Génie logiciel et gestion de projets

UML Overview & UML Class Diagrams

Roel Wuyts
ULB
2005/2006

http://decomp.ulb.ac.be/education/GL0506/

Object-Oriented Modelling

Question: given some problem, how to develop an object-oriented system to address that problem ?

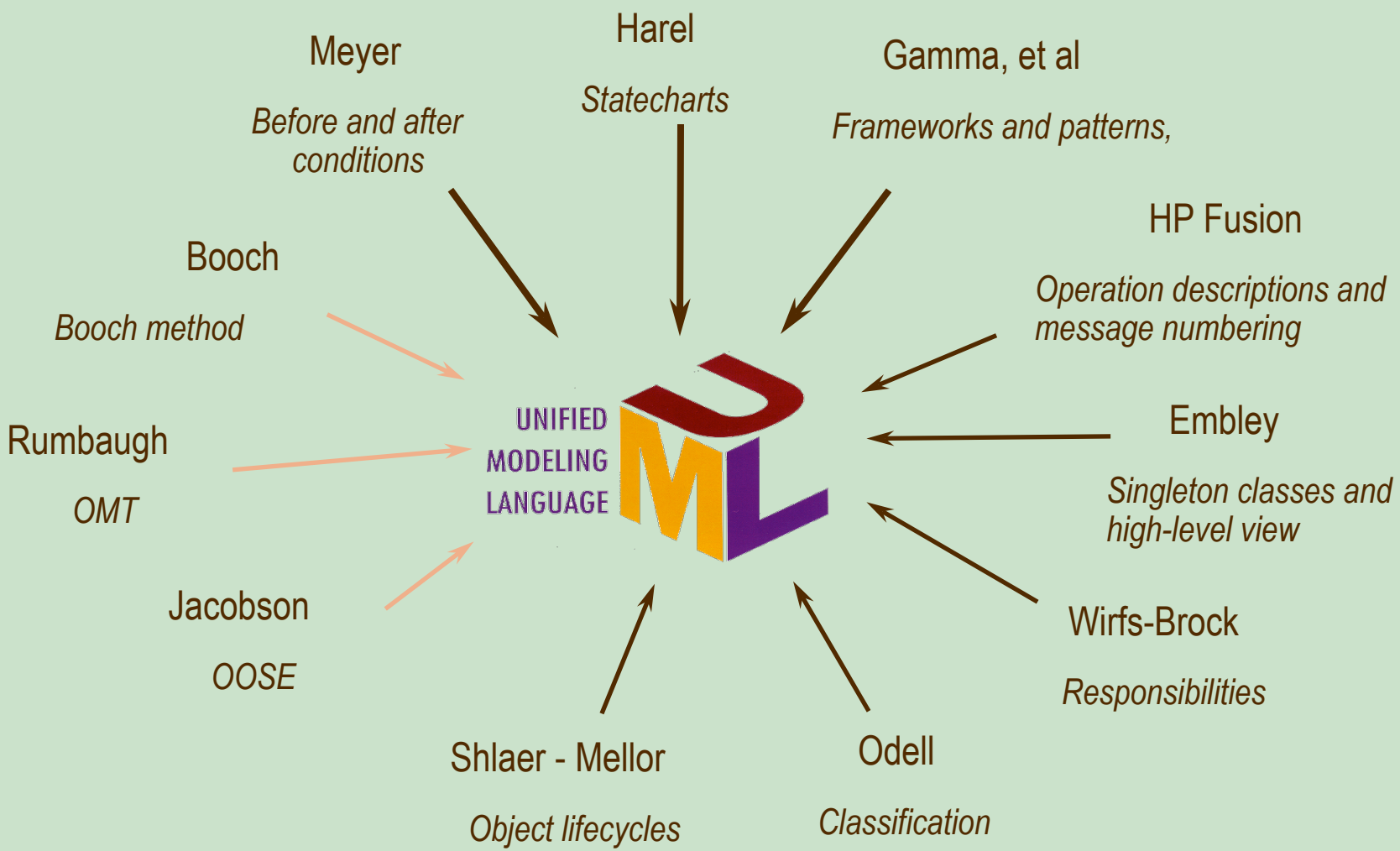
- Object-Oriented Modelling to get there
- Give object-oriented solution to problem
- Most can be used with different processes (waterfall, incremental, ...)
- Will use UML diagrams

Note: other approaches could be used

- e.g. functional decomposition, ER diagrams, ...

UML is ... unified

In 1994, more than 50 OO methods!



UML History

Designed by Booch, Rumbaugh, Jacobson (3 amigos)

- Started in 1994; version 1.0 finished in 1997
- Version 1.5 (1.4.2) since July 2004: current
- Version 2.0 in beta since 2004 - final late 2005

To end the OO method wars: standard

Standard adopted by OMG (also known from Corba)

General Goals of UML

Model systems using OO concepts

Establish an explicit coupling to conceptual as well as executable artifacts

To create a modeling language usable by both humans and machines

Models different types of systems (information systems, technical systems, embedded systems, real-time systems, distributed systems, system software, business systems, UML itself, ...)

UML Worldview

Two concepts:

- Views
- Diagrams

UML Views…

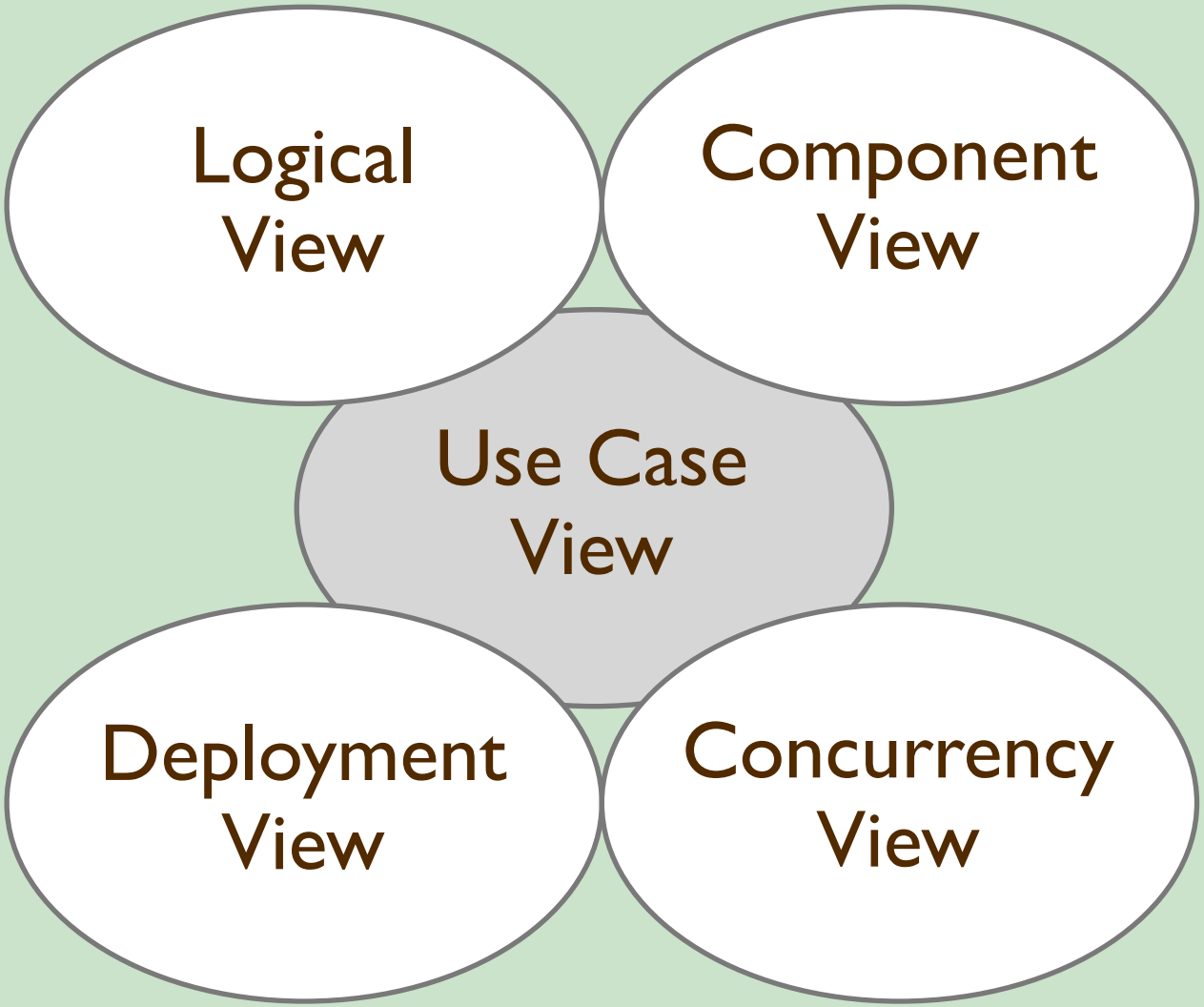
Each view is a projection of the complete system.

Each view highlights particular aspects of the system.

Views are described by a number of diagrams.

No strict separation, so a diagram can be part of more than one view.

UML Views…

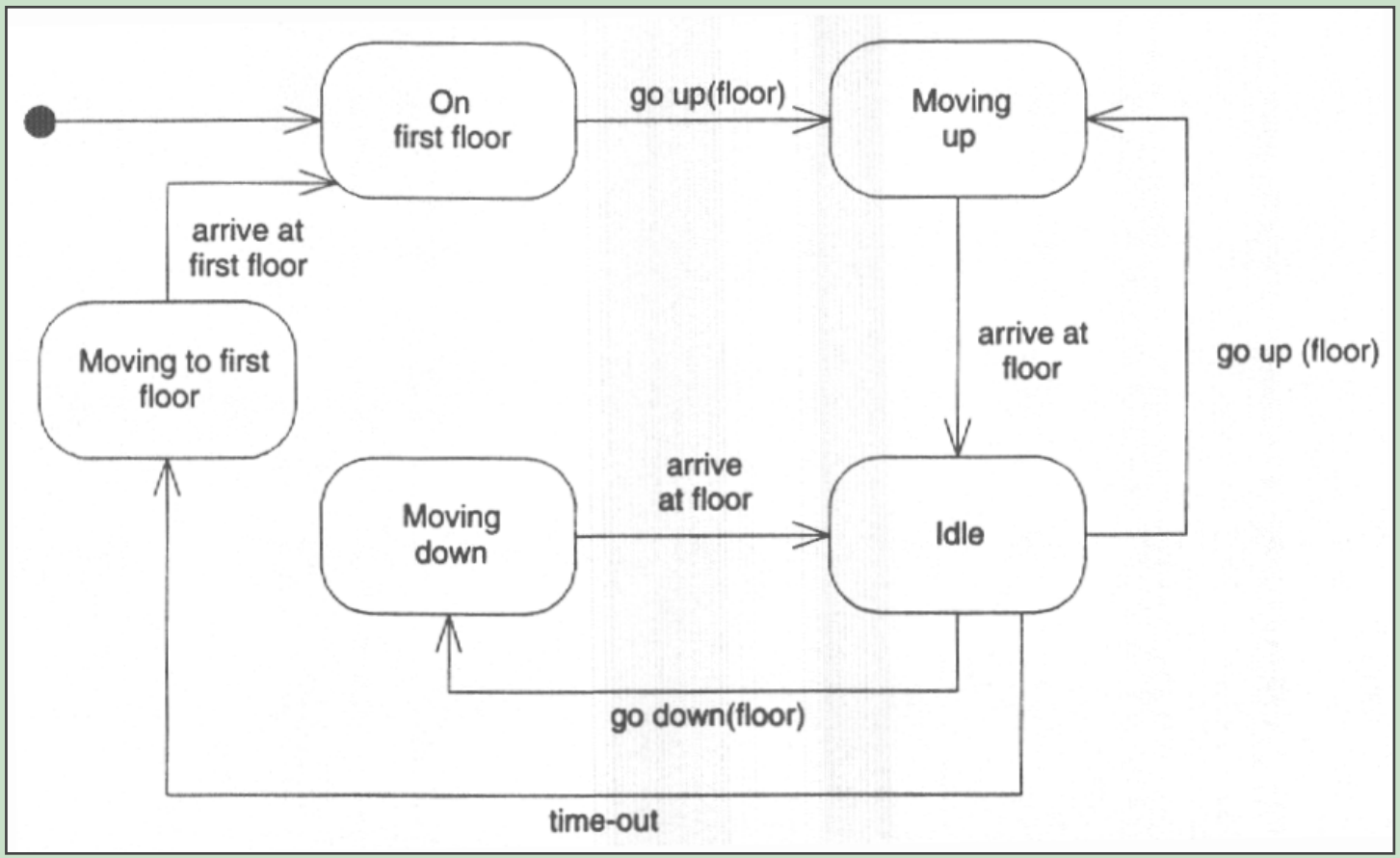

Logical View
Component View
Use Case View
Deployment View
Concurrency View

UML Diagrams...

