

Low level Models – data



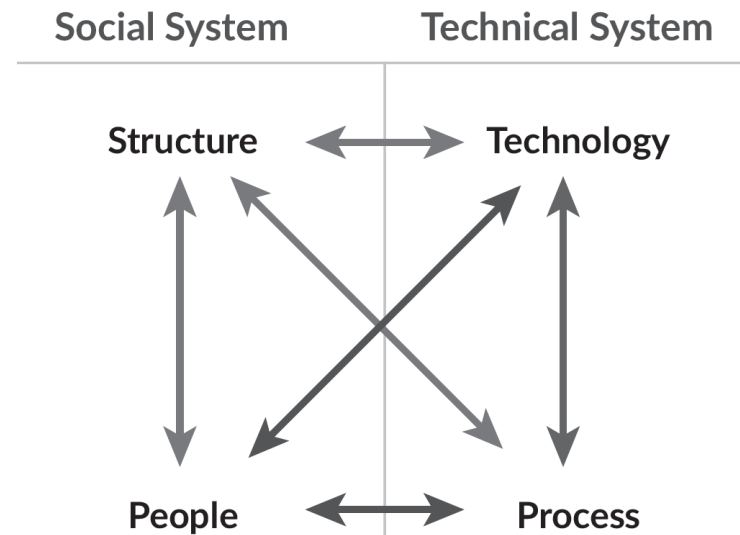
SoftEng
<http://softeng.polito.it>

© Maurizio Morisio, Marco Torchiano, 2012



Low level models

- Structure
- People
- Process
 - ◆ Process
 - ◆ Data
- Technology



Data modelling

- Goal: define information processed / exchanged by IS
- YES conceptual model
- NO database schema (RDB tables)
- NO data structures (programming language)

Data modelling

- Entity relationship
- UML class diagram



UML

- Unified Modeling Language
- Standardized by OMG
- Several diagrams
 - ◆ Class diagrams
 - ◆ Activity diagrams
 - ◆ Use Case diagrams
 - ◆ (Sequence diagrams)
 - ◆ (Statecharts)



Conceptual modeling

Process modeling

Functional modeling

Conceptual Modeling

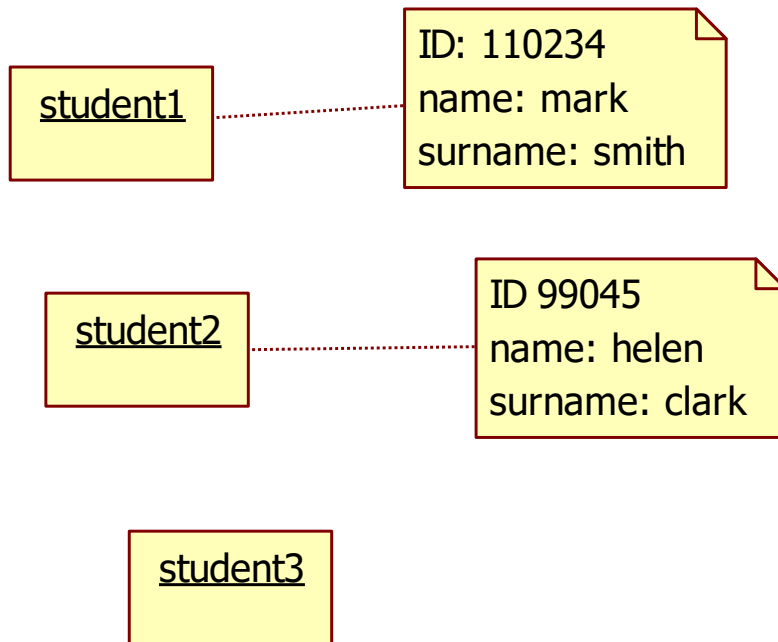
CLASS DIAGRAM

Goal: conceptual diagram

- Capture
 - ◆ Main (abstract) concepts
 - ◆ Characteristics of the concepts
 - Attributes associated to the concepts
 - ◆ Relationships between concepts

Object

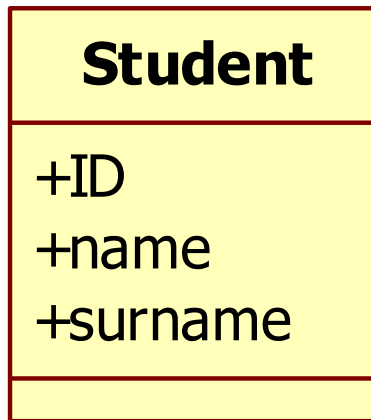
- Model of item (physical or within the software system)
 - ♦ ex.: a student, an exam, a window
- Characterized by
 - ♦ identity
 - ♦ attributes (or properties)
 - ♦ operations it can perform (behavior)
 - ♦ messages it can receive



Class

- They describe set of objects
 - ◆ Common properties (attributes, behaviours)
 - ◆ Autonomous existence
 - ◆ E.g. facts, things, people
- An instance of a class is an object of the type that the class represents.

Class – Examples



Attribute

- Elementary property of classes
 - ♦ Name
 - ♦ Type
- An attribute associates to each object a value of the corresponding type
 - ♦ Name: String
 - ♦ ID: Numeric
 - ♦ Salary: Currency

Usage of class diagram

- Model of concepts (glossary)
- Model of system (hw + sw) == system design
- Model of software classes (software design)

- Class in conceptual model (UML class diagram)
 - ◆ Ex Employee class
- Corresponding entities in software application
 - ◆ Data layer: Employee table in RDB
 - ◆ Business logic layer: Employee class in Java / C++, C#
 - ◆ Presentation layer: form to enter employee data, form to show employee data, and more

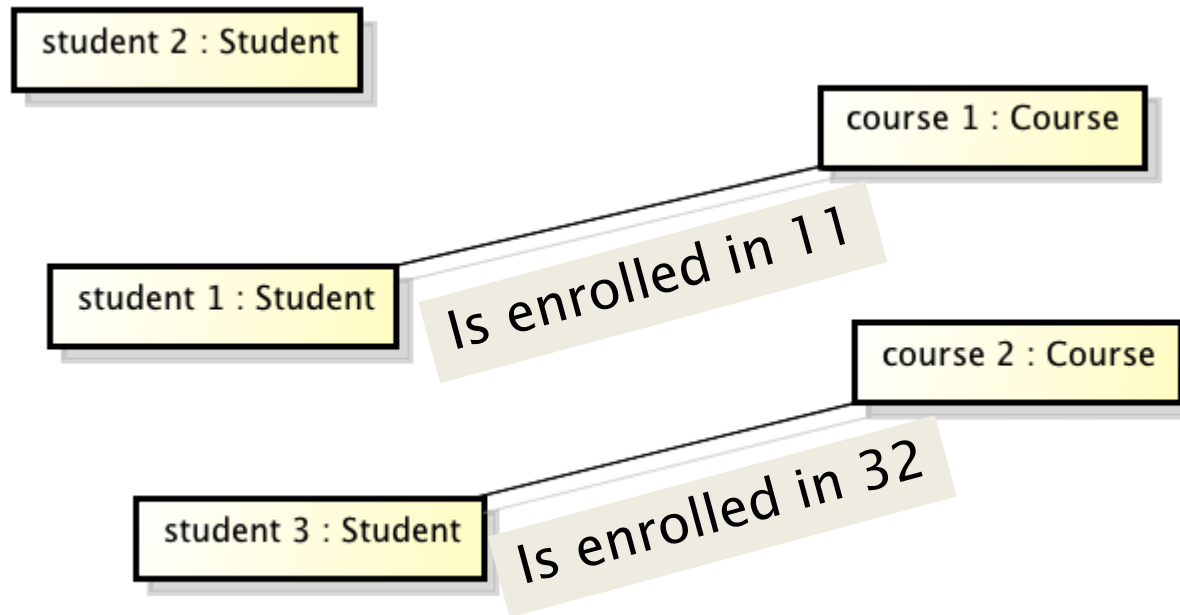
-
- Before doing a class diagram,
DECIDE WHAT YOU WANT TO MODEL

