Requirement Analysis with UML

Pierre-Alain Muller

pa.muller@uha.fr

Pierre-Alain Muller

- 15+ years experience in OO modeling
- Assistant professor at ENSISA (a French school of engineering)
- Former CEO ObjeXion Software
- Independent Consultant
- Author of “Instant UML”, Wrox Press 1997 and “Modélisation objet avec UML”, Eyrolles 2000

Course objectives

- Quick introduction to UML
- Requirement Analysis
  - Activity modeling
  - Use Cases modeling
  - Object-Oriented modeling

What is UML ?

- Unified Modeling Language for object-oriented developments
- De facto standard
  - Defined and endorsed by the OMG (Object Management Group)
  - Universally adopted by all major software vendors
  - The *Esperanto* for the object-oriented methods

A single language

- UML is the language for the description of the object-oriented models
- UML can be used by
  - Business analysts
  - Software engineers
  - Quality Assurance and Validation
  - End-Users and Customers

UML is a powerful notation

- Syntax and semantics formally defined in a metamodel
- 9 types of diagrams (from business modeling to component modeling)
- Everything is not needed by everybody
  - Many use a subset of UML

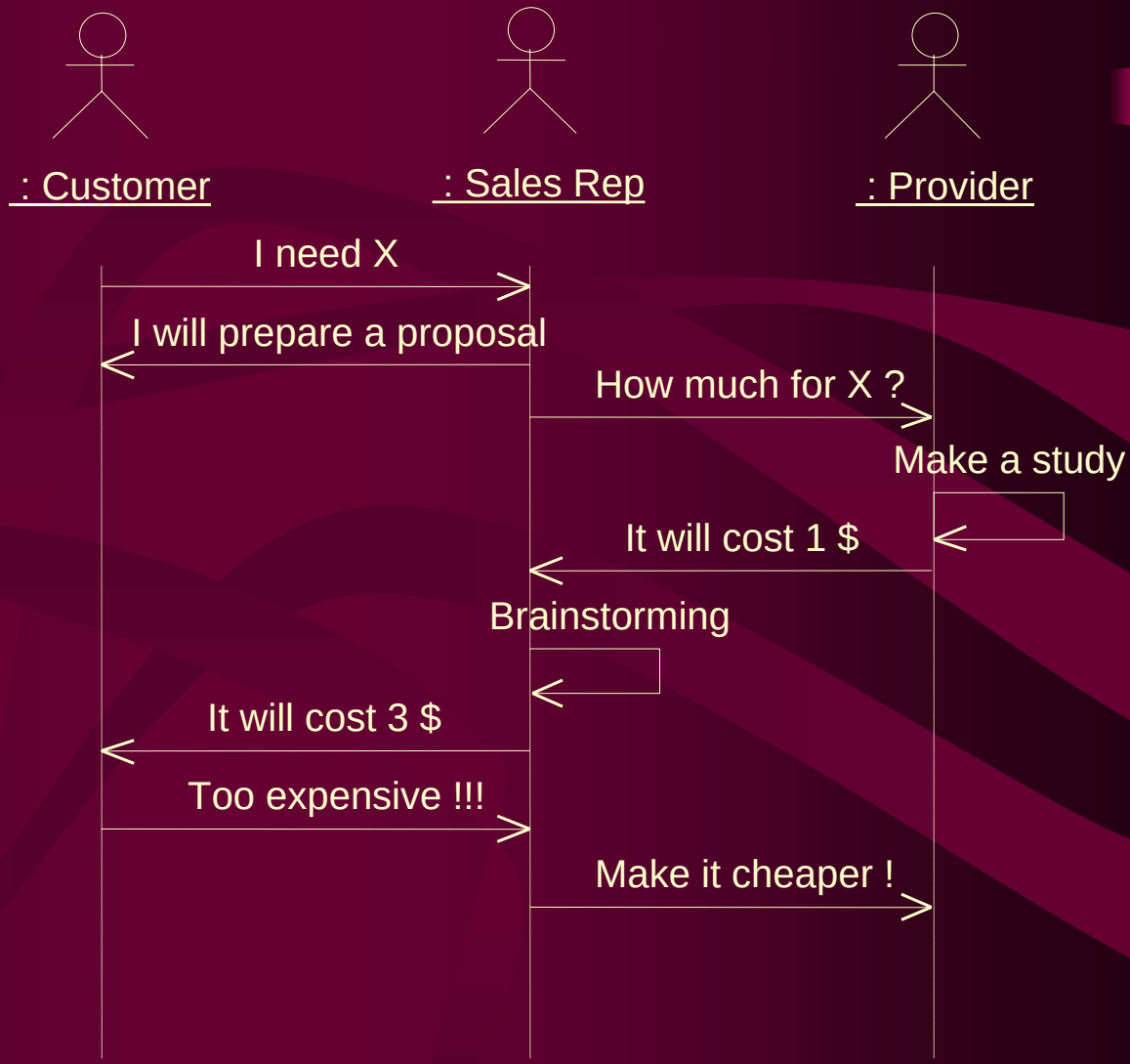
Activity Modeling

- Understand the business processes
- Find the actors in the business
- Find the activities performed by the actors

Scope

- The focus is on understanding *Who* is doing *What* and *Where* in the business
- Then the focus shifts on asking *What* should be provided by the system to help the actors to perform their tasks

Example


: Customer
: Sales Rep
: Provider
I need X
I will prepare a proposal
How much for X ?
Make a study
It will cost 1 $
Brainstorming
It will cost 3 $
Too expensive !!!
Make it cheaper !

Use Cases Modeling

- Communicate with users and customers
- Capture requirements
- Identify system boundaries
- Derive objects and objects interactions
- Design user interface
- Define test cases
- Outline the user documentation

What is usually wrong with requirement specification?

- The customer is focused on business details
- The SW development is focused on implementation details
- The customers introduces new ideas without realizing that they are out of the agreed scope

The root of the problem

- They are too many ways to interpret a requirement specification
  - All the details are correct, but they are out of context
  - Very few, if any, will indicate what the users really need

What is good with Use Cases ?