- Use-Case diagram (see later)
- Class diagram (see later)
- Object diagram
- State diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Component diagram
- Deployment diagram

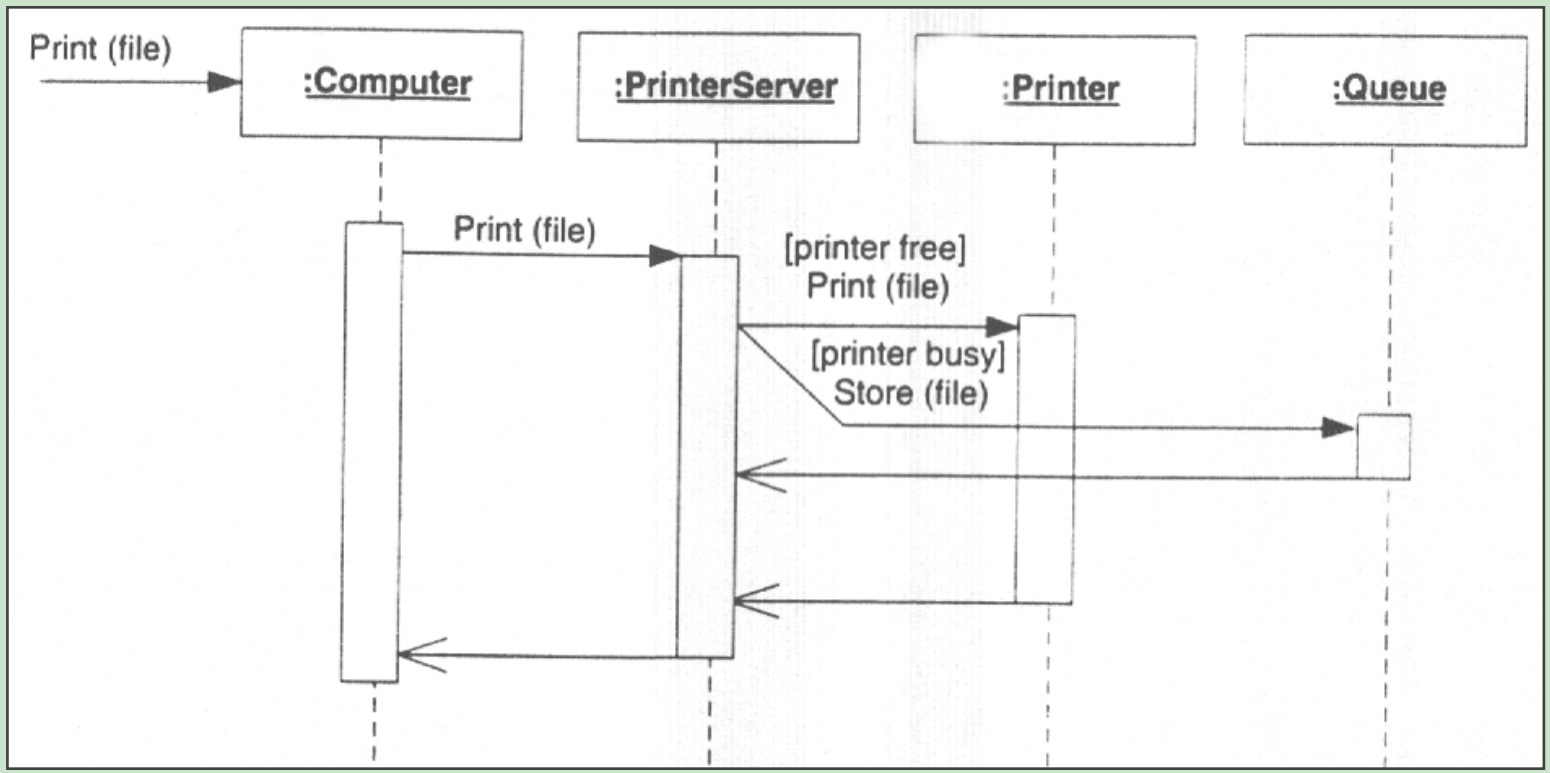
State Diagram

Represent the behaviour of a class in terms of (evolution of) its state.



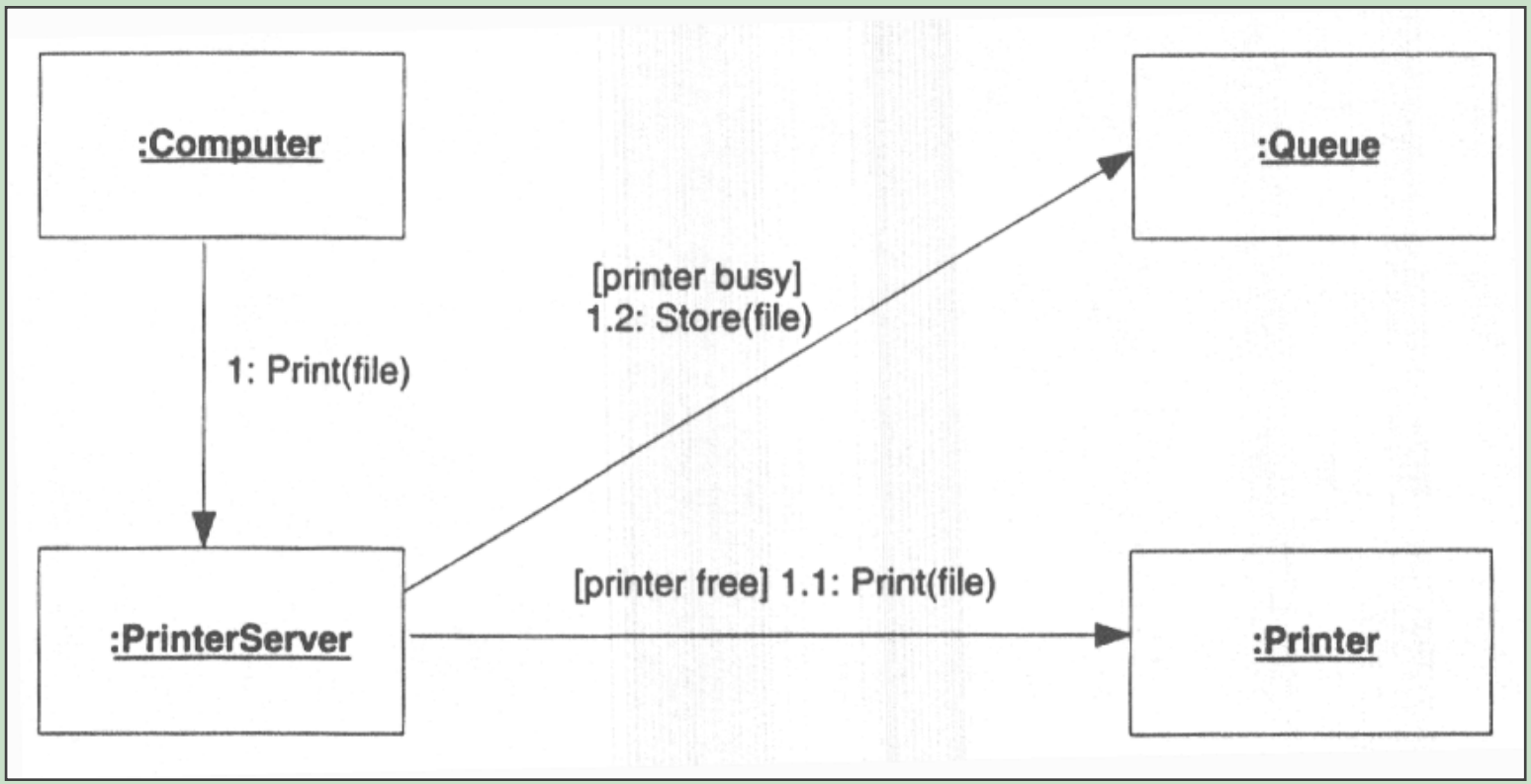
Sequence Diagram

Temporal representation of objects and their interactions.



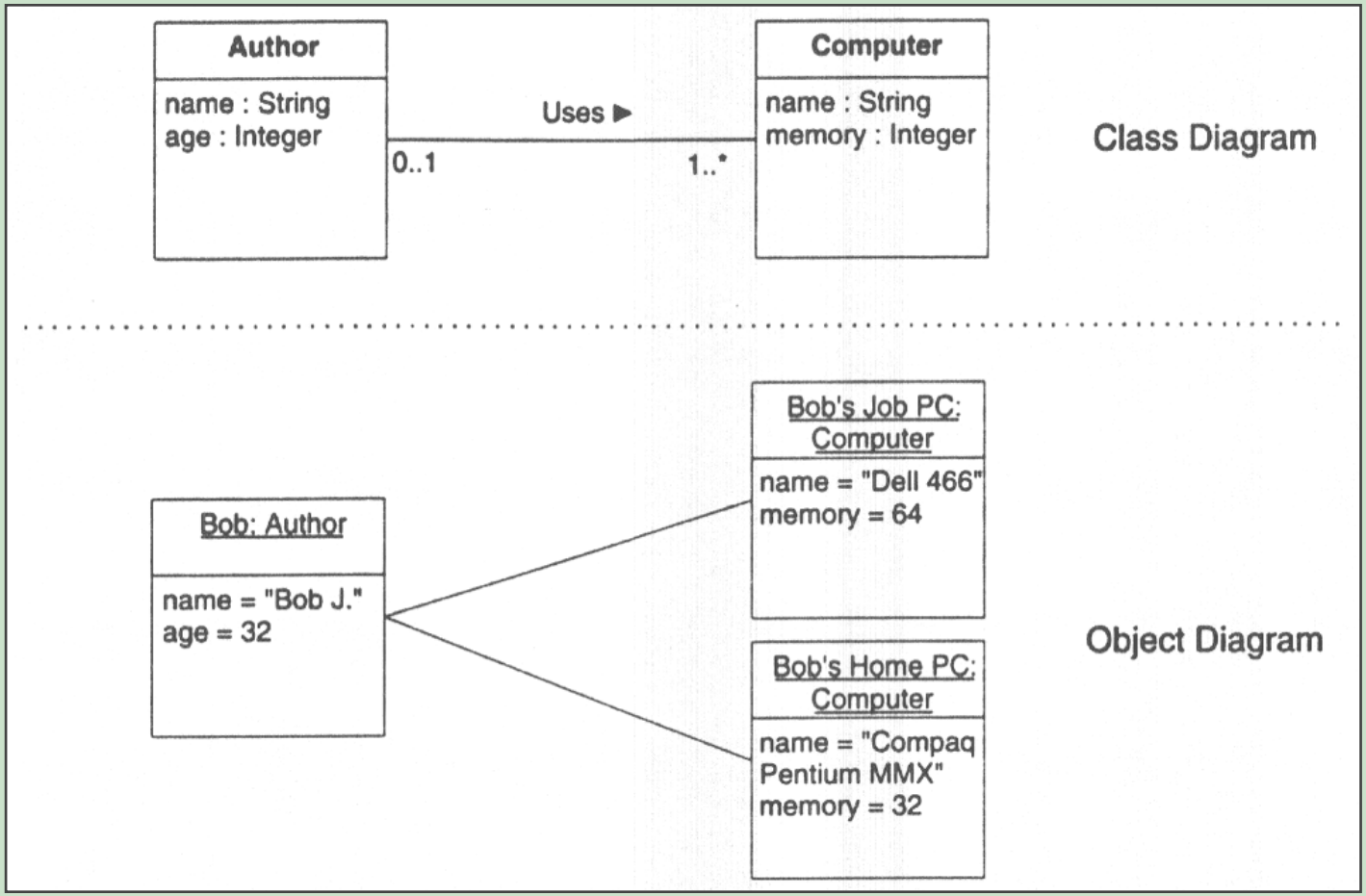
Collaboration Diagram

Spatial representation of objects, relations and interactions.



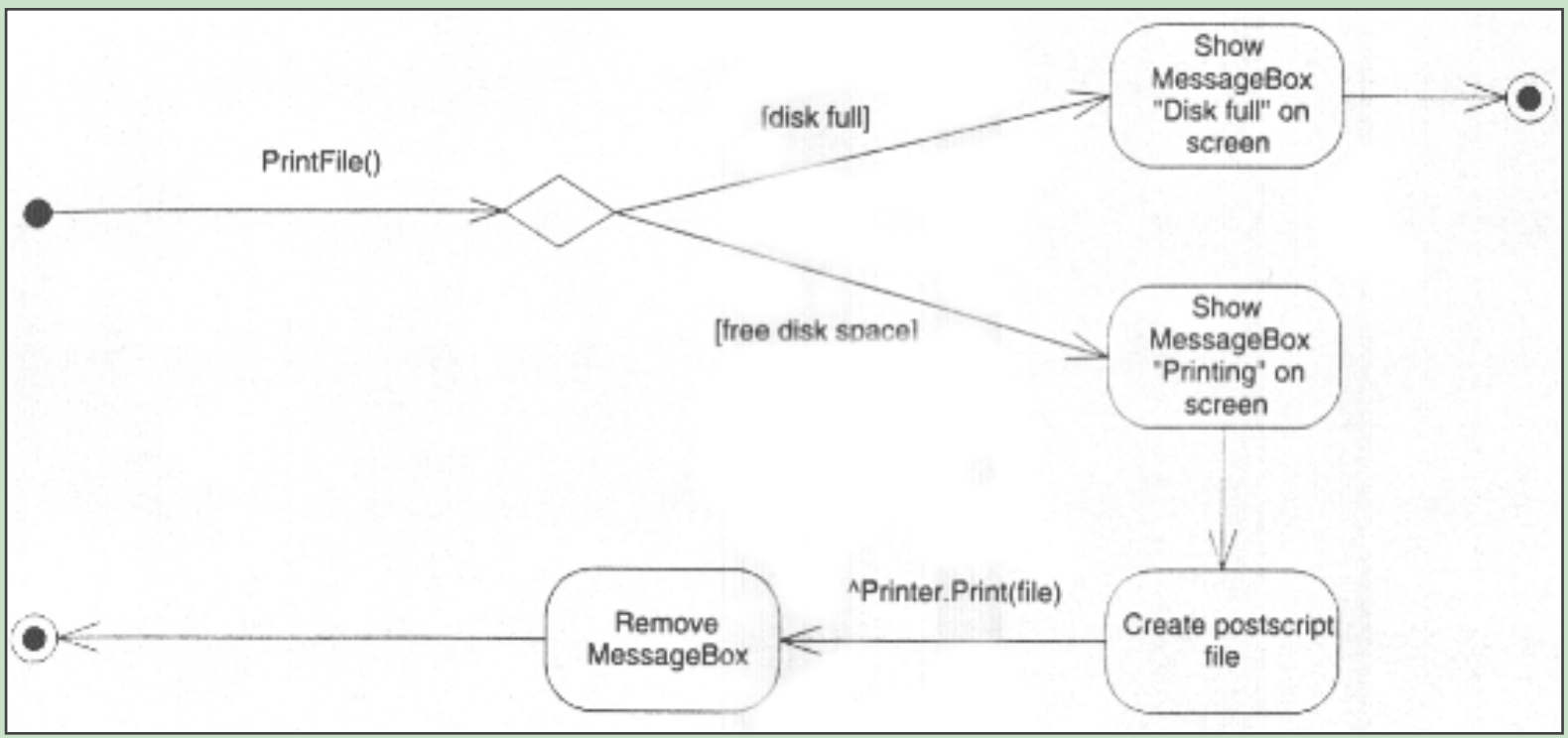
Object Diagram

Represents objects and their relations; corresponds to simplified collaboration diagrams (no message sends).



