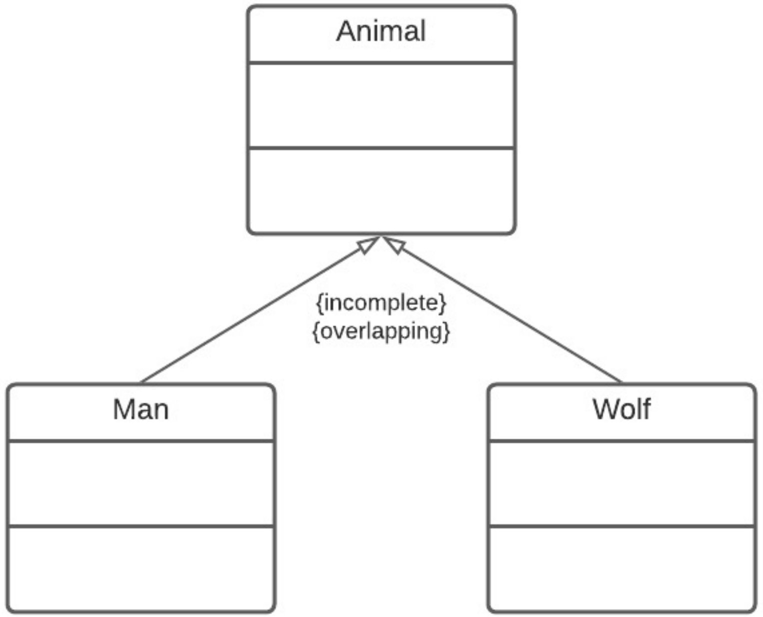
Constraints

- 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

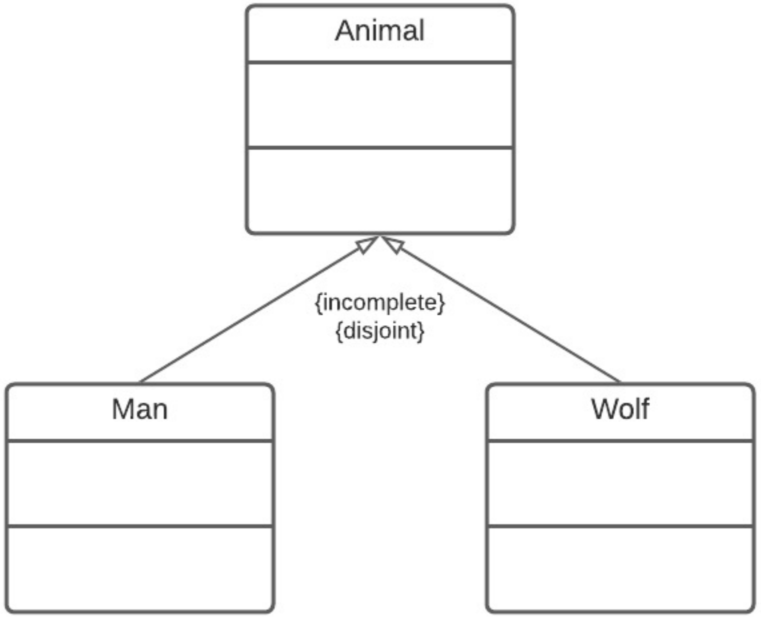
- Example:
- there are other animals than men and wolves, so the relation is **incomplete**.
- if you believe in werewolves, an instance can be both a man and a wolf, hence it is **overlapping**.



UML to model the structure

Constraints

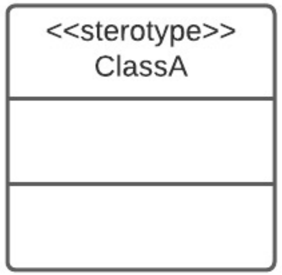
- Example:
- there are other animals than men and wolves, so the relation is **incomplete**.
- *Probably*, a man cannot be a wolf. So the relation is actually disjoint/