Classes in conceptual diagram

- Where to look for
 - ◆ Physical entities: Person, Car,
 - ◆ Roles: Employee, Director, Doctor,
 - ◆ Social / legal / organizational entities: University, Company
 - ◆ Events: Sale, Order, Request, Claim, Call
 - ◆ Time intervals: Car rental, Booking, Course, Meeting
 - ◆ Geographical entities: City, Road, Nation
 - ◆ Reports, summaries: weather report, bank account statement

Link

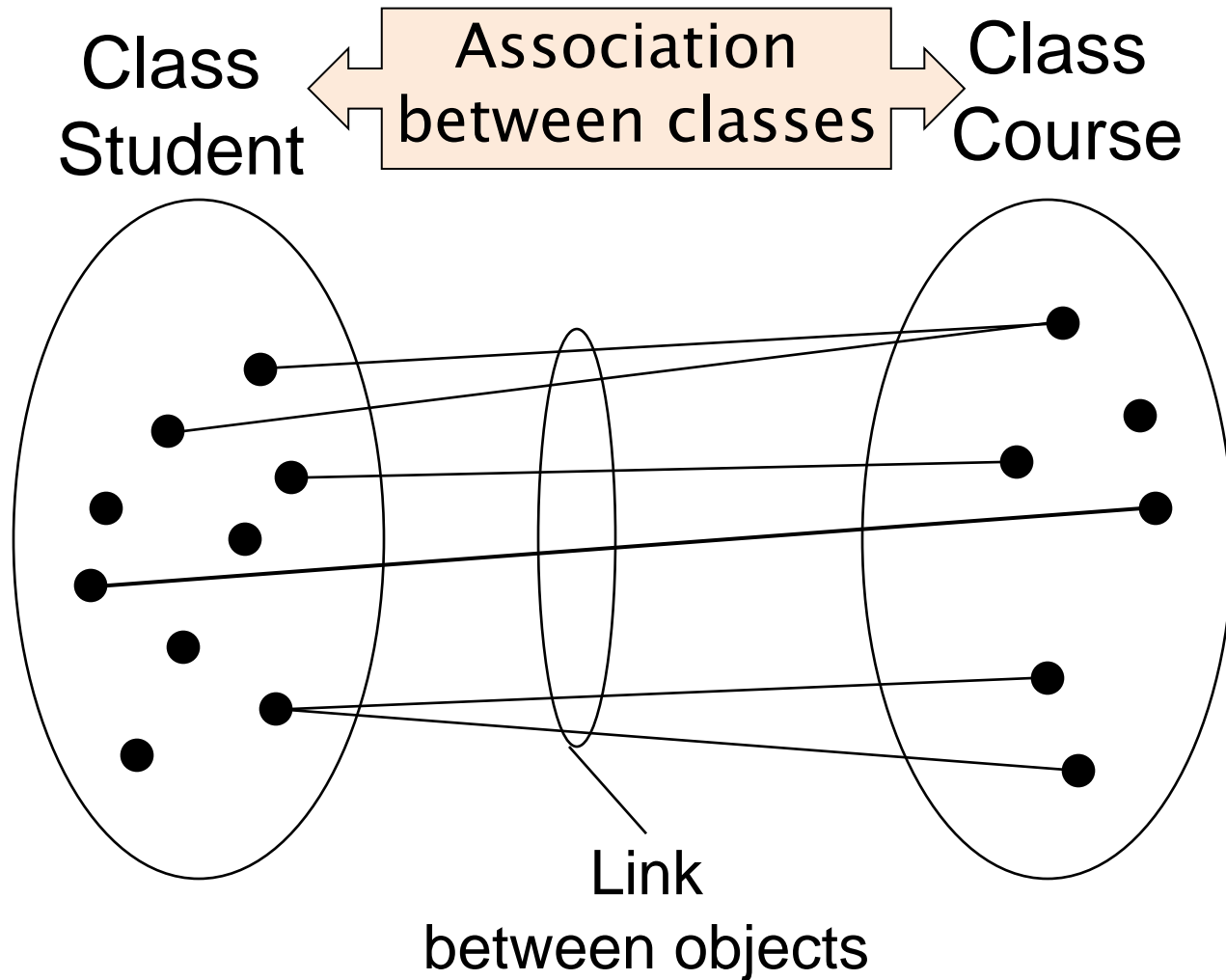
- Model of property between objects
 - ♦ A property that cannot be represented on one object only



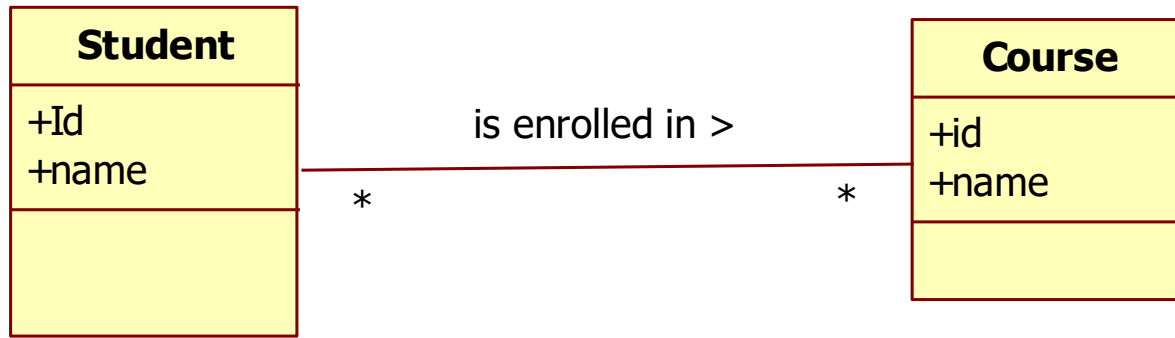
Association

- Represent set of links between objects of different classes.
 {Is enrolled in 11,
 Is enrolled in 32}
- Or pairs of objects (one per class):
 {student1 – course1,
 student3 – course2 }