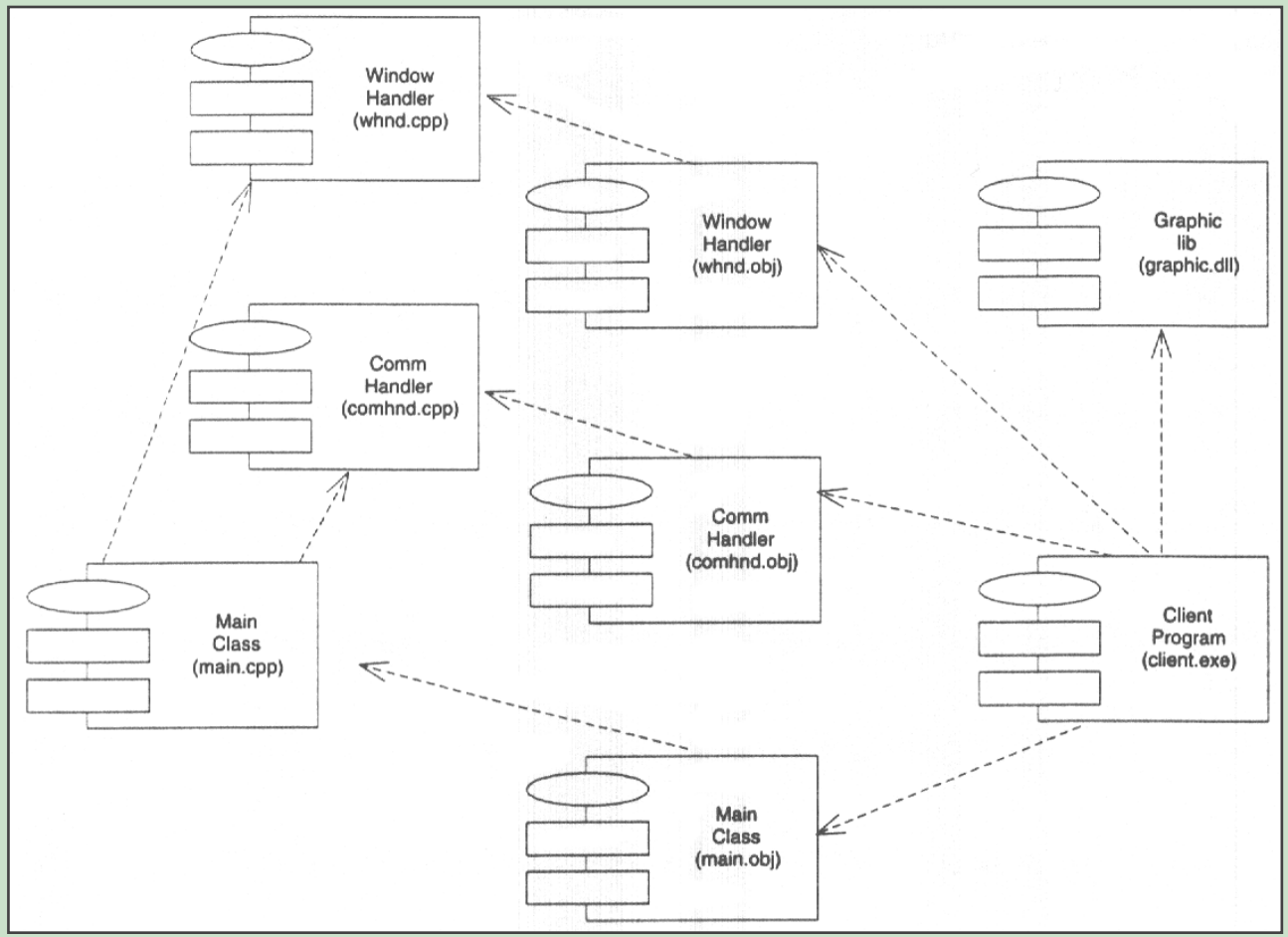
Activity Diagram

Represent the behaviour of one operation in terms of actions.



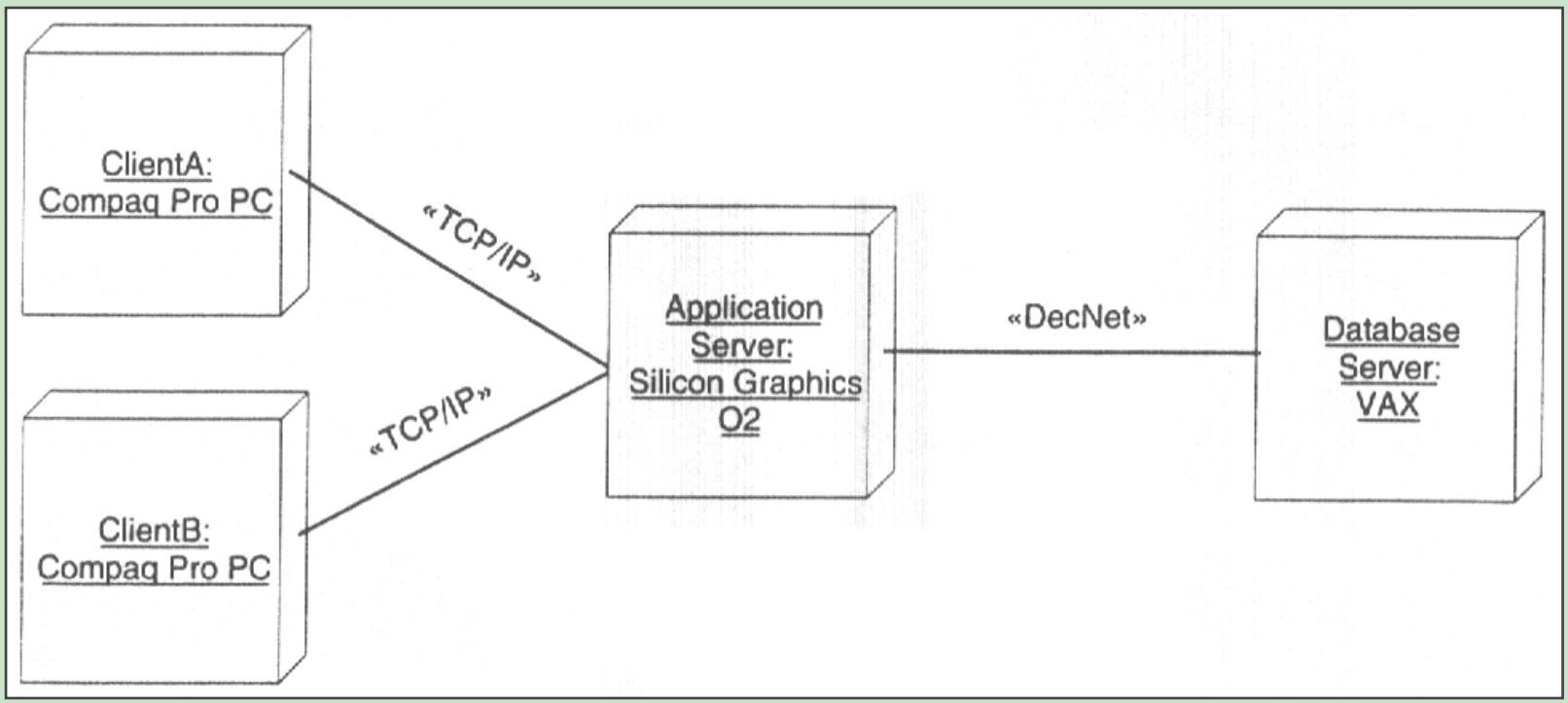
Component Diagram

Represent the physical components of a system.



Deployment Diagram

Represent the deployment of a system on hardware.



UML Metamodel

Some mental exercise:

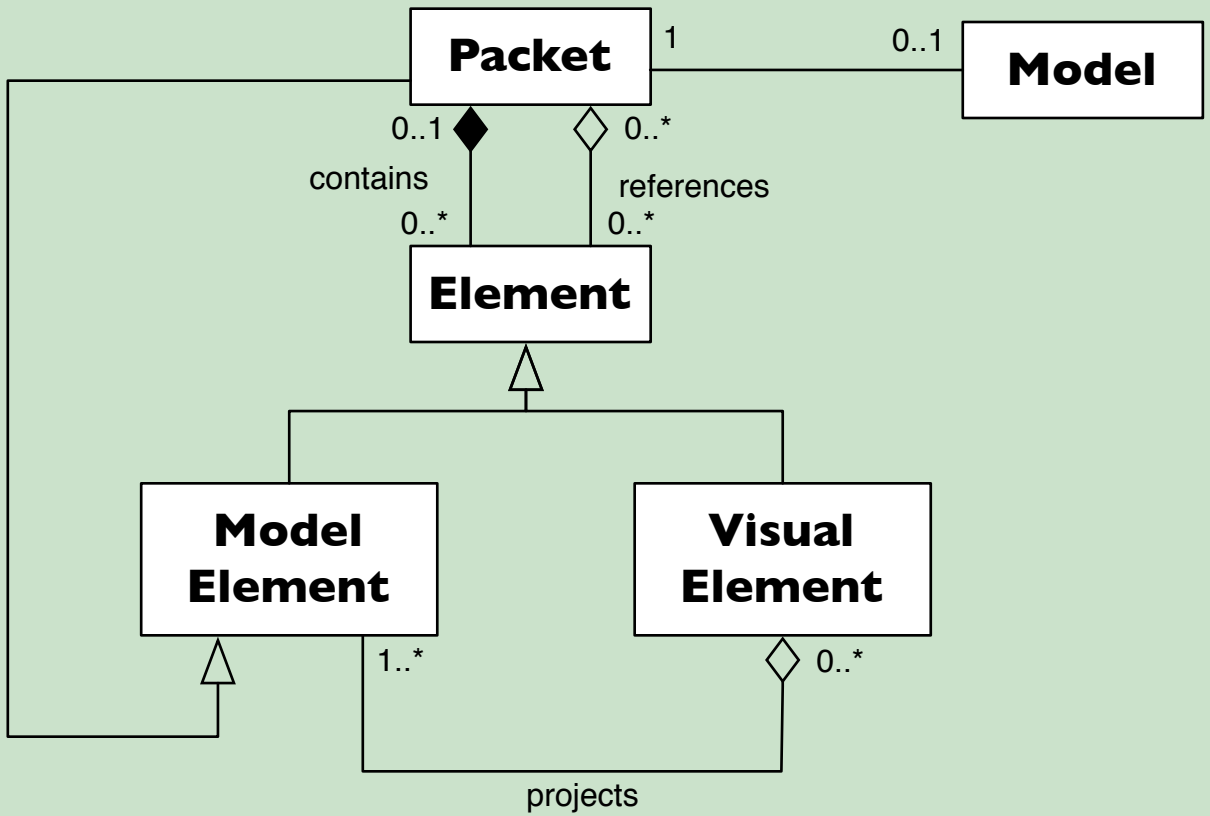
- Semantic of UML diagrams is described in UML!
  - Each diagram is an instance of a UML meta model
  - In other words: each diagram shows a simplified view of the meta model
  - UML Meta Model describes all possibilities

Common UML: Elements

Core : Element

- used for both model elements as visual elements
  - model element: represents system abstraction while modelling
  - visual element: textual or graphical representation of a model element that allows a user to interact

Meta-model part: Element

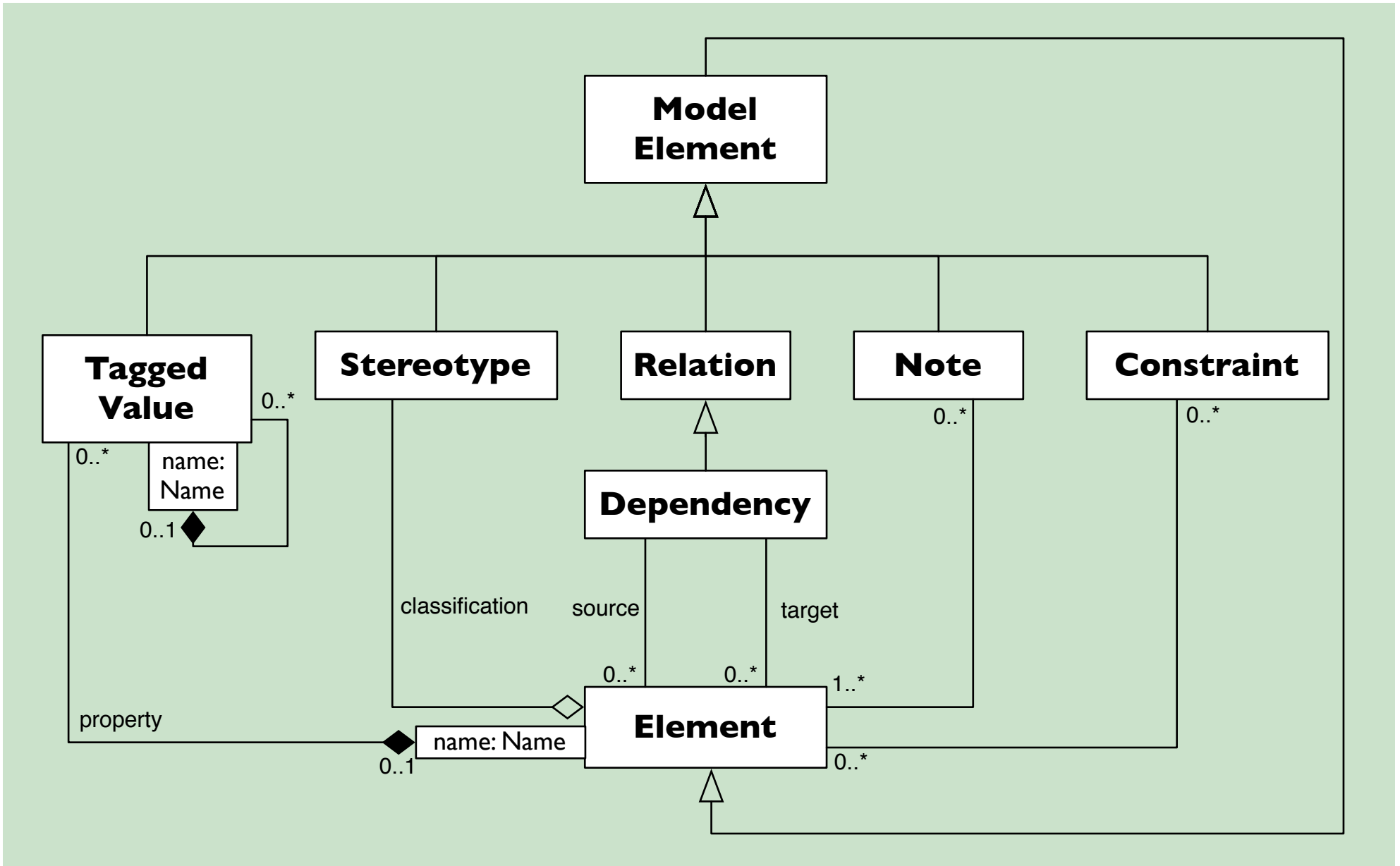


Common UML: mechanisms

Across UML, a number of common mechanisms are used:

- Stereotypes
- Tagged Values
- Notes
- Constraints
- Dependency relationships
- (type, instance) and (type, class) dichotomies