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

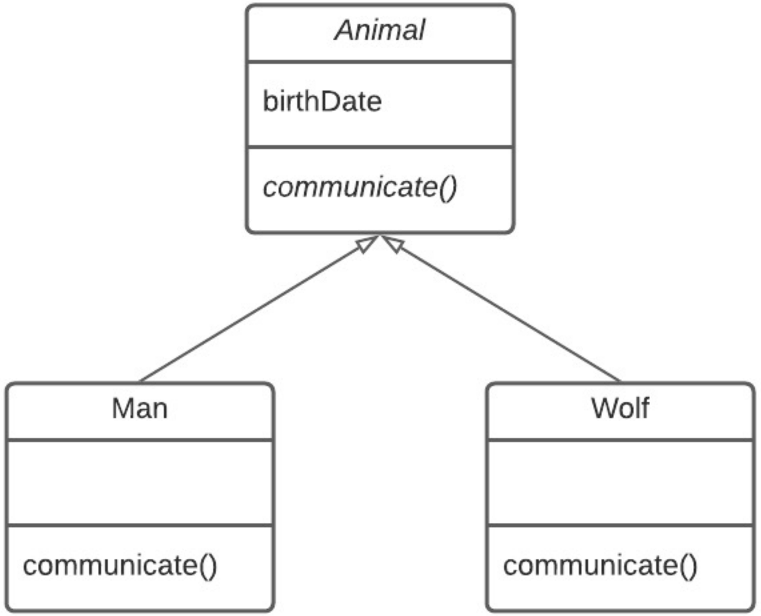
Stereotypes

- 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

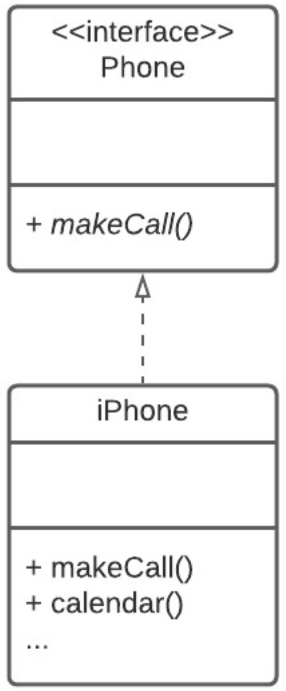
- Reminder: **Abstract** and **concrete** classes: abstract classes are classes that do not have instances (e.g. Mammal). Concrete classes do (e.g. Human).
- Abstract classes allow for class hierarchies and to group attributes and methods. They should have subclasses.
- Example: the method communicate 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

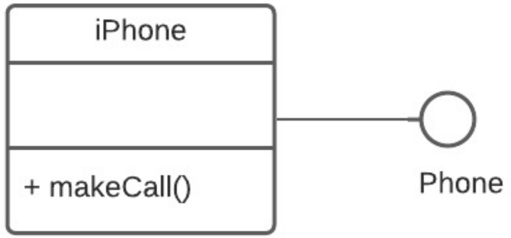
- Definition: an interface is a fully abstract class: it does not have any attribute and its methods are all public and abstract
- Syntax: stereotype + dashed empty arrow



UML to model the structure

Interface

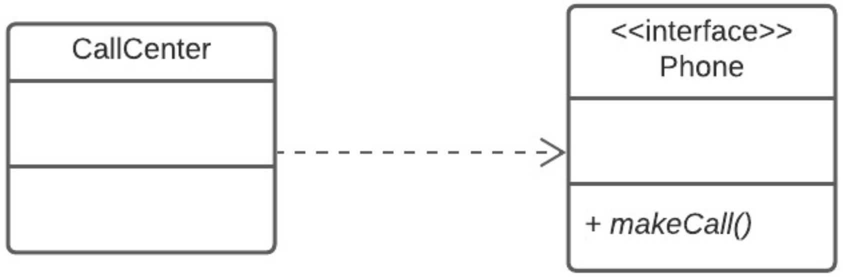
- Definition: an interface is a fully abstract class: it does not have any attribute and its methods are all public and abstract
- Syntax: stereotype + dashed empty arrow
- Alternative: lollipop



UML to model the structure

Interface

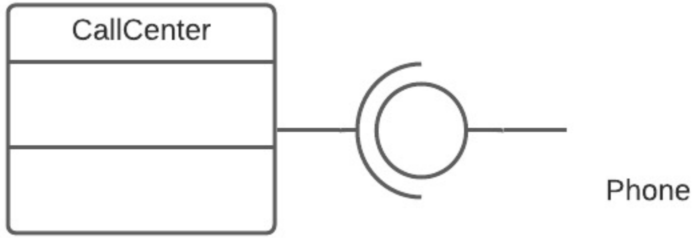
- When there is a dependency on an interface it can be noted “classically”



