

# Context

- Large programs for real cases: application in economy, large computing, simulation, automatisisation . . .
- Maintenance is **time and money consuming**.
- Creating new programs is not the most current task now.

# Context

- Large programs for real cases: application in economy, large computing, simulation, automatisisation . . .
- Maintenance is **time and money consuming**.
- Creating new programs is not the most current task now.
- Several ways to answer : conception level, programming level, data level ...

# What is the software design

- Context:
  - Object oriented paradigm.
  - Data analysis and structuring is the starting point and the foundation.
- Addition of task organisation and methods to formalize that.

# Data Analysis

- Real applications often imply complex data: management of bank account, hospital, employees of big companies or image/video analysis,
- Object Oriented paradigm is based on the analysis of data firstly.
- Does it exist some methods to help us to analyze these data?

# Data Analysis

- Real applications often imply complex data: management of bank account, hospital, employees of big companies or image/video analysis,
- Object Oriented paradigm is based on the analysis of data firstly.
- Does it exist some methods to help us to analyze these data?
- It exists **methodology** and many **recipes**.

# Bibliography

## For beginners :

- *Object-Oriented Analysis and Design with Applications*, G Booch (Addison-Wesley, 1992).
- *Object-Oriented Software Engineering*, I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard (Addison-Wesley, 1993).
- *Object-Oriented Methods* , 3rd dition, I Graham (thompson Publisher, 1997).
- Website [omg.org](http://omg.org) - specifications, catalog of spec ...

## Find the good classes

- De-composition / Composition: an object is an individual element, real or abstract with a well defined role in the problem domain.
- Find the right objects and the size of each object.
- Decide where to attach behaviors.
- Create modules and links between them.

# Basic Principles

- **Objects** : base units organized in classes and sharing common traits (attributes or procedures). They can be entities from real world, concepts of the application or of the concerned domain.
- **Encapsulation** :
  - Data structures and implementation details are hidden to the other system objects.
  - The only way to access to the object state is to send to it a message which activate the execution of one of its methods.
- **Abstraction** : is the way to apply encapsulation and encapsulation make abstraction useful.



# Basic Principles

- **Polymorphism** : possibility to use the same expression for different operations. Inheritance is a specific form of polymorphism typical to the object oriented systems.
- **Modularity** : program partition which creates well-defined (and documented) frontiers into the program in the goal to reduce complexity(Meyers). The choice of a good set of modules for a given problem is as difficult as the choice of a good set of abstractions.

# UML - a tool to design applications

- Just a graphical language, not a method.
- 1990 - Object Management Group : standardisation.
- Unification of the methods OMT (Booch) OOSE (Jacobson) et Rumbaugh : Unified Modeling Language (version 1.0 1997, version actuelle 1.3).

# Special UML

- *UML Distilled: A brief guide to the standard object modeling language* Martin Fowler (3rd edition).
- *Instant UML*,  
Pierre-Alian Muller.

## Internet sites

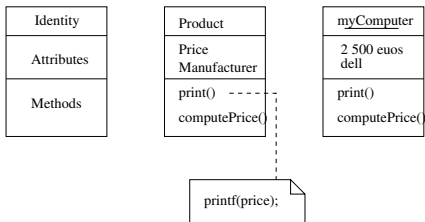
- Index to Object-Oriented Information Sources  
<http://www.oonumerics.org/oon> and [www.omg.org](http://www.omg.org)
- UML Resources  
<http://www.rational.com/uml/adobe.html>  
<http://umlcenter.visual-paradigm.com/LinksBooks.html>
- Aspect-Oriented Programming Home Page  
<http://aosd.net>  
<http://http://aspectj.org>

# Objects and classes

- Reminders :

Object = State + Behavior + Identity

- UML graphical conventions

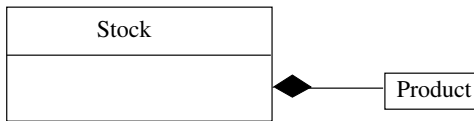


- Abstract classes are represented with the stereotype `<abstract>`

# Object composition

Object attributes can be themselves objects.

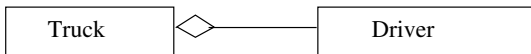
- **Composition by value** : responsibility to create and to delete the object.



- Creating an object implies to create its attributes by value too.

# Object composition

- **Composition by reference** : it is a reference link to an object, this object can be shared by several another object,



- Creating the container does not imply to create the referenced object.
- **NB** : the `diamond` is always at the using object side.

