Modeling with UML

Reda Bendraou
reda.bendraou{{@}}Lip6.fr
http://pagesperso-systeme.lip6.fr/Reda.Bendraou/

UML: Static/architecture viewpoint

- OO Basics
-Class Diagram
-Object Diagram
- Package Diagram

OO Basics

-OO Vision

- Main Concepts

OO Vision

- To consider a system as a set of objects interacting together to realize the system's functionalities. Each object encapsulates structured data and behavior

- Main Concepts
  - Object
  - Class
  - Messages & Methods
  - Generalization
  - Polymorphism

Objects

- **Objects represent entities from the real world**
- Can be concrete entities (customer) or abstract (banking account)

Objects

Identity

- Objects have a unique identifier, used to make reference to them

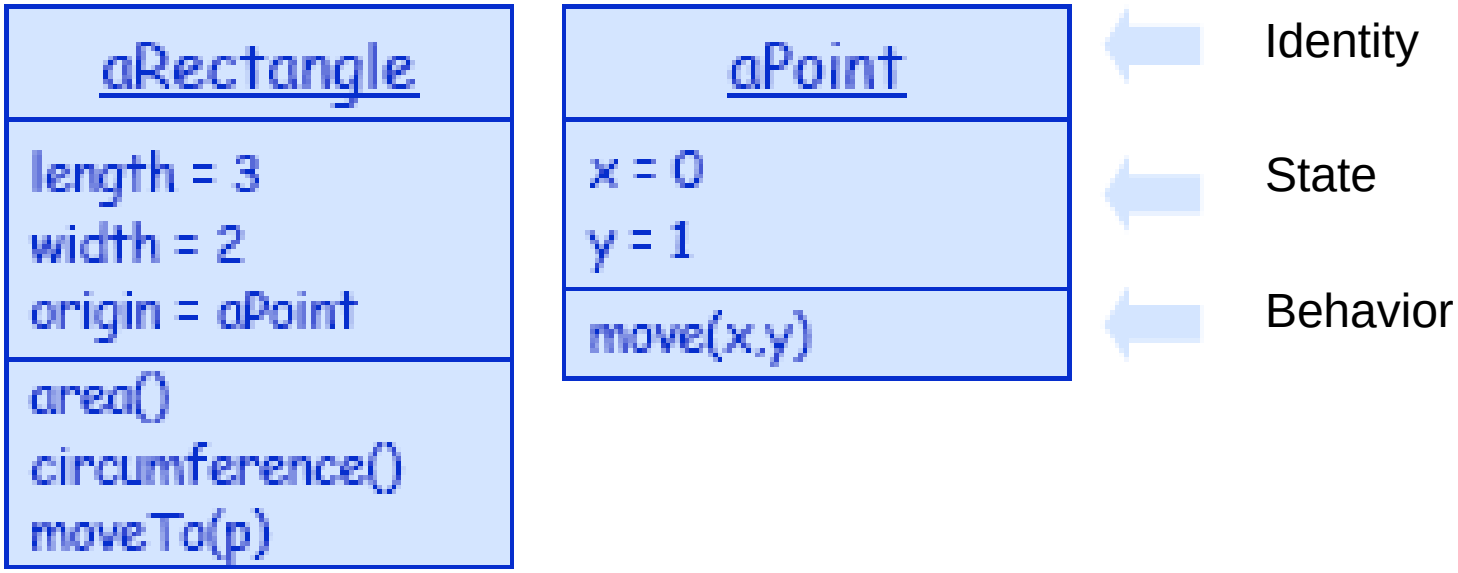
State

- Typed variables
- The variables values at a given time “t” determine the object’s state

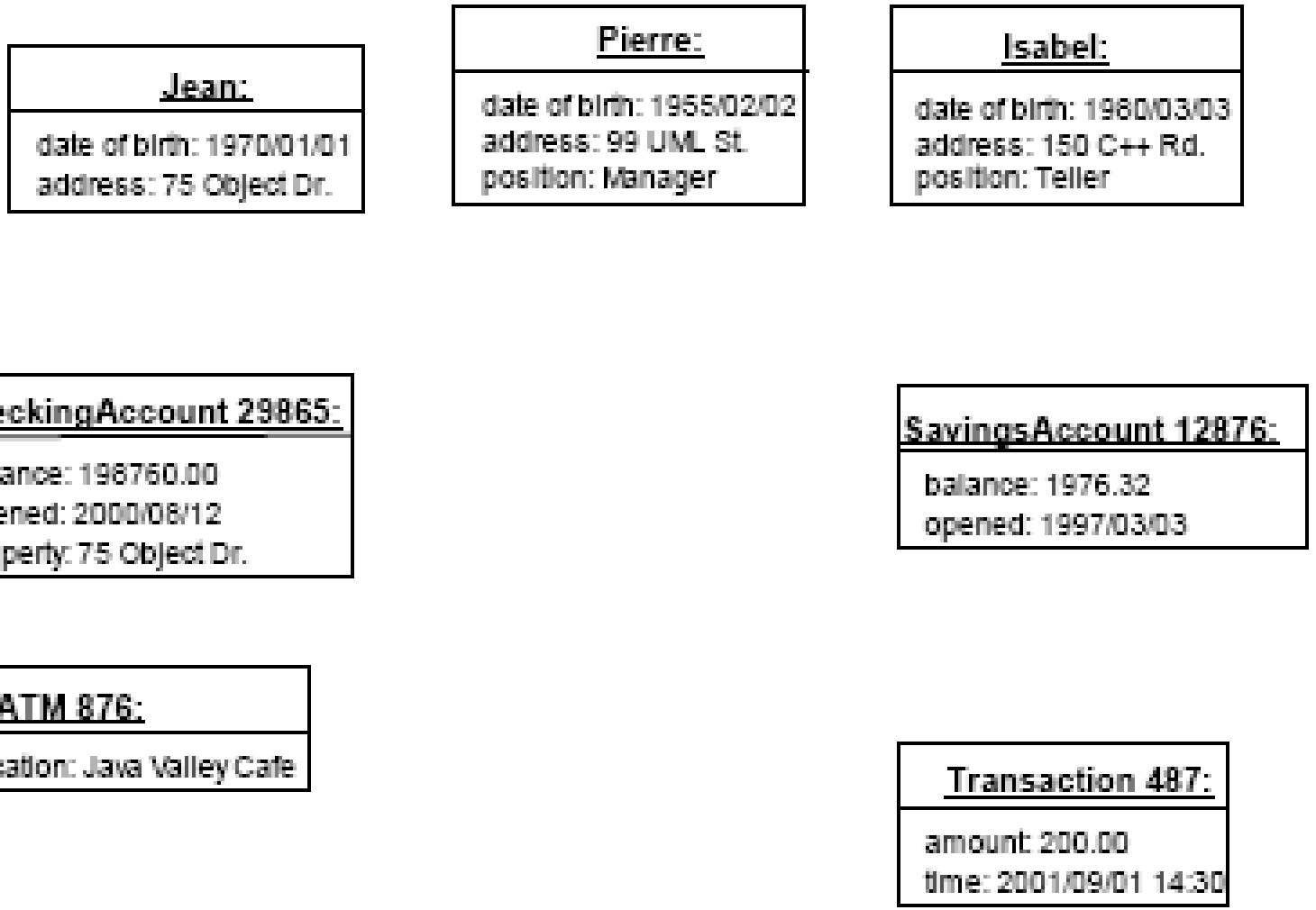
Behavior

- Object’s operations
- Offered through interfaces
- Can lead to a change in the object’s state (or not)

Object : Examples



Object: Examples



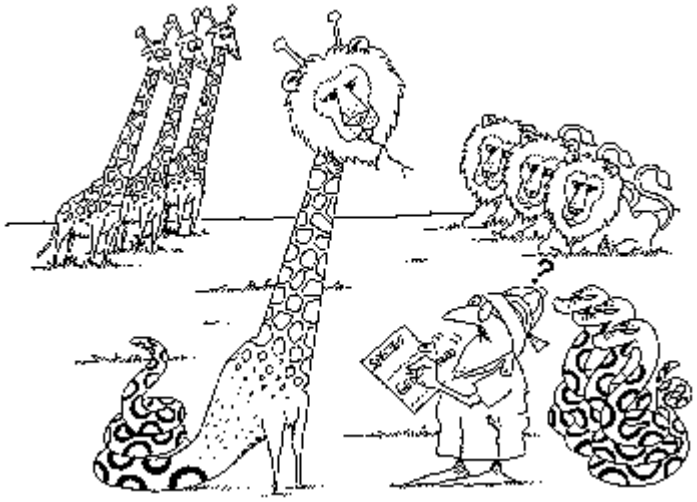
Messages & Methods

- **Messages**
  - The way objects interact with each others
  - Trigger the behavior of an object (Methods)

- **Methods**
  - Are the responses to the messages received by the object
  - Have access to the object's data