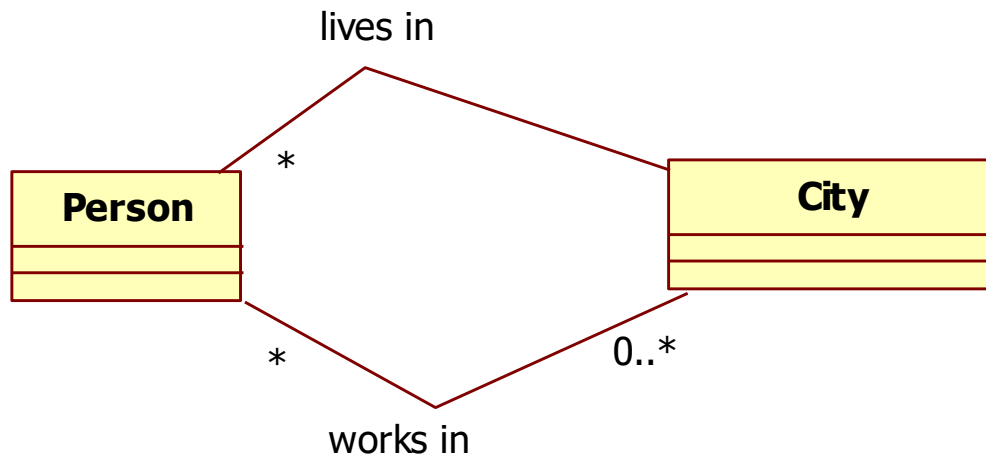
Associations



Association – Examples

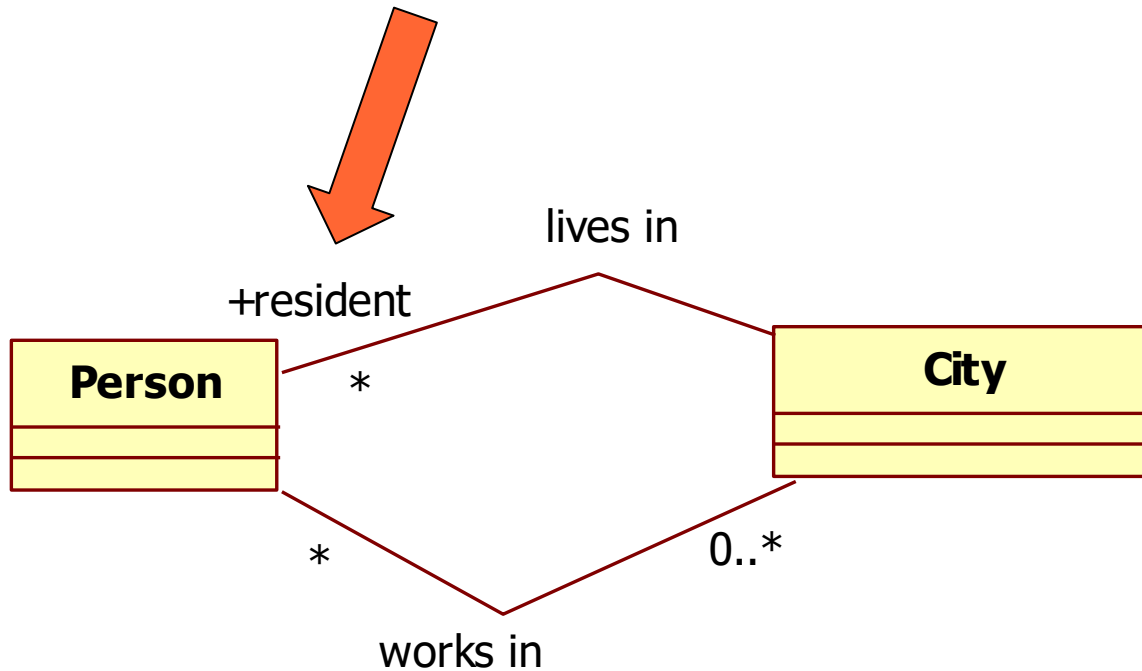


Association



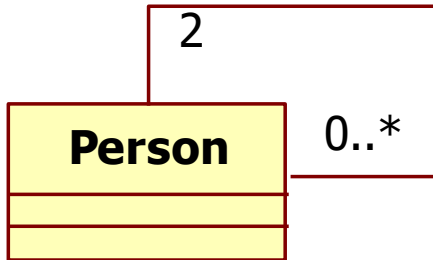
Role in association

- Name of one end of association

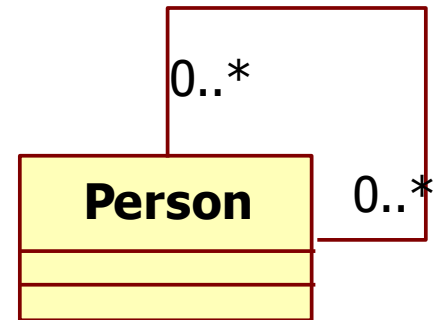


Recursive associations

is parent of

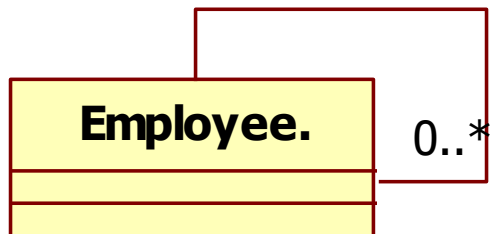


is friend of

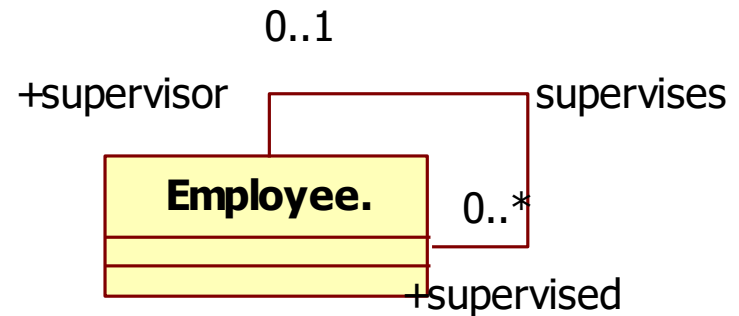
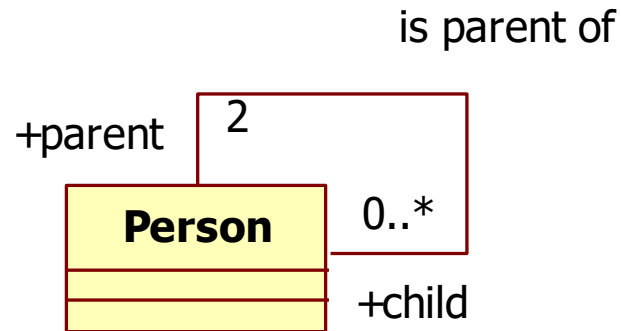