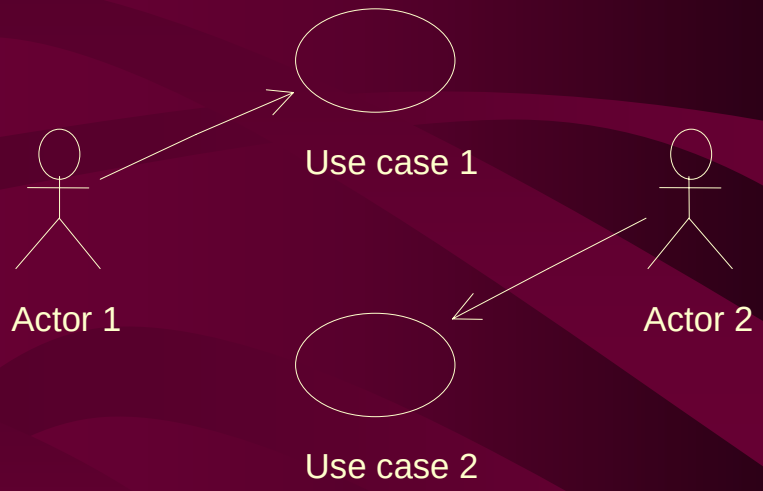
- The user is involved early in the process
- The user is aware when there is a change in the model
- The user sees the cost impact of a change in the model

Who writes the Use Cases ?

- A separate analysis team
- The designers
  - This is often the best way to get most from Use Cases

Use Case model concepts

- Actors
- Use Cases
- Scenarios
- System Boundaries
- Use Case description



What is an Actor ?

- An Actor
  - represents anything that interacts with the system (human, machine or another system)
  - represents roles a user can play
  - can give and receive information
  - is not part of the system

Identifying Actors

- Who will supply, use or remove information ?
- Who will use this functionality ?
- Who will support and maintain the system ?
- What are the system external resources ?

Description of Actors

- An Actor is described by:
  - Name
  - Brief description
    - Who or what the actor represents
    - Why the actor is needed
    - What interests the actor has in the system

What is a Use Case ?

- A Use Case is a complete and meaningful flow of events
- A Use Case is initiated by an actor to invoke a certain functionality in the system
- A Use Case models a dialogue between an actor and the system
- The collection of all Use Cases constitutes all possible ways of using the system

Identifying Use Cases

- What are the tasks of an actor ?
- How is the actor informed about certain occurrences in the system ?
- How is the system informed about certain occurrences in the business ?
- Does the system supply the business with the correct behavior ?

Use Case and Actors

- A Use Case can interact with many actors
- An actor normally interacts with several Use Cases of a system
- For each actor
  - Each way of using the system is captured in a use case
- For each use case
  - There is one actor initiating the use case

Use Case description

- A Use Case has:
  - Name
  - Brief description
  - Flow of events
    - One basic flow
    - Several alternative flows
  - Additional requirements
    - performance, reliability...

How to name a Use Case

- Name is taken from the actor point of view
- Name makes sense to the user (not to the system implementor)

Beware of functional decomposition

- Symptoms
  - A lot of small Use Cases
  - Difficult to understand the model
  - Names like
    - Operation + Object or Function + Data
- Actions
  - Raise the abstraction level (the larger context)
  - What value will the Use Case add ?

Interaction with the system

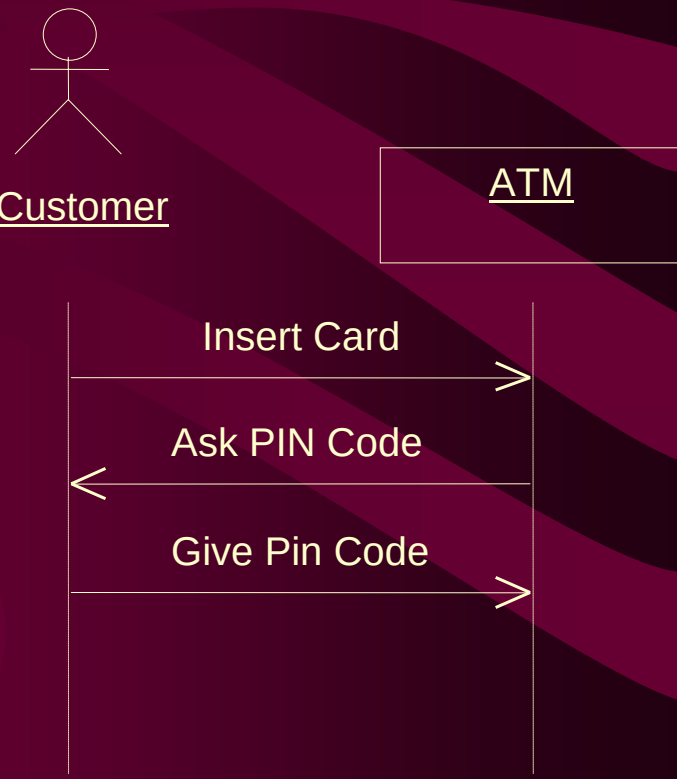
- A communication-association between an Actor and a Use Case indicates that they interact
- The arrow direction shows who started the interaction

Interaction description

- A Use Case is a set of flow of events
- These flows scenarios are named scenarios
- The scenarios can be described
  - Textually (in natural language)
  - Graphically (using UML sequence diagrams)
  - Usually both ways
- Between 5 and 15 pages of documentation

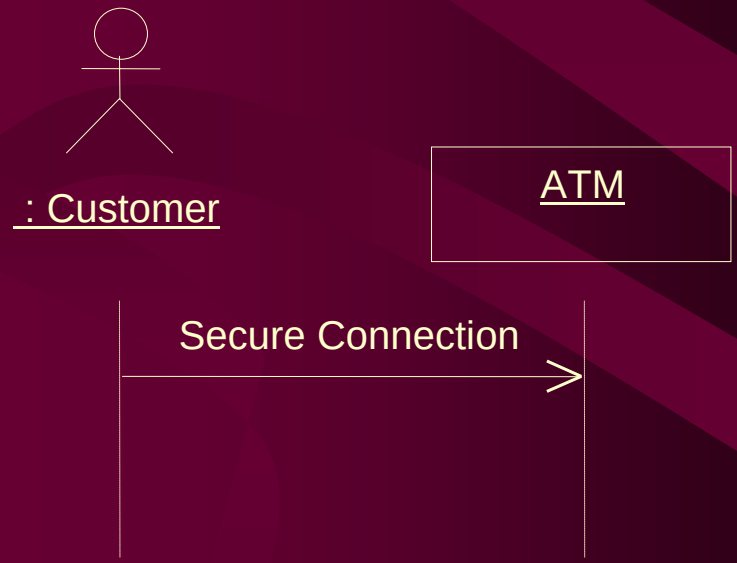
Example

- The user inserts his/her Credit Card in the ATM
- The ATM asks the PIN code
- The User provides the code



Essential Use Cases

- Focuses on semantics
- No GUI presentation details



Recurring Use Cases

- System start and stop
- System maintenance
- System evolution

Sources of information

- Requirements specification (if any)
- Customer, end users, and domain experts interviews
- Business model (if any)
- Internal standard practices and procedures