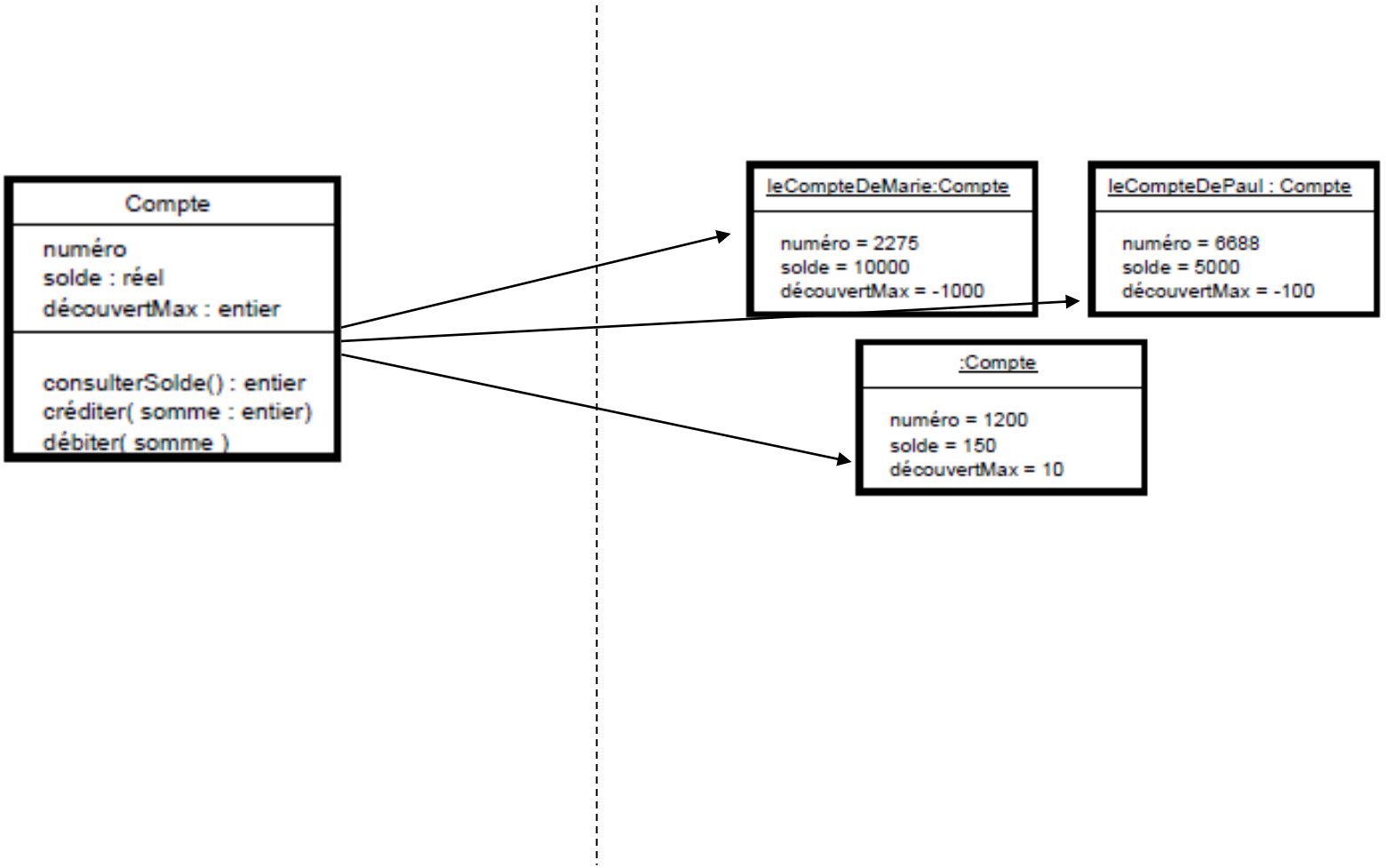
Class

- An abstraction unit
- A grouping, classification mechanism
  - A collection of similar objects
  - Each object is a class's instance
  - The object is typed by its class

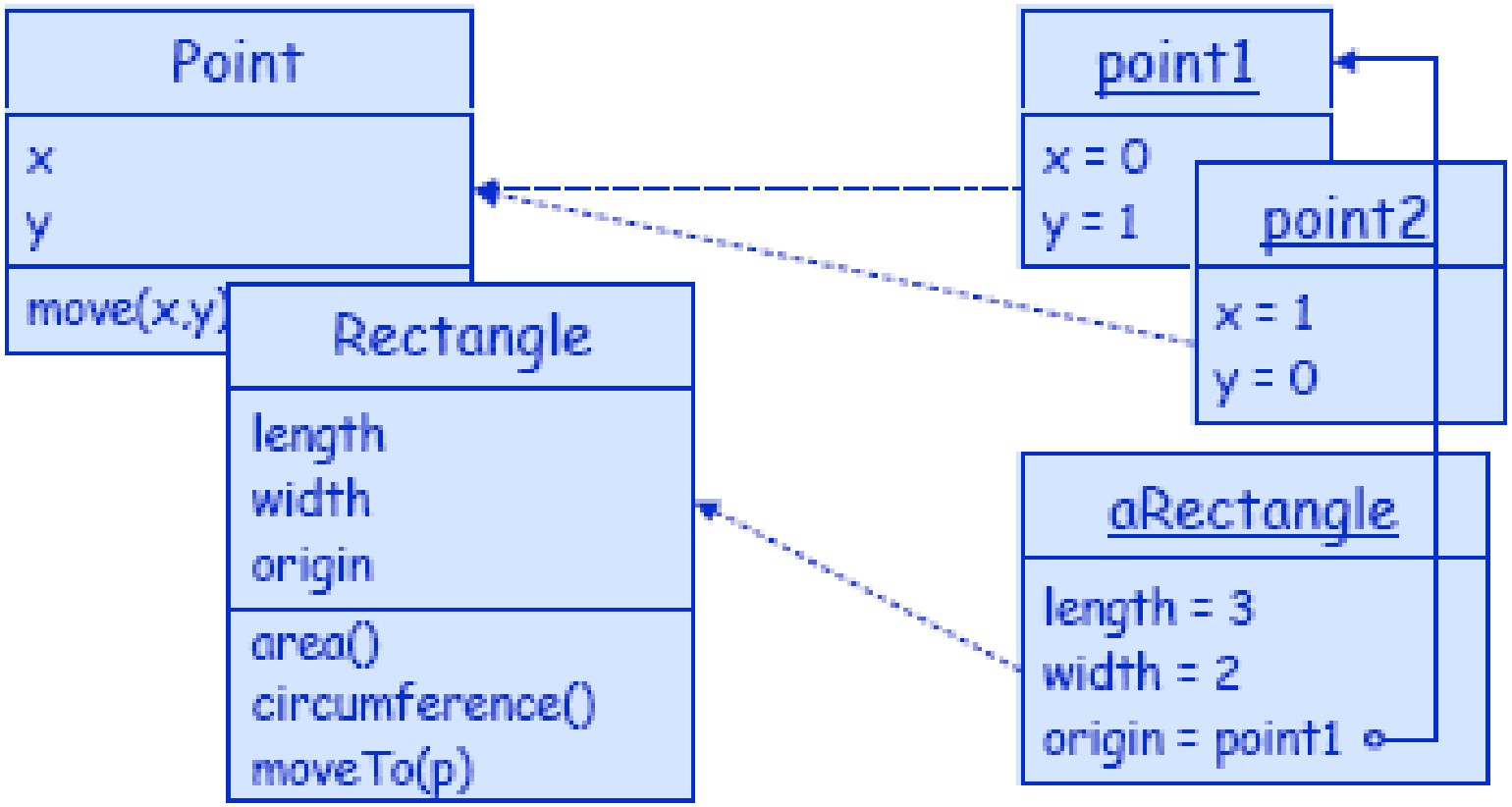


- Describes the common structure for all the objects in terms of properties (attributes) and methods

Class Vs. Objects



Classes & Instances



Instance Variables

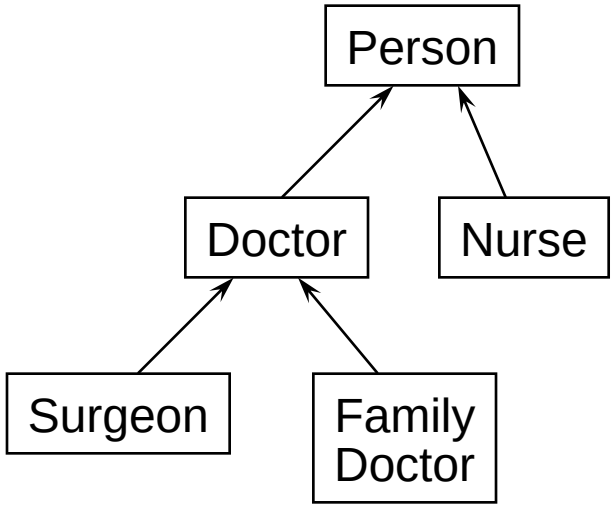
- Specific to each instance
- **Versus Class Variable: shared by all the class's instances**
  - *Notion of static* in Java or C++

Generalization

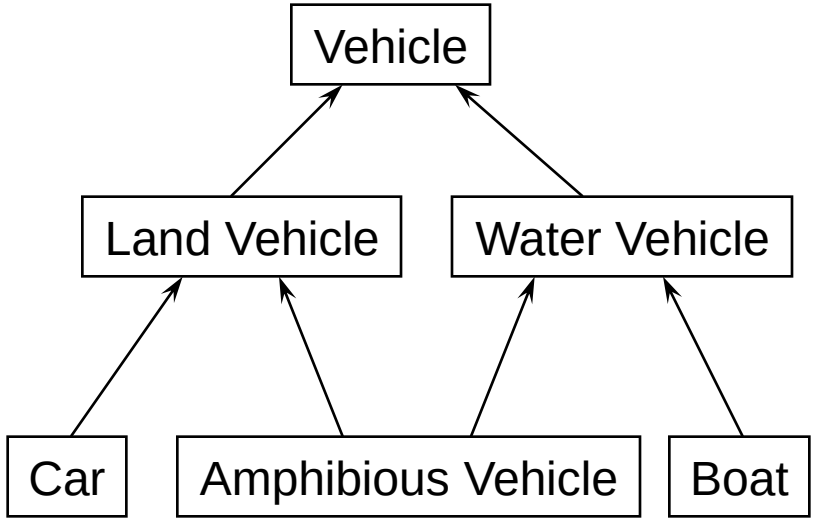
- Reusing a class's structure and behavior by other sub-classes

- Super-class
  - Defines common elements for all sub-classes
  - Sub-classes extend or redefine the super class's structure and behavior