0..1

supervises



Recursive associations + roles



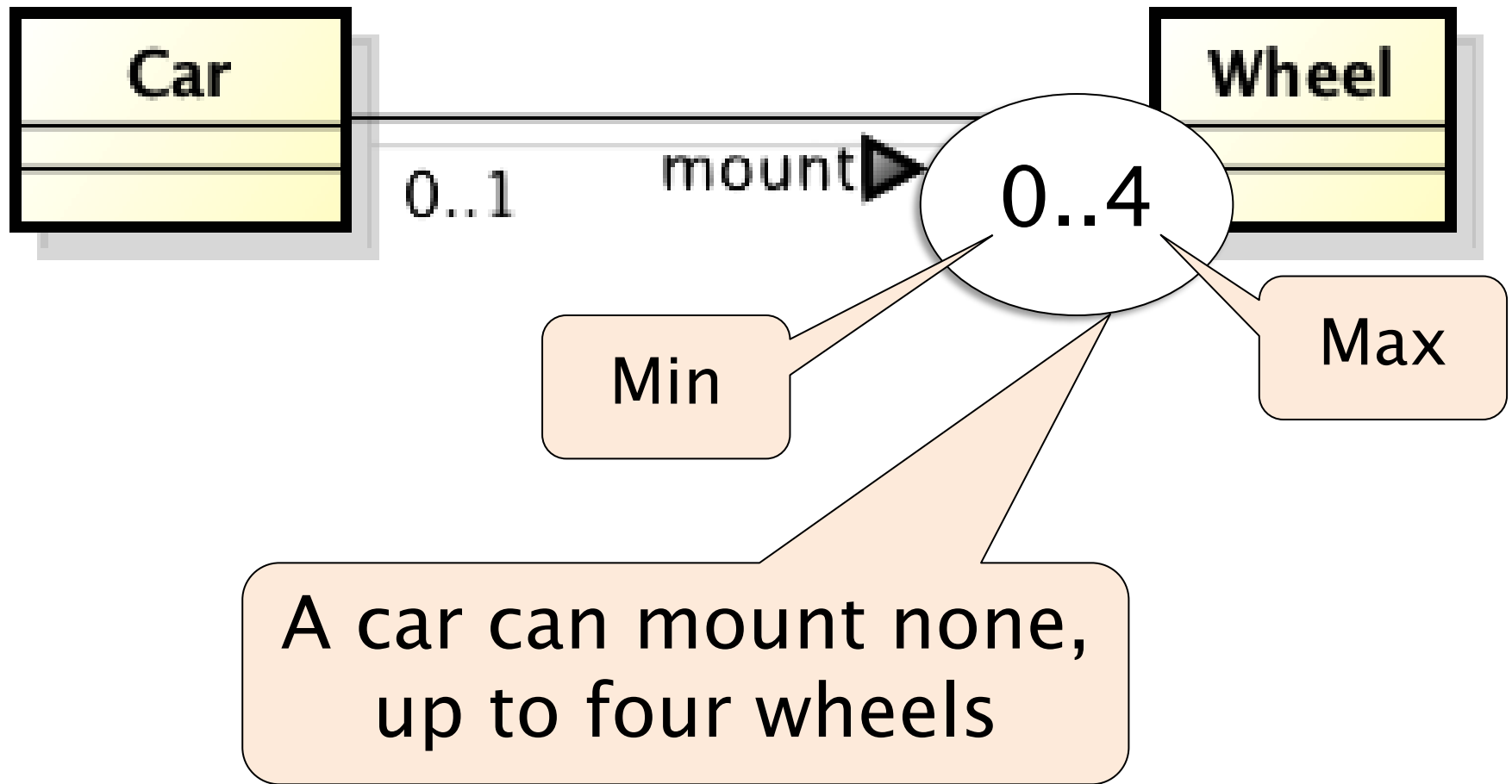
Style suggestions

- Class names
 - ◆ Singular noun
- Association name
 - ◆ Verb
- Attributes
 - ◆ Type of attribute not needed in conceptual model

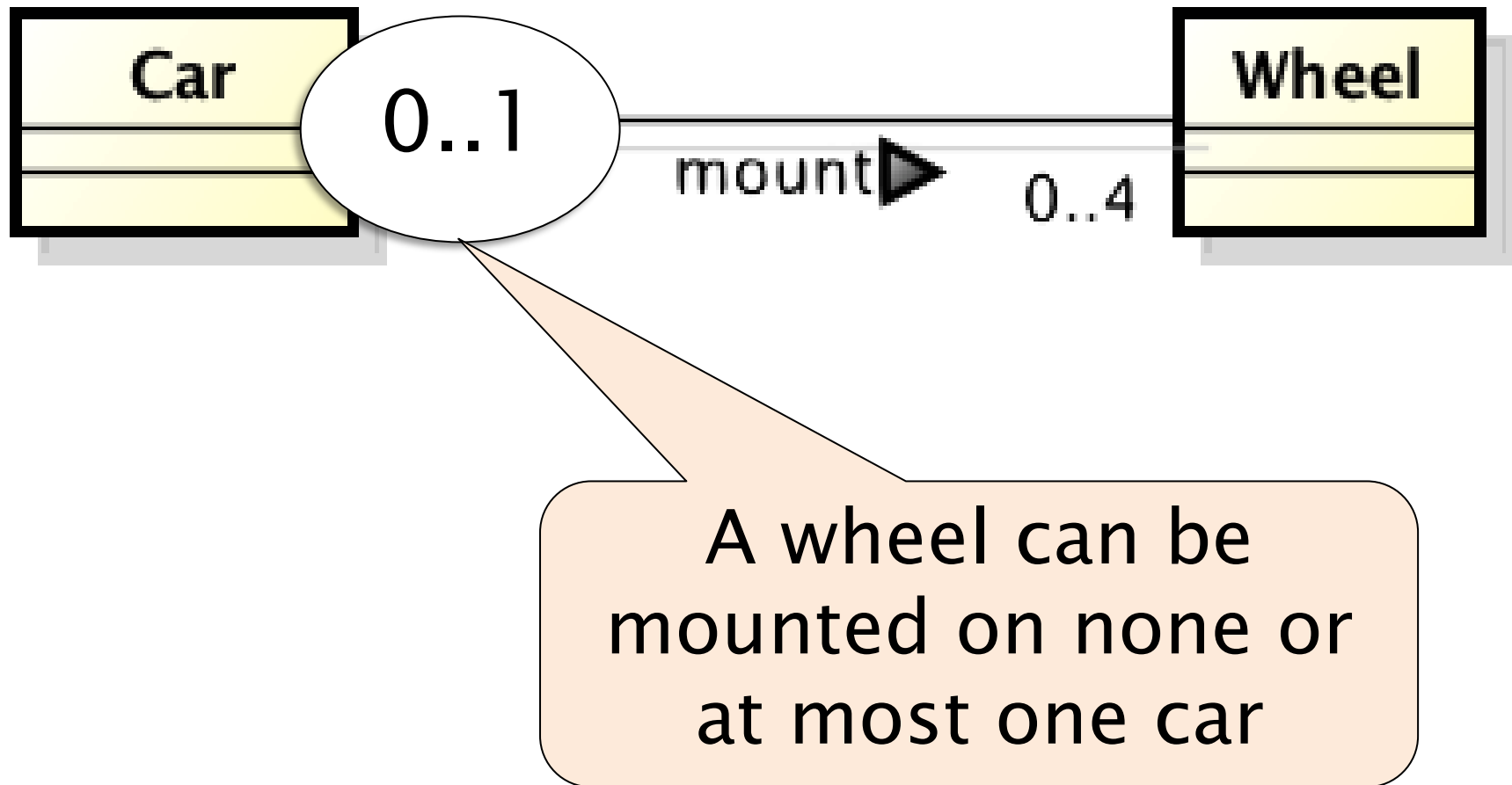
Multiplicity

- Describe the maximum and minimum number of links in which an object of a class can participate
- Should be specified for each class participating in an association

Multiplicity – Example



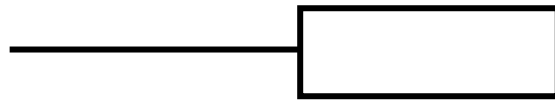
Multiplicity – Example



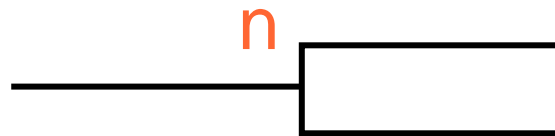
Multiplicity

- Typically, only three values are used: **0**, **1** and the symbol ***** (many)
- Minimum: 0 or 1
 - ♦ 0 means the participation is *optional*,
 - ♦ 1 means the participation is *mandatory*;
- Maximum: 1 or *
 - ♦ 1: each object is involved in at most one link
 - ♦ *: each object is involved in many links

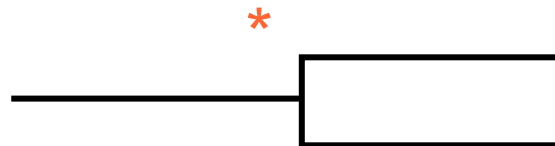
Multiplicity



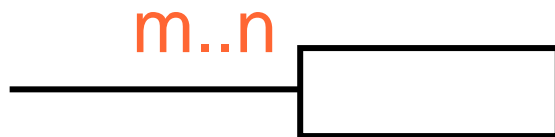
Exactly 1



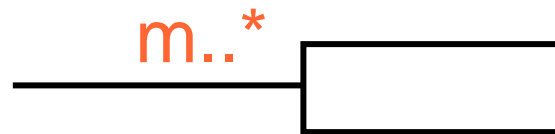
Exactly n



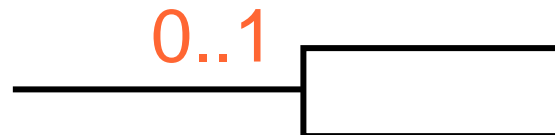
Zero or more



Between m and n (m,n included)

