Software Engineering with UML

Tewfik Ziadi
ziadi@ece.fr

Based on slides of:
1. R. Bendraou (UPMC)
2. J.Templemore (ECE)

Organisation of the course (1/4)

- Week 1 : Introduction to SE with UML
  - Part 1 : Course
  - Part 2 : Exercises/ Lab work
- Week 2 : Requirement analysis: Use case diagrams
  - Part 1 : Course
  - Part 2 : Exercises/ Lab work
- Week 3 : Structural modeling class diagrams + design pattern
  - Part 1 : Course
  - Part 2 : Exercises/ Lab work
- Week 4: Structural modeling : Lab work and work on project.
  - Part 1 : Exercises/ Lab work
  - Part 2 : Work on the project.

Organisation of the course (2/4)

- Week 5 : Sequence diagrams + state machines
  - Part 1 : Course
  - Part 2 : Exercises/ Lab work
- Week 6 : Sequence diagrams + state machines: Lab work
  - Part 1 : Exercises/ Lab work
  - Part 2 : Work on the project
- Week 7 : Code generation & reverse engineering
  - Part 1: Course
  - Part 2: Exercises/ Lab work
- Week 8 : Software Testing
  - Part 1 : Course
  - Part 2 : Exercises/ Lab works

Organisation of the course (3/4)

- Week 9 : Software Product Lines
  - Part 1 : Course
  - Part 2 : Exercises/ Lab work

Organisation of the course (4/4)

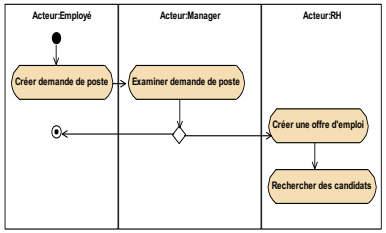
Evaluation

- Intermediate Written Test (Week October 16) ==> (20%)
- Long Mini Project (30%)
- Final Exam (50%)

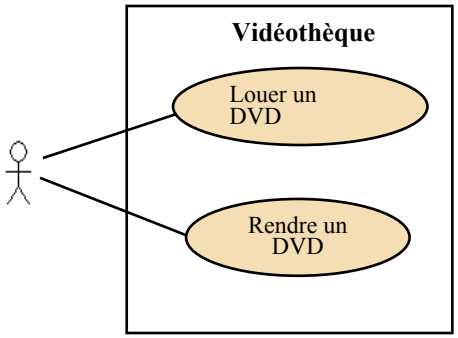
Long Mini Project (using the Modelio UML tool)

- Applying the concepts to develop a system InterimECE (the specification document is available on the campus web page).
- Groups of 2 students (maximum 3).
- One final report (deadline before Week 8).

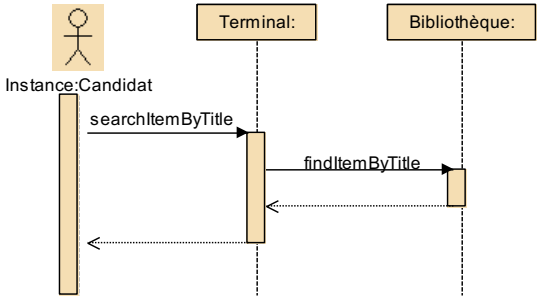
The UML Is a Language for Documenting



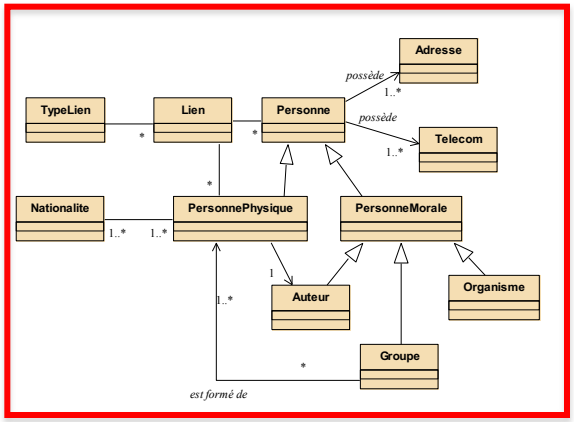
Activity diagrams



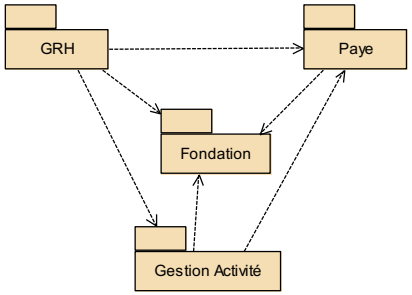
Use case diagrams



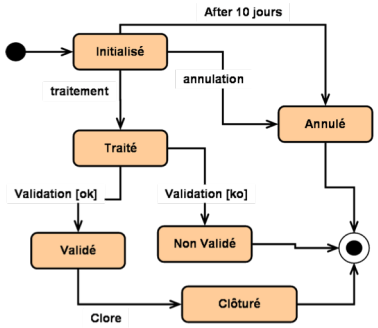
Sequence diagrams



Class Diagrams



Package Diagrams



State machine diagrams

Modeling systems with the UML

Structural modeling with UML class diagrams

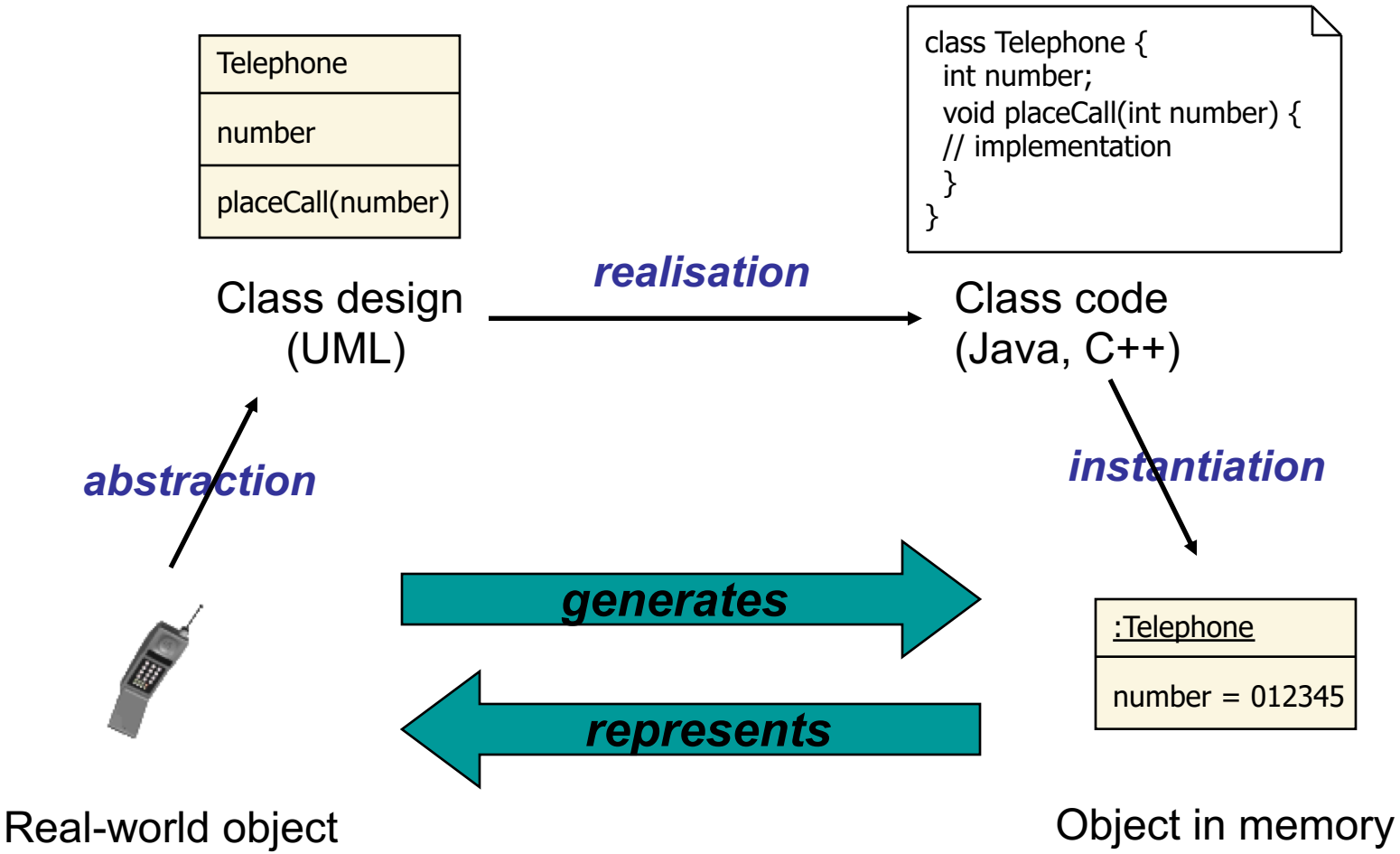
Structural modeling

- Defines some aspect of the *static structure* of a system
- **Logical structure** describes the data that is managed by the system as a graph of related classes.
  - UML class diagram
  - UML objet diagram

Logical data structure

- Describes the static structure of data managed by an IS.
- Expressed as a class diagram that defines a set of complex data **classes** (*types*), and the **relationships** between these classes.
- A class diagram is one of the core object model elements
  - It can be translated to a database structure
  - It can be used to generate a code skeleton

Objects and classes: Reminder



Class diagrams elements

- Classes
  - Properties(attributes)
  - Operations(methods)
- Relationships between classes
  - association (for linking classes)
  - aggregation (for decomposition)
  - generalization (for factorization and polymorphism)
- Relationship properties
  - multiplicities
  - roles
  - navigability

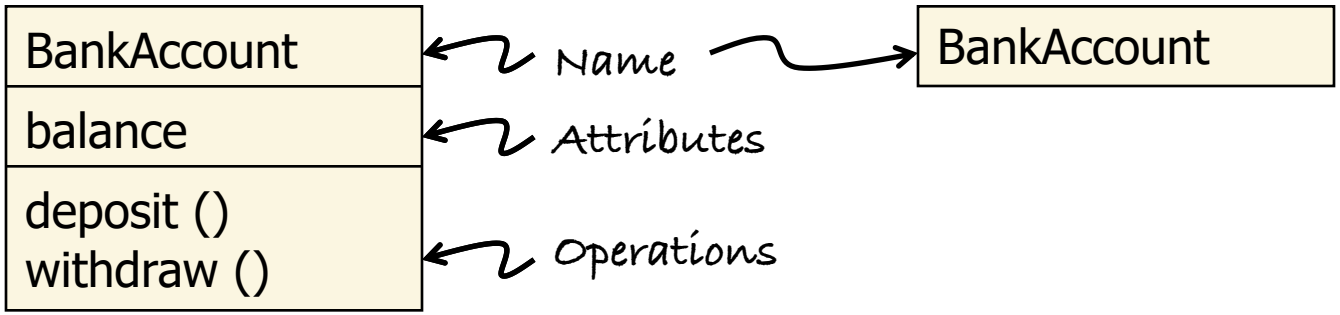
Class

- *A class is a pertinent abstraction of a category / type of real world objects found in the problem domain.*