Summary

- Use Cases capture the system requirements
- Use Cases are user-focused
- Use Cases are readable by the end-user
- Use Cases are families of scenarios

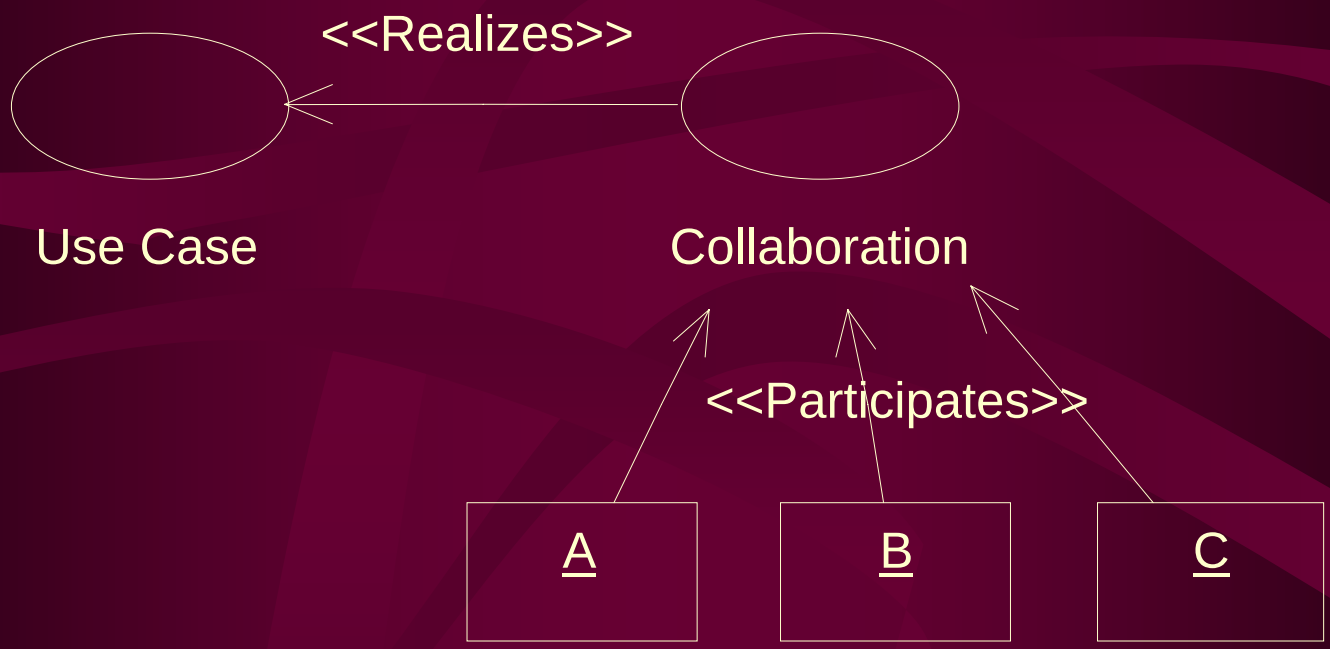
Object-Oriented Modeling

- Move from requirements analysis to object-oriented analysis
- Prepare the smooth transition to implementation

Object-Oriented Modeling

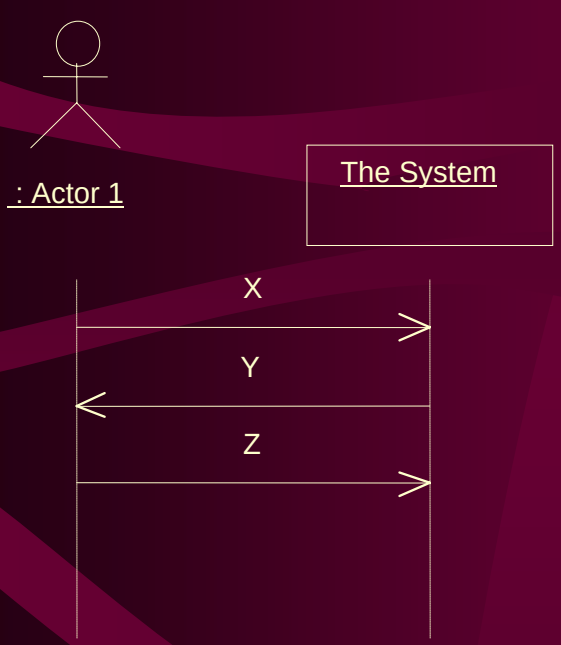
- Use cases are *realized* by societies of collaborating objects
- These objects are originally drawn from the problem domain