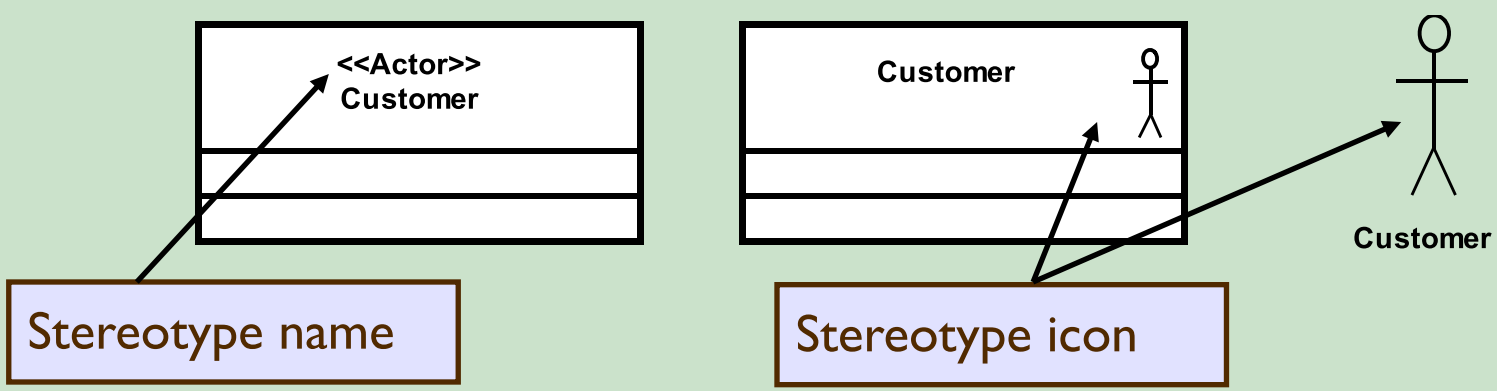
Meta model part: common mechanisms



Stereotypes

Stereotypes

- Allows to define a new kind of model element based on an existing one
- Basically adds extra semantics
- There are predefined stereotypes



Tagged Values

Tagged values

- Name-value pairs of information
  - (name, value)
- Hold additional information about elements

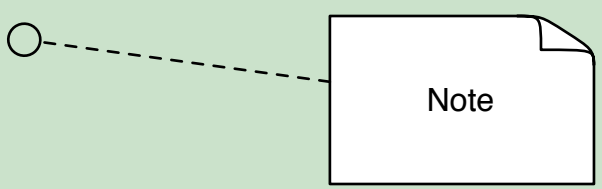
Notes

Comment attached to one or more elements

Holds no semantic information, just information

- ✦ use stereotypes for semantic information

Graphically:



Constraints

Restrictions that limit the usage of an element or the semantics of an element

No syntax specified

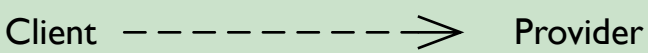
- Can be natural text, pseudo-code, mathematical expressions, OCL (object constraint language), ...

Dependency Relationships

One-directional usage relationship between two model elements (called *source* and *target*)

Notes or constraints are valid sources for dependency relationships

Graphical notation:



(client depends on provider)

Common UML: Primitive Types

Primitive types:

- *Boolean*: enumerated type {true, false}
- *Expression*: string with some semantics
- *List*: ordered sequence, possible indexed
- *Multiplicity*: see next
- *Name*: string used to indicate element
- *Point*: tuple (x,y,z) that indicates point in space
- *String*: list of characters designated with a name
- *Time*: represents absolute or relative time
- *Non-interpreted*: blob

Primitive type: multiplicity

Non-empty set of positive integers

Syntax:

multiplicity ::= [interval | number] { ‘ , ’ multiplicity}

interval ::= number “.. ” number

number ::= positive_number | name | ‘*’

Examples:

1

1, 3..4, 5..8

0..*

Common UML : Packet

Groups model elements

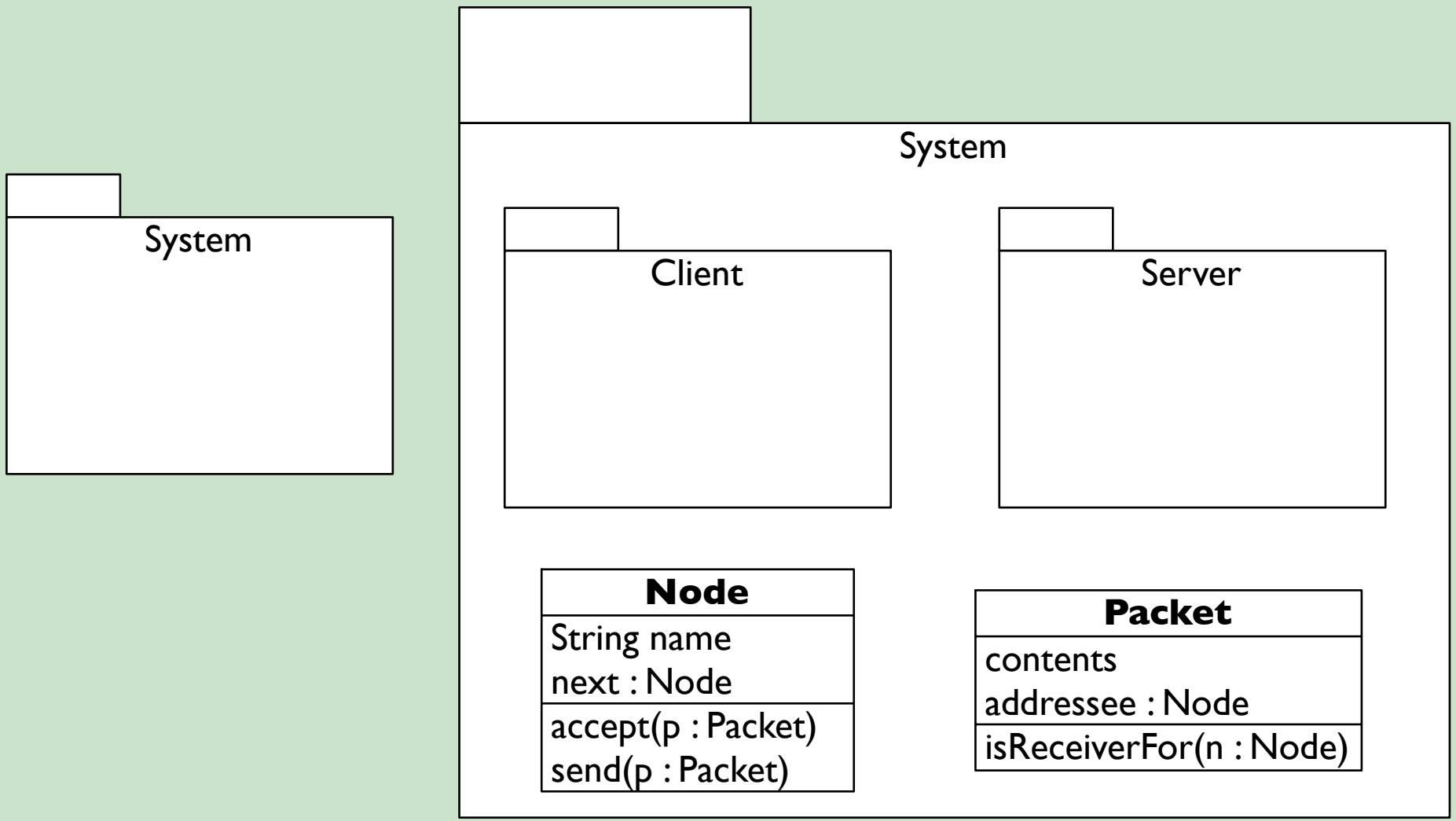
- All elements belong to a package
  - There is a root package for the system
- Can contain other packets, since packets are model elements themselves

Enforces namespace

- Two elements in a different package can have the same name

Packet

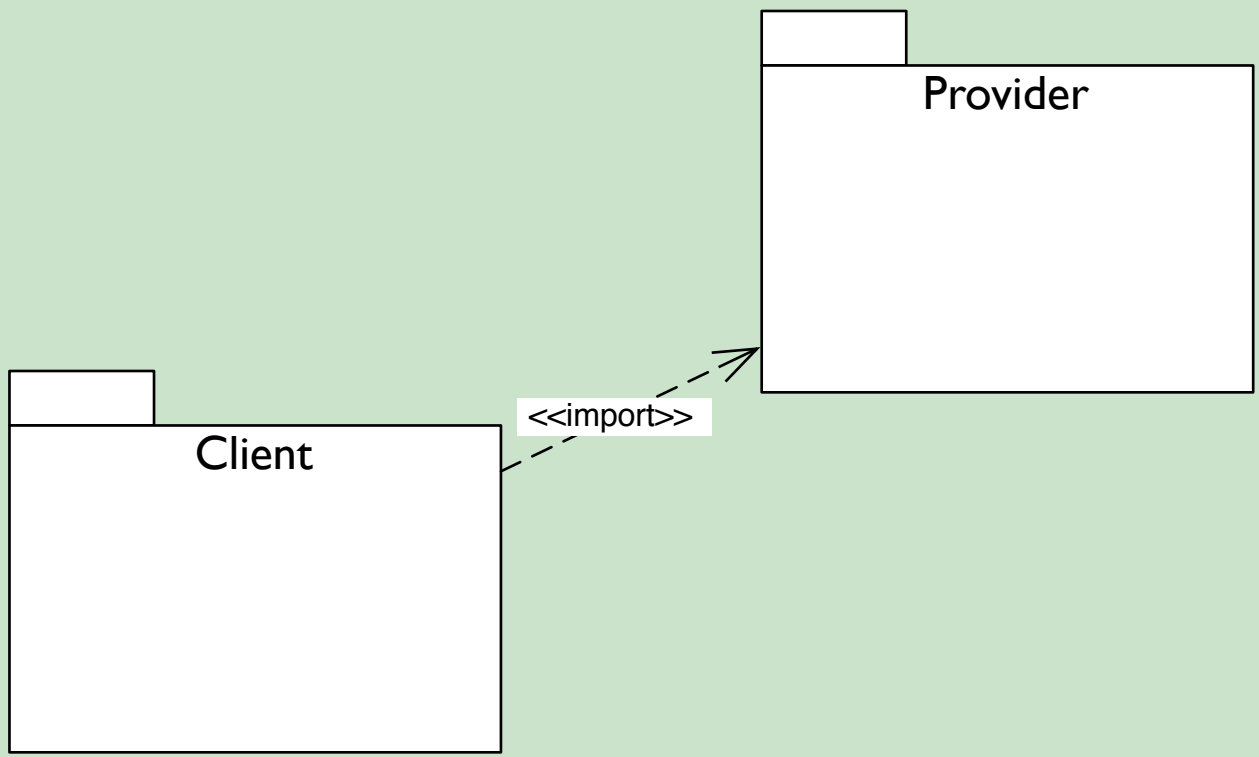
Graphical notation



Package import

Elements can be shared between packages

The import relation between packages is modelled using a *dependency relationship* stereotyped with *import*



Package visibility

Elements contained in a package are not visible to the outside world

- except when they are declared *public* (using stereotypes, for example)
- otherwise they are *implementation* (not visible)

Class Diagrams

Static model type

- A view of the system in terms of classes and relationships

Classes not only describe attributes but also behaviour !

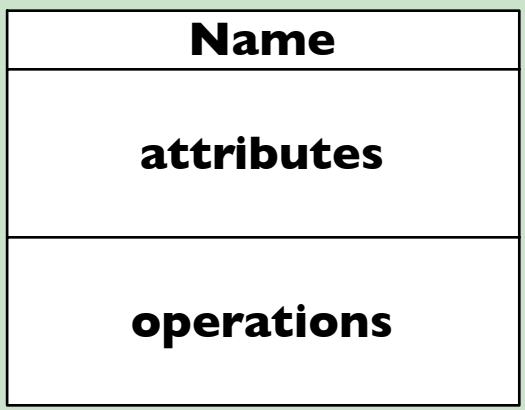
Description of object types .

- Attributes and behaviour of a type of objects
- All objects are instances of a certain class

A Class in UML…

A rectangle divided into 3 compartments :

- name, attributes, operations



Uppercase, bold

lowercase
+, -, # for visibility
= for defaults
{ } for enumerations
underline : static

Some examples

Invoice
+ amount : Real + date : Date = Current date + customer : String - administrator : String = "Unspecified" - number of invoices : Integer + status : Status = unpaid {unpaid, paid}

Figure
size : Size # pos : Position
+ draw() + scaleFigure(percent : Integer = 25) + returnPos() : Position

Relationships Between Classes

An *association* is a connection between classes

- "usage"

A *generalization* is a relationship between a more general and a more specific element

- "inheritance"

A *refinement* is a relationship between two descriptions of the same thing but at different levels of abstraction

A *realization* is a relationship between elements where one carries out what the other specifies

Associations

Specify structural relationships

Specifies that objects are interconnected

Can be implemented in a lot of ways

- through instance variables
- through arguments of methods
- through auxiliary classes
- ...