## Methods and attributes visibility

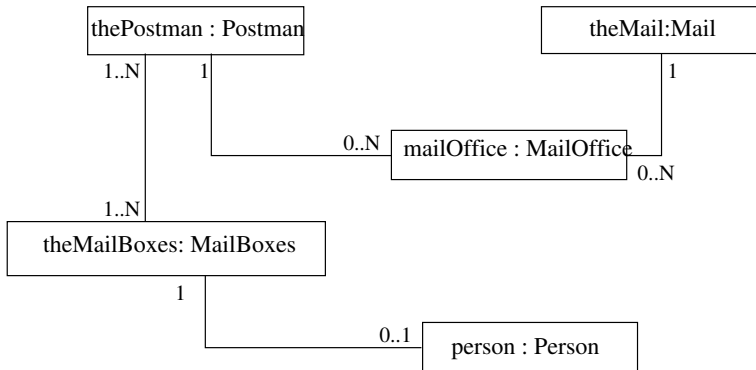
- **Public**: a public attribute or method is specified with the + sign.
- **Private**: a private attribute or method is specified with the – sign.
- **Protected**: a protected attribute or method is specified with the # sign.

## Association/Composition Arity

1	1 only one
0..1	0 or 1 association
M..N	from M to N (M and N integer)
*	from 0 to several
0..*	from 0 to several
1..*	from 1 to several

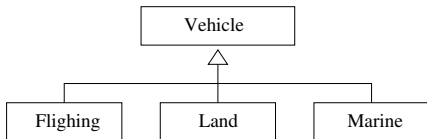


## Example of arities placed in diagram



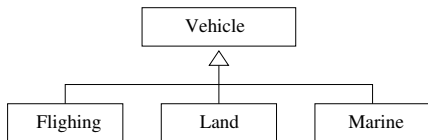
# Inheritance

- Simple inheritance

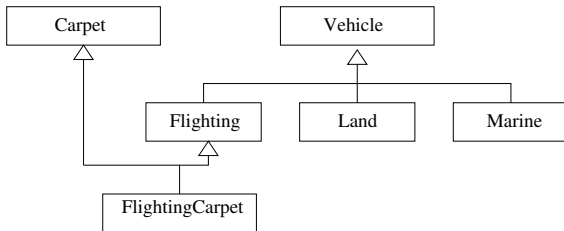


# Inheritance

- Simple inheritance



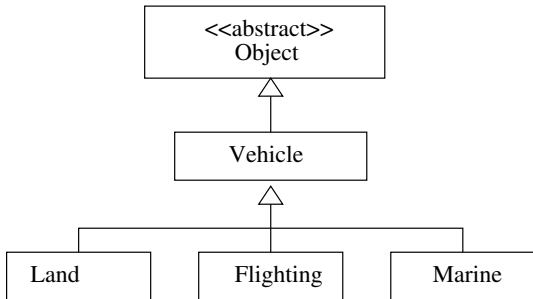
- Multiple inheritance



# Meta-Classes

- Definition :
  - A meta-class defines class characteristics, it is a generic model of classes (attribute or behavior).
- Example :
  - abstract class, interface, by extension each class of class.
- We note that a meta-class is also an object of which the class is the reference base class from which all objects are built (`Object` class in Java).
- At the analysis and conception step, it does not exist difference between a class and a meta-class.

# Meta-Class



# Now You have to work !

- The problem:
  - Modeling a library application to manage the subscription lends.
  - There are two kinds of subscribers: adult and child.
- Rules to take books or video or CDs:
  - Adult: 7 things - with a maximum of 5 books, 3 videos and 3 music CDs.
  - Child: 5 things - with a maximum of 3 books for child and 2 videos.
- All the lends are for a period of 3 weeks maximum.
- **Goal:** give the class diagram for this application.

# Classes Diagram

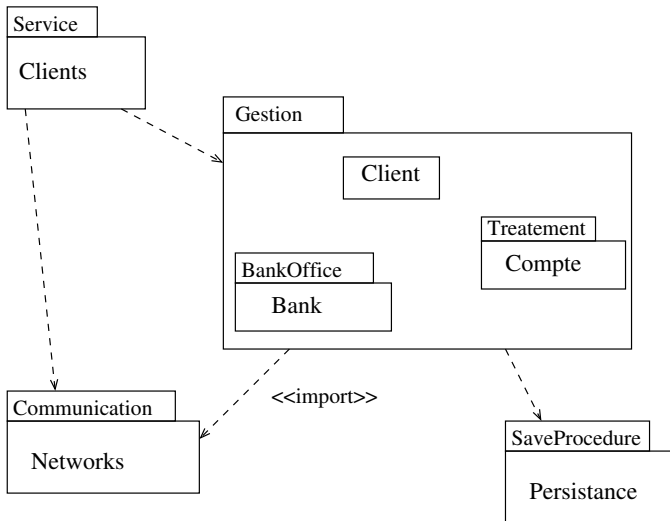
- Classes diagram is a **static** view of the model.
- It describes the internal structure of classes and their relations (inheritance, dependancy, composition . . .).
- The terms : *static structural diagram* and *class diagram* are equivalent in UML terminology.
- It is a collection of declarative elements of the model. These elements are classified by typing mechanism.
- **Note**: it is possible to build a diagram which contains only interfaces and abstract classes. In this case, it is called a **Meta-Model**