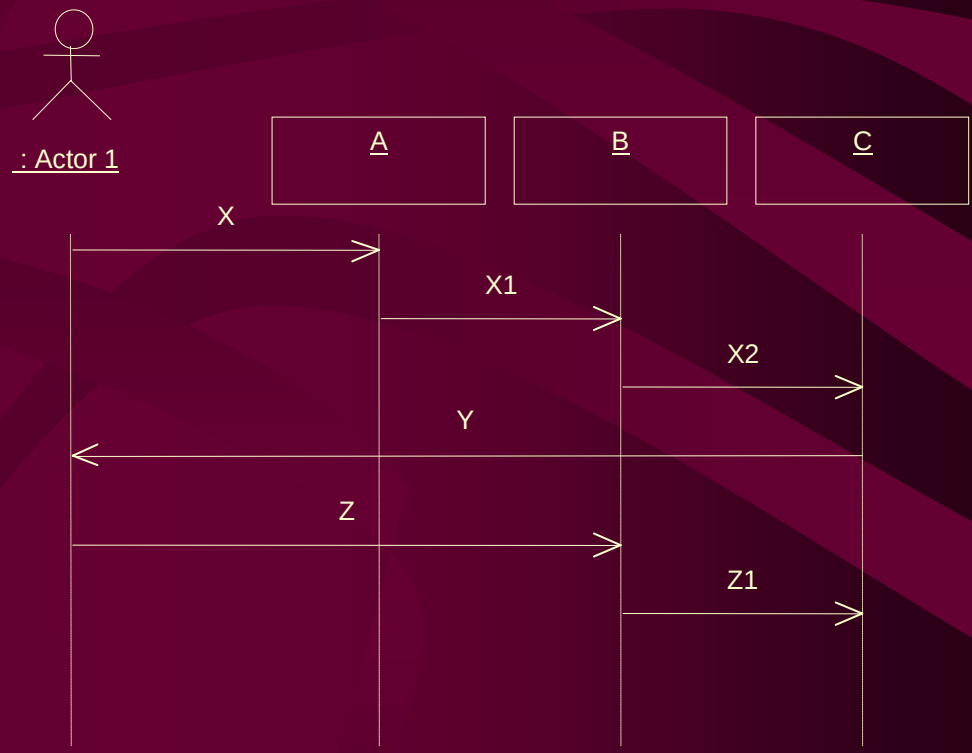
The move to the objects



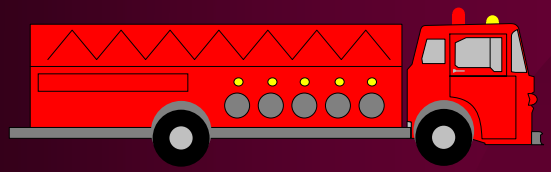
Sequence diagrams

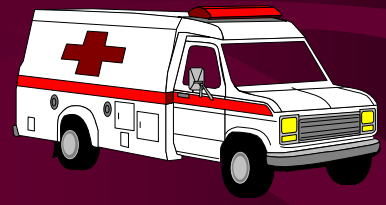
- Zoom into the system

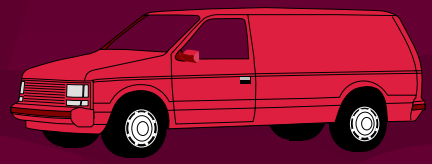


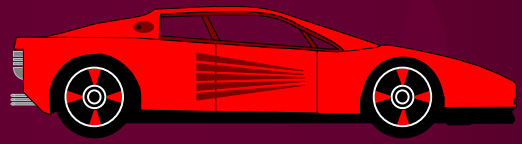


Object chaos









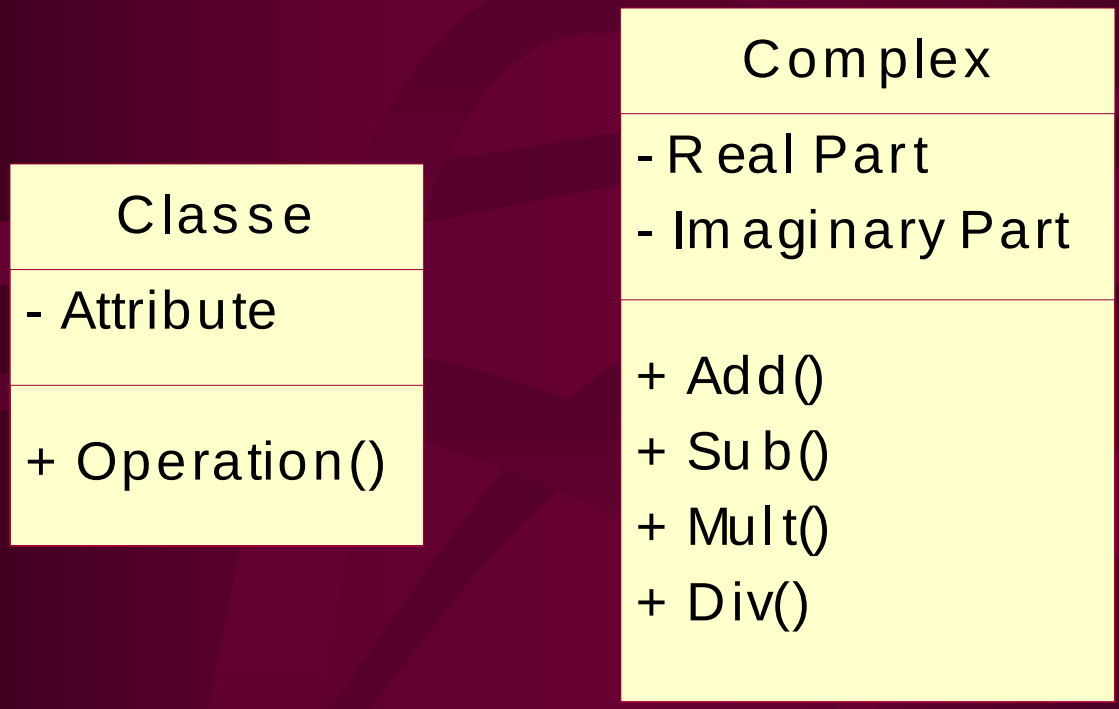
Object chaos

- There are many objects in the real world
- To understand the world, we group objects by similarities
- Making groups is known as classifying
- Humans have classified animals, flowers, mushrooms, atoms…
- Classification is the way humans deal with complexity

Classes

- A class is an abstract definition of a set of objects
- Common objects elements are factored out in the class

Graphical representation



Class Description

- Split in two parts
  - The specification (the *what* part)
  - The implementation (the *how* part)

Representation of the static structure

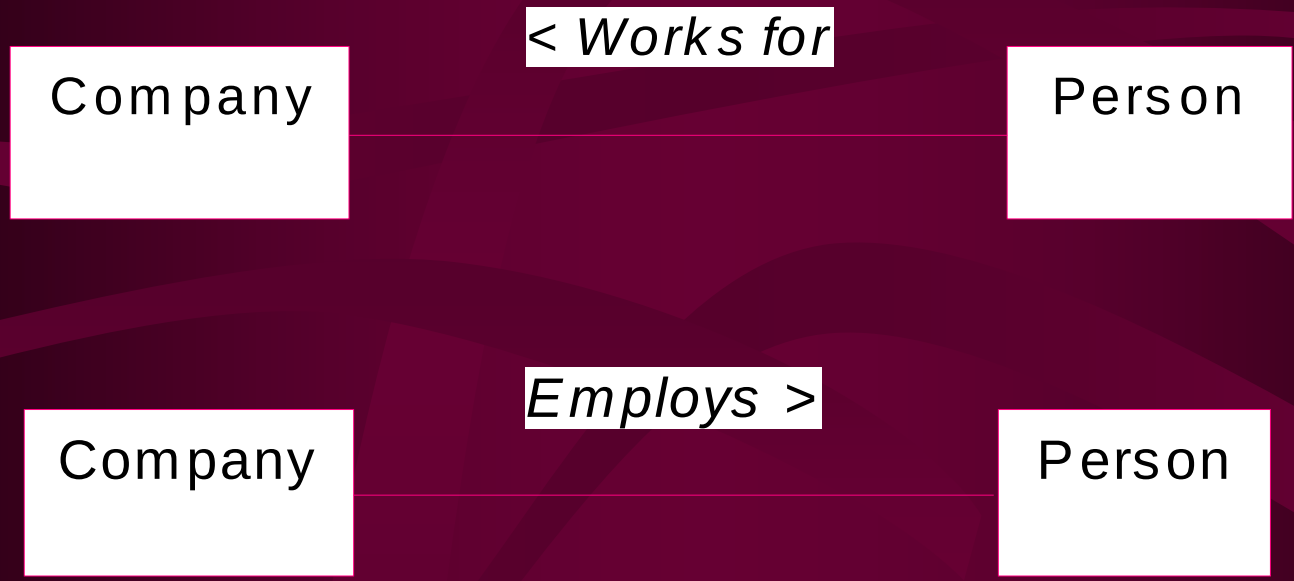
- Class diagrams
  - Classes
  - Relations between classes
    - Association
    - Aggregation
    - Generalization
    - Dependence