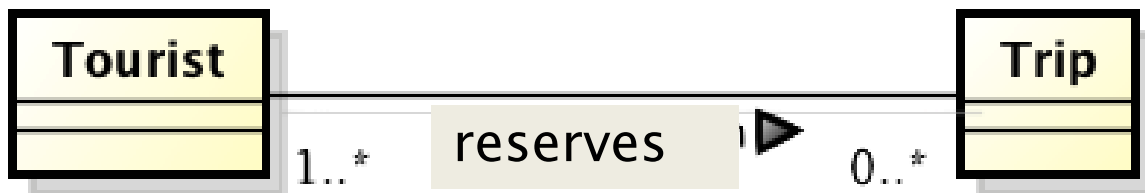
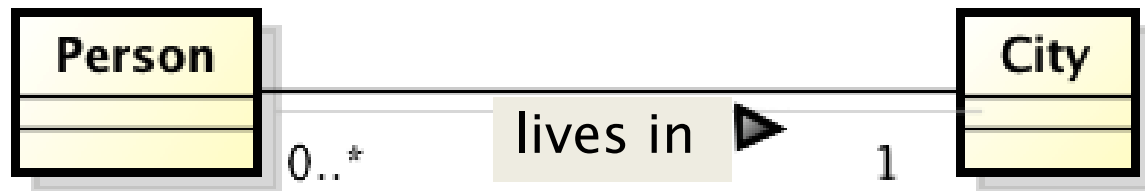


From m up



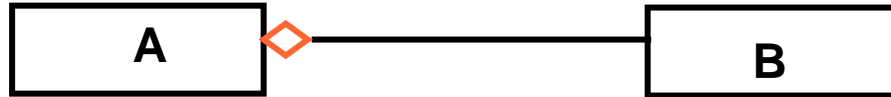
Zero or one (optional)

Multiplicity

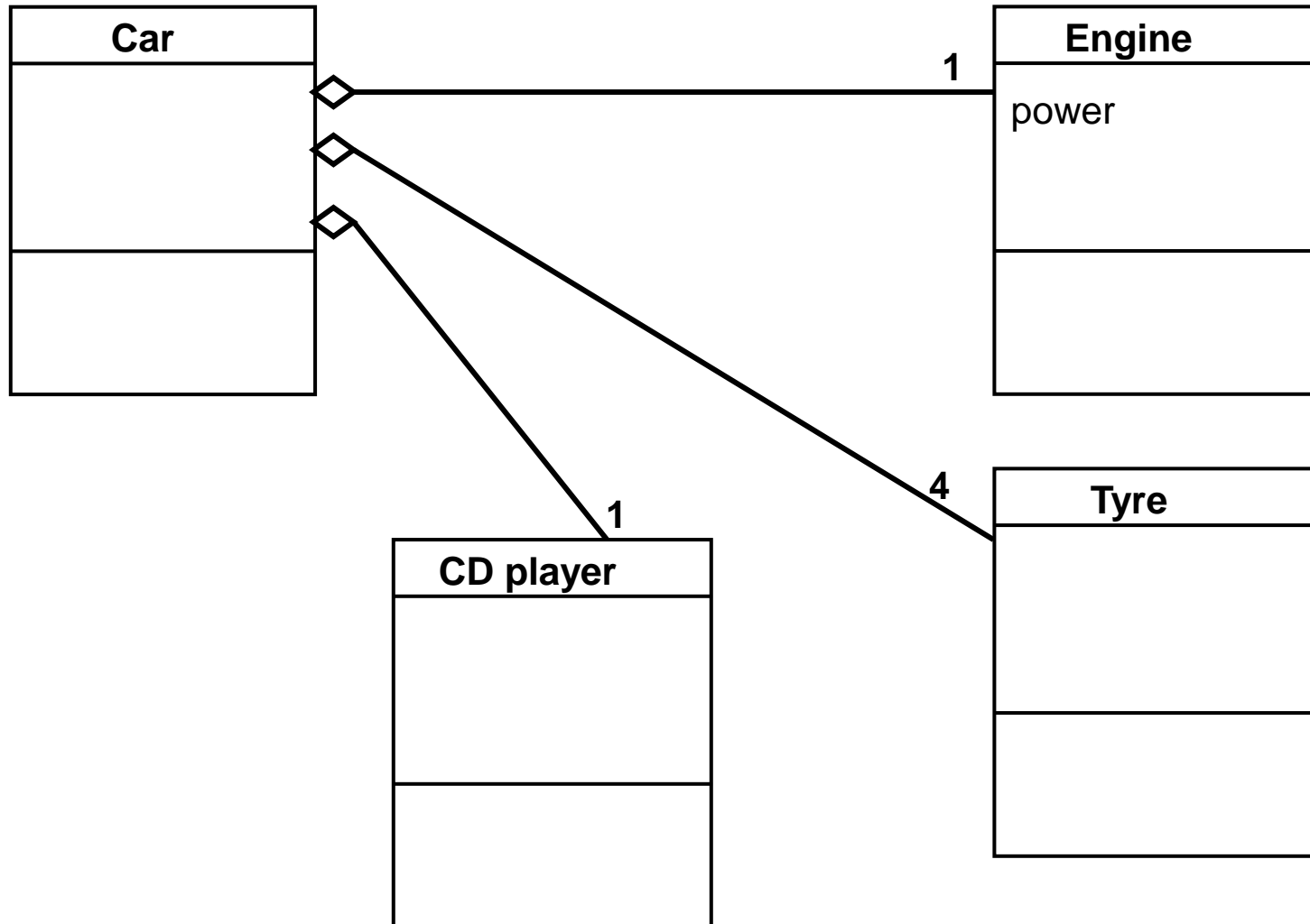


Aggregation

- Association with specific meaning
- *B is-part-of A* means that objects described by class B are part of (are components of) objects described by A
- Has
- Part-of