- A class defines
  - a name, describing the responsability of each object
  - attibutes, defining the data structure of each object
  - operations, defining the services offered by each instance

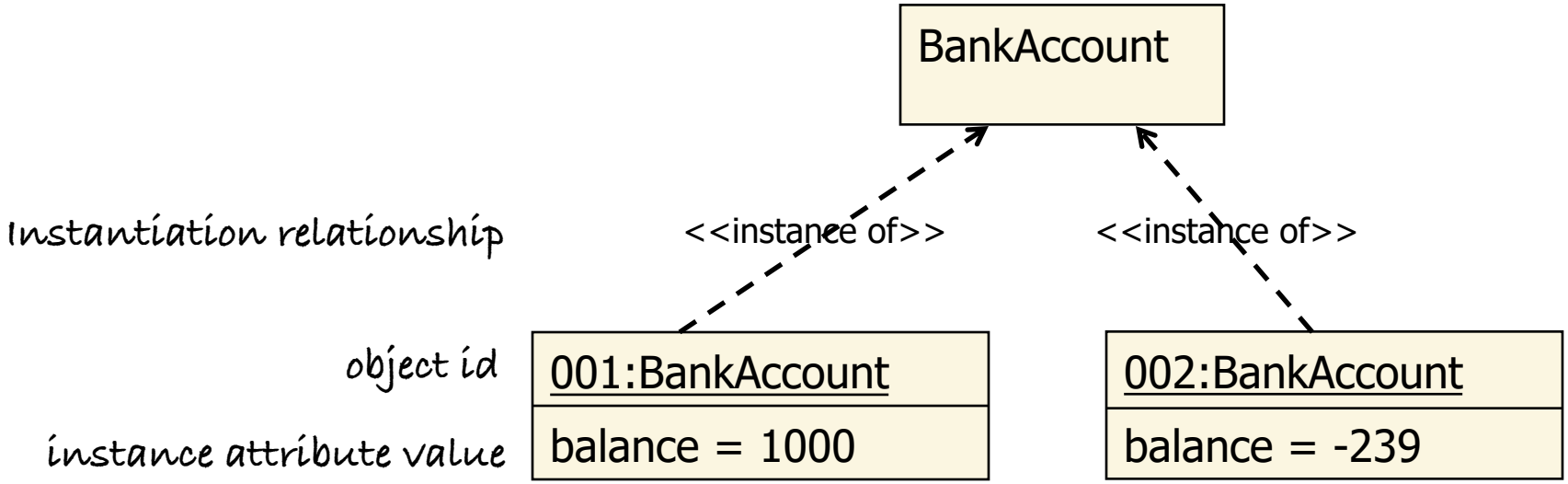
- A class is represented as a rectangle with separate "boxes" for its name, attributes and operations.

Class



- Most details can be hidden if useful
- Only the class name is obligatory

Object


BankAccount
Instantiation relationship
<<instance of>>
<<instance of>>
object id
instance attribute value
001:BankAccount
balance = 1000
002:BankAccount
balance = -239

Attributes

- An attribute defines a value that qualifies an object.
  - A Car's colour attribute indicates that Cars have colours.

- An attribute defines data needed by an object to fulfil its responsability

- Attributes are usually types with:
  - Primitives UML types: integer, string, char, boolean and real.

Attribute syntax

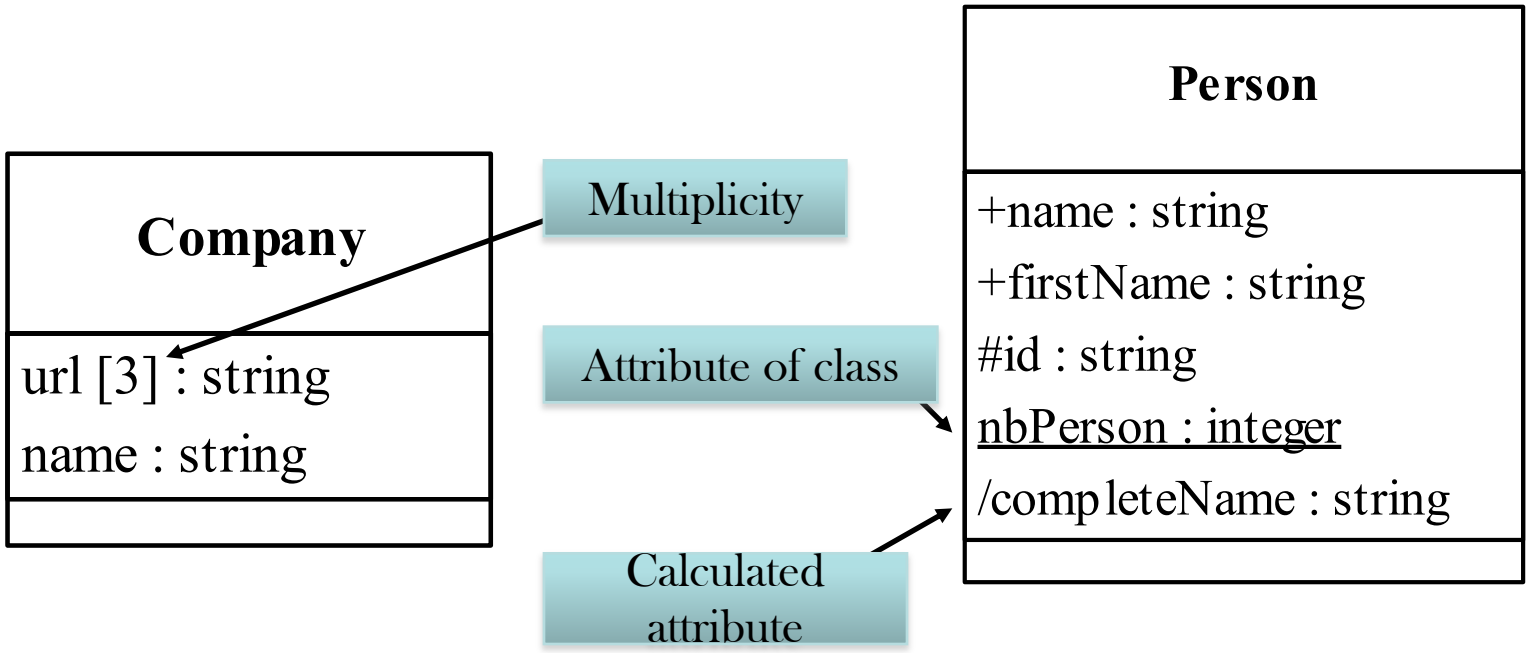
```
visibility name : type = defaultValue
```

- The amount of detail shown depends on the phase of the project.

- Examples
  - seconds : 0..59
  - name : string
  - quantity : integer = 0

- Visibility will be discussed later

Attributs: Examples



Operations

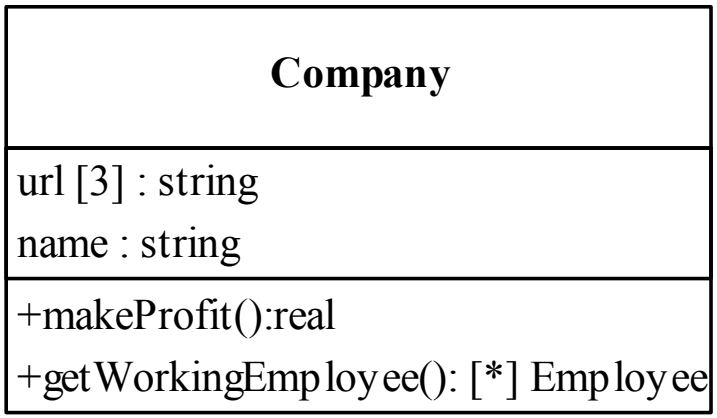
- An operation defines a service offered and carried out by an object.
  - SavingsAccount offers the operations creditInterest(), deposit(), withdraw()
  - Operations read / change the data of the object

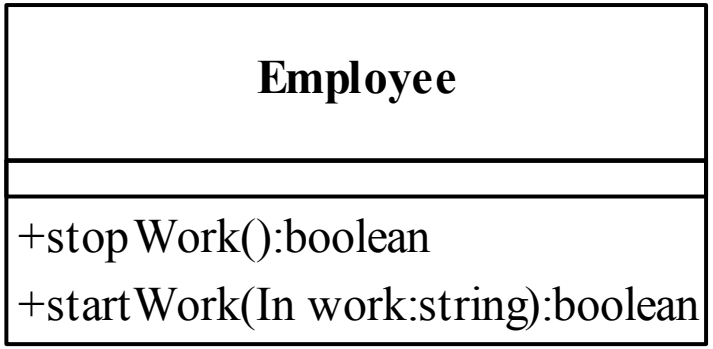
- Conceptually, operations describe the responsibilities of a class.
- But more obviously, operations correspond to the programmatic methods defined in a class.

Operations

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

Operations: Examples





Visibility

- Visibility is used to specify encapsulation
  - Specify the *public interface* and *hidden implementation*

- UML defines three levels of visibility :
  - + (public) : feature may be read or set by any object (interface)
  - - (private) : feature may only be read or set only by the owner object
  - # (protected) : feature may be read or set by the owner object or instances of sub-types of that object's type.