Association Relationship

Drawn as line between classes

By default bidirectional

- ✦ but particular direction can be indicated

Can contain multiplicities

- ✦ a range that tells us how many objects are linked

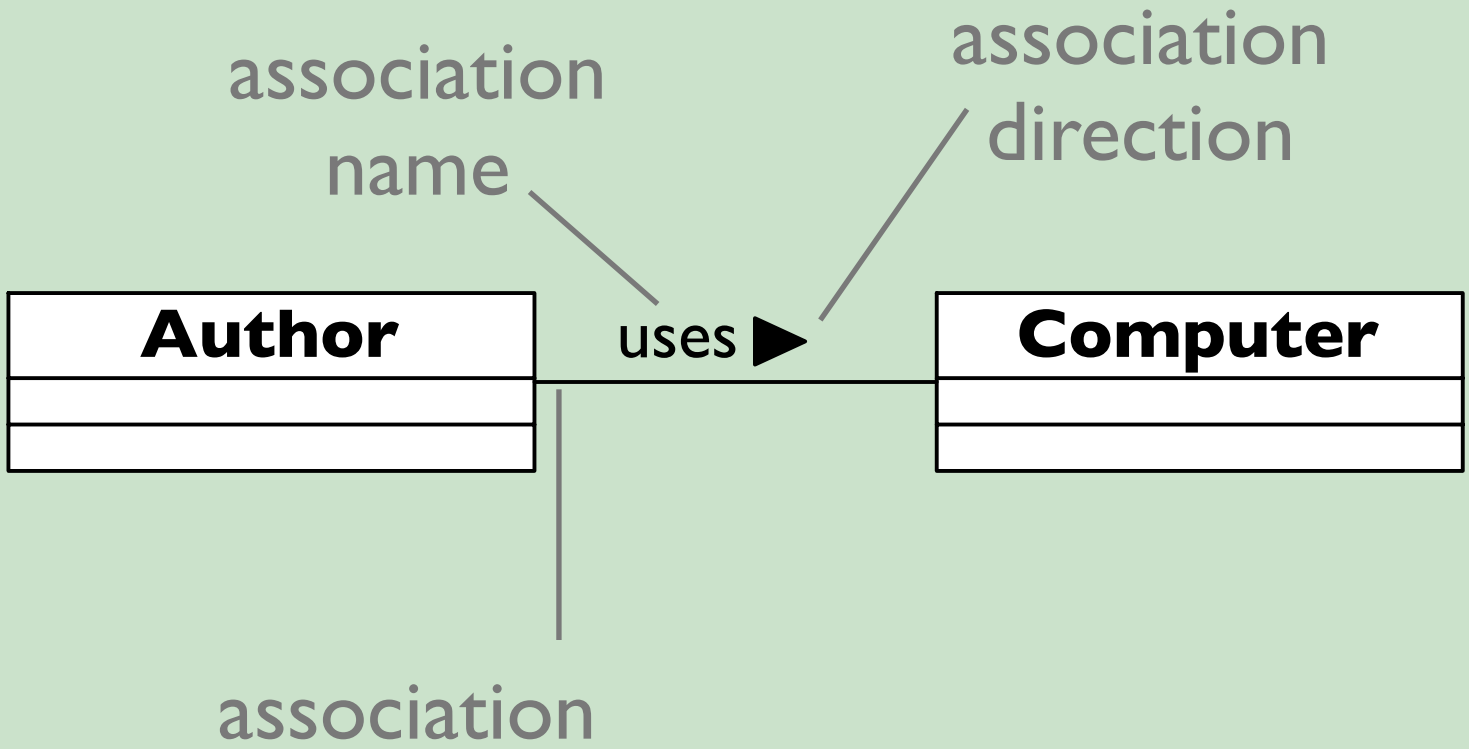
9 examples to show different aspects and give some additional information...

Example 1

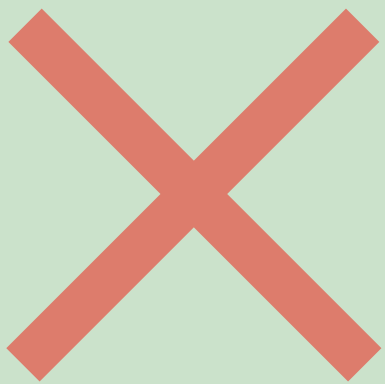
Model the following:

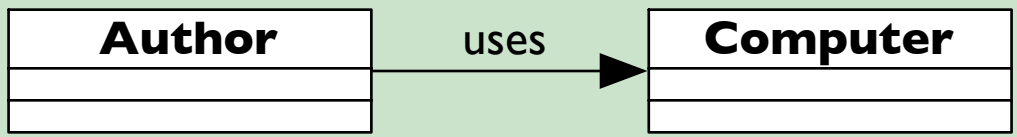
- An author can use a computer.

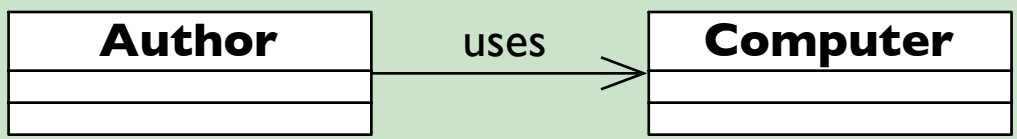
Example 1 Solution

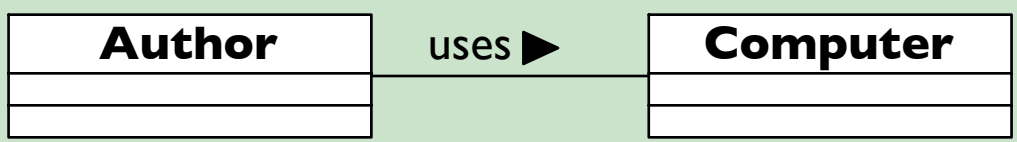


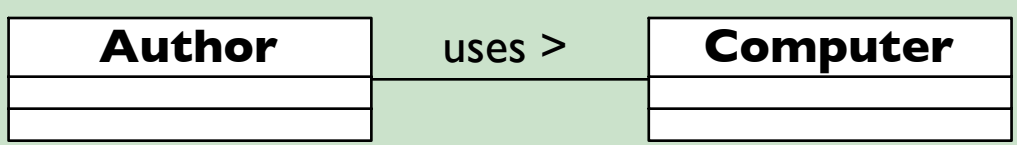
On association directions...









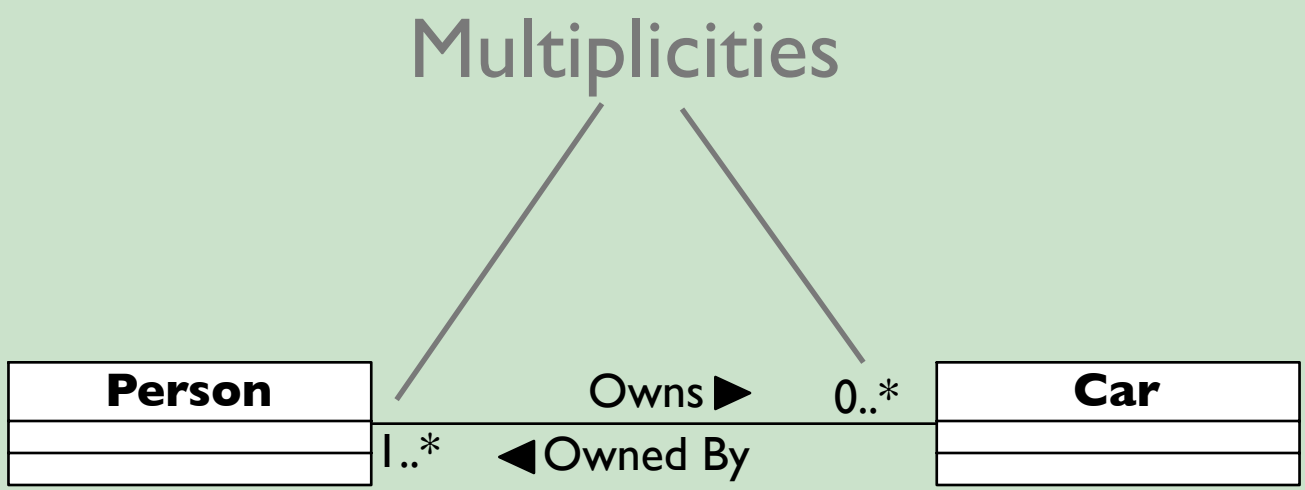


Example 2

Model the following:

- A user can own 0 or more cars. Each car is owned by 1 or more persons

Solution Example 2



A Person Owns 0 or many Cars
A Car is Owned by 1 or many Persons

Common Multiplicities

Multiplicity	Notation
optional	0..1
zero or more	0..* or *
at least one	1..*
exactly one	1 or left blank

(See primitive types for all possibilities)

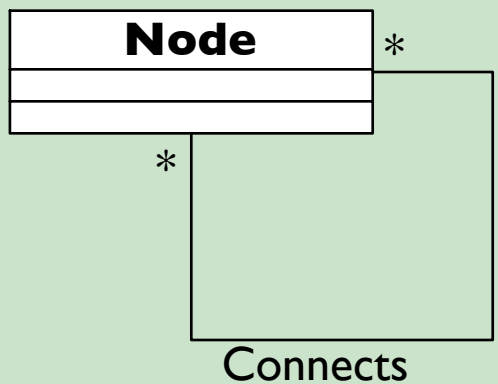
Example 3

Model the following:

- Nodes can be connected to other nodes.

Recursive Associations

Recursive association: connecting a class to itself



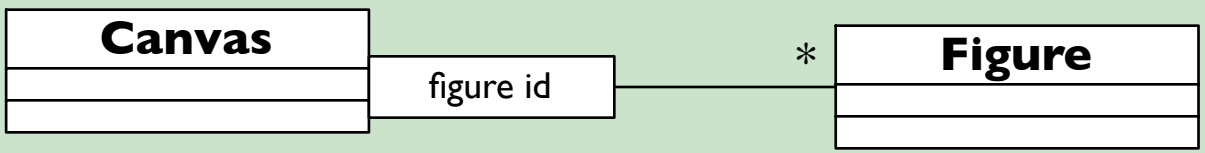
Example 4

Model the following:

- A Canvas contains many Figures which are identified by an identifier

Qualified Associations

Specifies how a certain object at the many end is identified (+/- a key)

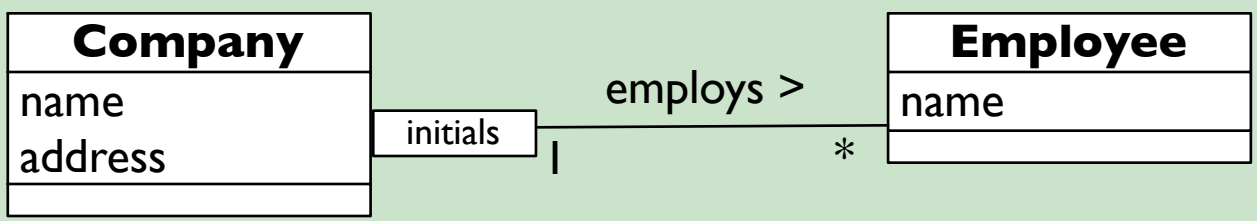


A Canvas contains many Figures which are identified by a figure id

Qualified associations as restrictions

Qualified association subdivides the referenced set of objects into partitions where, viewed from the initial object, each partition may occur only once

Example: employees are partitioned: all employees with the same initials belong to one partition



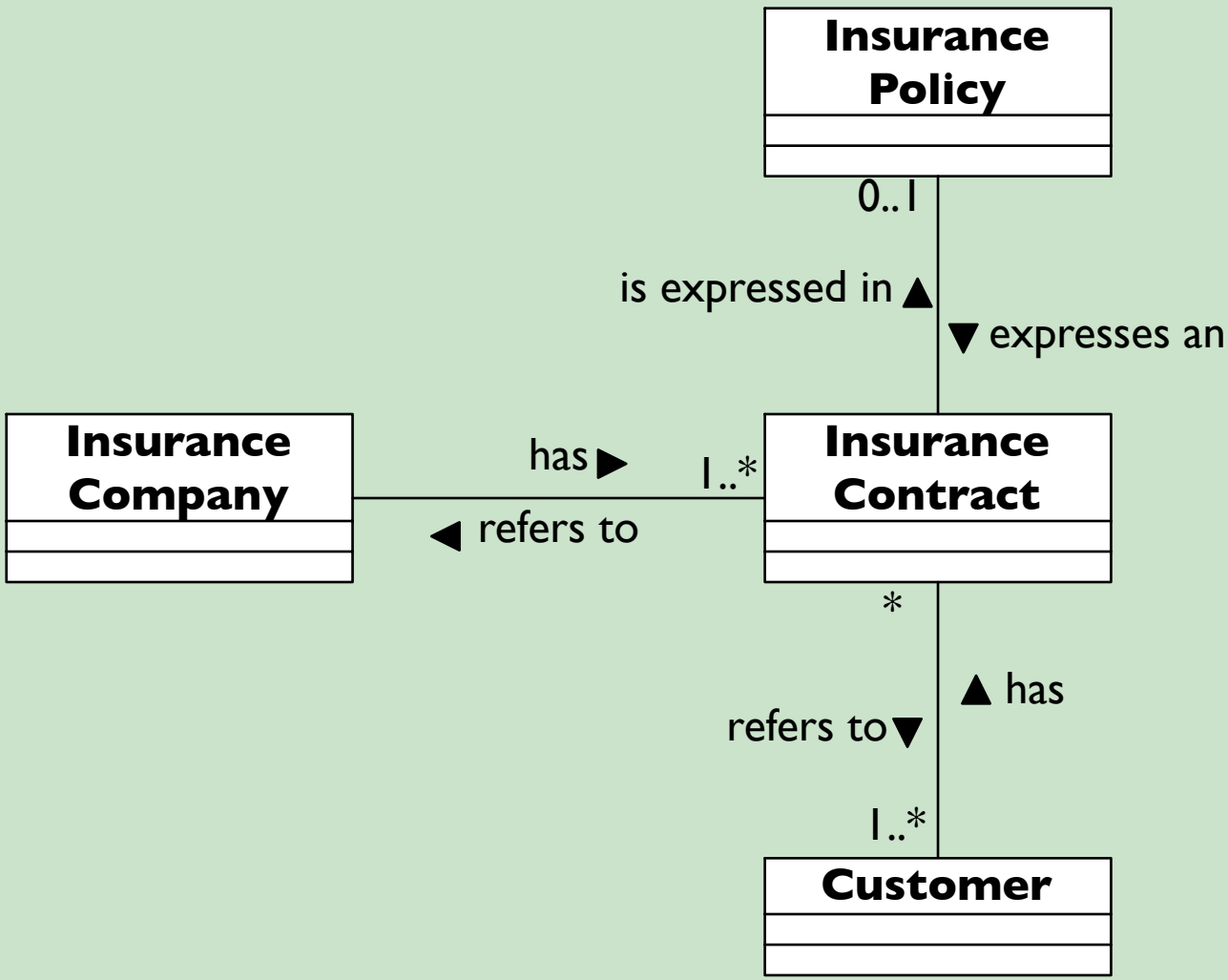
company	initials	employee
ULB	RW	Roel Wuyts
ULB	TM	Thierry Massart
UMH	TM	Tom Mens

Example 5

Model the following:

- An insurance company has insurance contracts, which refer to one or more customers.
  - A customer has insurance contracts (zero or more), which refer to one insurance company.
  - An insurance contract is between an insurance company and one or more customers. The insurance contract refers to both a customer (or customers) and an insurance company.
  - The insurance contract is expressed in an (zero or one) insurance policy (a written contract of insurance). The insurance policy refers to the insurance contract.