# Objects Diagram

- It is the instance graph for the all object class.
- It is itself an instance of the class diagram.
- It is only used to illustrate examples.

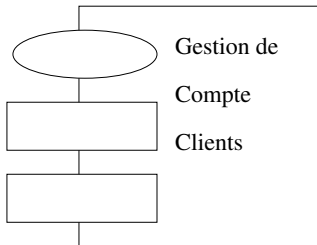


# Package representation



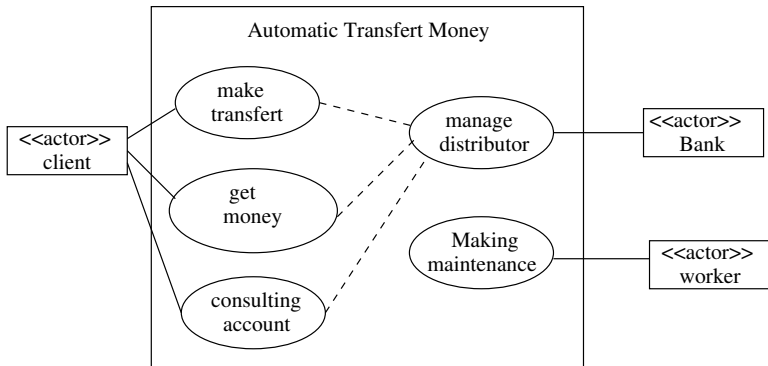
# Module graph

Modules are compiling sub-units. Some programming languages do not be able to implement this concept.



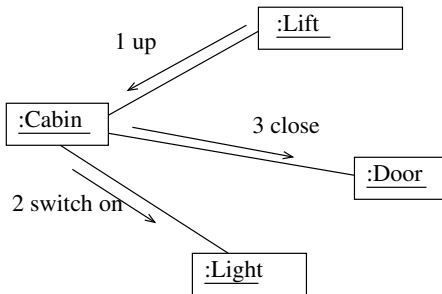
# Use cases diagram

- Communication, event or data flux diagram between external entities and the system.
- Give the external interface of the system.
- Only two kinds of represented objects: external actors and components which interact directly with the actors.



## Collaboration diagram

- This diagram shows the interactions between objects and the structural relations which allows these interactions.
- The numerotation gives the order of messages.
- Time is not represented.

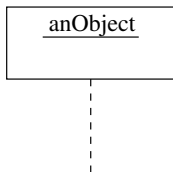


# Collaboration diagram

- Give the context for an objects set and the interactions between these objects (messages sendings).
- Messages are given on the links which associate objects, these links are oriented from the caller to the destination.
- Allows to represent actors or external element.
- The links in the collaboration diagram have not the same semantic than the composition links into the class diagram.

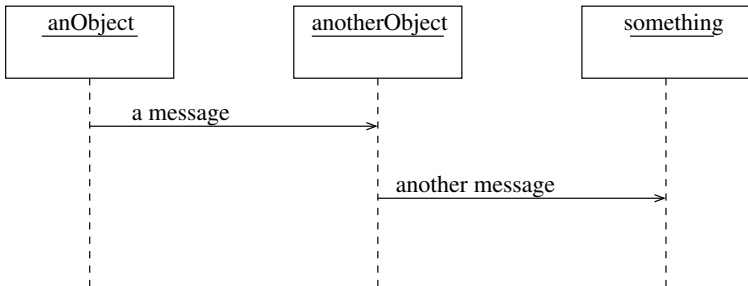
# Sequence Diagram

- Show interactions between object from the time point of view.
- Notation <sup>1</sup> :
  - An objet is materialized by a rectangle and a vertical line called *the life line*



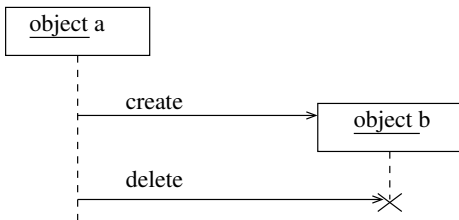
# Sequence diagram

- The rank of the message sending is done by the position on the vertical axe.