Basic relationships

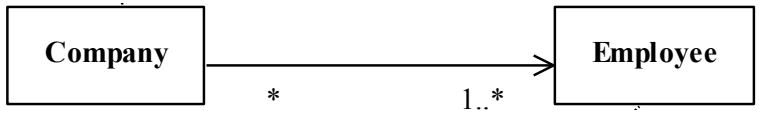
- Association
  - Defines a simple relationship between classes
  - *Used in 80% of relationships*

- Aggregation / composition
  - Aggregation defines an all/part relationship, used to decompose a complex class
  - Composition is a "strong" aggregation in which the parts depend entirely on the whole

- Generalisation / specialisation
  - Defines a subclass / subcategory / a type of a type
  - Used to explicitly show similarities differences
  - Defines a hierarchy of classes, allows factorization and polymorphism

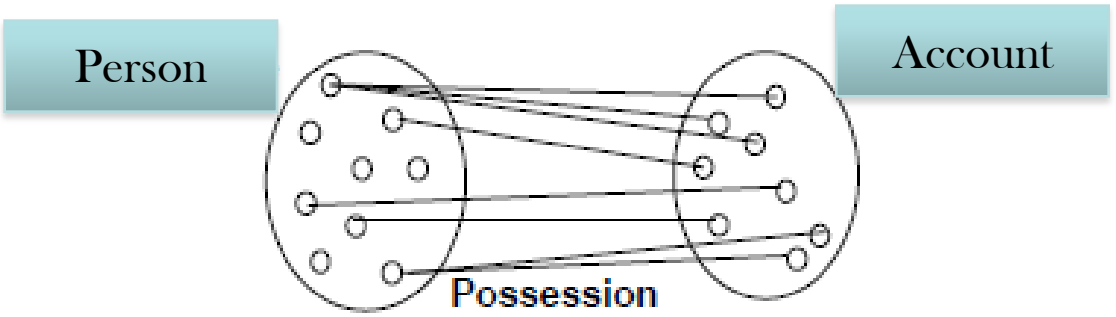
Associations

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



Associations

- An association specifies links between the set of objects.

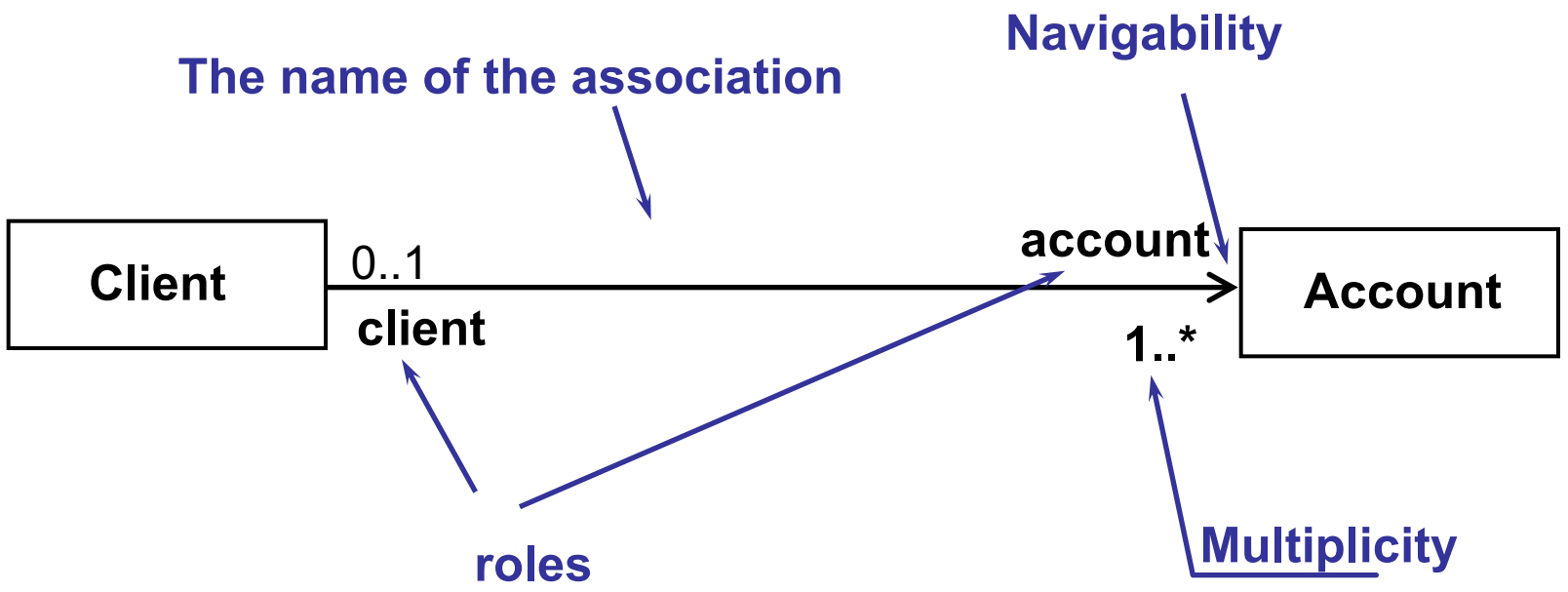


Les Associations

- A binary association is composed of two association ends.

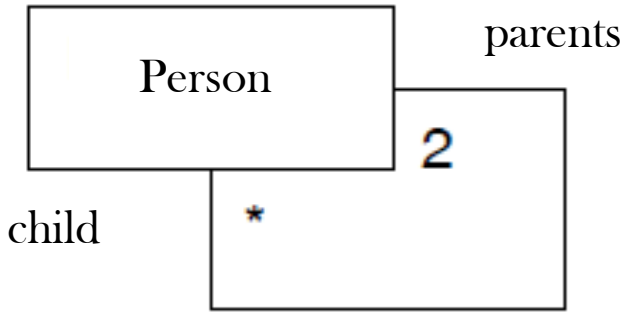
- An association end is defined by:
  - A name (the role played by the connected entity)
  - Multiplicity (0, 1, *, 1..*, ...)
  - The kind of aggeragation (composite, aggregation, none)
  - Others properties: isNavigable, isChangeable, etc.

Association: Notation



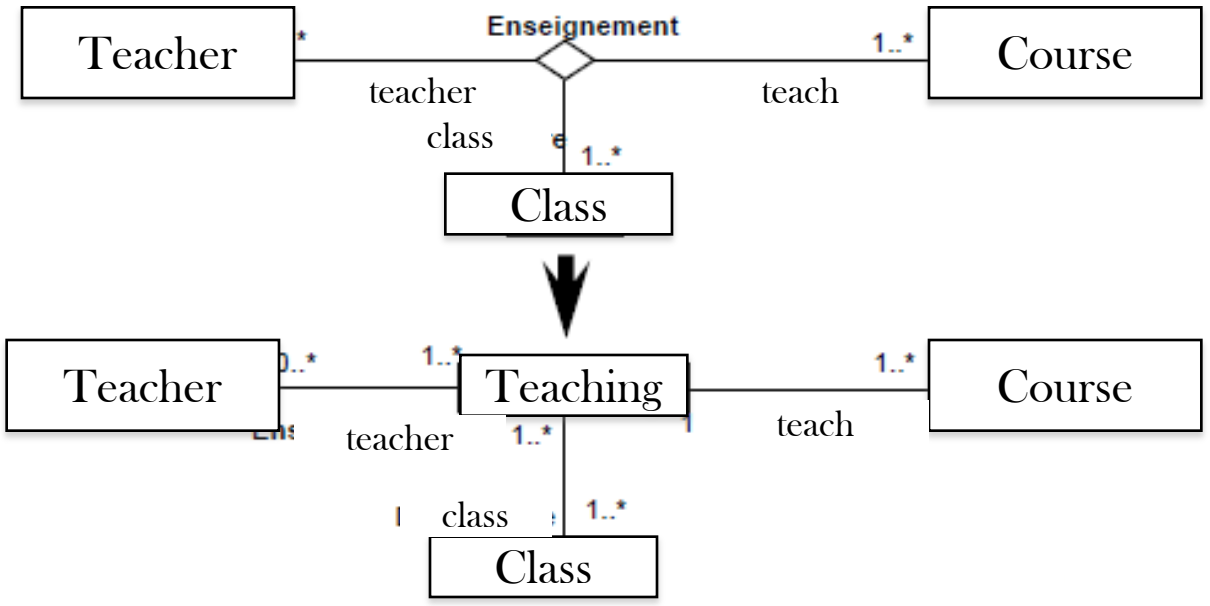
Reflexive association

- A reflexive association links objects of the same class



N-ary Associations

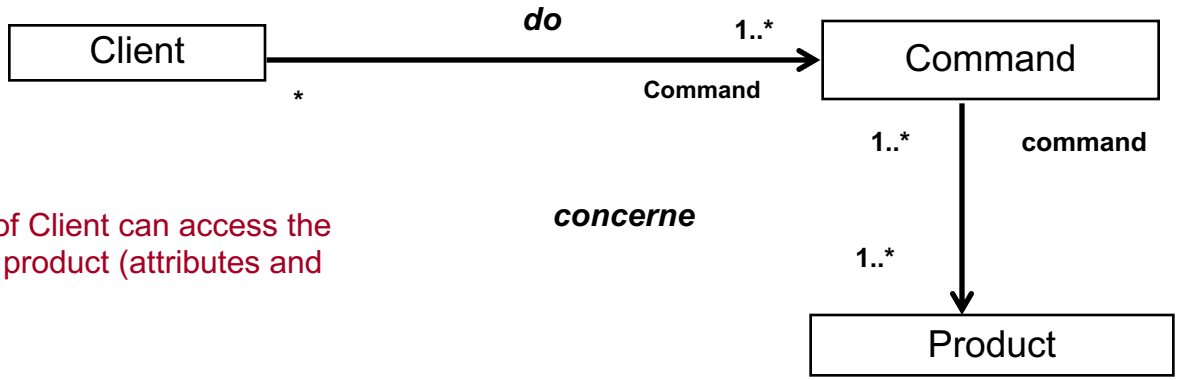
- Relationship between more than two classes
- Can always be represented differently using binary associations



Association: Navigability

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

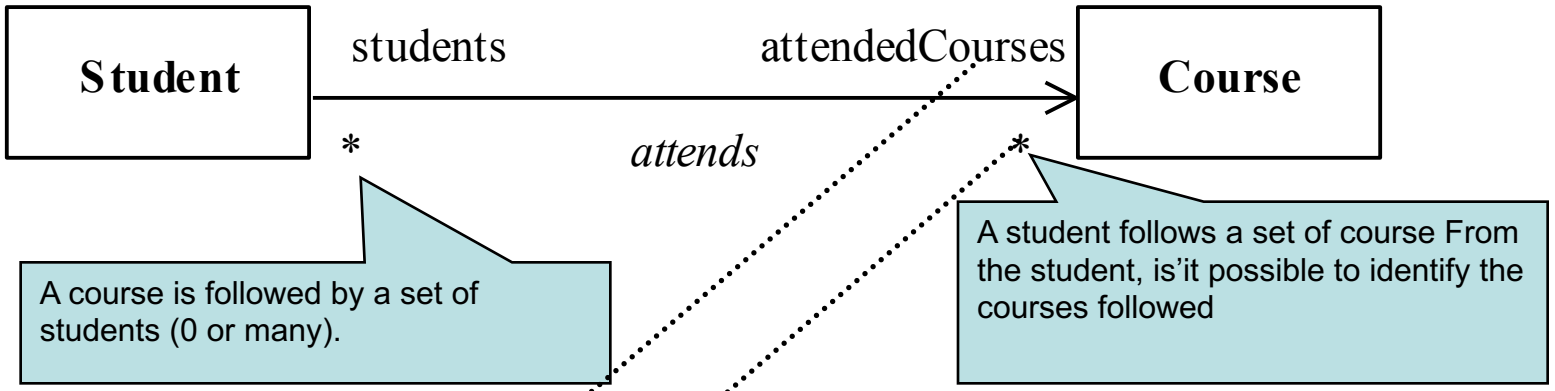
- Represented by an arrow at the association end.