Example 5 solution



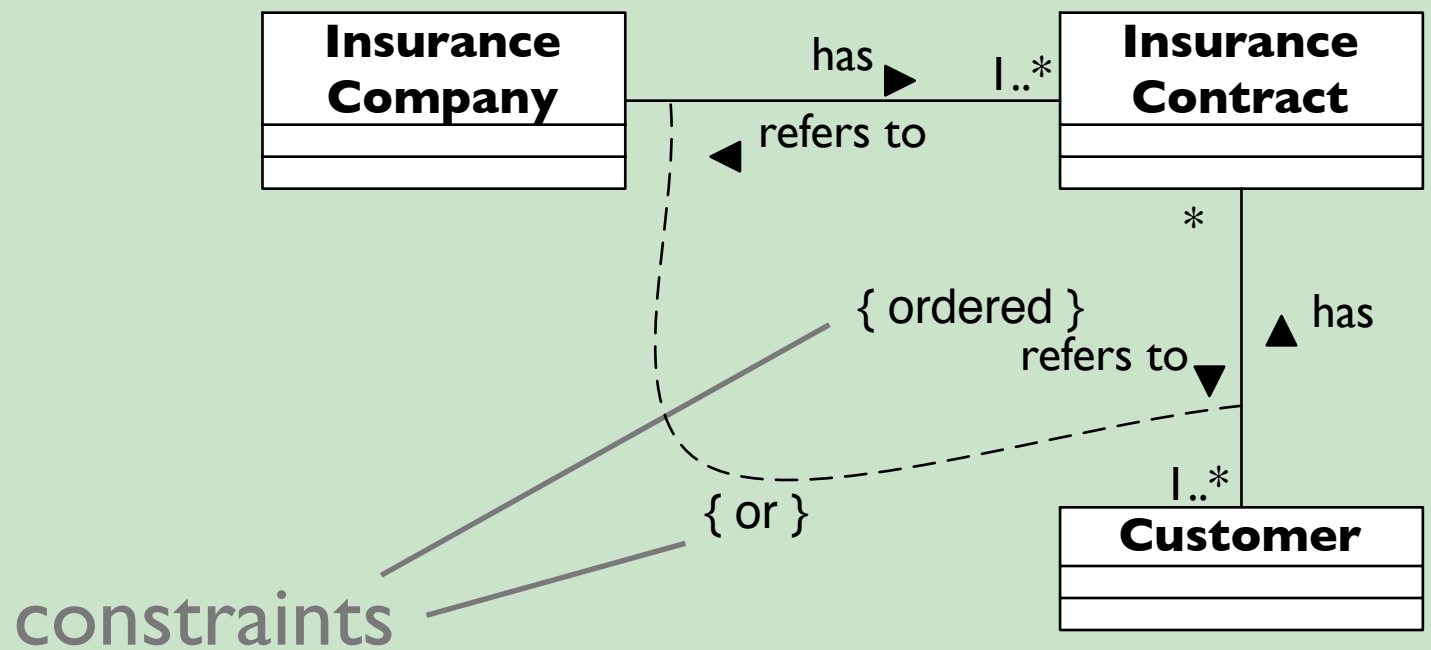
Example 6

Model the following:

- An insurance contract belongs to a customer or to a company. A customer can have multiple insurances. An insurance contract has an ordered collection of 1 or more customers.

Example 6: 'or' associations

Put constraint between the associations



Other constraints

Some widely used constraints:

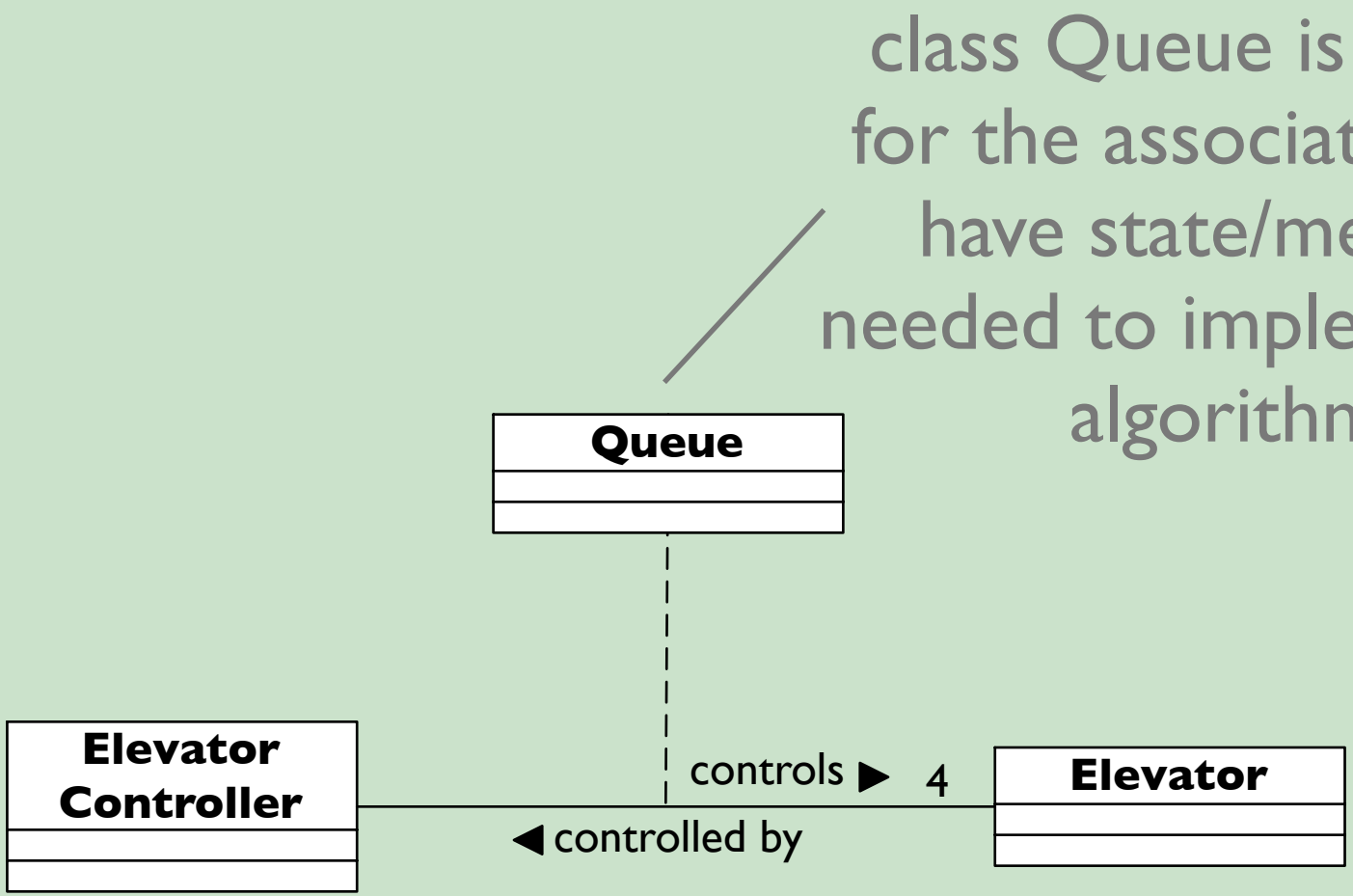
- {ordereded}, {or}, {xor}
- {implicit} : specifies that the relationship is not manifest but, rather, is only conceptual
- {changeable}: Links between objects may be added, removed, and changed freely
- {addonly}: New links may be added from an object on the opposite end of the association
- {frozen}: A link, once added from an object on the opposite end of the association, may not be modified or deleted

Example 7

Model the following:

- An elevator control manipulates four elevators. On each link between the elevators and the elevator conrol, there is a queue. Each queue stores the requests from both the elevator control and the elevator itself. When the elevator control chooses an elevator to perform a request from a passenger outside the elevator, the elevator control reads each queue and choses the elevator that has the shortest queue. The choice could also be made using some clever algorithm.

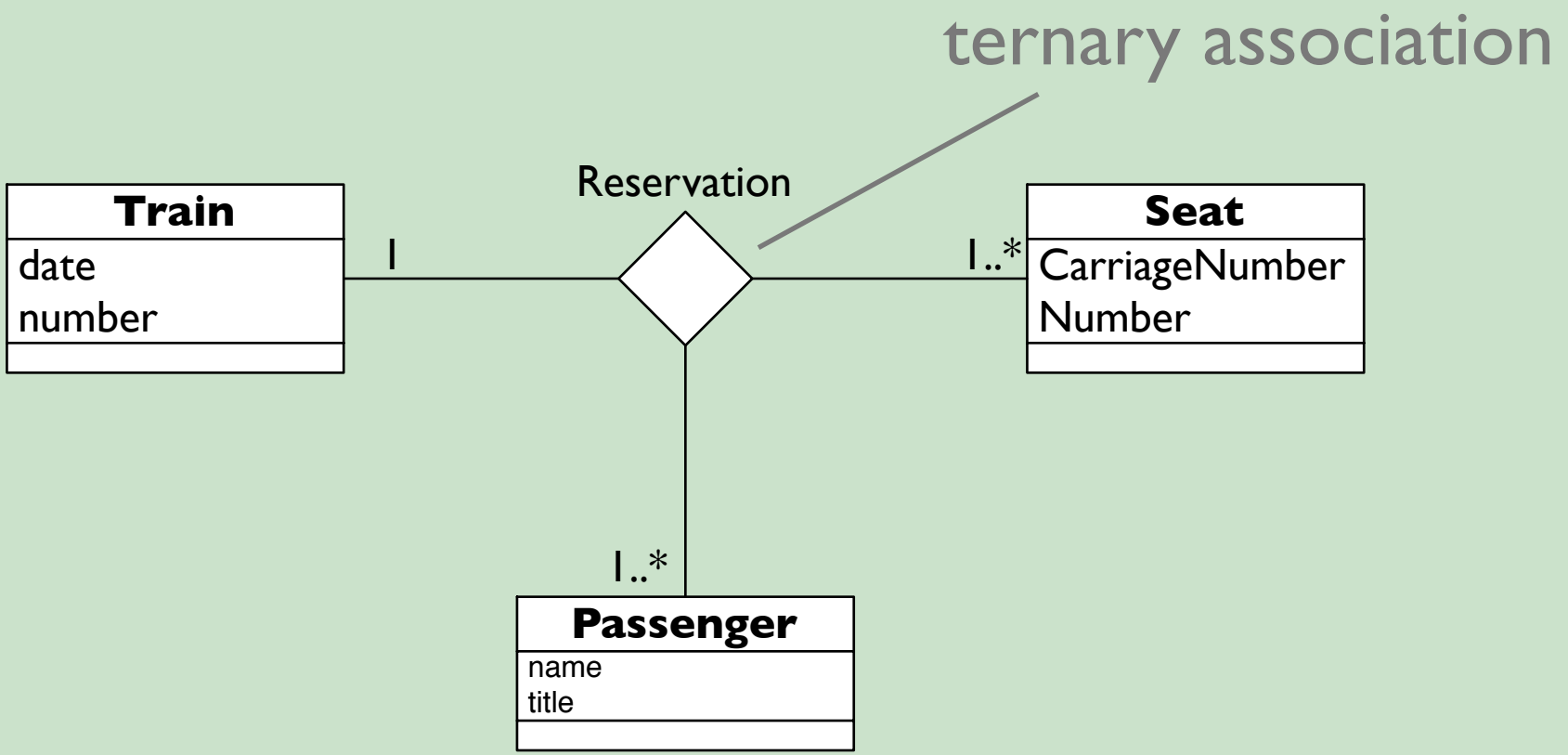
Example 8: association classes



Example 8

A reservation on a passanger train consists of a passanger (for whom the seat is reserved), a seat (which is being reserved), and a train (time of reservation). Besides such simple reservations, group reservations are also allowed. Who will actually occupy which seat is left over to the group.

Example 9: ternary associations

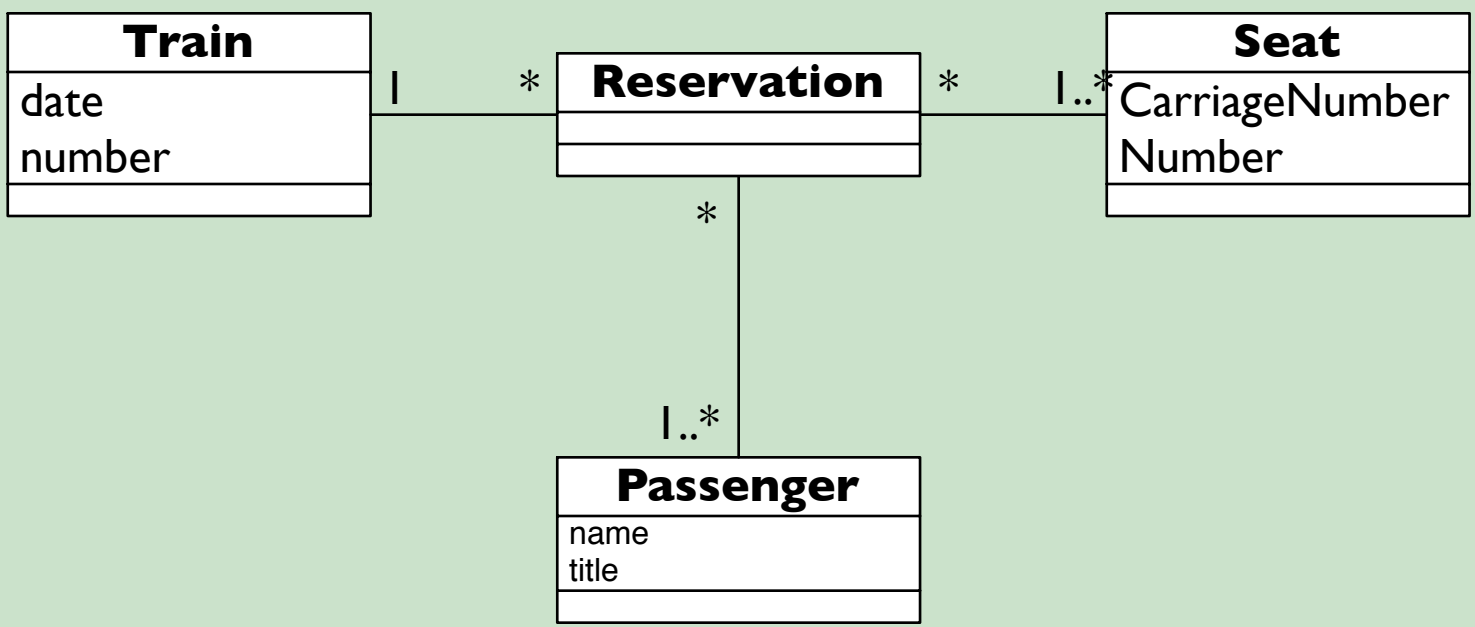


n-ary associations

Associations in general relate n classes

- Most of them are binary, but three (as in previous example) or more are possible