# Sequence Diagram

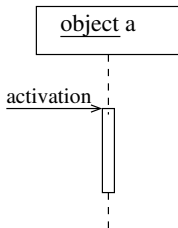
- Creation / destruction of object





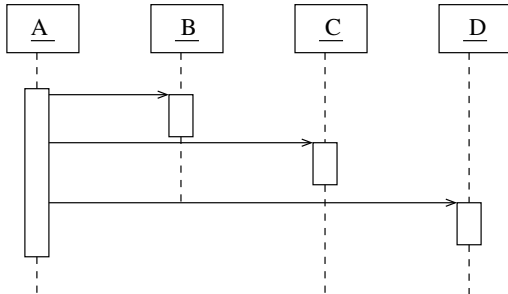
# Sequence diagram

- Representation of activity periods for objects = working time for this object.
- Beginning and the end of the vertical band correspond to the beginning and the end of the activity period.



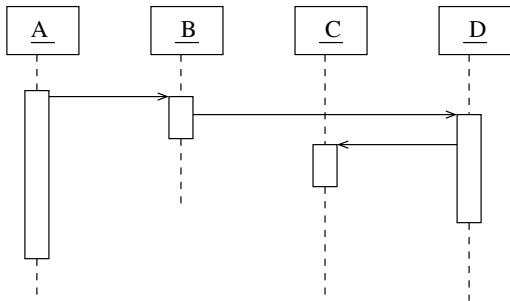
# Centralized - decentralized mode

- Sequence diagrams can show the control structure choice.

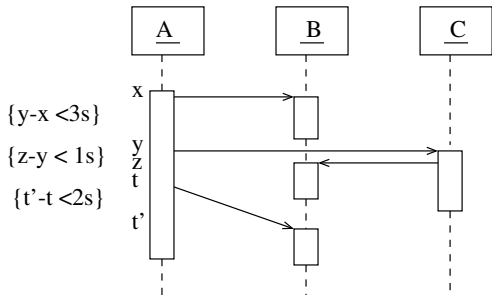


# Centralised - decentralised mode

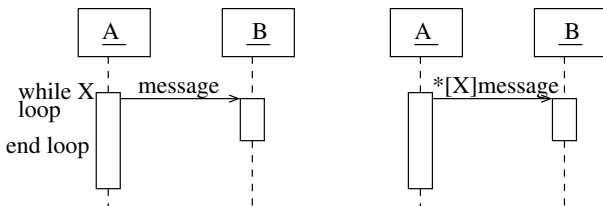
- Decentralised sending of messages.



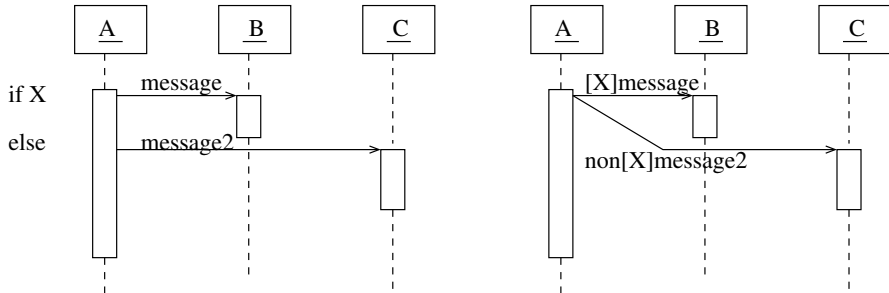
# Temporal constraints



# Loops and branchings

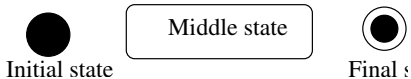


# Conditional branching



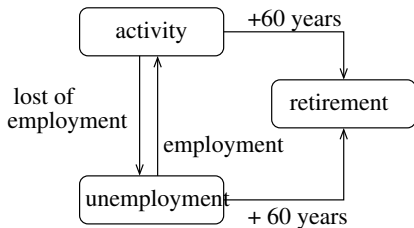
# State-transitions diagram

- An object is in a given state at each step.
- The state of an object is given by the values of its attributes



## State-transitions diagram

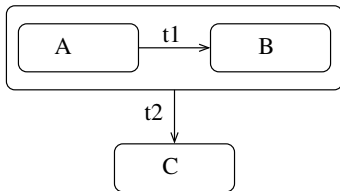
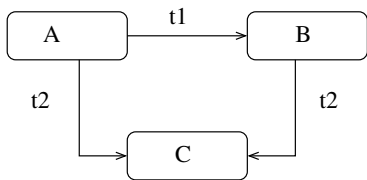
- The object change from a state to another one by using transition.
- Fired by event, transitions allows the change immediatly.