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

Association : Navigability

- The impact of the navigability, names and roles on code generation

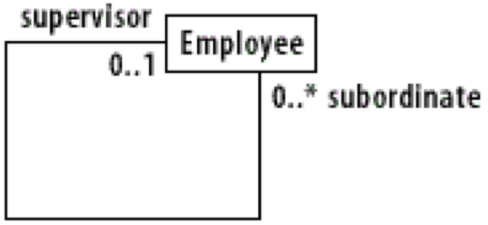



```
public class Student
{
   public Course attendedCourses[];

   public Student()
   {
   }
}
```

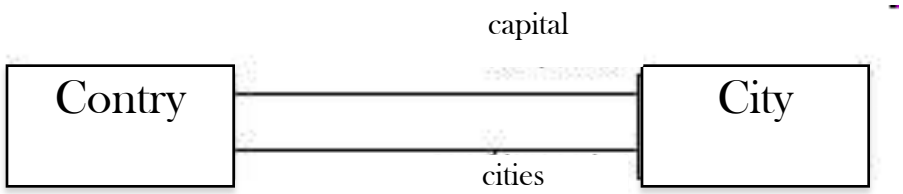
```
public class Course
{

   public Course()
   {

   }
}
```

Associations: Roles Indispensable

- In the case of a reflexive association

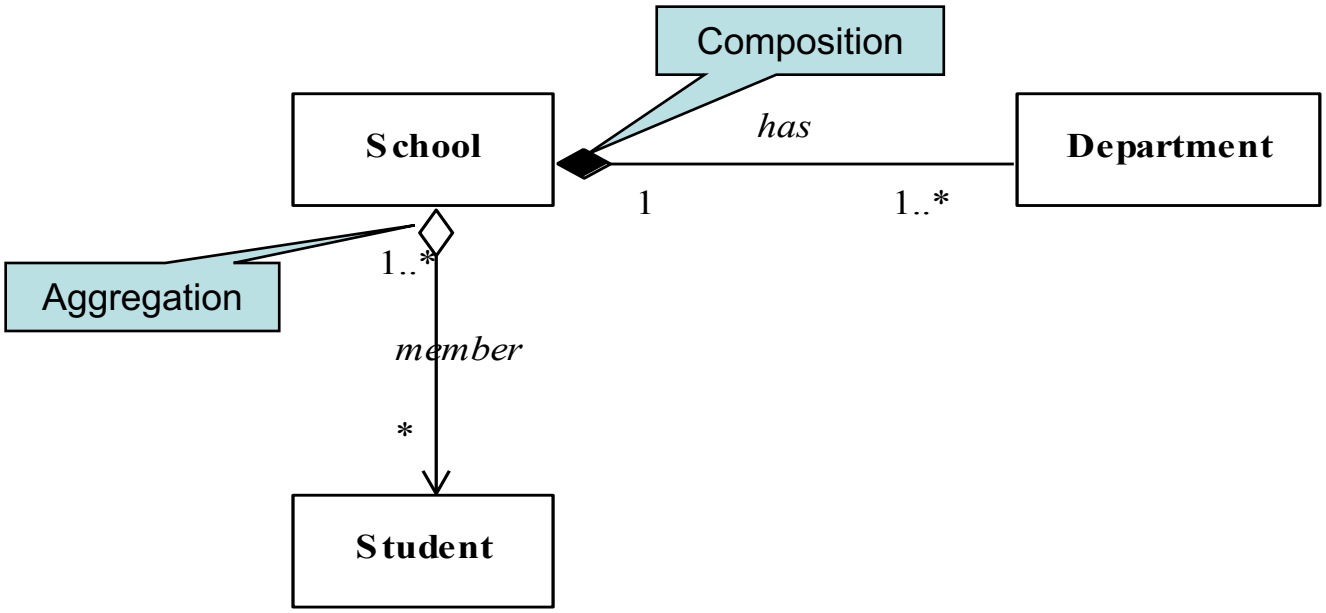


- Case of multiple associations between the same two classes



Associations: Aggregation & Composition

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

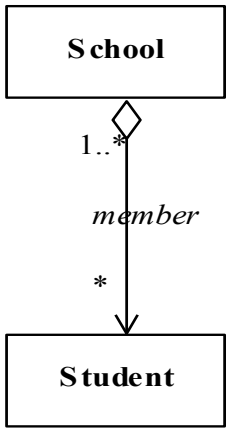


Associations: Aggregation

- Aggregation expresses a week composition (shared)
- A component can ne part of multiples composites.
  - The multiplicity on the side of composite may be (1...*)

- The life cycle of the component is not linked to that of the composite.

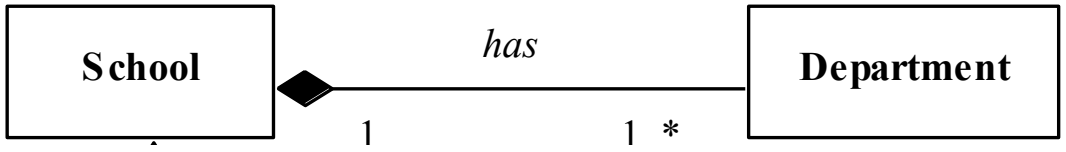
if the composite is delete, the components remain



Associations: Composition

- Specifies a strong composition (not shared)
- The component is a part of a single composite
  - Multiplicity 1 on the side of composite
- The life cycle of the component is linked to that of the composite

  if the composite is delete, the components will be also.



Generalization(Inheritance)

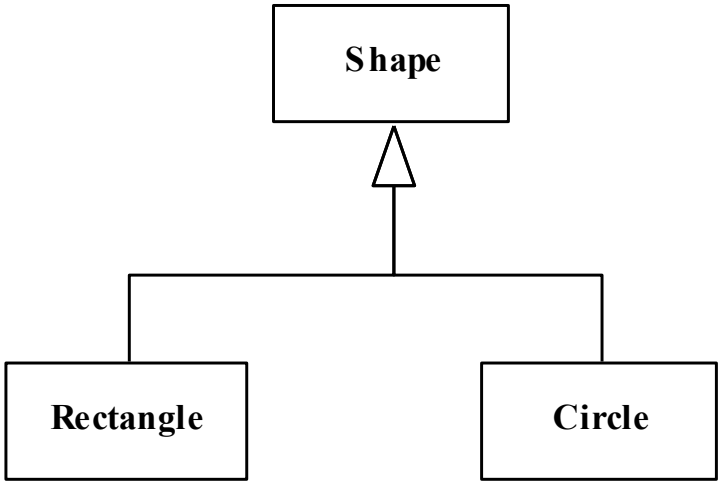
- Inheritance is a type of relationship in UML
  - And not a type of association,

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

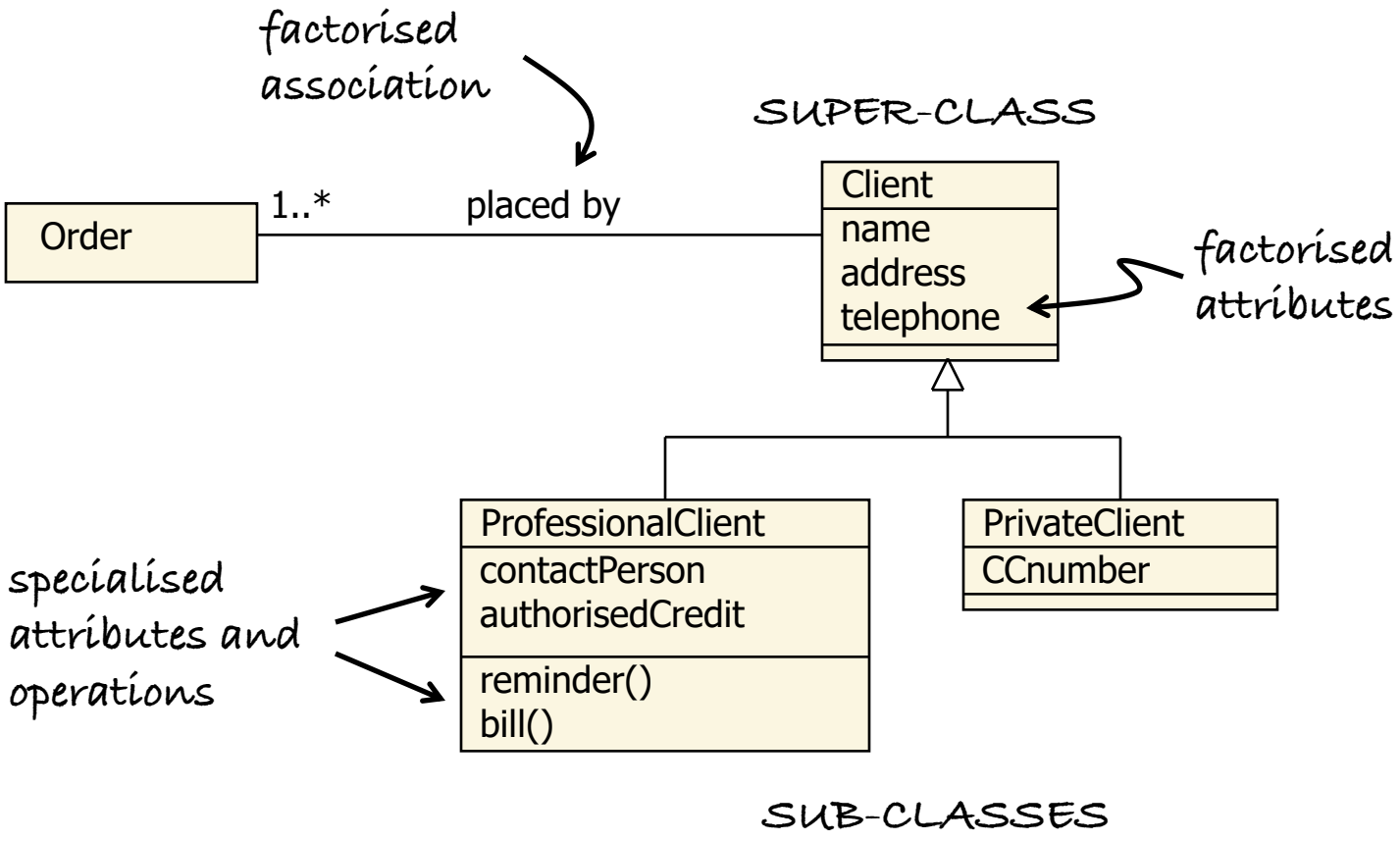
- Identifiable with words such as "is a kind of"