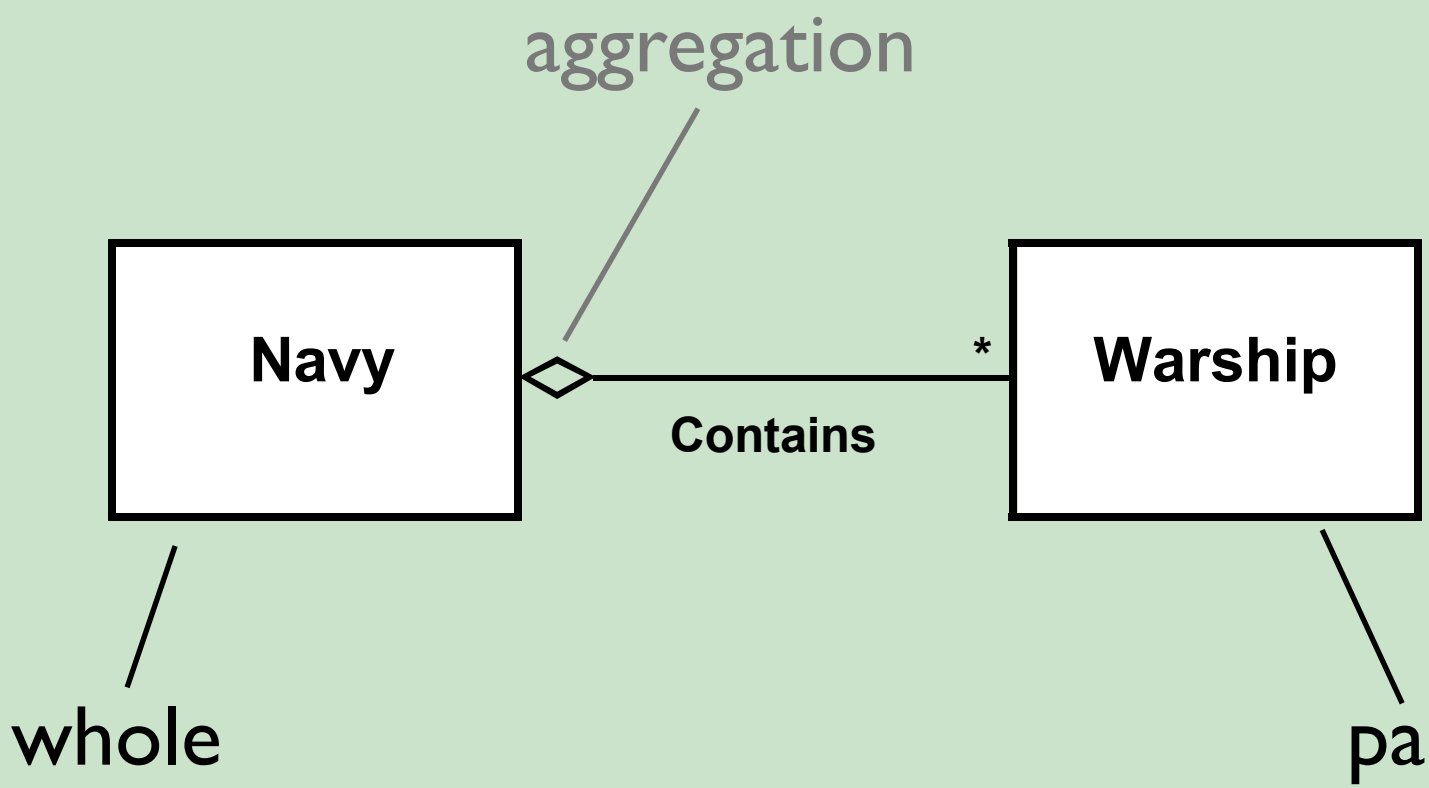
Can usually be transformed in binary associations:



Aggregation

A whole-part association (whole owns the part)

Describes different levels of abstraction

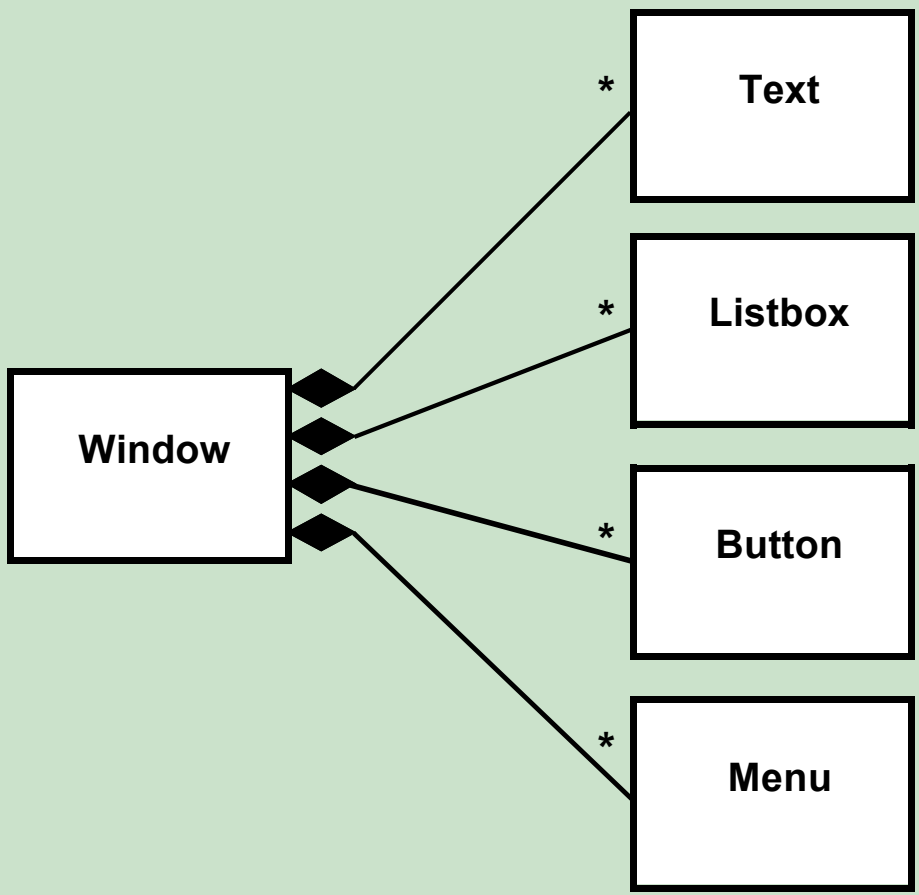


Note: Diamond can only be at one end!

Composition Aggregation`…

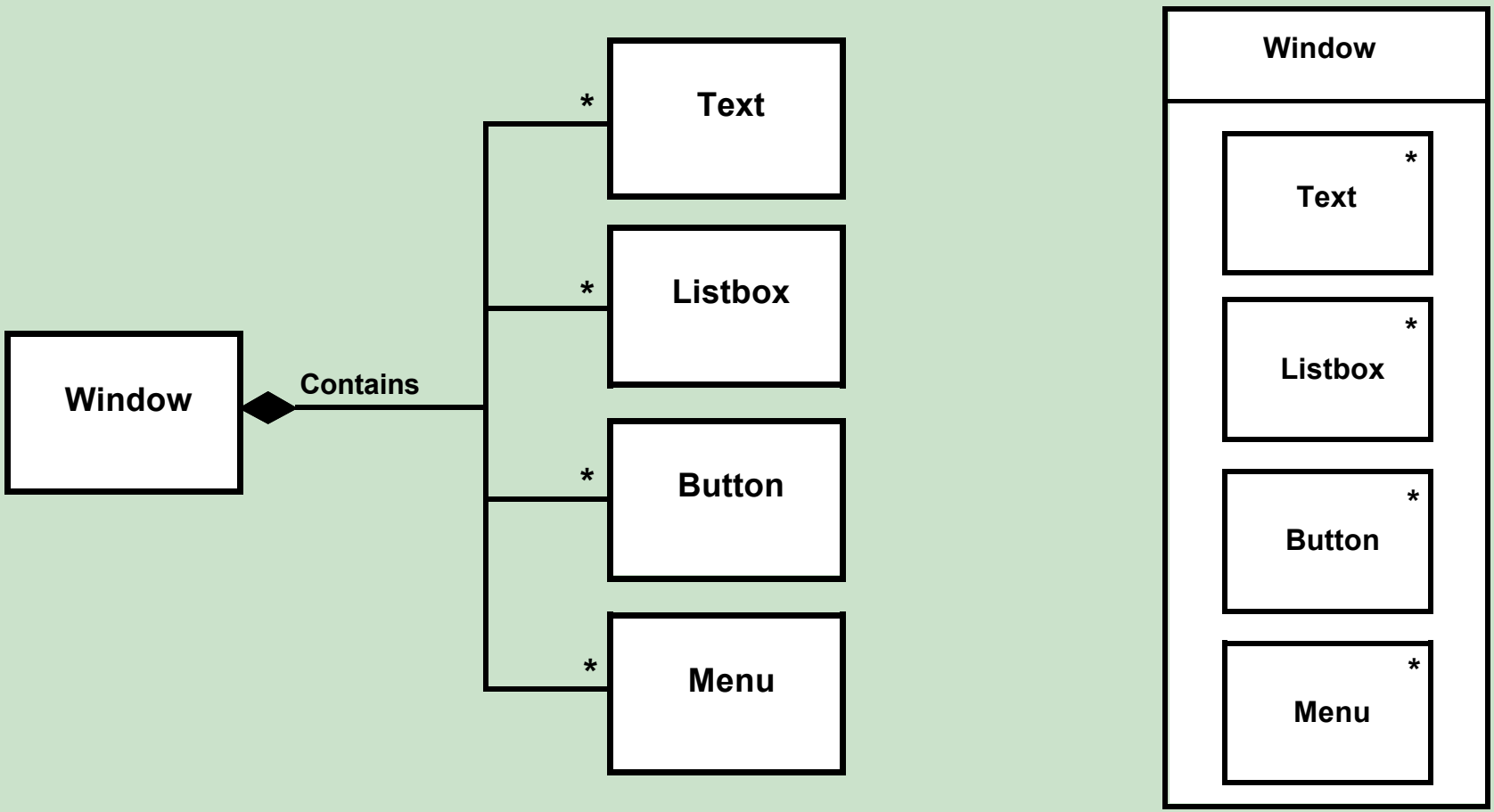
The parts can only exist if the whole exists

- they are destroyed with the whole



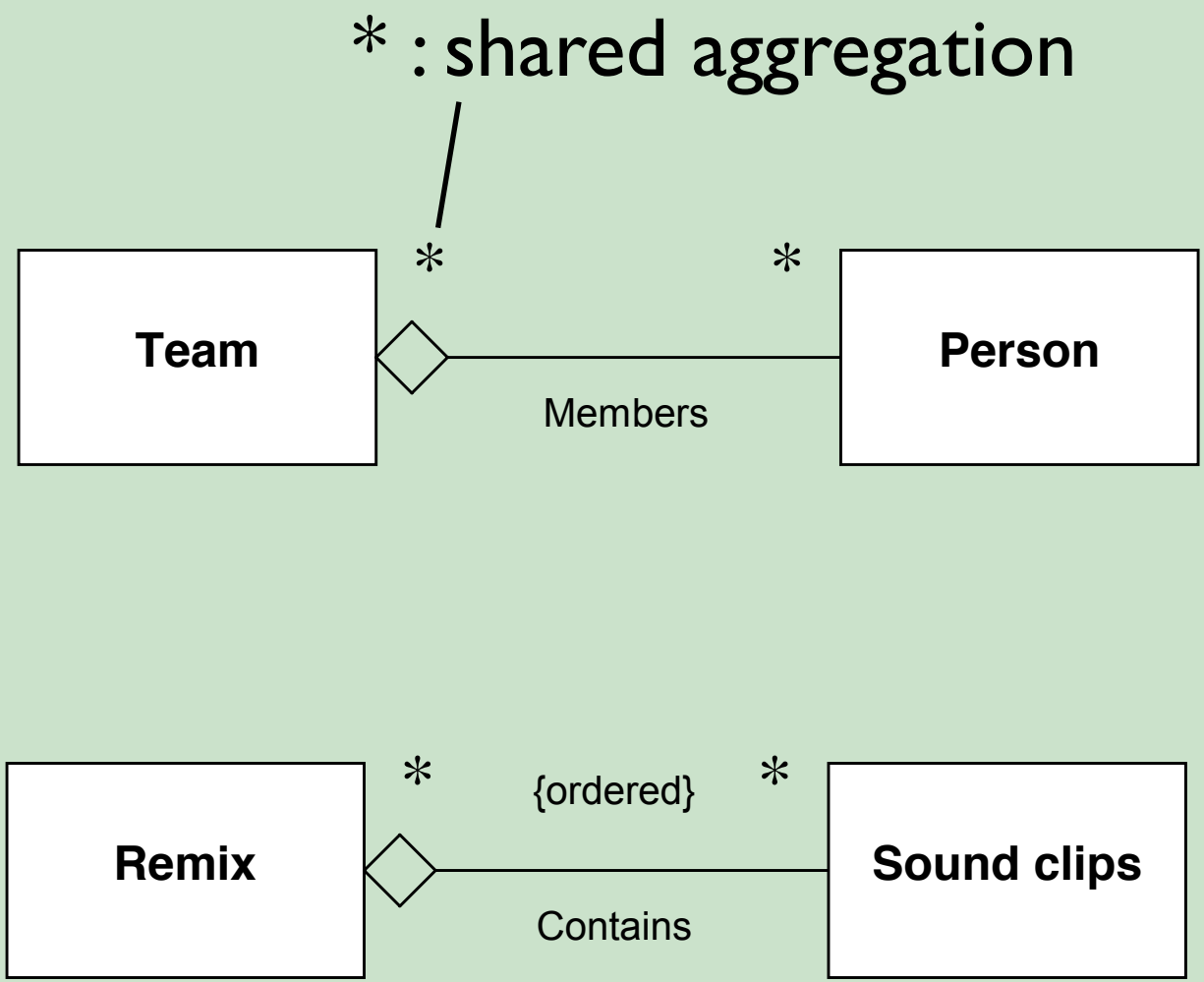
Composition Aggregation…

Other notations expressing the same:



Shared Aggregation

Part may be part of several wholes



Association or Aggregation?

The decision to use association or aggregation is a matter of judgment and is often arbitrary

E.g. What type of relationship should be used to model a car with its tires?

- If the application is a service center, the only reason you care about the tire is because it is part of the car, so use an aggregation
- If the application is a tire store, you will care about the tire independent of the car, then the relationship should be an association.