Association

- Bi-directional symmetric semantic connection between classes
- An abstraction of all the links between the instances of the associated classes
- Represented by a line drawn between the classes

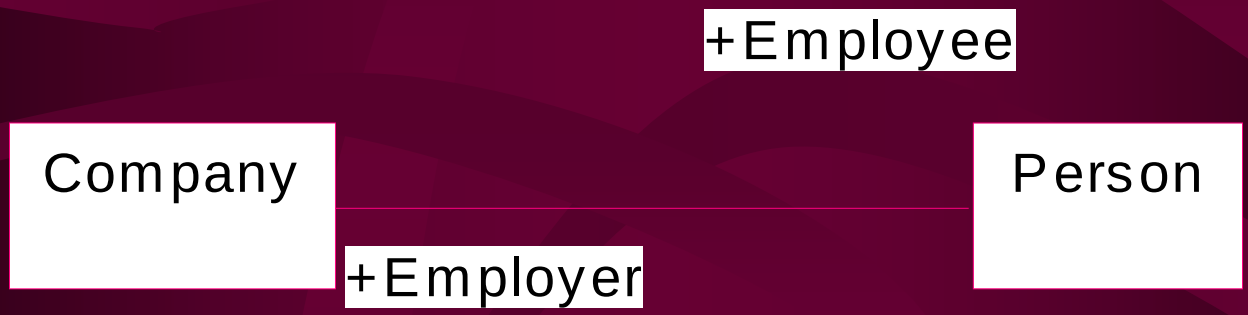
Exemple

Naming of associations

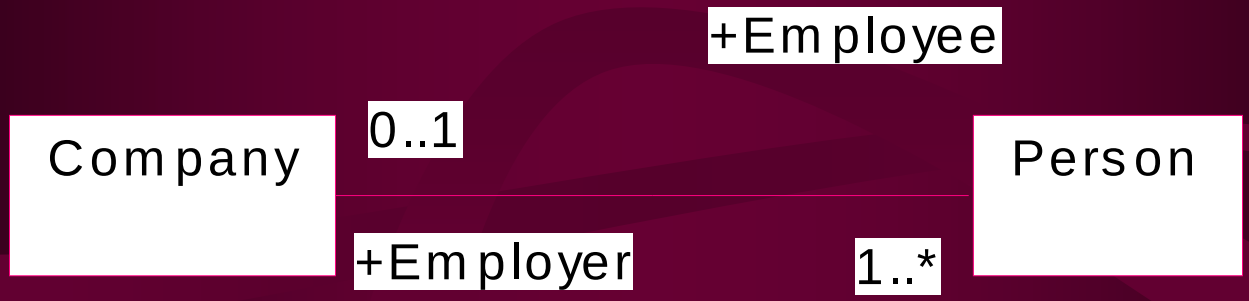


Naming of the roles

- A role describes one end of an association



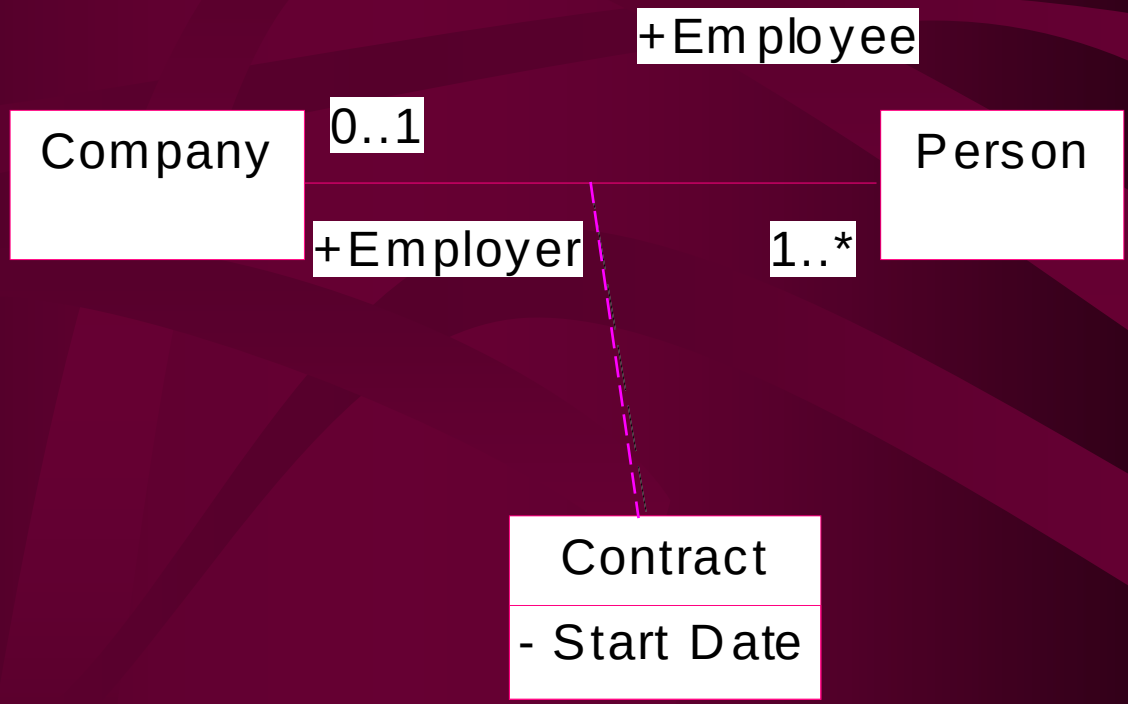
Role multiplicity



1	One and only one
0..1	Zero or one
M .. N	From M to N (naturals)
*	Many
0 .. *	From zero to many
1 .. *	From one to many

Class-associations

- Adding attributes and/or operations to the relation



Aggregation

- Bi-directional asymmetric semantic connection between classes
- Some kind of “stronger” association
- Represents
  - master and slave relations
  - whole and part relations
  - composite and component relations