Generalization: Example



simple

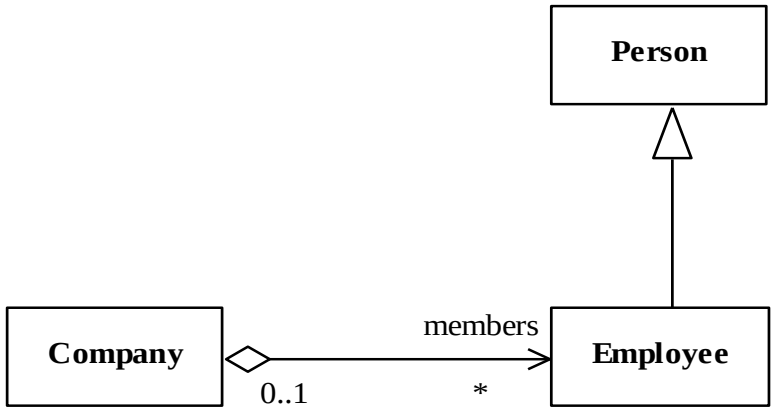


multiple

Class Diagram

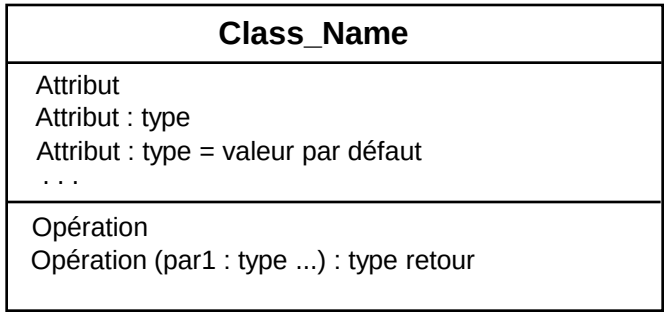
Class Diagram

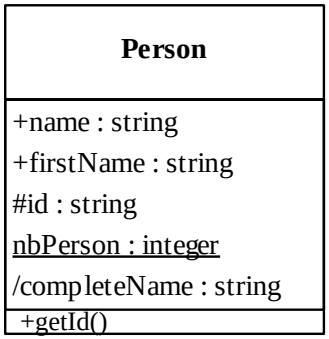
- A class diagram is a graph of elements connected by relations
- Gives the static aspects of your system (structure, architecture, main entities, relations, etc.)



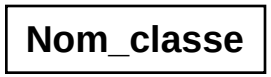
Class

- Detailed representation

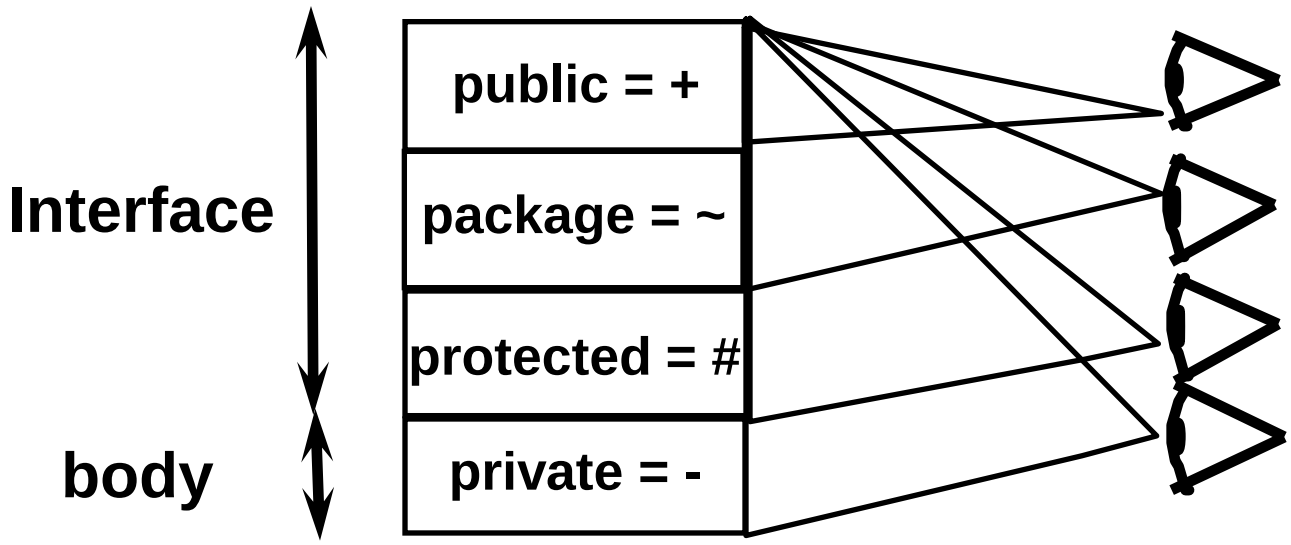




- Simplified representation



Visibility



Attributs

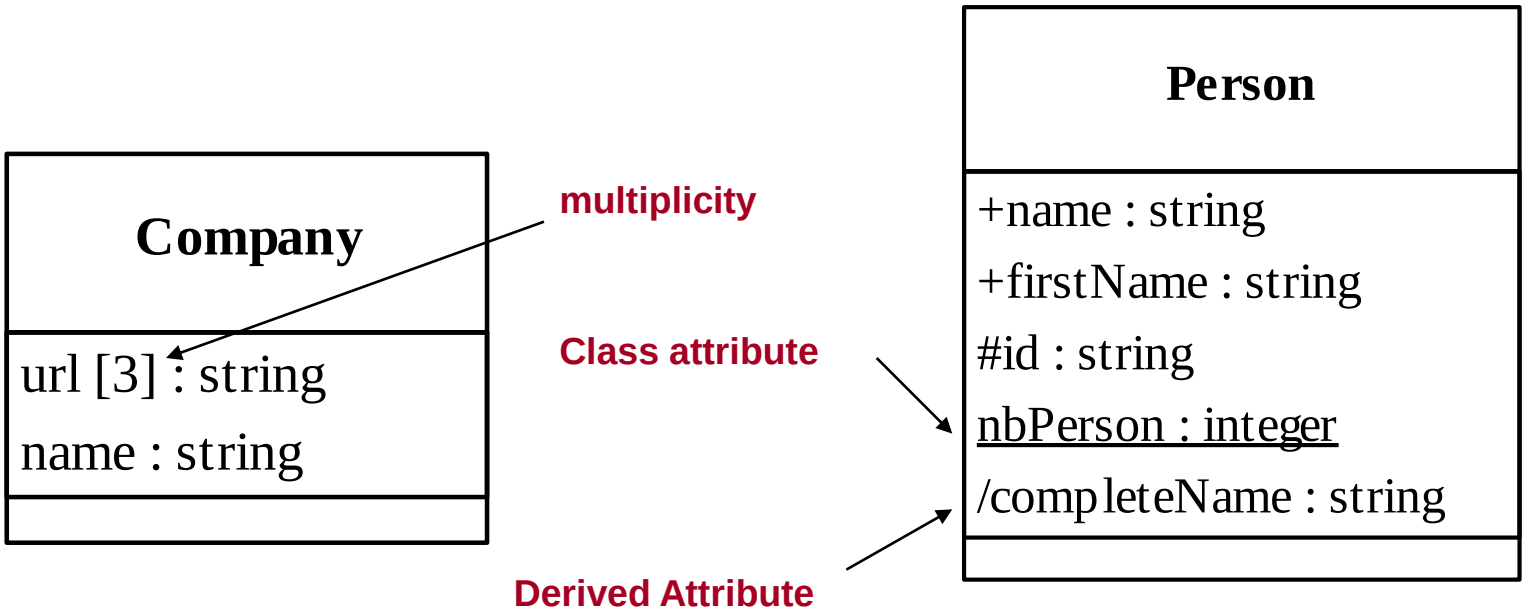
Syntax:

```
visibility name : type [= defaultValue]
```

- Visibility:
  - '+' public
  - '#' protected
  - '-' private
- UML predefined types
  - Integer, real, string, …
- Can be a Class attribute (static) must be underlined.

- Can be derived (calculated), it is then prefixed by'/'

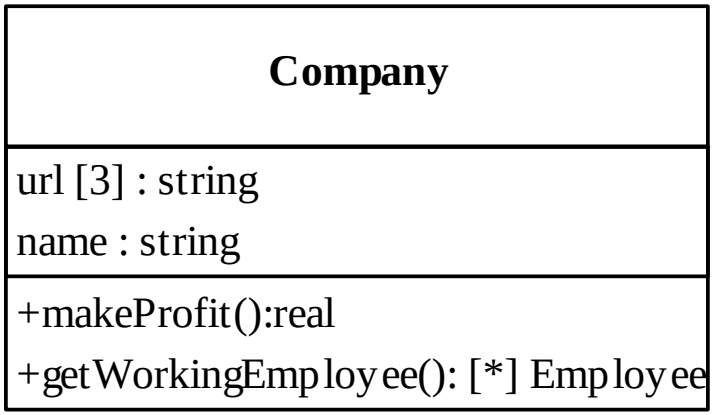
Attributes: Examples

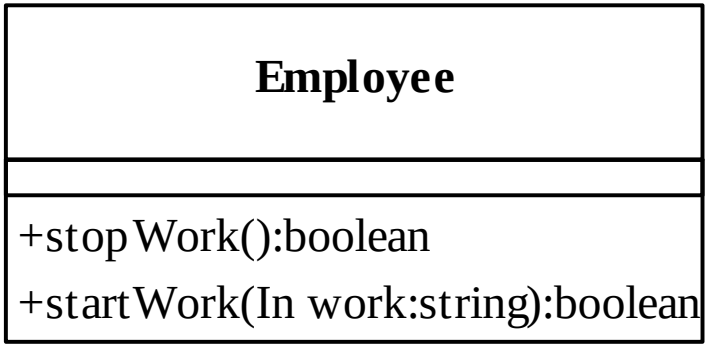


Operations

- Operation is defined as:
  visibility name(parameter):return
- Parameter is defined as:
  kind name : type
- Kind can be:
  - in, out, inout

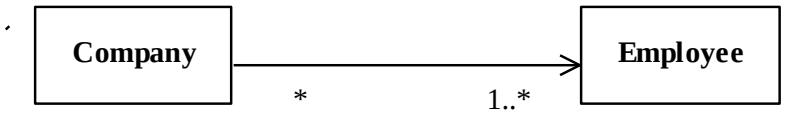
Operations: Examples





Associations

- A very important concept in UML
- A relation between classes
- **Very important: An association is a stable link (persistent) between two objects**