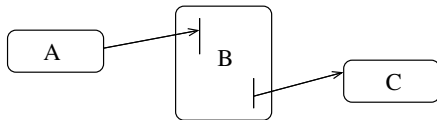
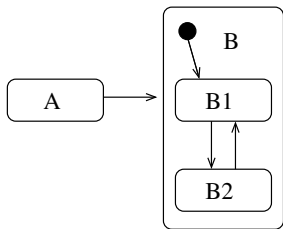
## Generalisation of states

- To simplify the diagram, states can be put together into subclasses of a generic class.
- A state can be decomposed into several separated (exclusive) sub-states, object can be only in only one state at the same time.



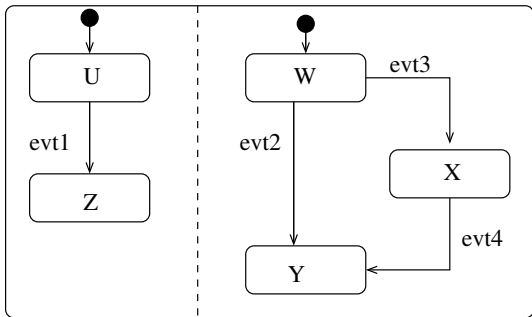
## Exemple of agregation of states

- How to decompose states without breaking the abstraction frontier.

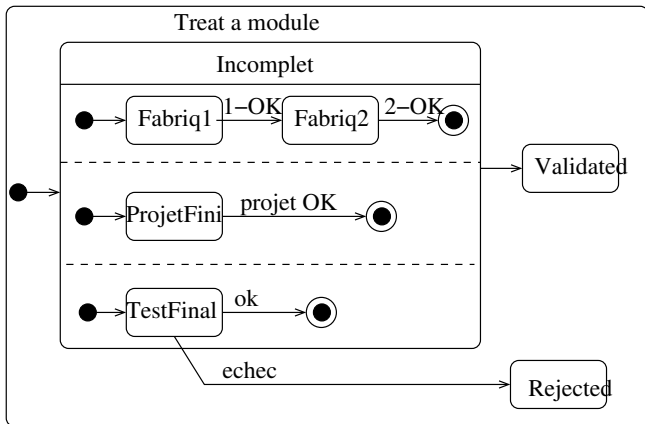


## Agregation of states

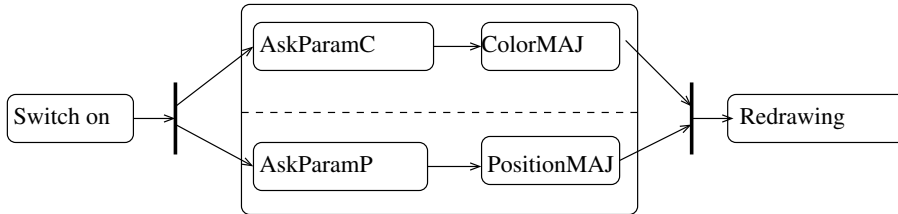
- Agregation of several automatas which work simultaneously and independently



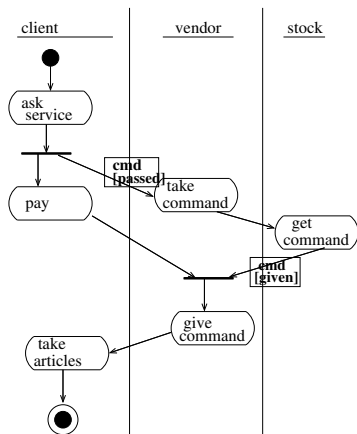
# Agregation of state



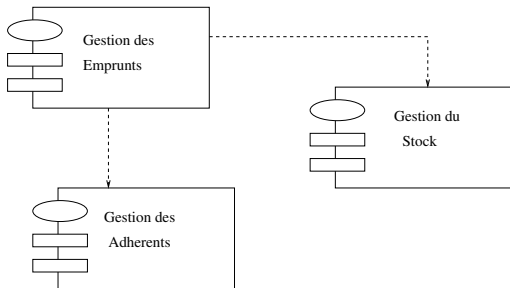
# Complex transitions



# Activity diagram



# Composants Diagram



# Deployment Diagram