UML to model the structure

Interface

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

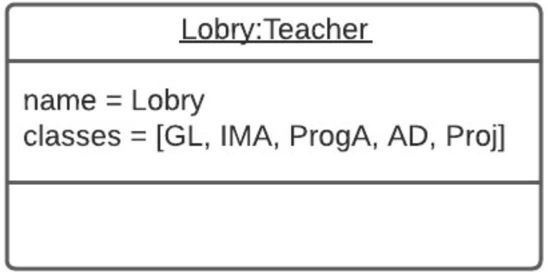


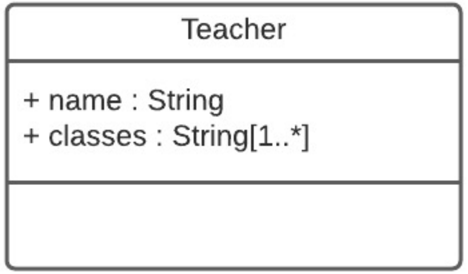
Menu of the day

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: name_of_the_instance:ClassName

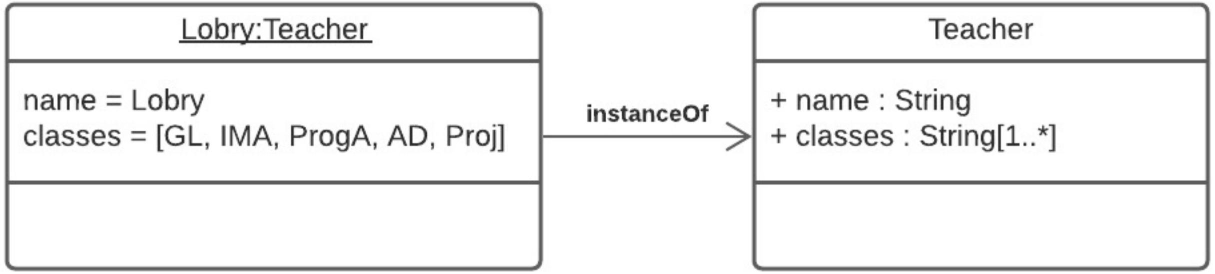




UML to model the structure

Representing an object

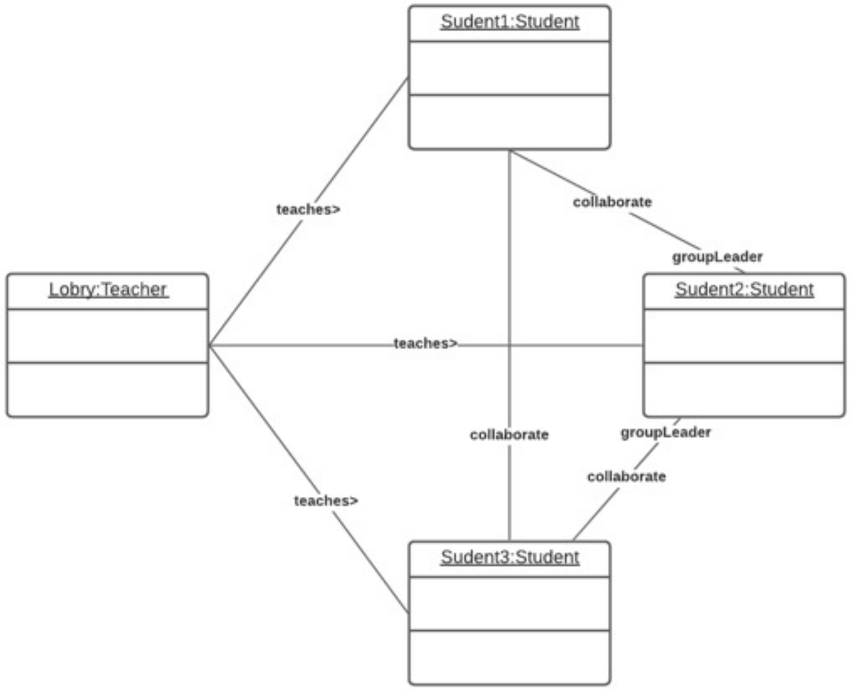
- Class diagram represent a static view of the structure
- Object diagram can show a snapshot of the system:
- Object diagram shows instances and values of their attributes
- Syntax: <u>name_of_the_instance:ClassName</u>
- Optionnally “instanceOf” link



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)
  - Try to get feedback!
  - Not a unique good solution
- Refining the diagram
- Requires practice

UML

UML Conclusion

- Complete model = diagrams + documentation
- Diagrams to model the behavior (Use case diagram), the interactions (Sequence diagram) and the structure (Class diagram)
- Center the diagrams around use cases

- Document:
    1. Practical information (authors, date, version)
    2. Context of the project
    3. Introduction to the model (choices, which views, discussion)