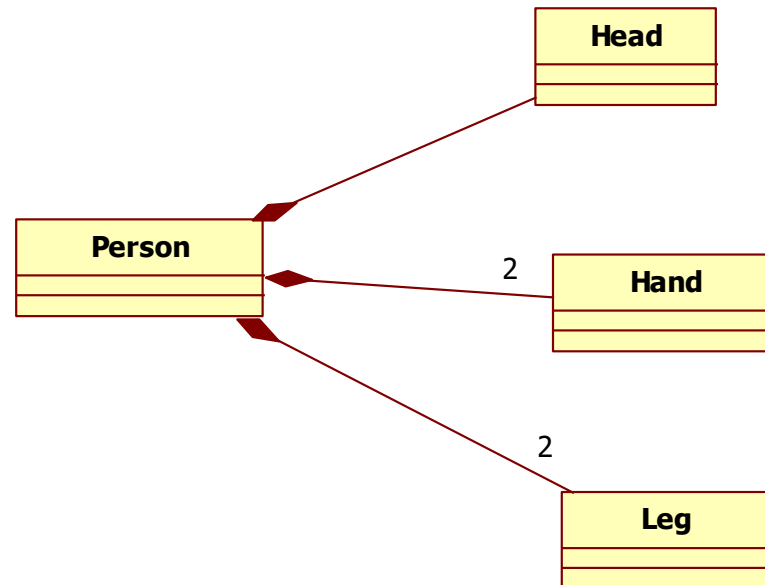


Example



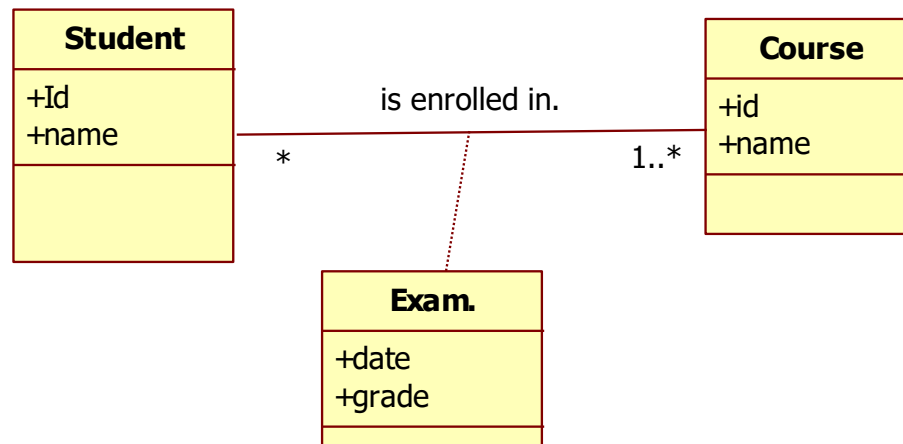
Composition

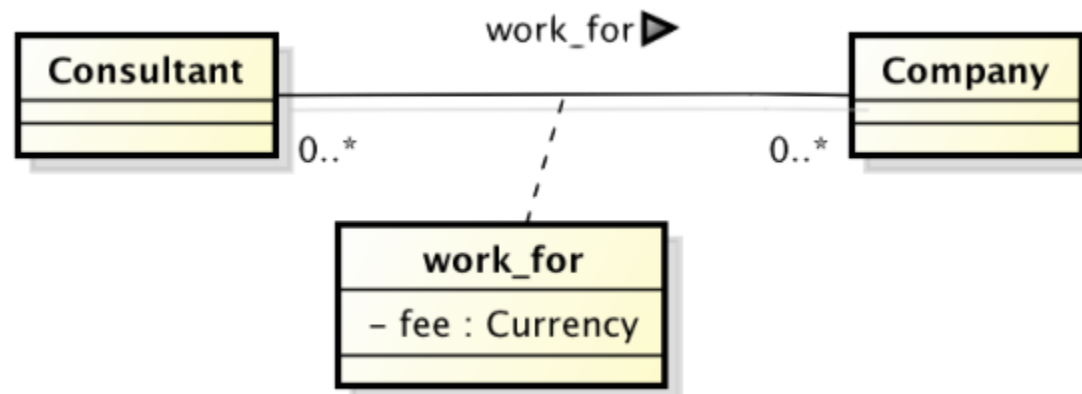
- An aggregation where the link part / whole is more strict: lifecycle of both classes is the same
 - ◆ if object Person disappears, so the corresponding 2 objects Leg, Hand



Association Class

- The association class allows to attach attributes to the association
- A link between two object includes
 - ◆ The two linked objects
 - ◆ The attributes of the link





Consultant	Company	fee
------------	---------	-----

consultant1	company2	300
-------------	----------	-----

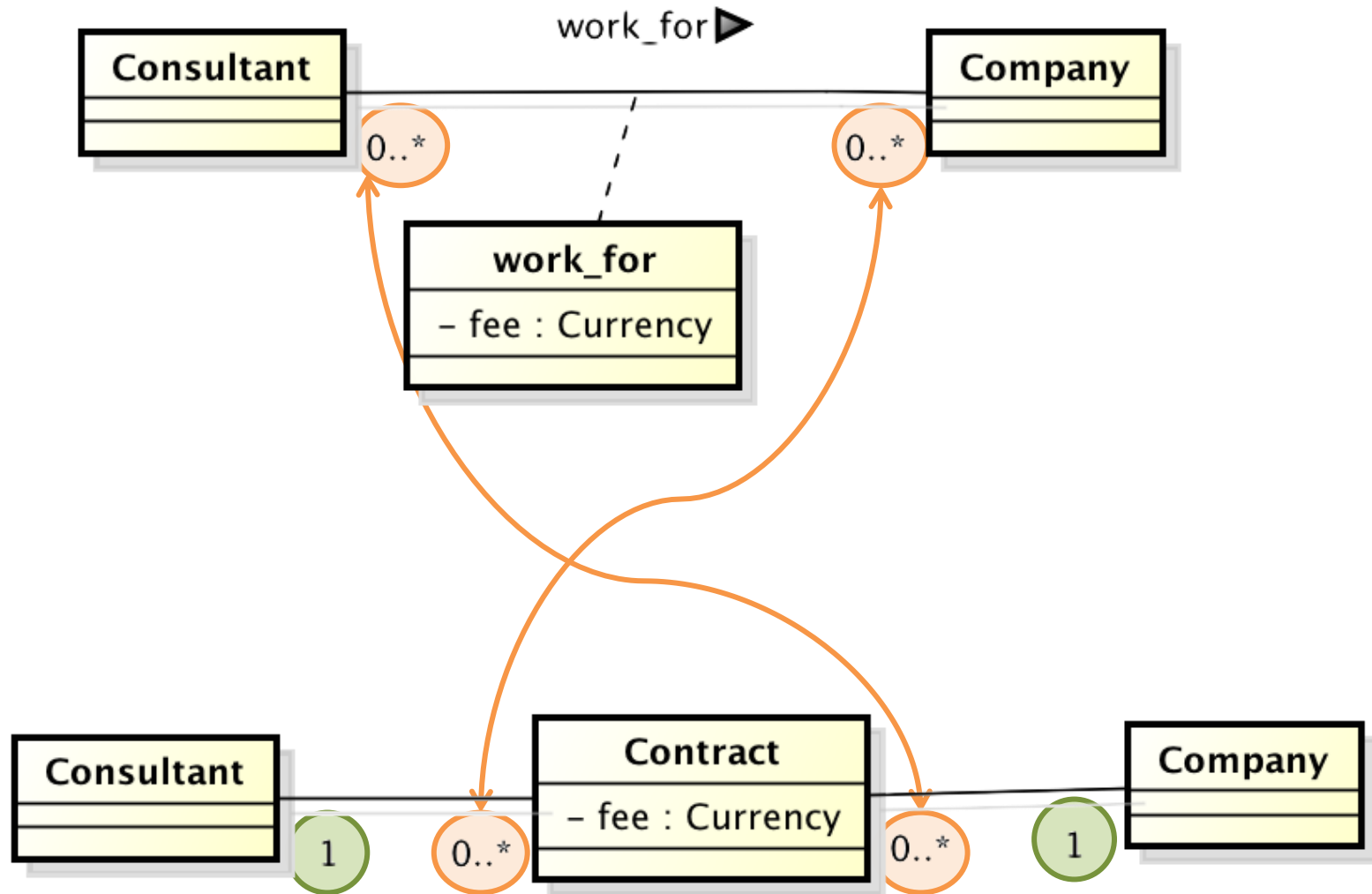
consultant1	company3	200
-------------	----------	-----

consultant1	company3	250
------------------------	---------------------	----------------

consultant2	company2	100
-------------	----------	-----

consultant3	company2	300
-------------	----------	-----

Instead of Association class



Consultant	Company	Contract
------------	---------	----------

consultant1	company2	c1 300
-------------	----------	--------

consultant1	company3	c2 200
-------------	----------	--------

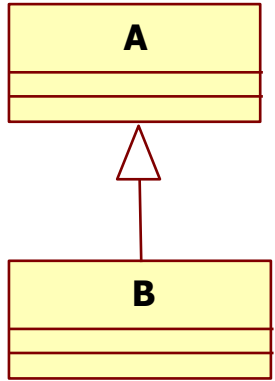
consultant1	company3	c3 250
-------------	----------	--------