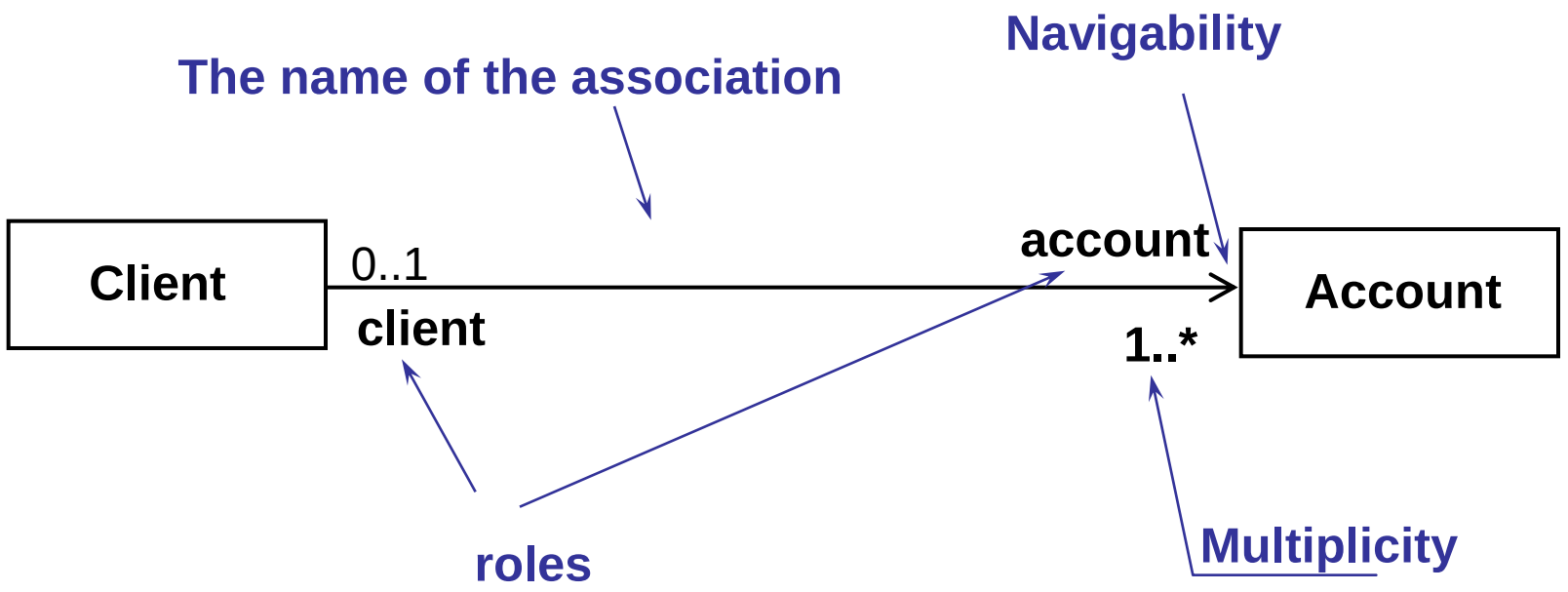


Associations

- A binary association is composed of two association ends.

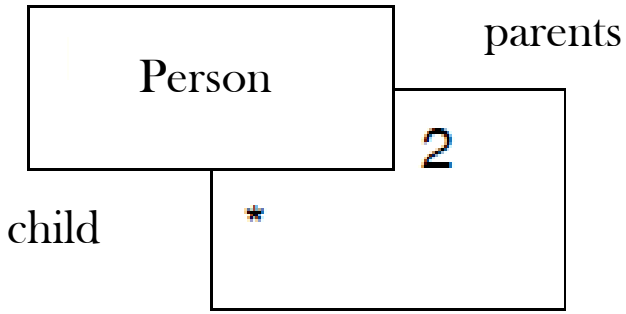
- An association end is defined by:
  - A name (the role played by the connected entity)
  - Multiplicity (0, 1, *, 1..*, ...)
  - The kind of aggeragation (composite, aggregation, none)
  - Others properties: isNavigable, isChangeable, etc.

Association: Notation



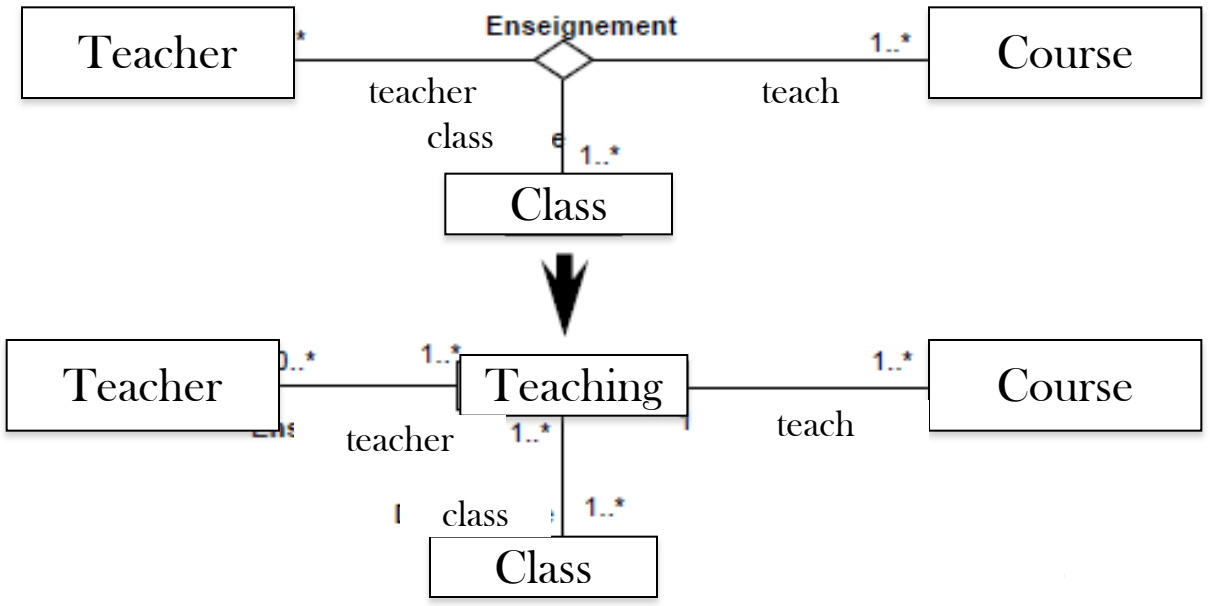
Reflexive association

- A reflexive association links objects of the same class



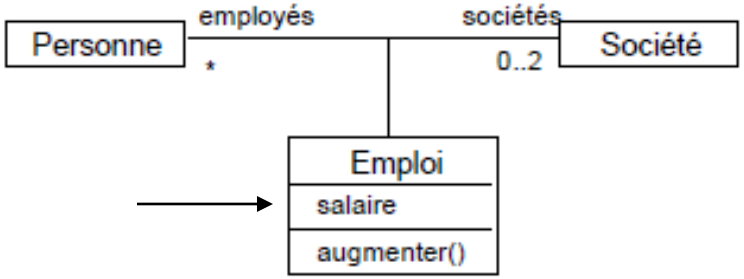
N-ary Associations

- Relation between more than two classes
- Can always be represented differently using binary associations

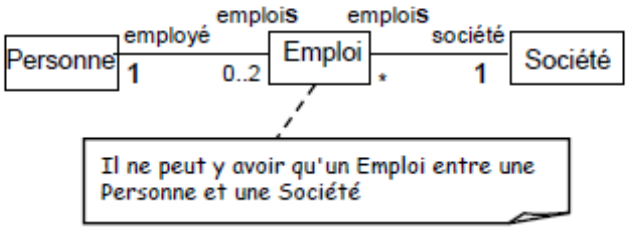


Association's class

- When the association contains data



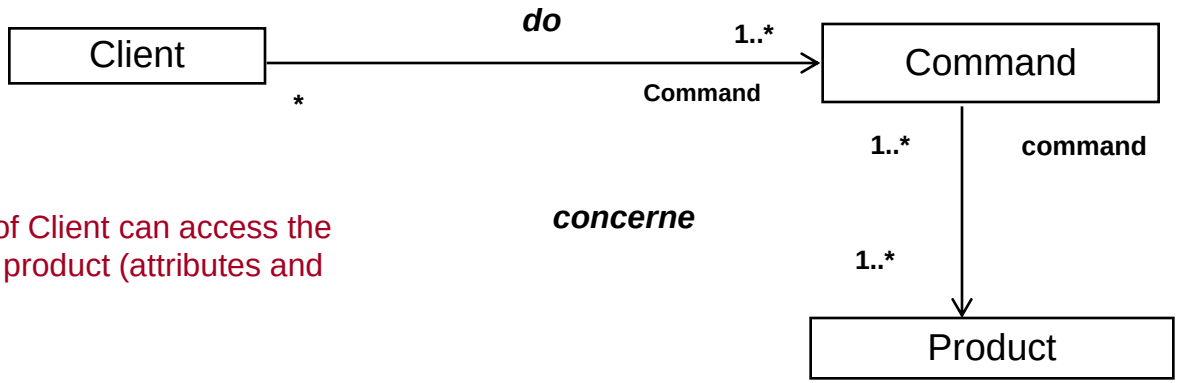
Traduction



Association: Navigability

- The way to access the properties (attributes and operations) of other classes.