Generalization: Notation

- We talk on Generalization / Specialization
- Super classe, sous-classes



Factorisation example

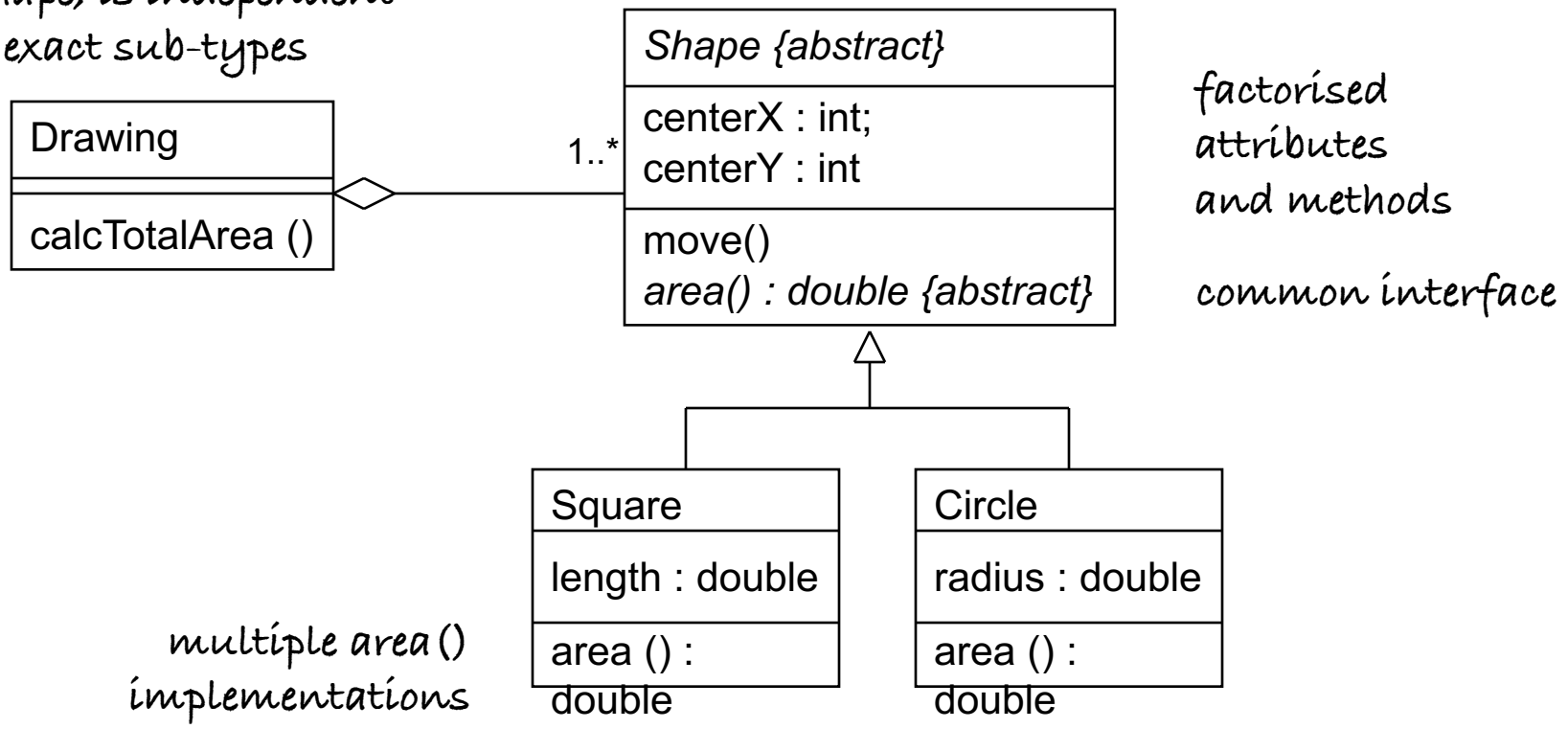


Abstract Classes and Opérations

- 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

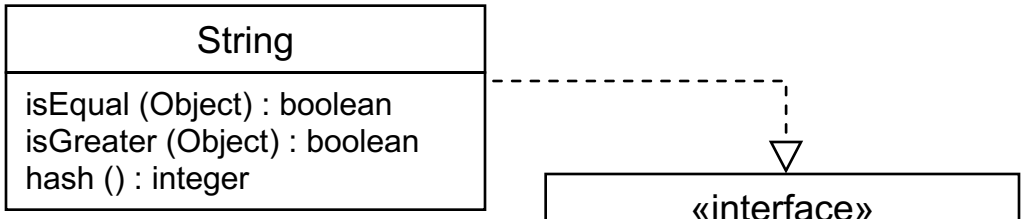


Interfaces

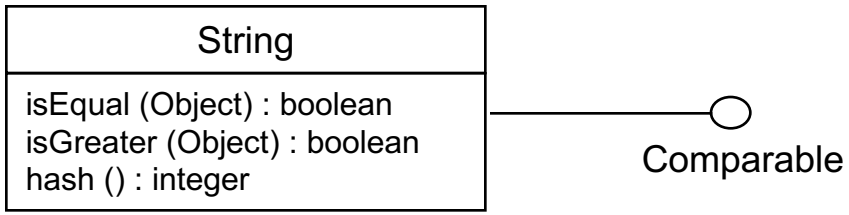
- A set of operations without body
  - Just signatures
  - Can be seen as an abstract class from which all operations are abstract

- A class can implement one or several interfaces.

Interfaces: Notation



Or:

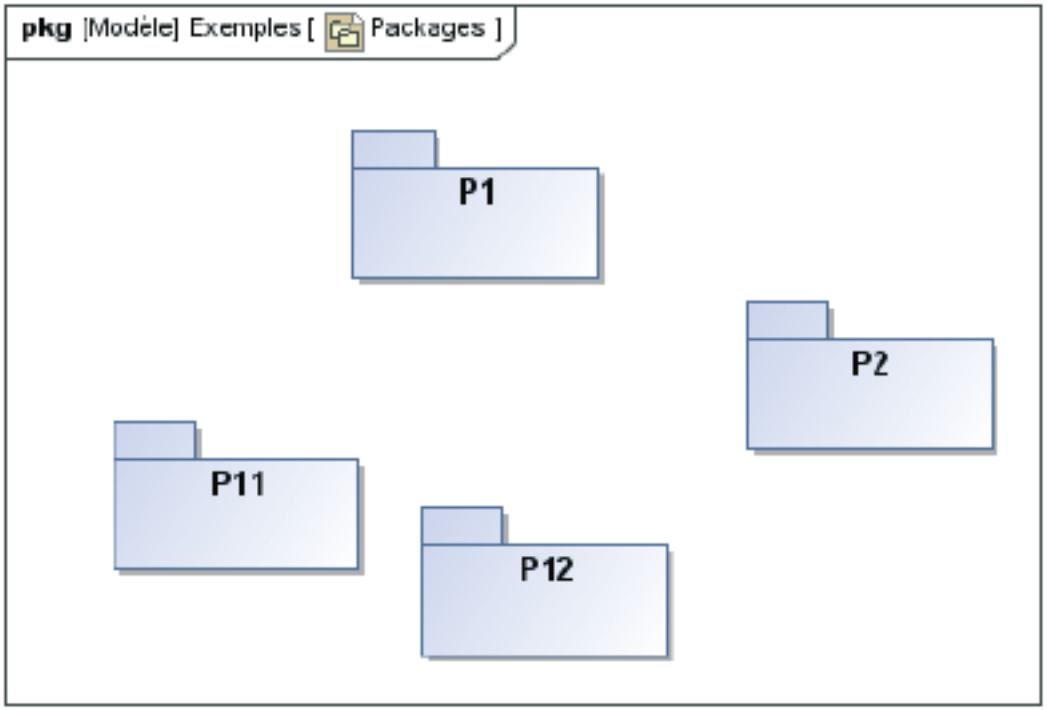


Packages

- A package allows to gather a set of elements
  - Classes, use case, etc.
- A package is a namespace (namming)
  - Two classes with the same name can not be in the same same package.
- A package can contain others packages.
- A package can import ou use others packages.

Package diagrams

- Package: a container of classes.



Packages

- Dependency : package P1 needa another package package P2
- Dependency between packages is generated by the depndendencies entre the classes :
  - Dependency between classes is generated by :
    - Generalisation
    - Association with navigability
    - Attributes types with classes
    - Operations with parameters

Dependencies etween packages

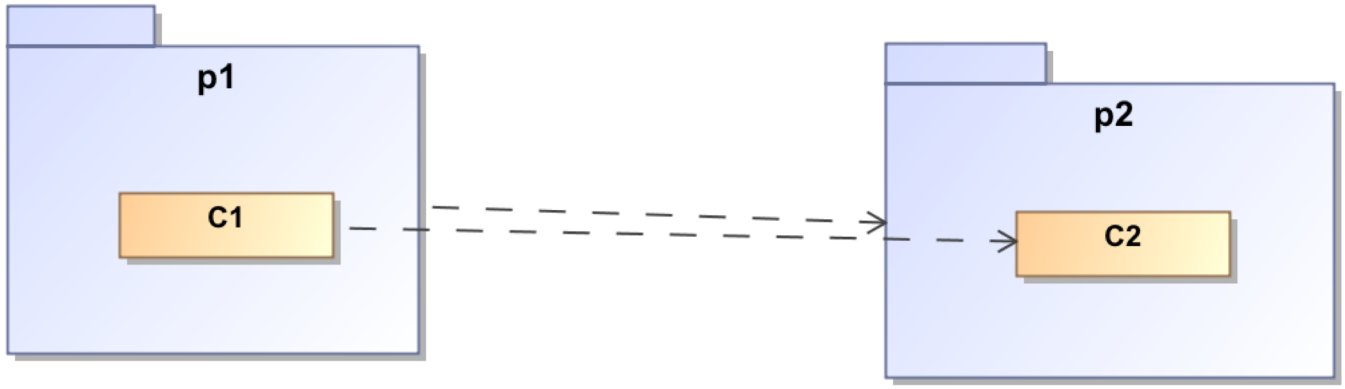
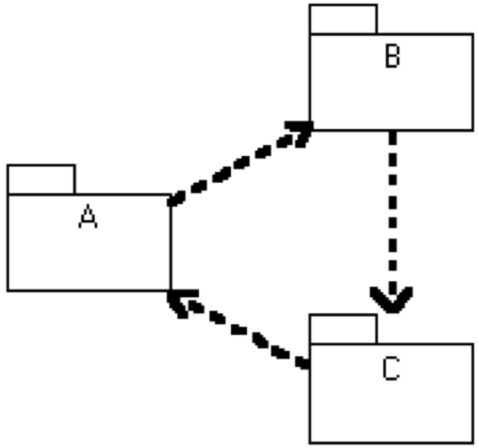