Recap: associations

Connect classes

- between 1, 2, 3, ... classes

Can contain multiplicities, directions, role names

Constraints can be used to express additional requirements (more on constraints later)

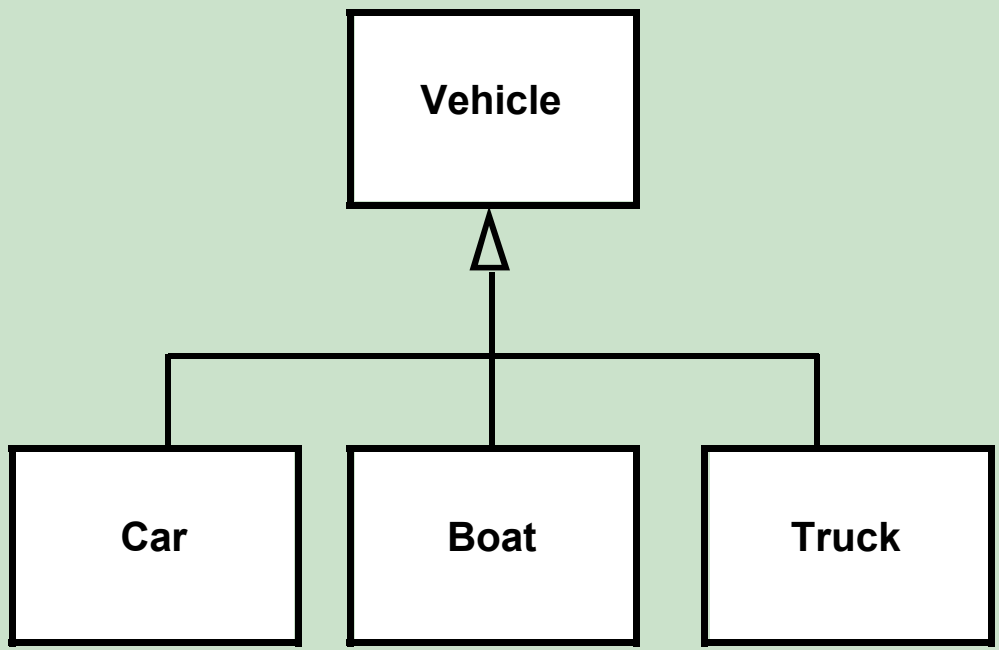
Association classes can be used for more difficult associations that have behaviour from their own

Aggregation: part-whole composition

Generalization…

Inheritance, 'is-a' relationship

The more specific may be used where the more general is allowed



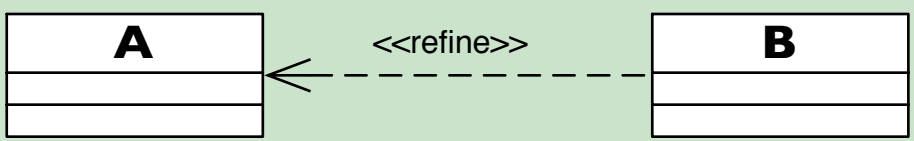
Remember : Using Inheritance for code reuse, or just because it looks nice is a dangerous practice !

Refinement

"A relationship that represents a fuller specification of something that has already been specified at a certain level of detail."

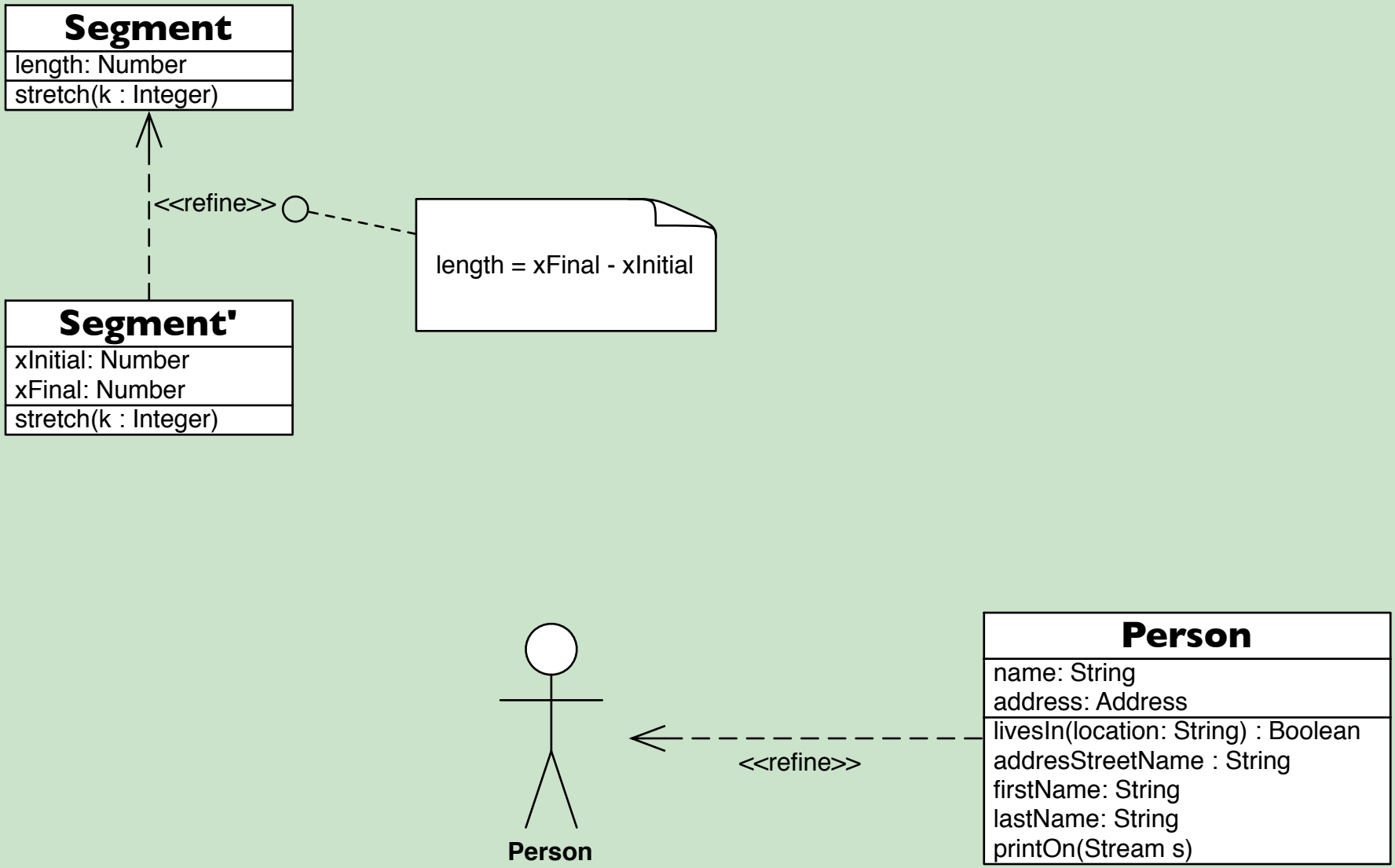
Usefull for modelling

- a relation between the analysis version and the design version, or between a clean implementation and an optimized but potentially difficult variation



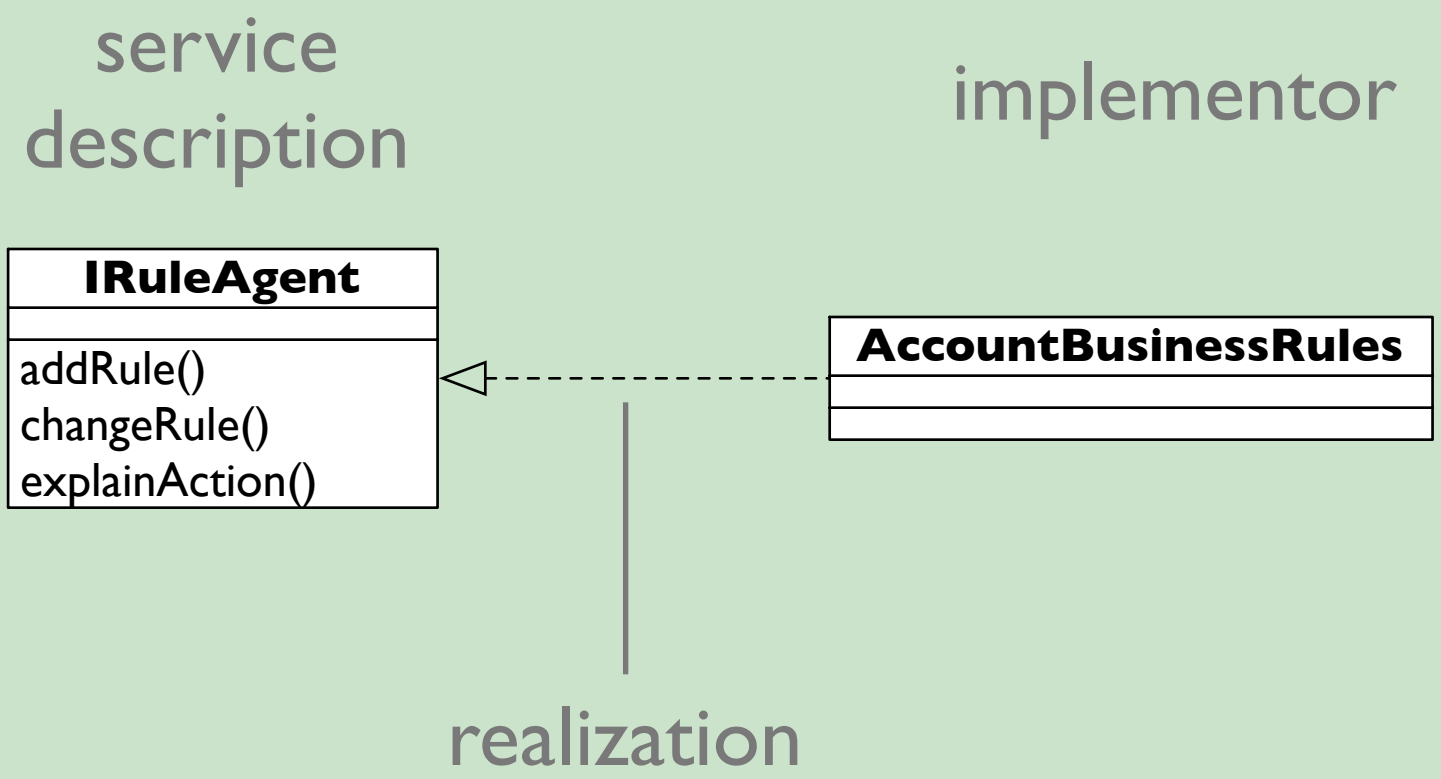
B refines A

Class refinement examples



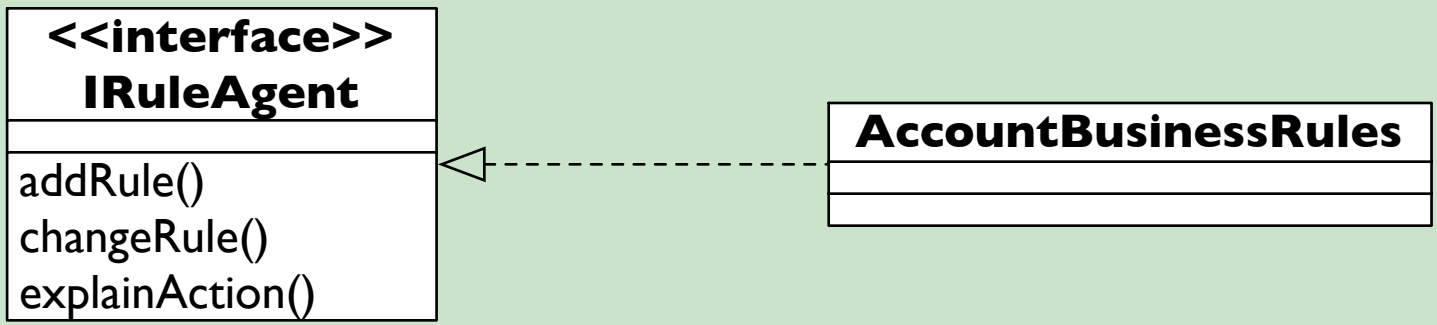
Realization

"A semantic relationship between classifiers in which one classifier specifies a contract that another classifier guarantees to carry out"

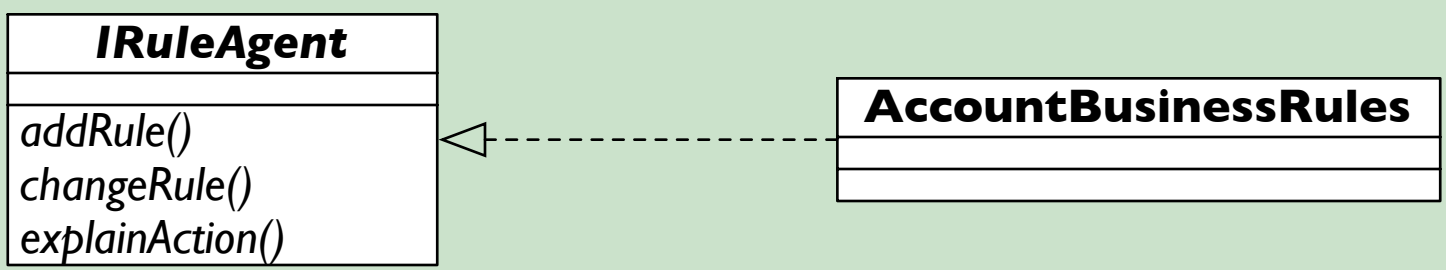


Implementing realization

In Java: implemented with interfaces



In C++: implemented with fully abstract classes



Take care

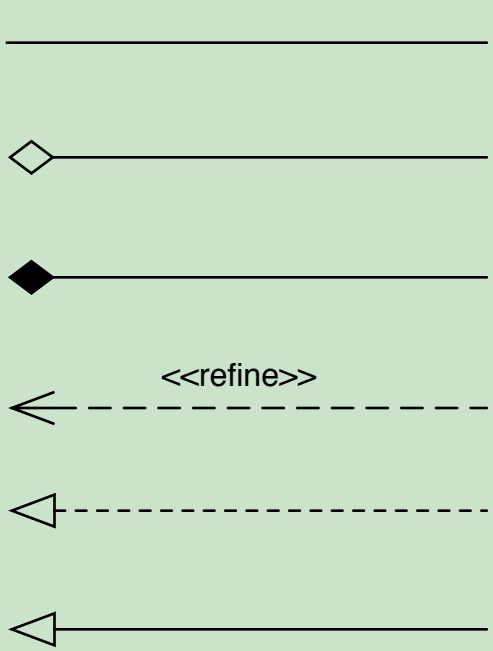
In older versions of UML, the notation of refinement was the notation which is now used for realization

Wrap-up

Class Diagrams model statical elements (classes and their relationships)

For use by architects and developers

Relations between classes:



association

aggregation

composition aggregation

refinement

realization

inheritance

Wrap-up

UML models all kinds of *systems*.

It is a modelling language, not a process!

It consists of

- Views (5)
- Diagrams (9)

It is based on a meta model and hence the different diagrams share similar elements

- and they are extensible