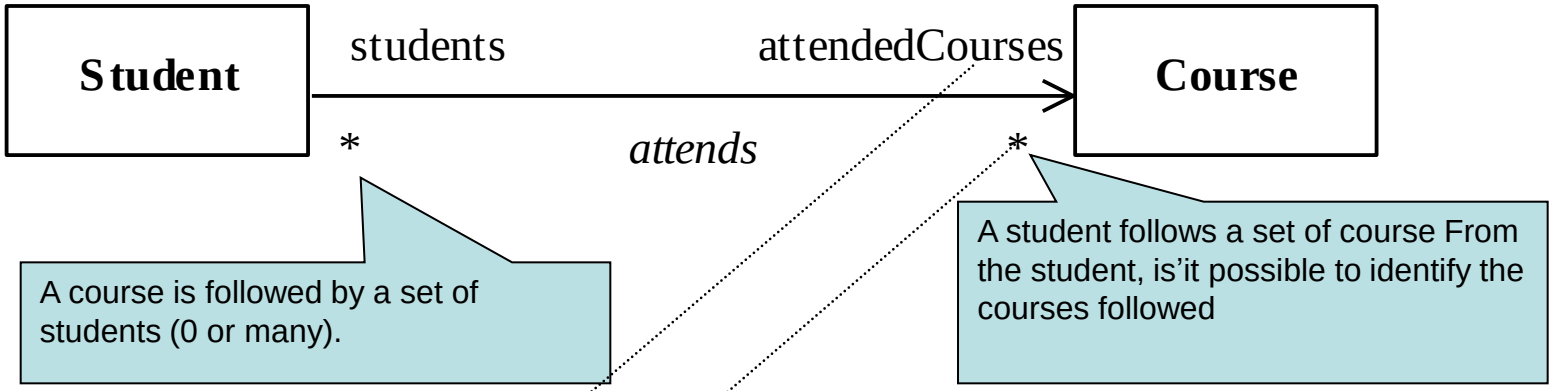
- Represented by an arrow at the association end.



An instance of Client can access the properties of product (attributes and operations)

Association : Navigability

- The impact of the navigability, multiplicity and role's names on the generated code

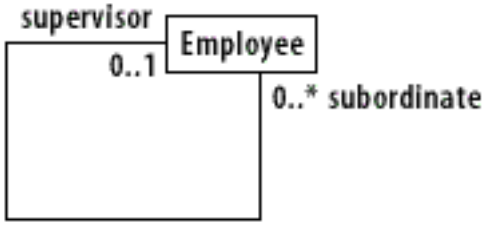



```
public class Student
{
  public Course attendedCourses[];

  public Student()
  {
  }
}
```

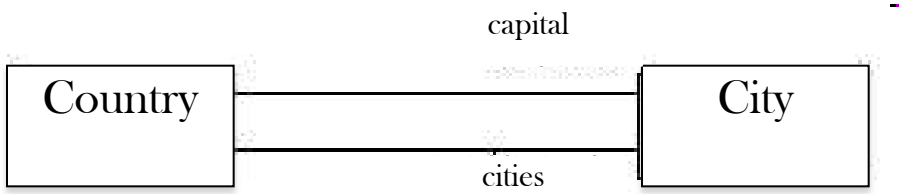
```
public class Course
{

  public Course()
  {

  }
}
```

Associations: Mandary Roles

- In the case of a reflexive association

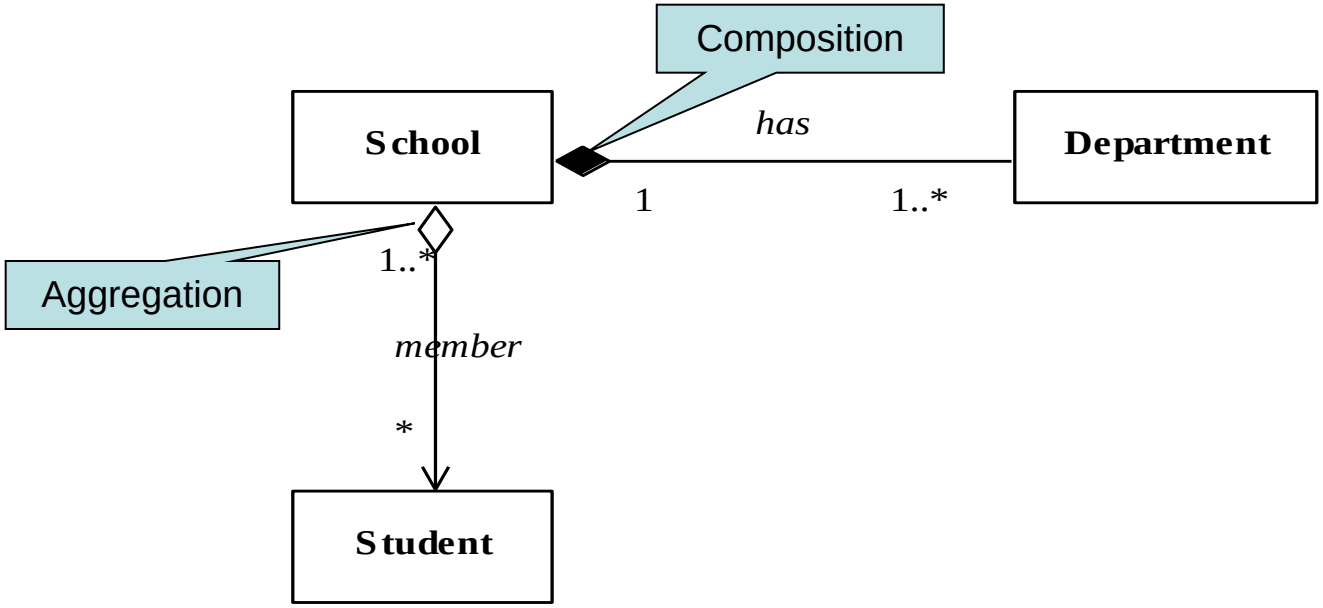


- Case of multiple associations between the same two classes



Associations: Aggregation & Composition

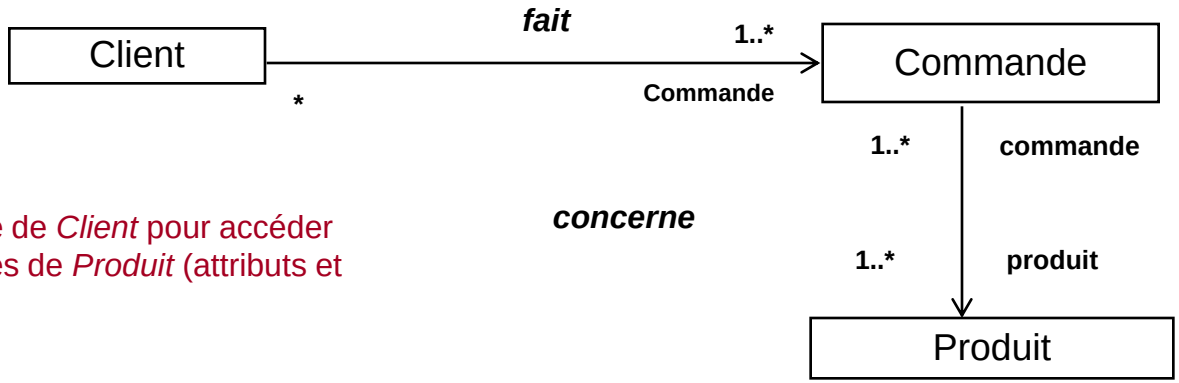
- Notion of « composed of », « contains » « is constrcuted from», ...
- Reinforces the association semantics (a set of objects that belong to another object)



Association : Navigabilité

- Le moyen d'accéder aux propriétés (attributs et opérations) d'autres objets à travers le graph d'objets représentant l 'application

- Représentée par une flèche
  - Attention à la notation en cas de navigabilité dans les 2 sens



Une instance de *Client* pour accéder aux propriétés de *Produit* (attributs et opérations)

Associations: Aggregation & Composition

- Don't overuse/ misuse of these association kinds!

- Aggregation is not very used => very similar to simple association
  - Main point: cycles are not allowed, comparing to associations

- **Avoid specifying your diagrams with questions such as : "If this class has to be deleted should this one be deleted too"? This will result in a class diagram full of compositions !!**

  This kind of association must stay exceptional

Generalization(Inheritance)

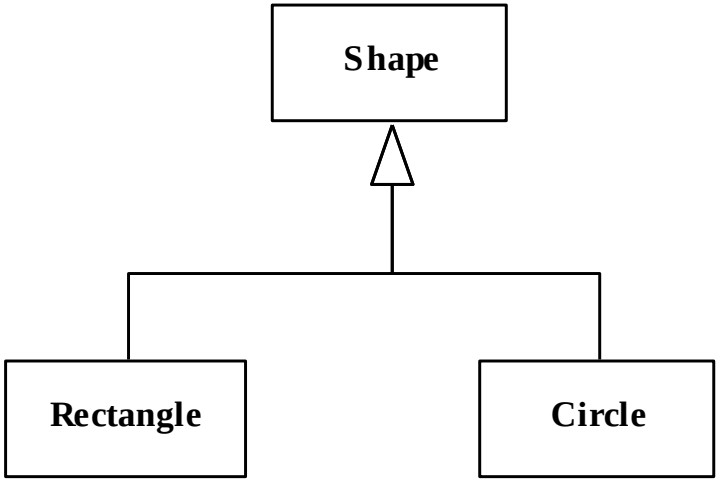
- Inheritance is a type of relation in UML
  - And not a type of association,

- Inheritance allows to share common (attributes, operations and associations), and preserves differences
- Can be simple or multiple
  - In Java, only simple inheritance