Diagramme de packages

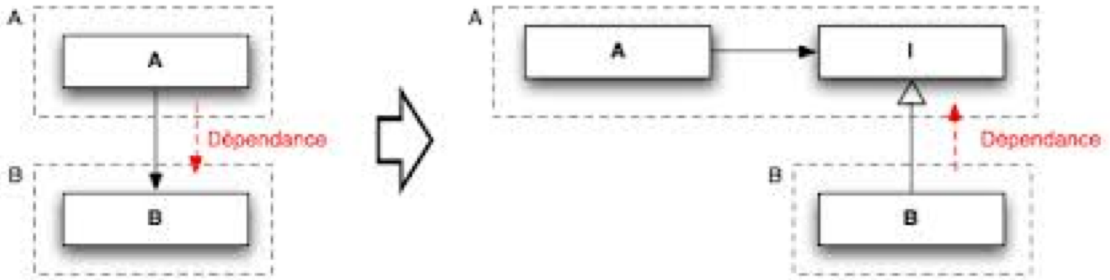
- Dependency cycle: A design problem!!



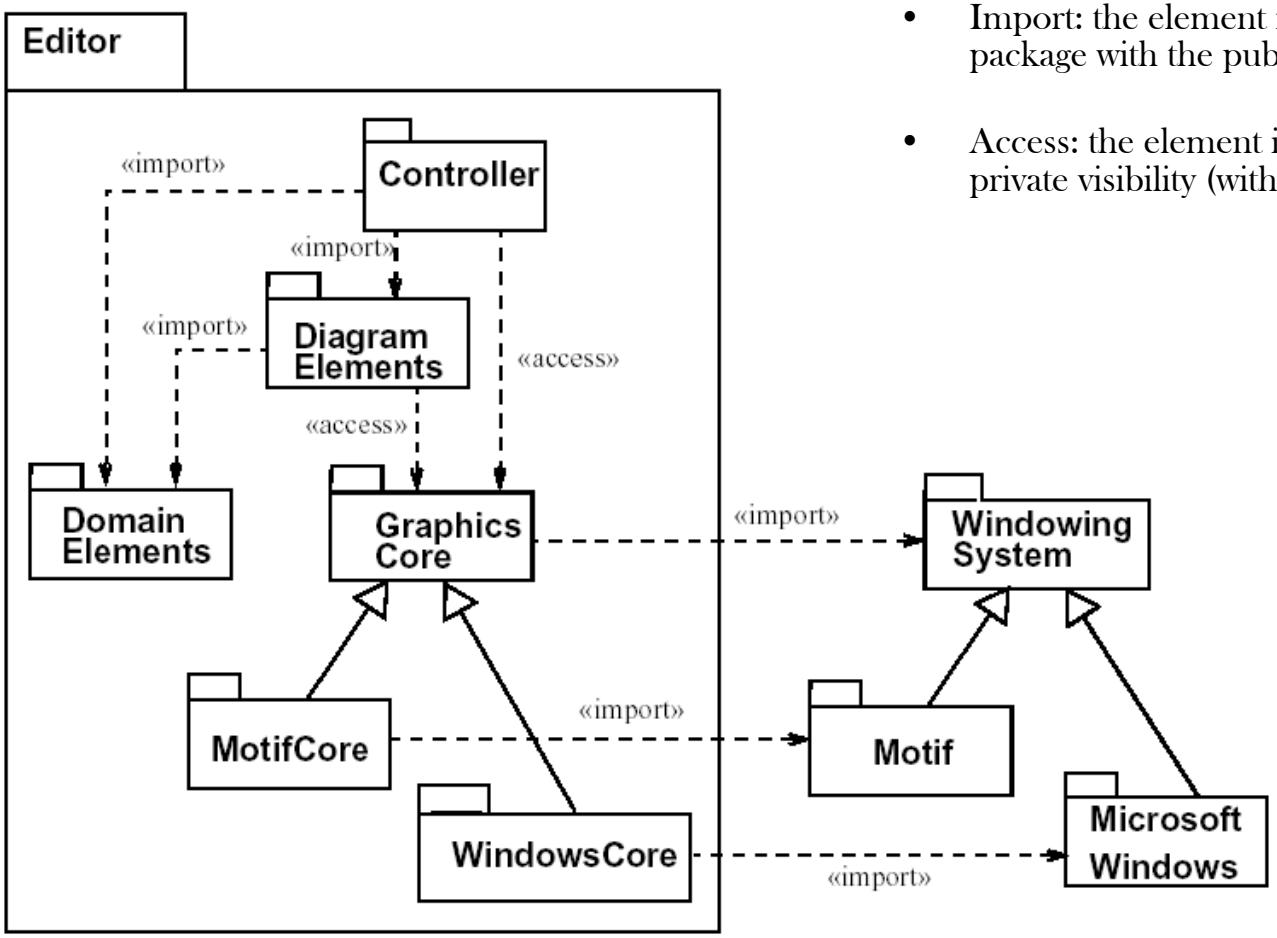
- How we can break dependency cycle?

Diagramme de packages

- The use of inheritance to break cycles



Packages: Example

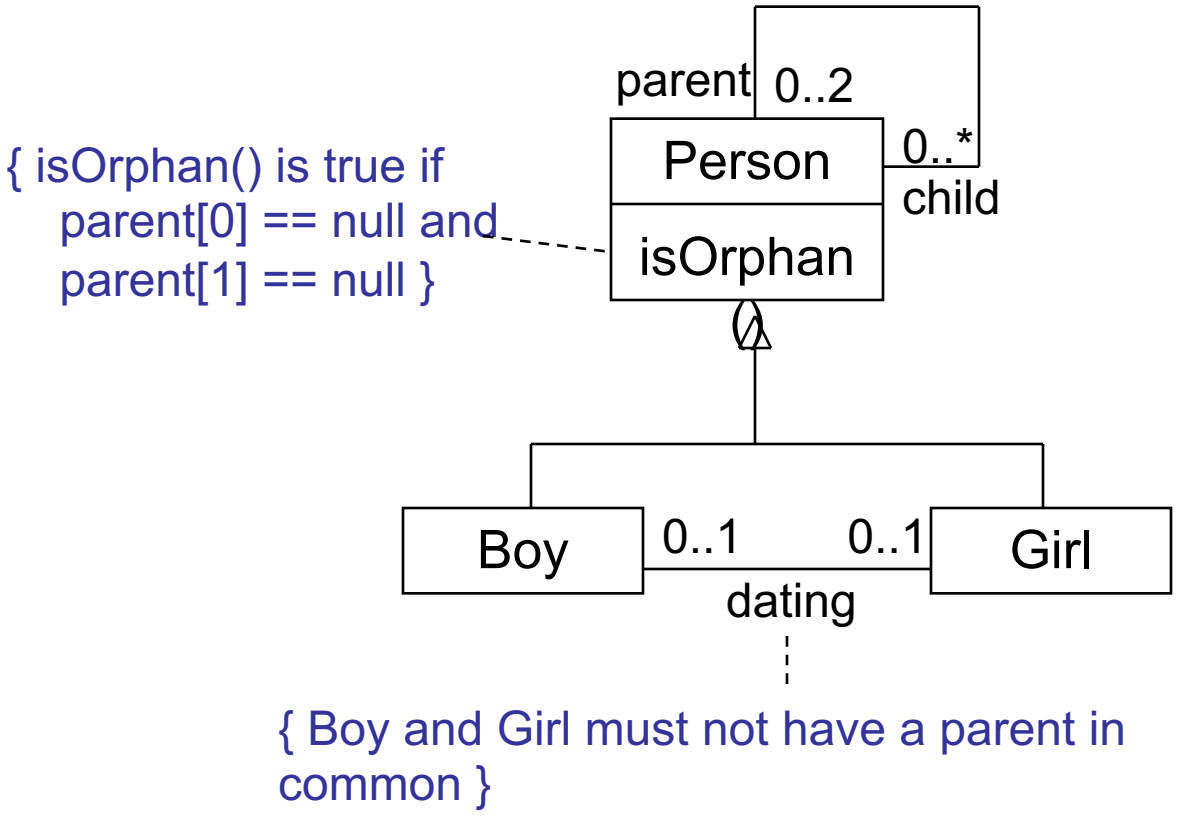


- Import: the element is imported in the package with the public visibility.
- Access: the element is imported with the private visibility (without transitivity)

Constraint rules

- A class diagram is all about constraints.
- But the basic structures of association, attribute and generalisation cannot specify every constraint, particularly functional integrity constraints.
- You can add additional constraints to a class diagram by placing expressions between accolades
  - { your constraint here }
- A constraint may be expressed in any language : Informal english / french, or more formal, like the OCL (Object Constraint Language).