consultant2	company2	c4 100
-------------	----------	--------

consultant3	company2	c5 300
-------------	----------	--------

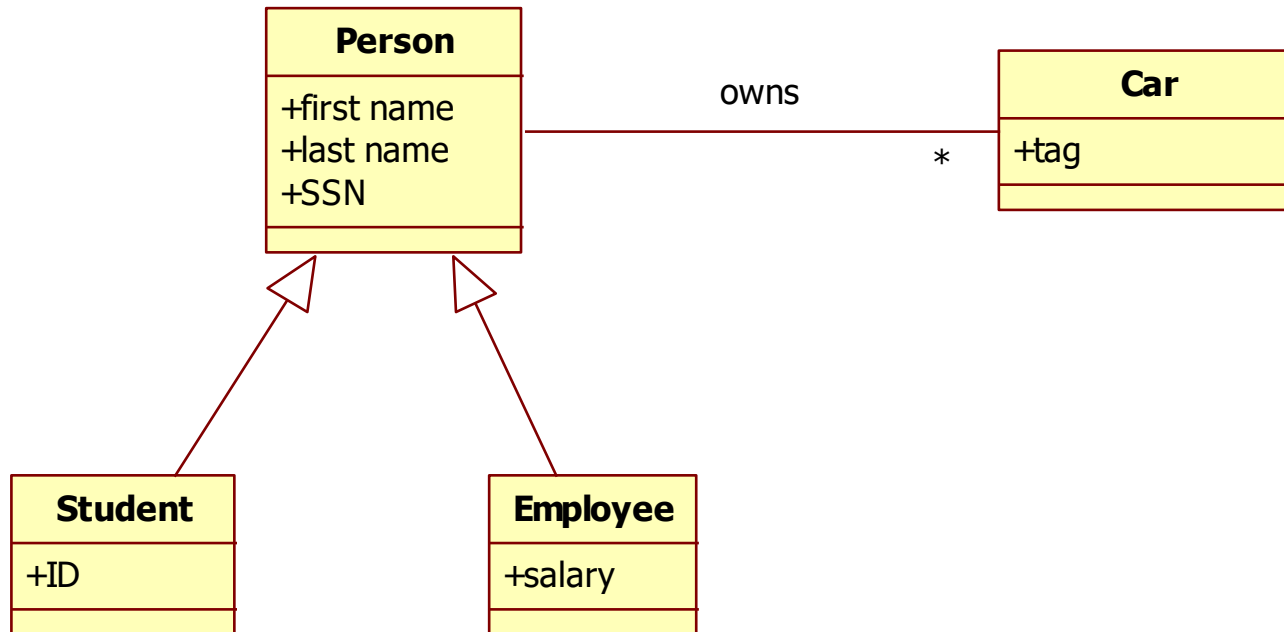
-
- The two options are equivalent, except
 - Intermediate class:
 - ◆ More than one value for a link
 - Association class:
 - ◆ Only one value for a link

Specialization / Generalization



- *B specializes A* means that objects described by B have the same properties of objects described by A
- Objects described by B may have additional properties
- B is a special case of A
- A is a generalization of B (and possibly of other classes)

Generalization



-
- Specialization can be used only if it is possible to state
 - ◆ B is-a A
 - Employee is-a Person – yes
 - Student is-a Person – yes
 - Head is-a Person – no
 - Person has-a Head – yes

Inheritance terminology

- Class one above
 - ◆ Parent class
- Class one below
 - ◆ Child class
- Class one or more above
 - ◆ Superclass, Ancestor class, Base class
- Class one or more below
 - ◆ Subclass, Descendent class, Derived class

Effects of inheritance

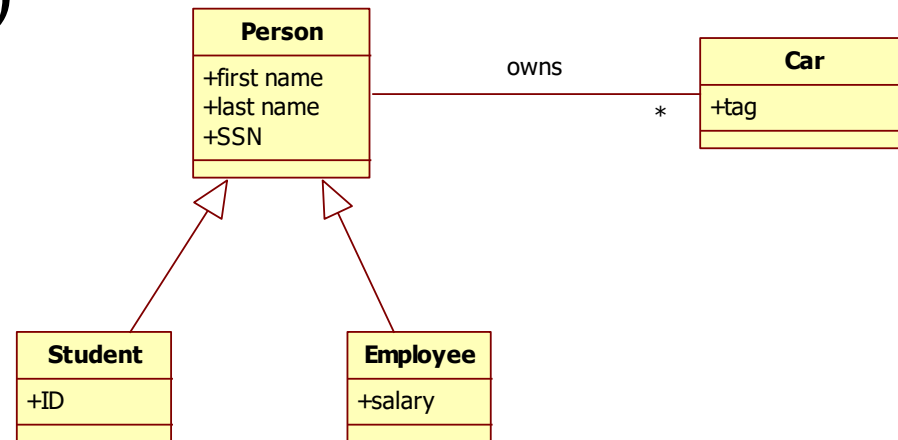
- The subclass inherits
 - ◆ All attributes
 - ◆ All associations
- Of all ancestors

- Employee has properties

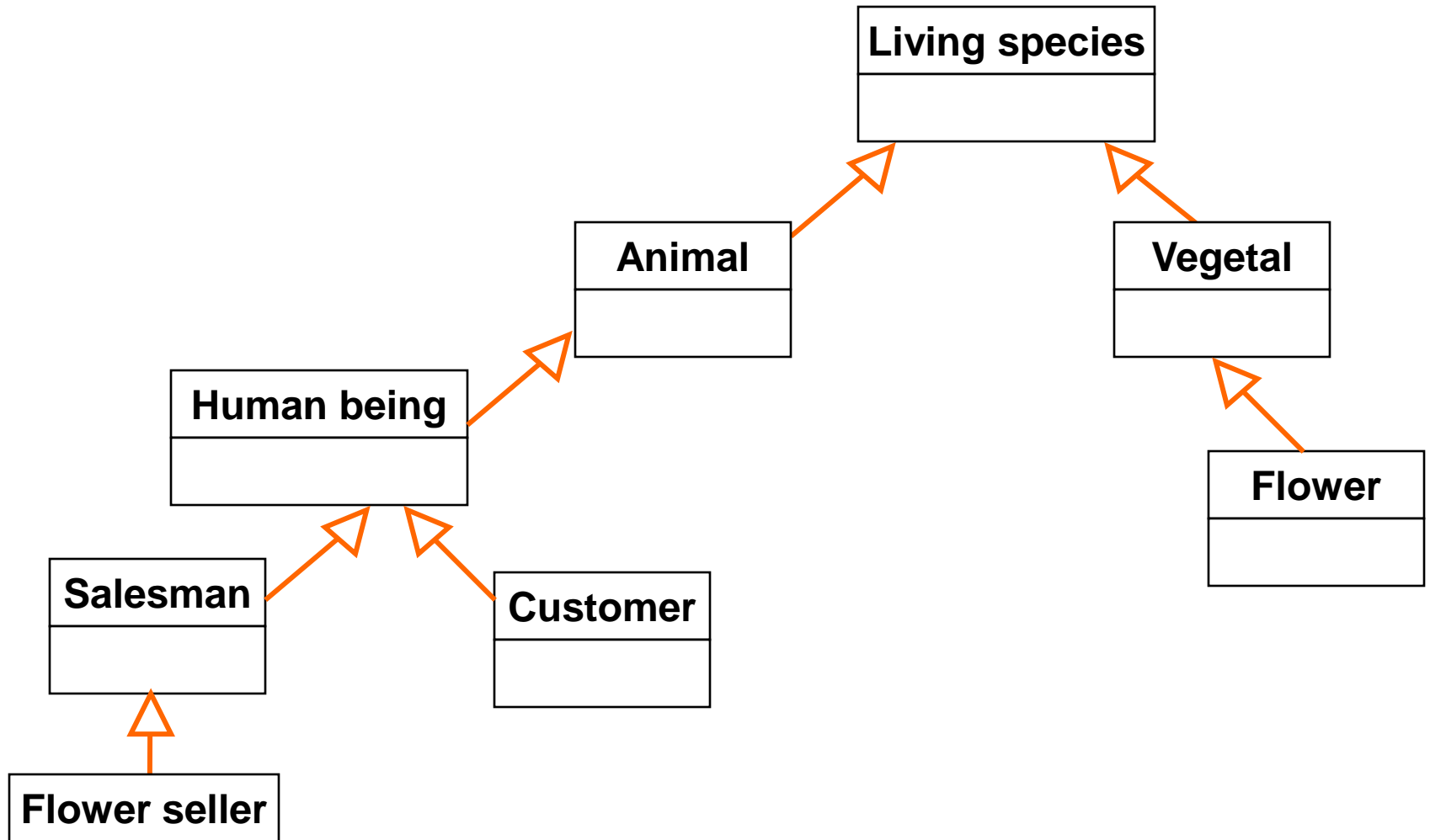
- ◆ First name (inherited)
- ◆ Last name (inherited)
- ◆ SSN (inherited)
- ◆ Salary