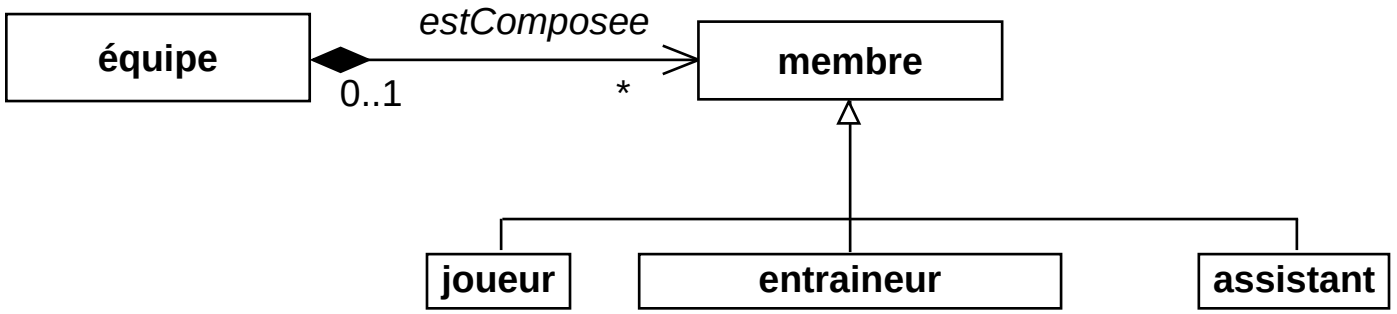
- Identifiable with words such as "is a kind of"

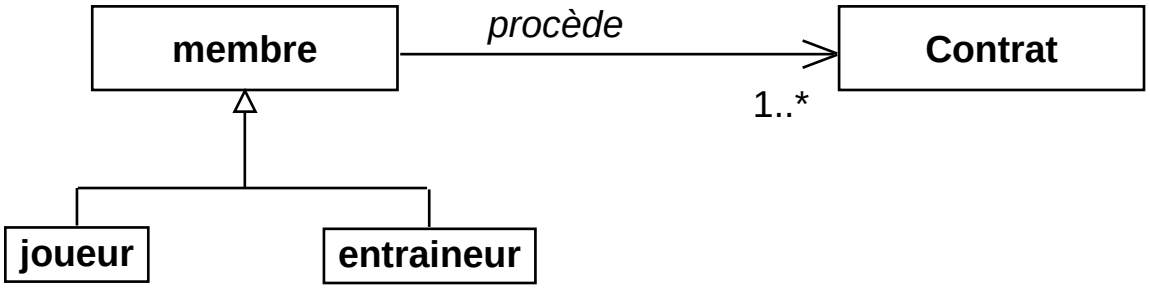
Generalization: Notation

- We say Generalization / Specialization
- Super classe, sub-classes



Generalization: Example (with association)





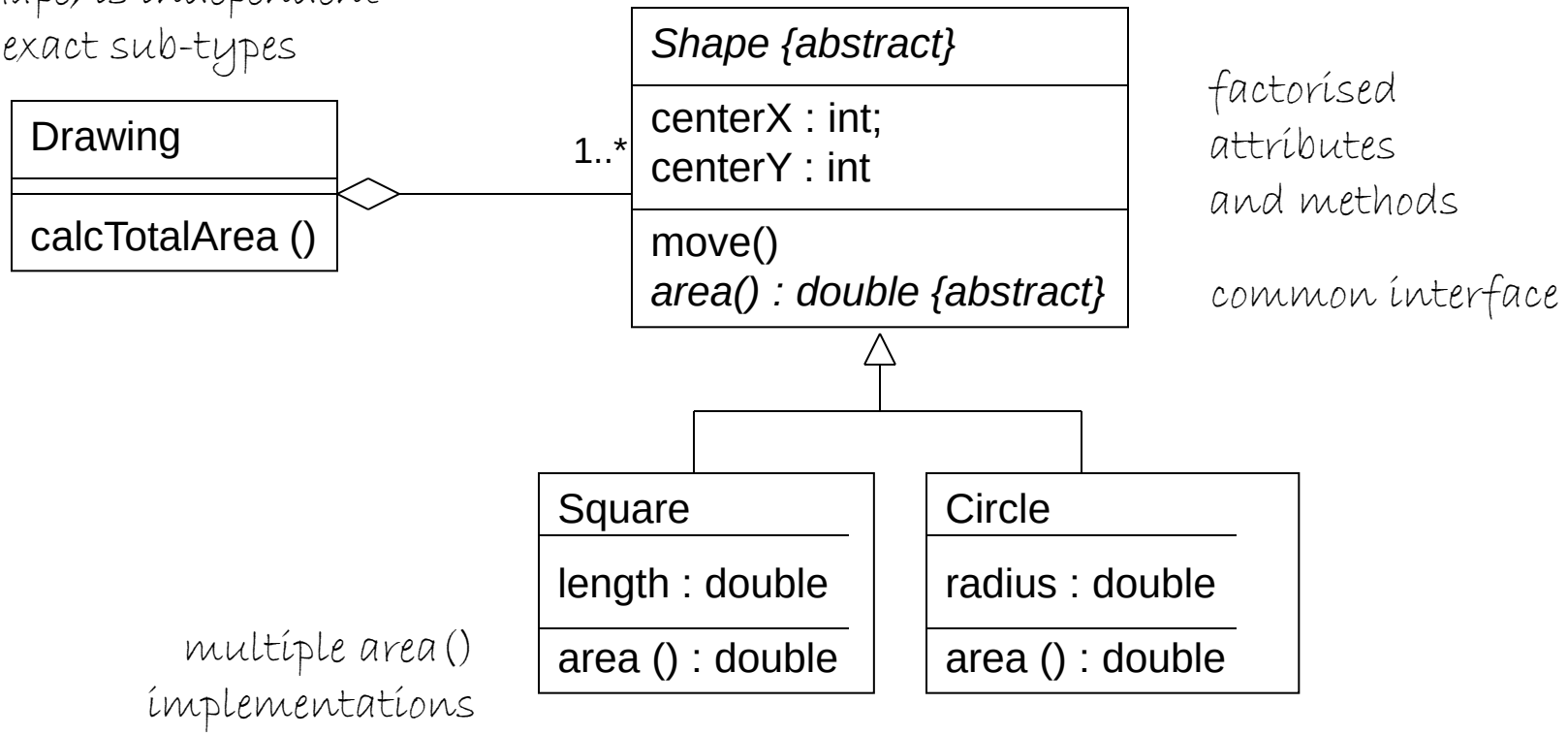
Abstract Classes and Opérations

- An abstract class is a class that contains at least one abstract operation
  - Capture common behaviors
  - Used to structure the system
  - Can not be instantiated

- An abstract operation is an operation whose implementation is left to subclasses

example

Drawing works with Shape, is independent of exact sub-types



factorised attributes and methods

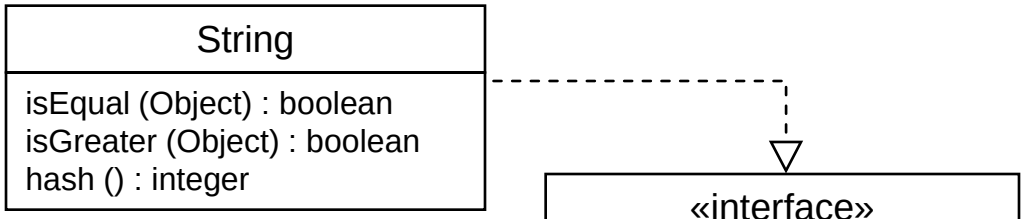
common interface

multiple area () implementations

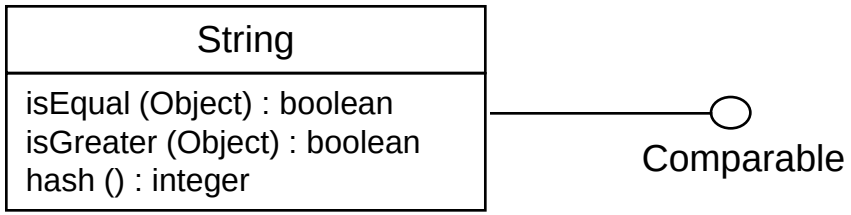
Interfaces

- A set of operations without implementation
  - Just signature
  - Can be viewed as an abstract class where all the operations are abstract
  - May contain constants
- A very powerful Typing mechanism
- A Class can realize one or multiple interfaces
  - Has to give an implementation for each of its operations

Interfaces: Notation

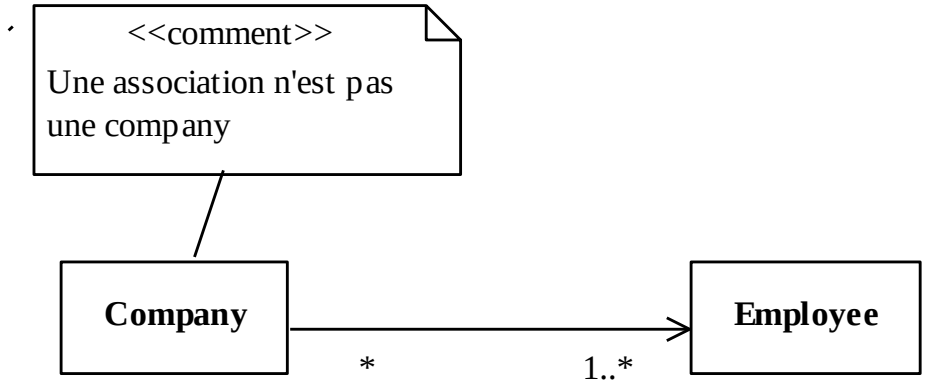


Or



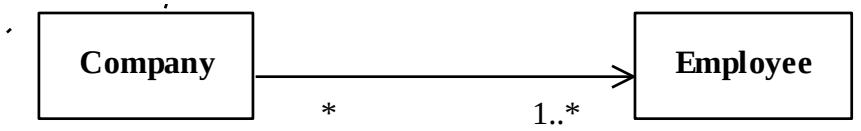
Notes (comments)

- Can be attached to any UML element for more precision / details
  - Some tools use them to put code inside for 100% code generation from the model
- Graphical Notation

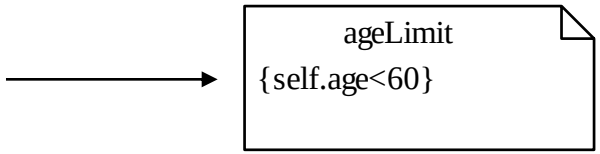


Constraints

- Can be business rules, structural constraints, etc.
- Can be expressed using natural language in notes or some predefined UML Constraints ({ordred}, {frozen}, etc.)
- Can be formalized using UML OCL(Object Constraint Language), OMG standard (not addressed in this lecture)