Constraint example



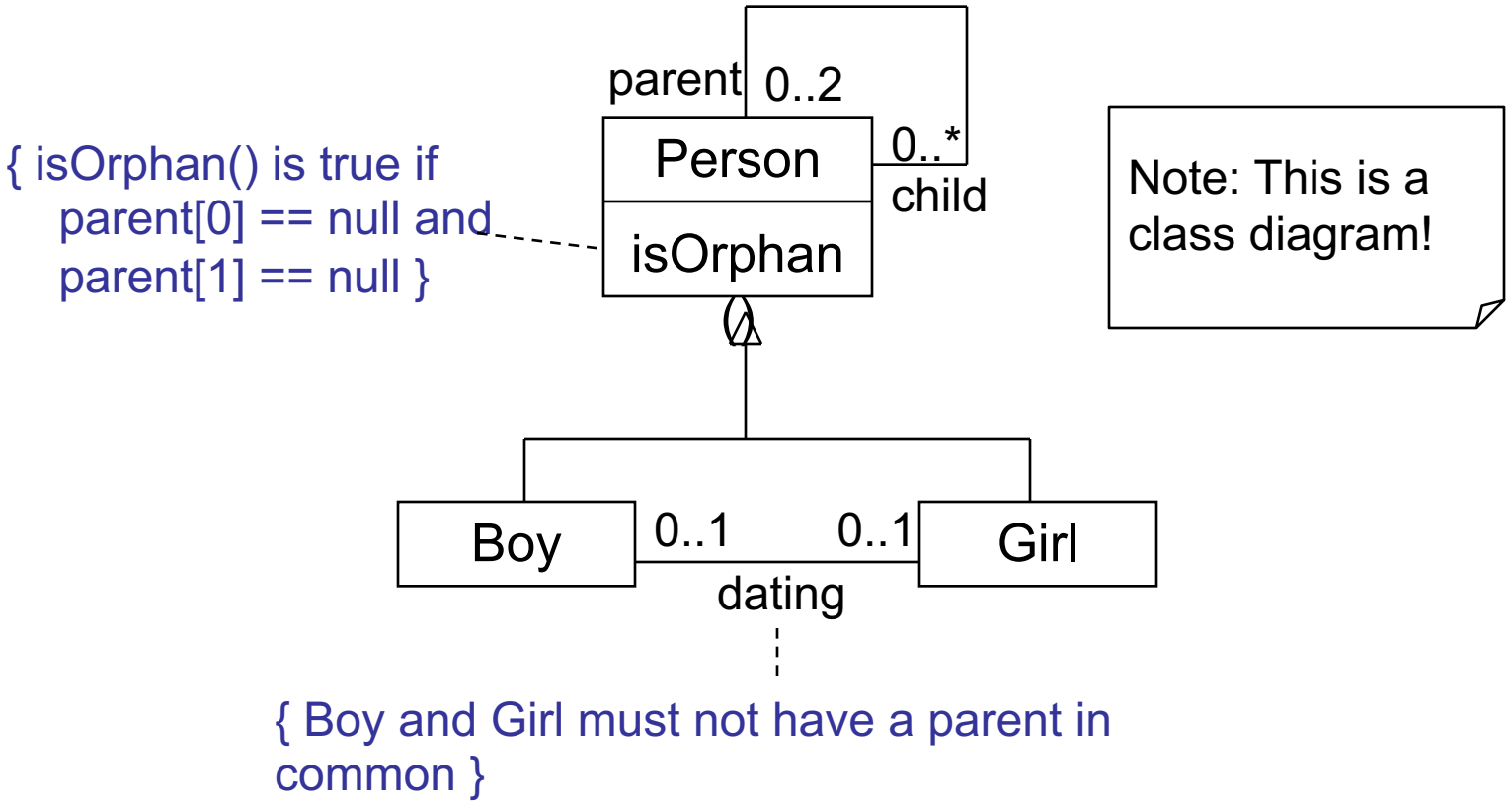
Object diagram

- An object diagram shows a snapshot of the objects and connections between them at a point in time.

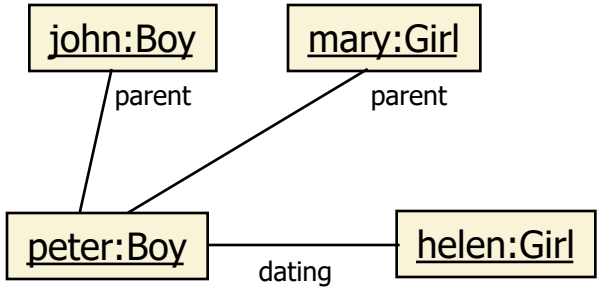
- Used to test and demonstrate complex data structures
  - Find a general class structure from specific object structures
  - Test that the class structure generates all legal structures, but not illegal ones.

- Objects are shown as rectangles in which the name takes the form **instanceName:className**, and is underlined.
  - Example : 

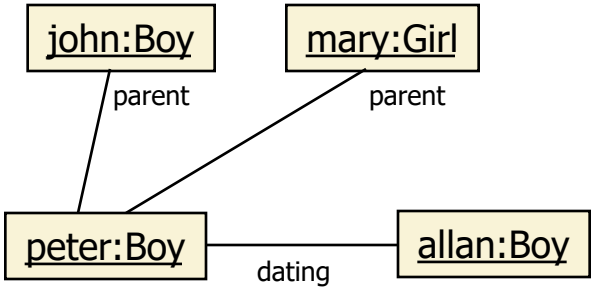
Object diagram example (1)



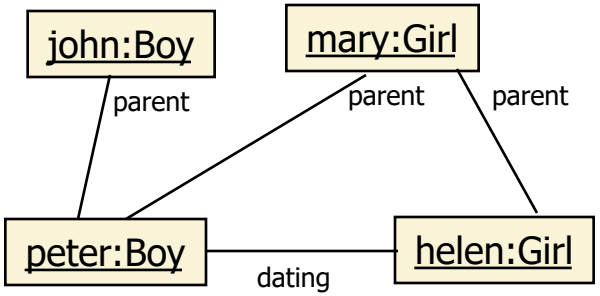
Object diagram example



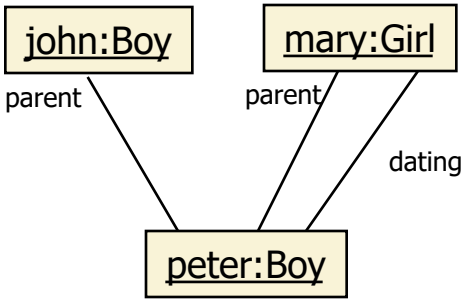
Structure allowed: Peter is dating an orphan named Helen.



Structure prohibited by Boy-Girl association "dating".



Structure prohibited by constraint : A Dating pair cannot share a parent



Structure allowed by current model, but probably unwanted; requires new model!

Design Patterns

”Patterns expose knowledge about software construction that has been gained by many experts over many years. All work on patterns should therefore focus on making this precious resource widely available. Every software developer should be able to use patterns effectively when building software systems. When this is achieved, we will be able to celebrate the human intelligence that patterns reflect, both in each individual pattern and in all patterns in their entirety. ”

From ”Pattern oriented software architecture” by Buschmann et al.

Pattern: Definition

« Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice »

C. Alexander, "*The Timeless Way of Building*", 1979

- A generic solution for a recurrent problem in a given context

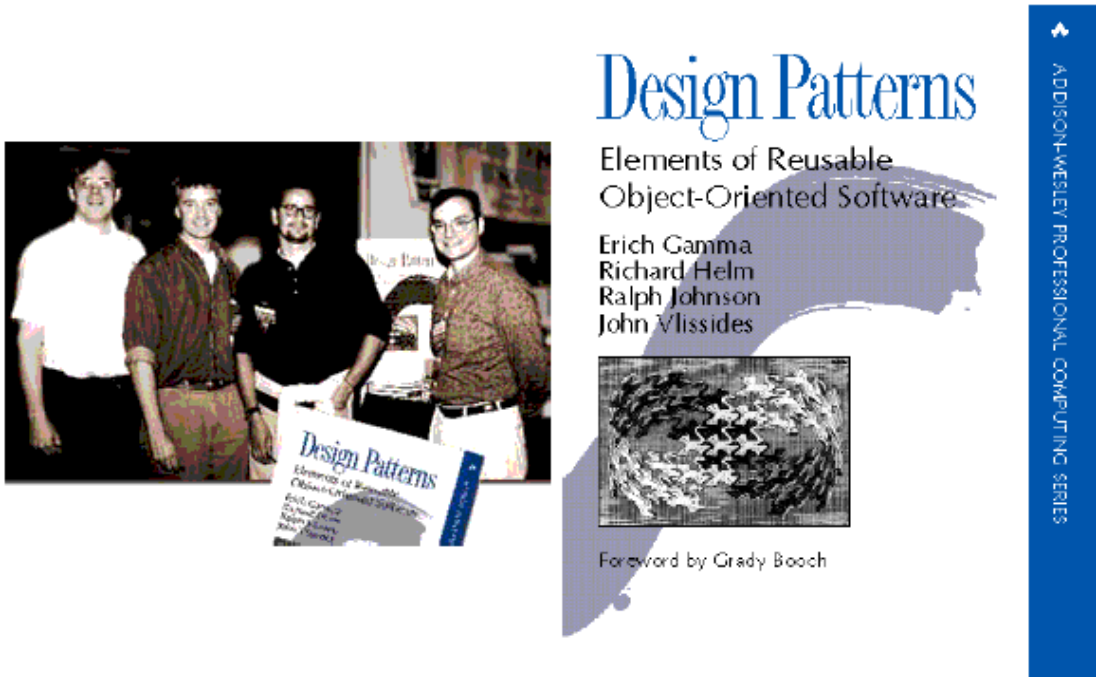
Why Design Patterns?

- Make available and explicit good practices of design

- Naming and making explicit an abstract structure (independently from the code).
- Create a common vocabulary for software developers.

Design Patterns in computer sciences: GoF

- 23 Patterns



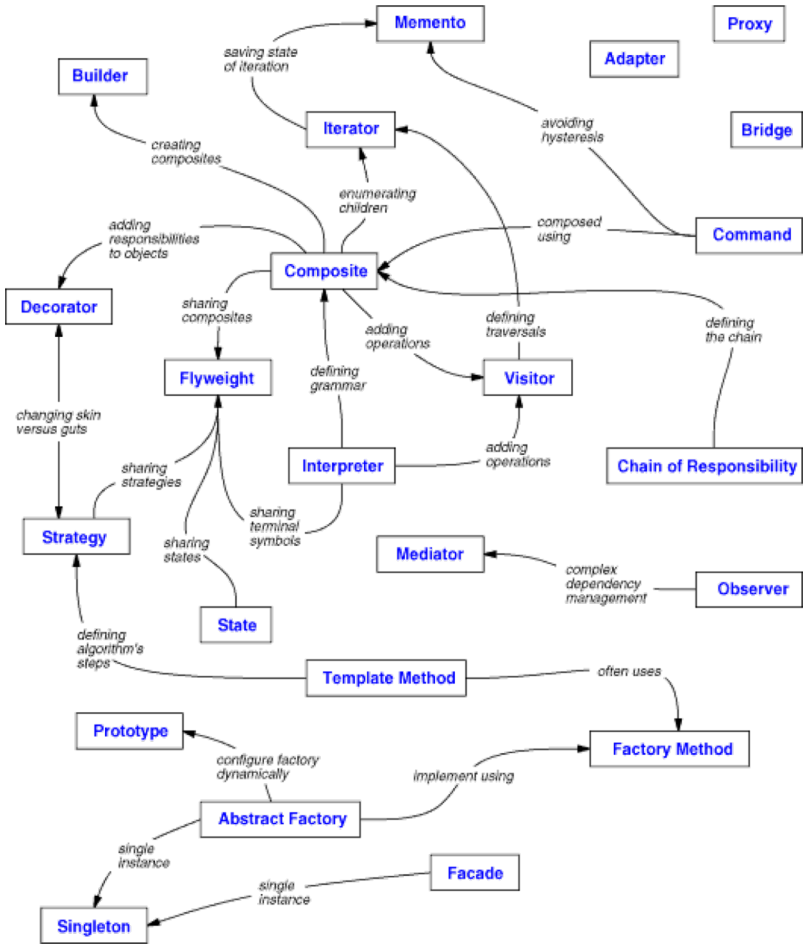
Documentation of DP in GoF

- Name and classification
- Intention
- Other names
- Motivation (scenario)
- Applicability
- Structure (UML)
- Participants (classes...)
- Collaborations
- Consequences
- Implementation
- Example of code
- Related Patterns

GoF Design Pattern: Classification

PURPOSE			
SCOPE	**CREATIONAL**	**STRUCTURAL**	**BEHAVIOURAL**
CLASS	Factory Method	Adapter (class)	Interpreter
			Template Method
OBJECT	Abstract Factory	Adapter (object)	Command.
	Builder	Bridge	Iterator
	Prototype	Composite	Mediator
	Singleton	Decorator	Memento
		Facade	Observer
		Flyweight	State
		Proxy	Strategy
			Visitor
			Chain Of Resp.