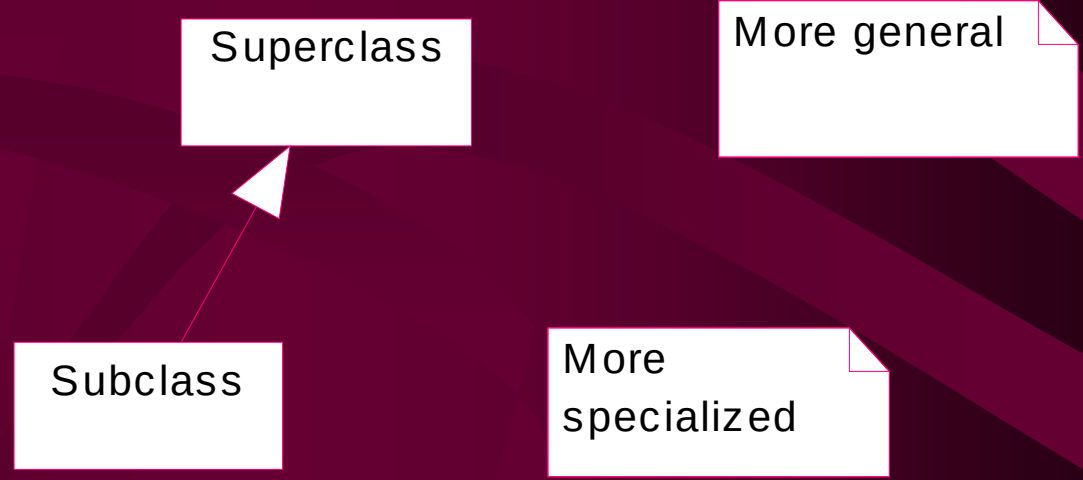
Examples

Correspondences

- An object is instance of a class
- A link is instance of a relation
- Links connect objects, relations connect classes
- A link between two objects implies a relation between the classes of these objects

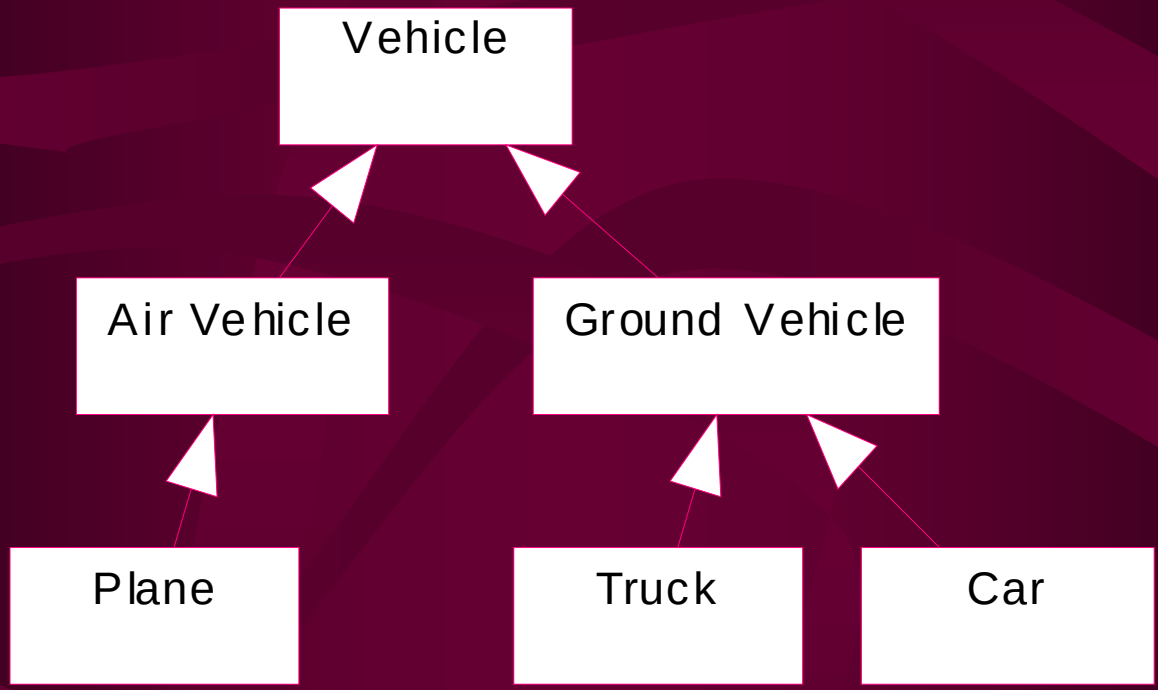
Class Hierarchies

- Master complexity
  - Hierarchies of abstractions
- Generalization
  - Superclasses
- Specialization
  - Subclasses



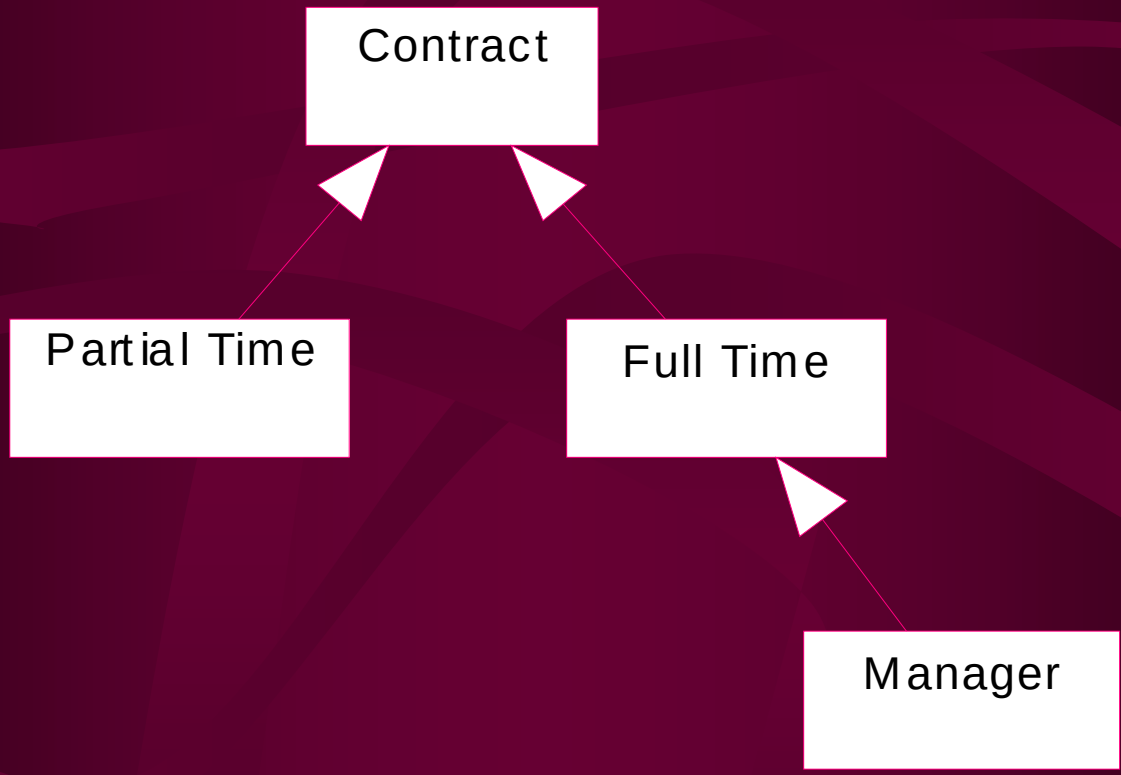
Generalization

- Factor out the common elements
  - attributes, operations, relations, constraints