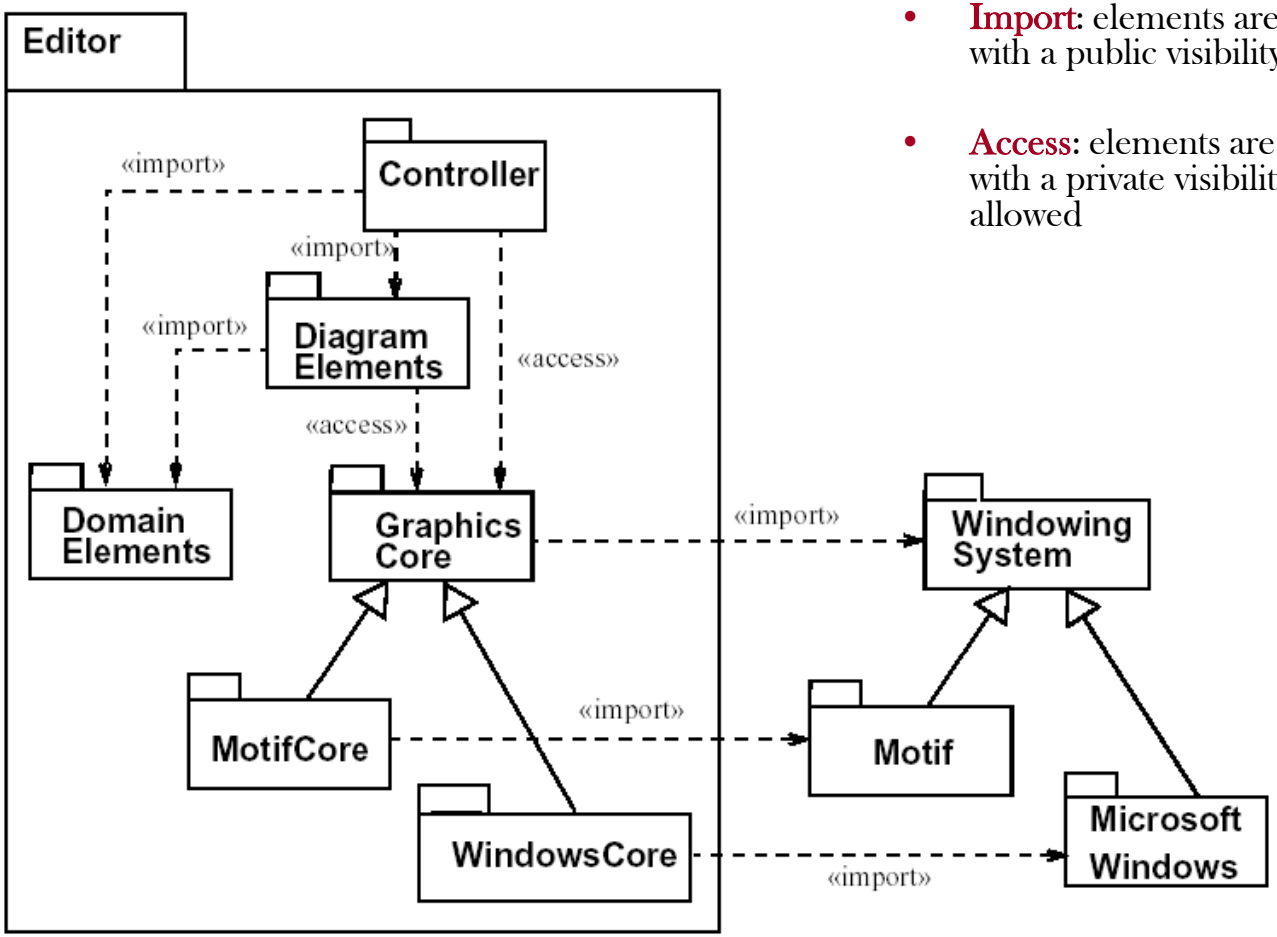
Example of an OCL constraint



Packages

- A grouping element for
  - Classes, use case, diagrams, etc.
- Serves as a Naming space
  - Two classes with the same name can't belong to the same package
- A package can import other packages
- Generalization is also possible

Les Packages: Example



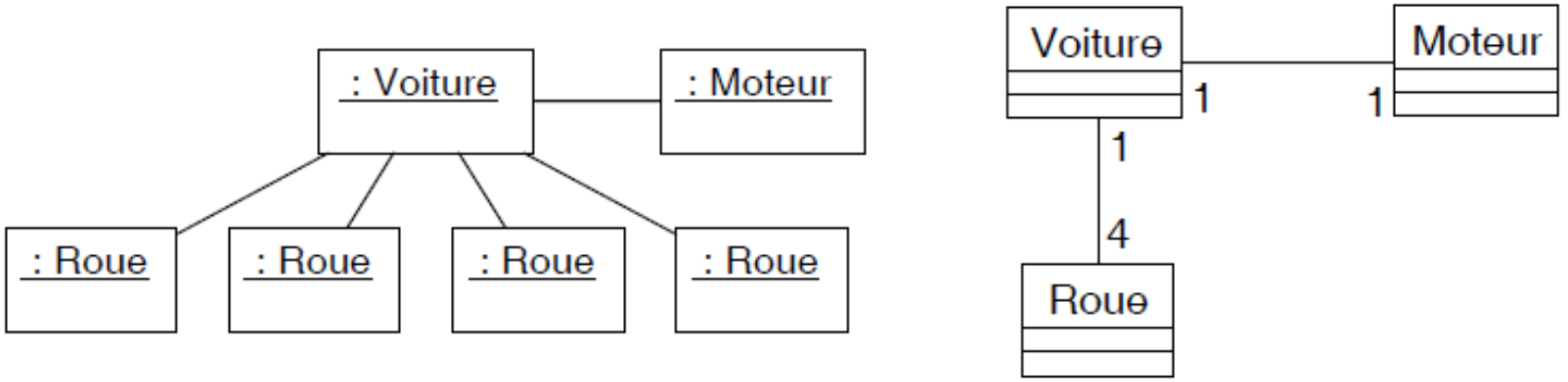
- **Import**: elements are imported to the package with a public visibility and it is transitive
- **Access**: elements are imported to the package with a private visibility. Transitivity is not allowed

Object Diagram

Object Diagram

- Is an instance of class diagram
- We talk about objects and links and not classes and associations
- Association roles are optional
- Useful to validate multiplicities in your class diagram, to give examples
- Not used very often in the industry

Object Diagram: Example



UML: Point de vue Dynamique

-Diagramme de Séquence
-Diagramme de Collaboration
-Diagramme d'État/Transition
-Diagramme d'Activité

Components diagram

Components

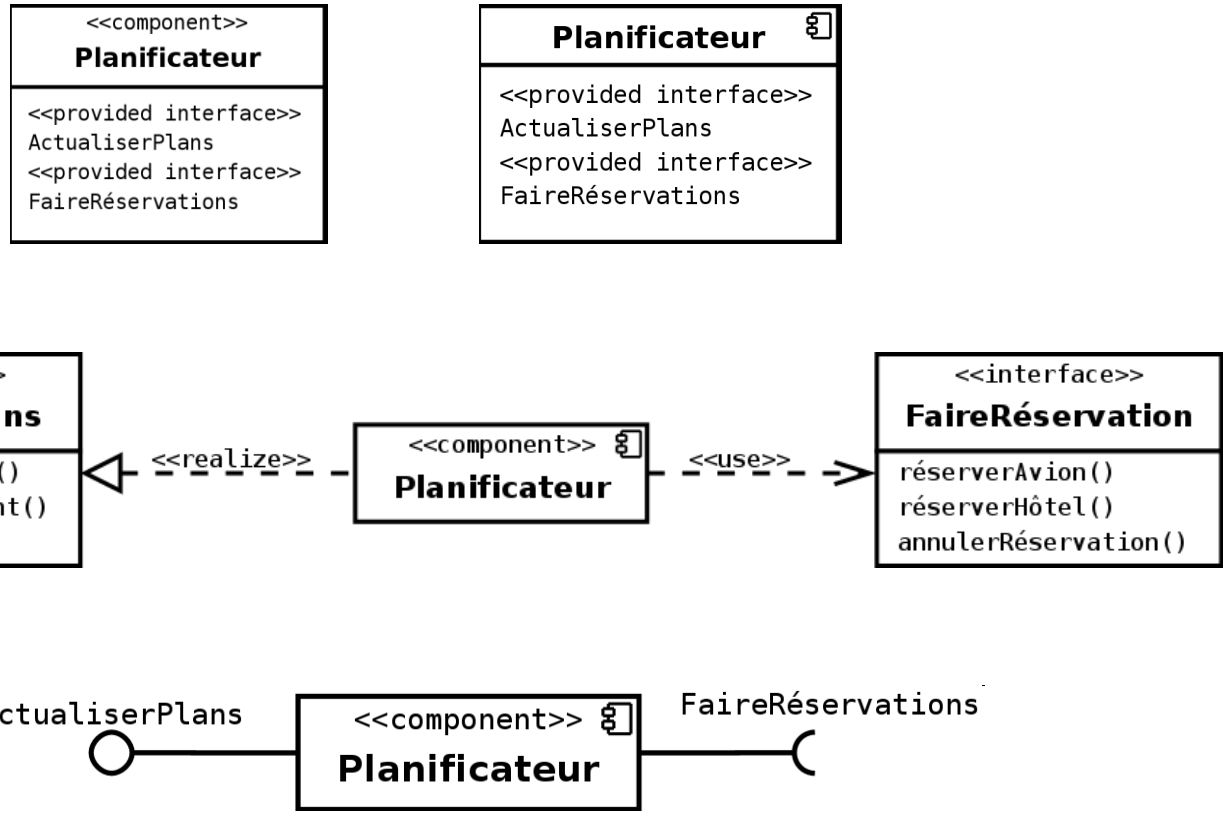
- « A component is a self contained unit that encapsulates the state and behavior of a number of classifiers » [UML 2.0, OMG]