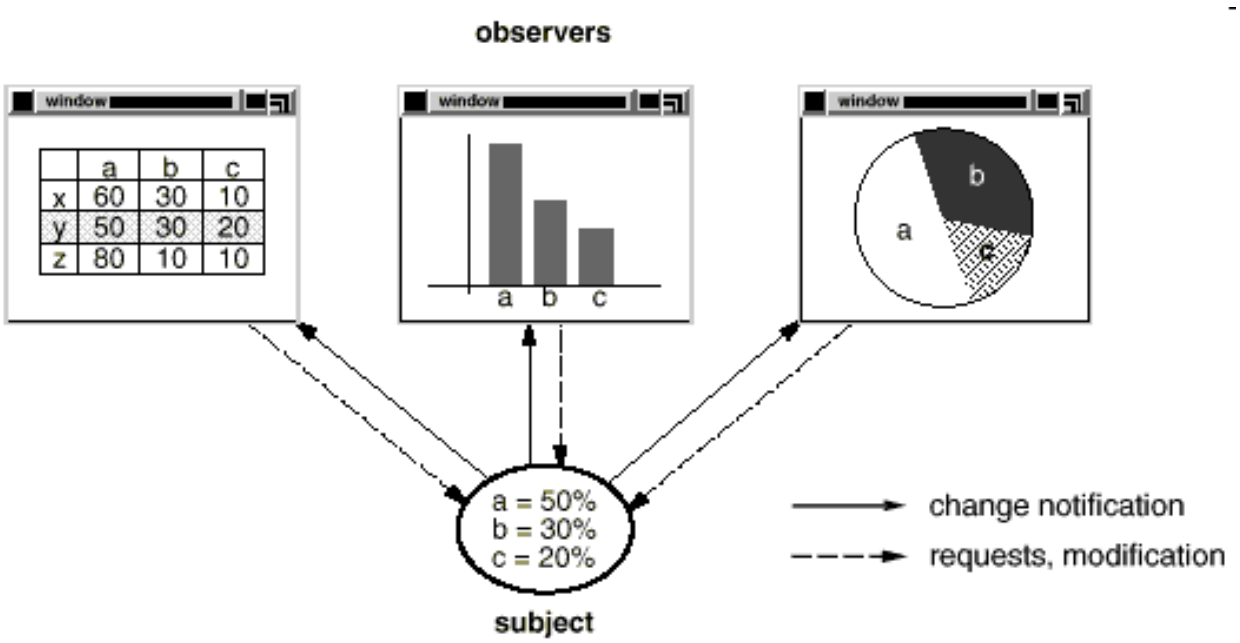
Design patterns are related



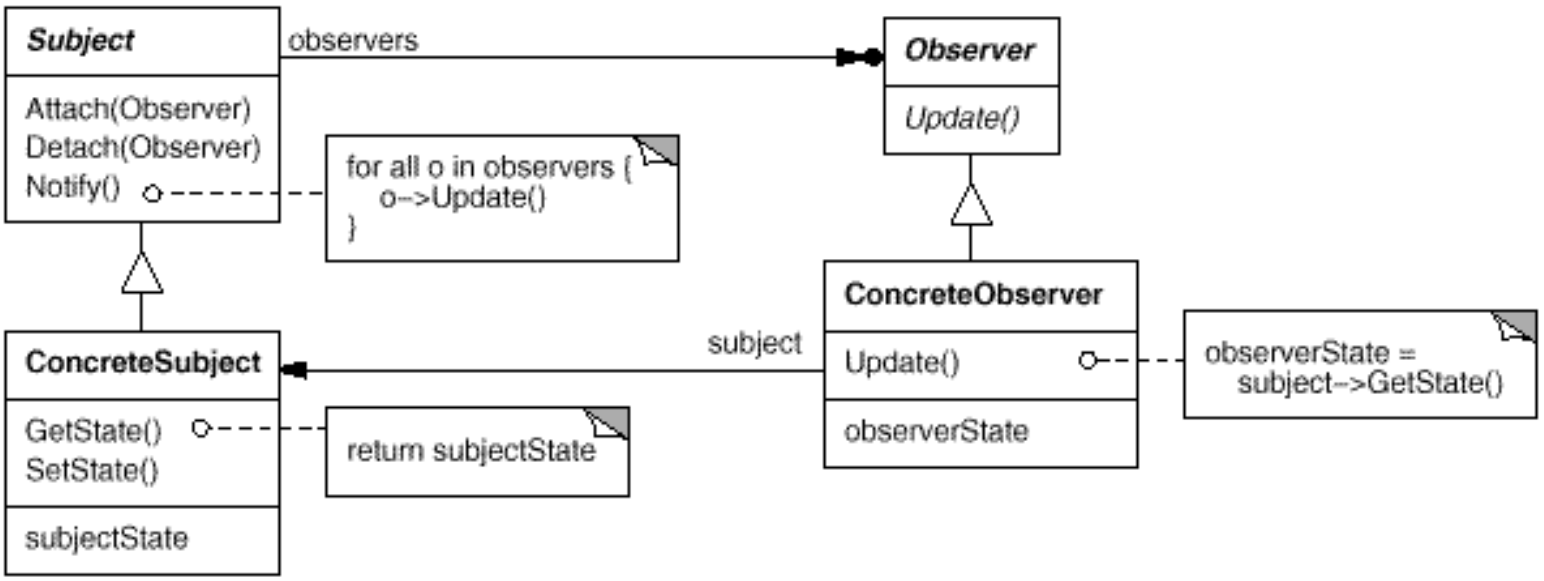
Design pattern: Example

- Observer

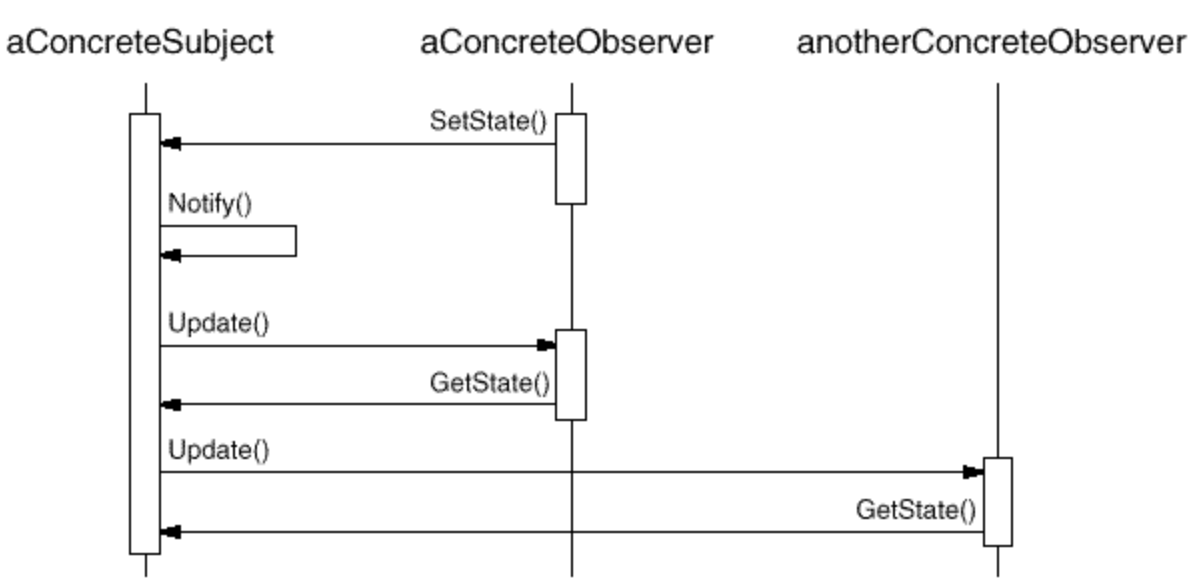
Pattern Observer: Autre exemple d'emploi



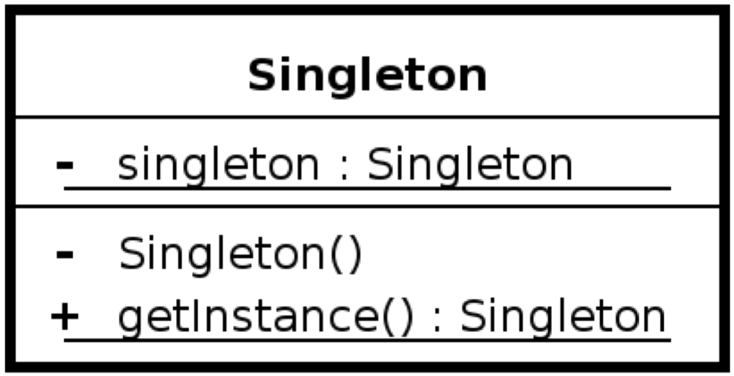
Pattern Observer: Diagramme statique générique



Pattern Observer: Diagramme de séquence



Singleton: Solution (Structure)



Singleton: Solution (Code)

```
public class Singleton
{
private static Singleton theInstance = null;

// Le constructeur en privé pour interdire l'instanciation de classe de
    //l'extérieur
private Singleton() {}
// On passera par cette méthode pour instancier la classe

public static Singleton getInstance()
{
    if (theInstance == null)
            theInstance = new Singleton();
    return theInstance;
}
}
```

documentation

- 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/

- 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