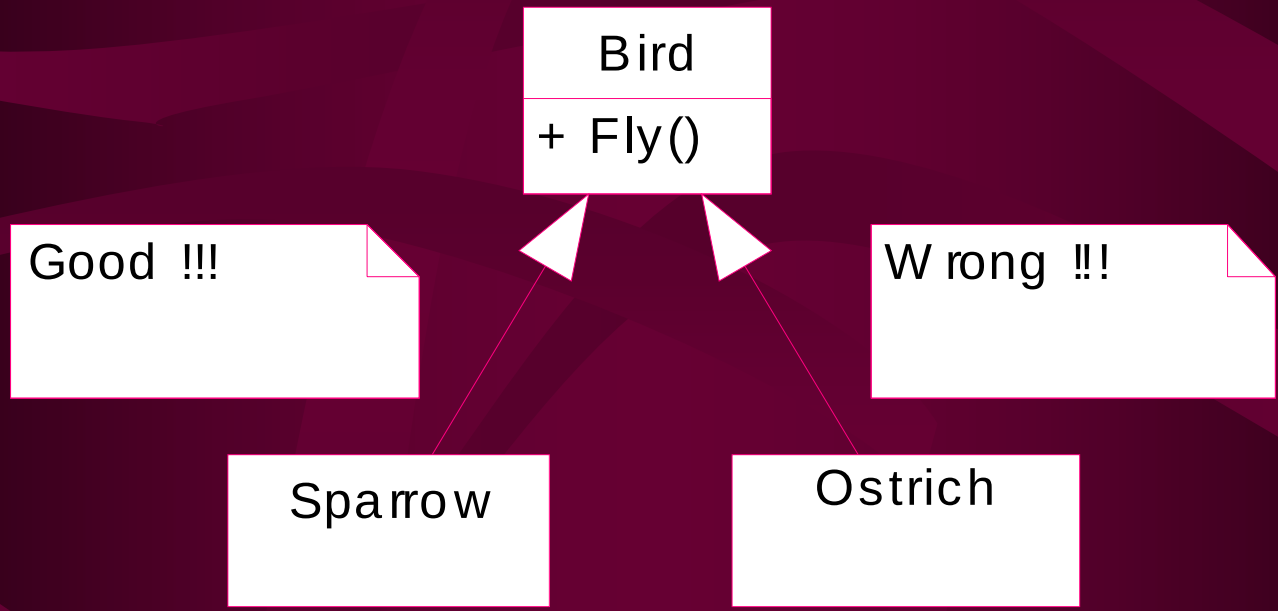
Specialization

- Extension of a set of classes



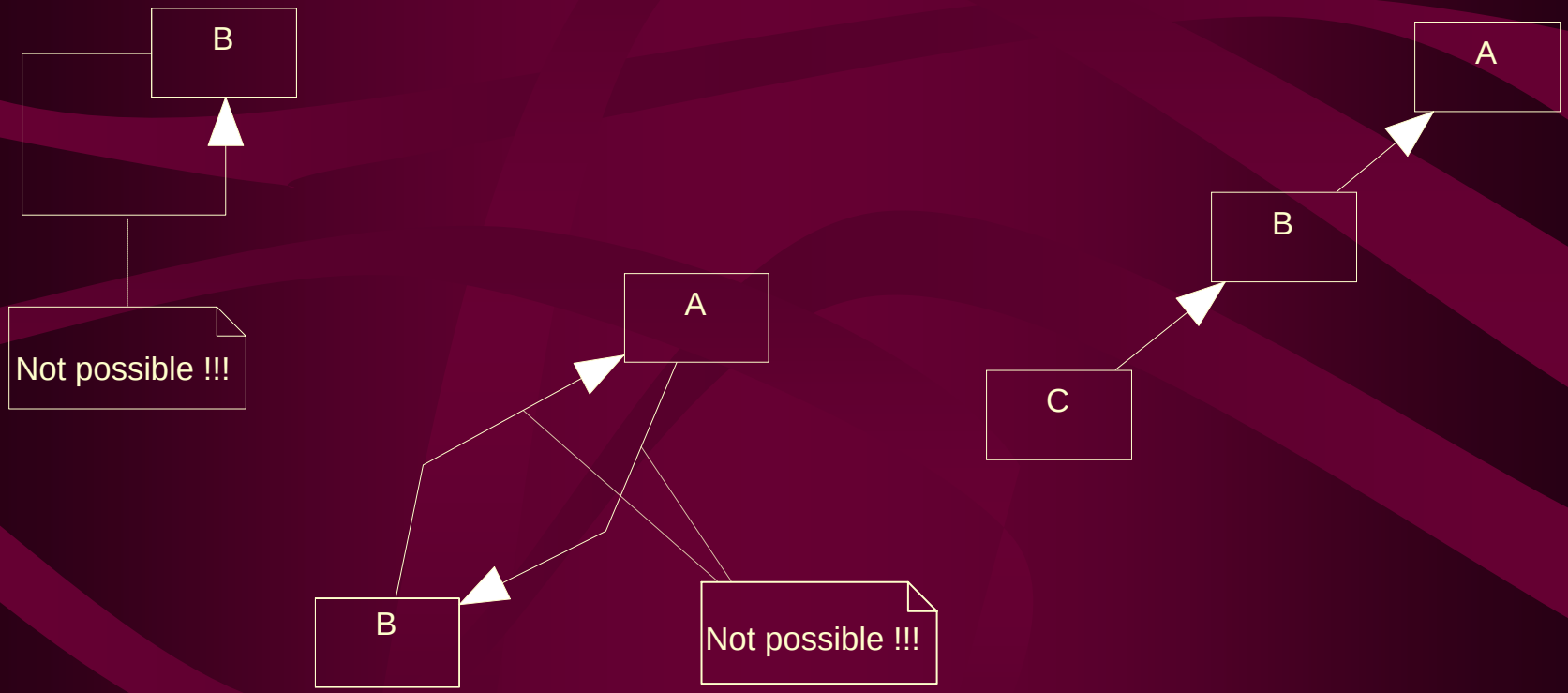
Properties of generalization

- Always means : *is a* or *is a kind of*



Properties of generalization

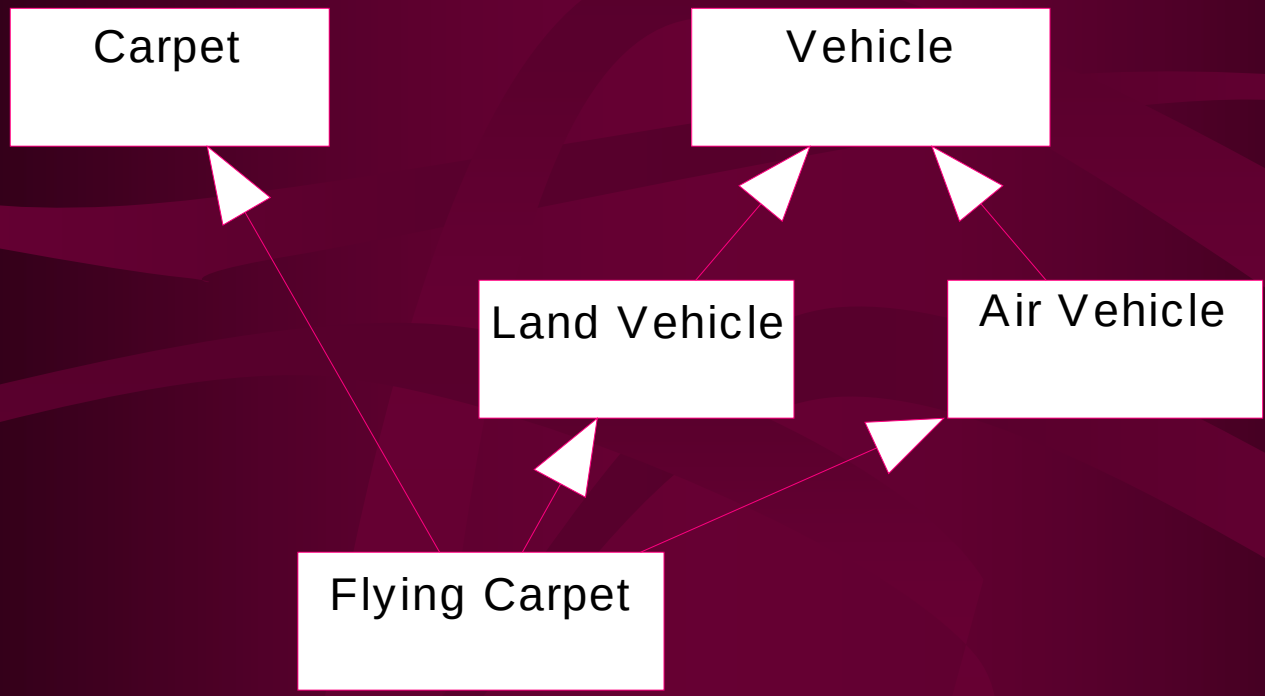
- Non-reflexive, non-symmetric, transitive



Substitution principle

- *It must be possible to substitute any object instance of a subclass for any object instance of a superclass without affecting the semantics of a program written in terms of the superclass.*

Multiple Generalization

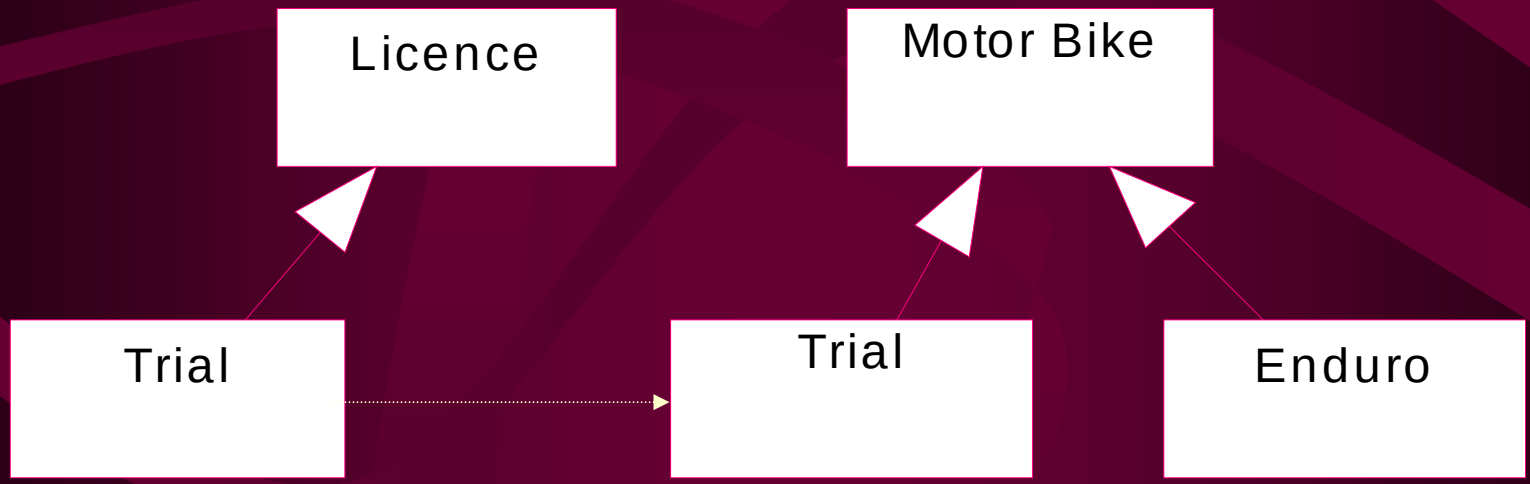


Inheritance

- Most often used technique for implementing generalization
- Build one class from another by sharing attributes, operations, relations and constraints within a class hierarchy
- Not always available in the implementation language / environment

Dependence

- Poorly semantically loaded relation
- Dependence shows obsolescence rules
  - `Include, With, Instantiation`



Summary

- Classes are connected by relations
  - Associations are bi-directional symmetric semantic connections between classes
  - Aggregations are bi-directional asymmetric semantic connections between classes
  - Generalization orders classes within hierarchies
  - Dependence shows obsolescence rules

Summary

- UML is the standard language for object-oriented models
- UML provides notational support for requirement analysis
  - Business modeling
  - Use Cases modeling
  - Domain Modeling