- A lot of definitions around the notion of Component

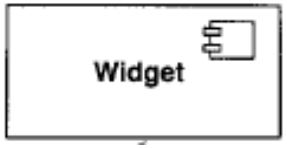
  "Components are not a technology. Technology people seem to find this hard to understand. Components are about how customers want to relate to software. They want to be able to buy their software a piece at a time, and to be able to upgrade it just like they can upgrade their stereo . They want new pieces to work seamlessly with their old pieces, and to be able to upgrade an their own schedule, not the manufacturer's schedule . They want to be able to mix and match pieces from various manufacturers. This is a very reasonable requirement. It is just hard to satisfy". Ralph Johnson

- Components diagram gives an overview of the application's architecture in terms of components, interfaces and dependencies between components (through required/provided interfaces)

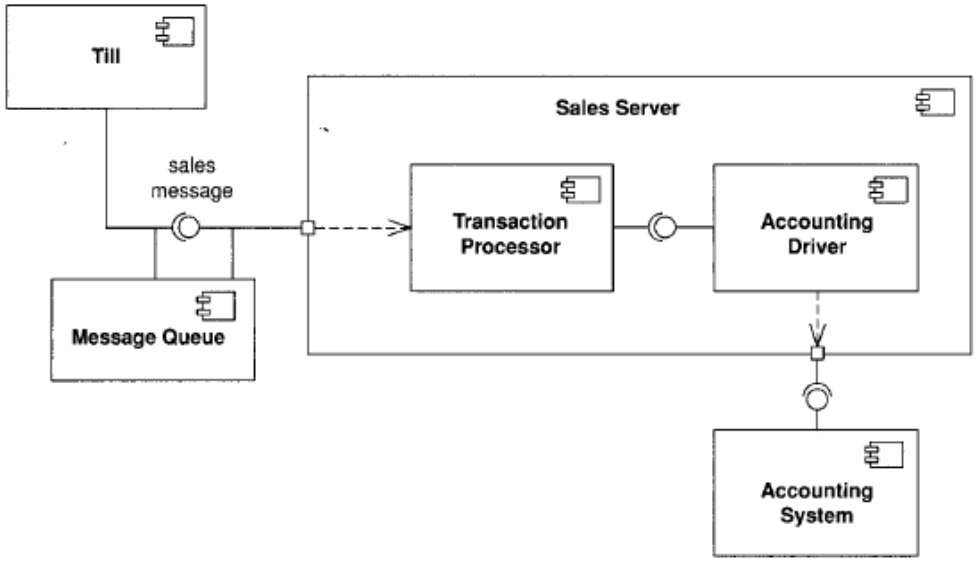
Components diagram: Notation



Components diagram : Example



Notation UML 2.0

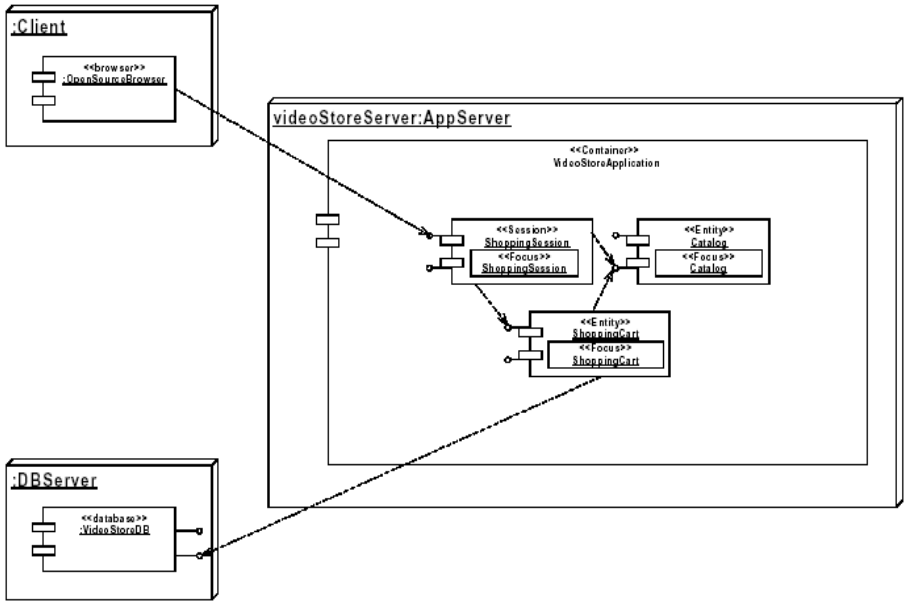


Un Exemple de diagramme de Composants

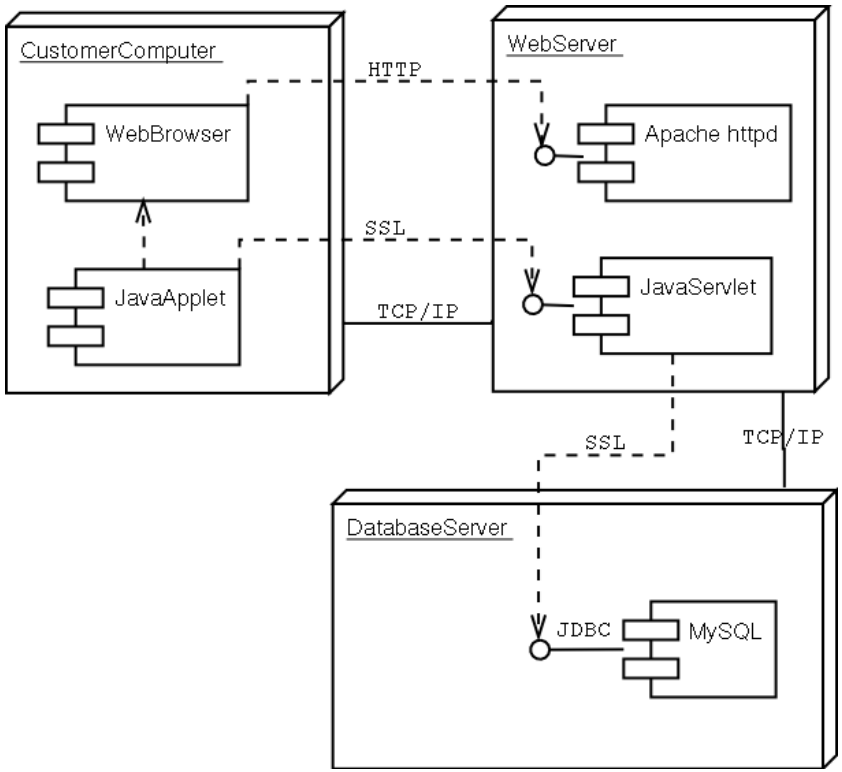
Deployment diagram

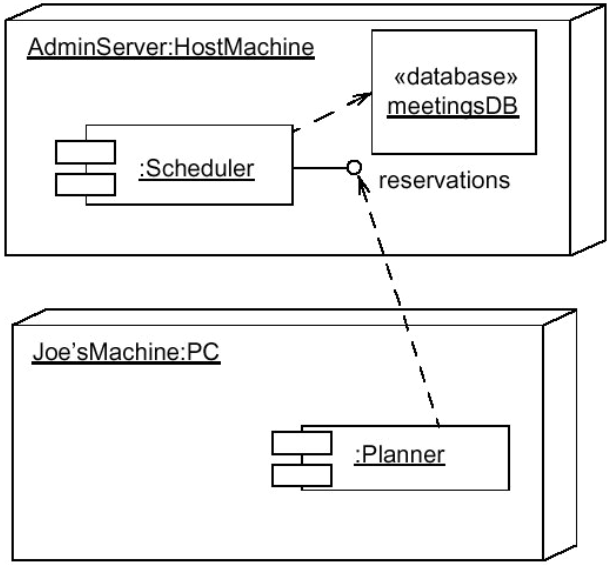
Deployment diagram

- Shows how application's components are physically deployed in the application's environment
  - Physical elements (servers, departments, etc.)
  - Components
- Very useful to think about distribution, performances, hardware, required, protocols, etc.



Deployment diagram: Examples





Readings

- Software Engineering,
  - Ian Sommerville, Addison Wesley; 8 edition (15 Jun 2006), ISBN-10: 0321313798
- The Mythical Man-Month
  - Frederick P. Brooks JR., Addison-Wesley, 1995
- Cours de Software Engineering du Prof. Bertrand Meyer à cette @:
  - http://se.ethz.ch/teaching/ss2007/252-0204-00/lecture.html
- Cours d'Antoine Beugnard à cette @:
  - http://public.enst-bretagne.fr/~beugnard/

- UML Distilled 3rd édition, a brief guide to the standard object modeling language
  - Martin Fowler, Addison-Wesley Object Technology Series, 2003, ISBN-10: 0321193687
- UML2 pour les développeurs, cours avec exercices et corrigés
  - Xavier Blanc, Isabelle Mounier et Cédric Besse, Edition Eyrolles, 2006, ISBN-2-212-12029-X
- UML 2 par la pratique, études de cas et exercices corrigés,
  - Pascal Roques, 6ème édition, Edition Eyrolles, 2008
- Cours très intéressant du Prof. Jean-Marc Jézéquel à cette @:
  - http://www.irisa.fr/prive/jezequel/enseignement/PolyUML/poly.pdf
- La page de l'OMG dédiée à UML: http://www.uml.org/
- Cours de Laurent Audibert sur http://laurent-audibert.developpez.com/Cours-UML/html/Cours-UML.html

---

- Design patterns. Catalogue des modèles de conception réutilisables
  - Richard Helm (Auteur), Ralph Johnson (Auteur), John Vlissides (Auteur), Eric Gamma (Auteur), Vuibert informatique (5 juillet 1999), ISBN-10: 2711786447