- Employee

- ◆ Owns Car (inherited)



Example of inheritance tree



DOs in Class Diagram

- Decide goal of model
 - ◆ In context of this course, conceptual model

Dos – consider:

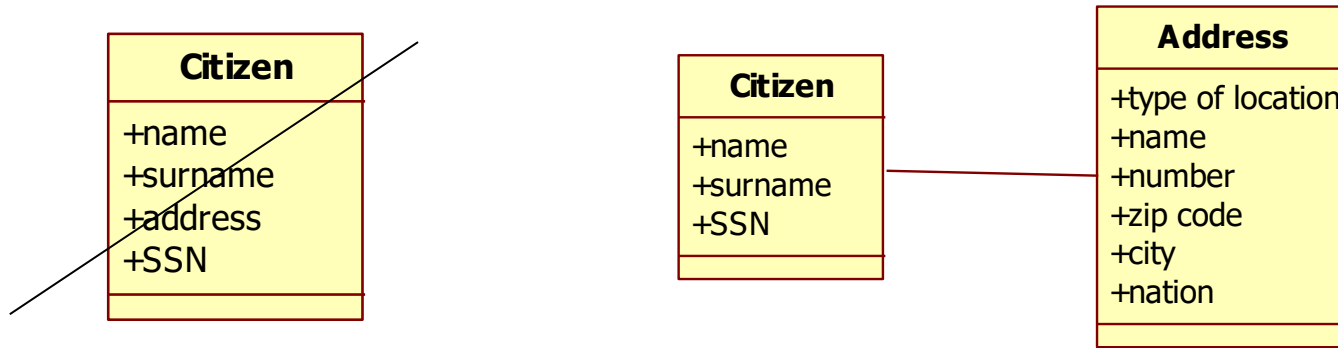
- ◆ Physical entities: Person, Car,
- ◆ Roles: Employee, Director, Doctor,
- ◆ Social / legal / organizational entities: University, Company, Department
- ◆ Events: Sale, Order, Request, Claim, Call
- ◆ Time intervals: Car rental, Booking, Course, Meeting
- ◆ Geographical entities: City, Road, Nation
- ◆ Reports, summaries, paper documents: weather report, bank account statement, travel request

DO NOT in class diagrams

- Use plurals for classes
 - ◆ Person yes, PersonS no
- Forget multiplicities
- Forget roles / association classes, when needed

DO NOT in class diagrams

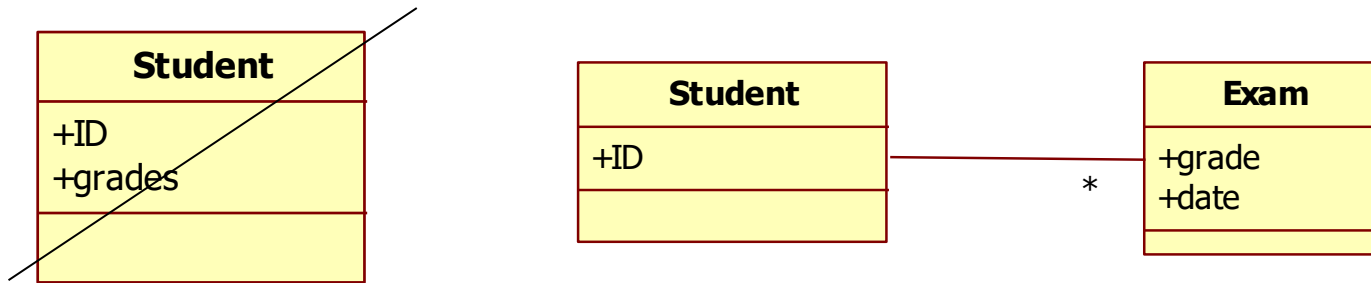
- Use class as an attribute



If address has many attributes it should be modelled as a class, not as an attribute

DO NOT in class diagrams

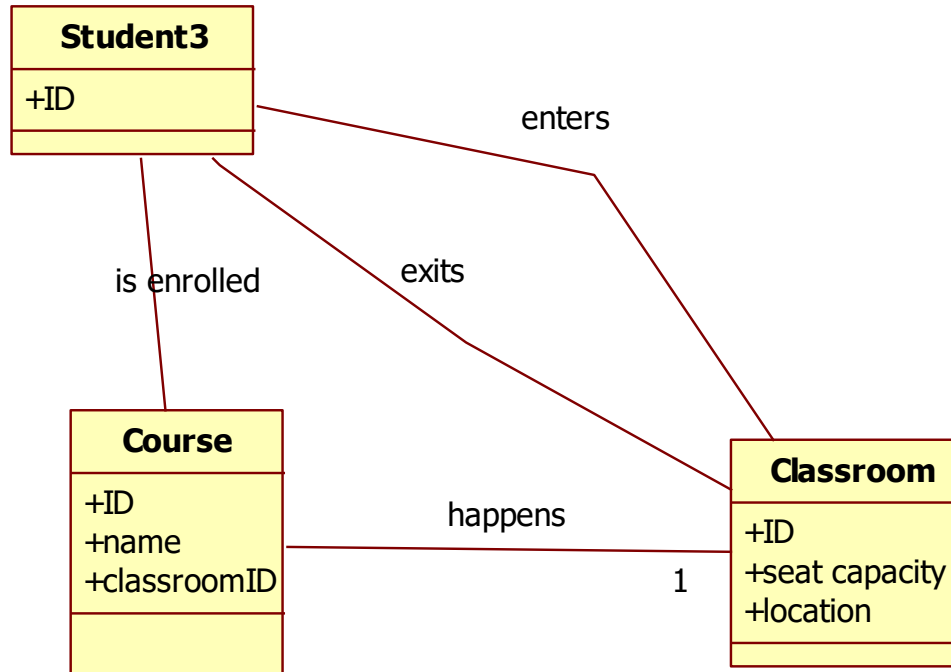
- Use attribute that represents many objects



‘grades’ is not an attribute, but an association with multiplicity *

DO NOT in class diagrams

- Use transient (dynamic) relationships that represent events

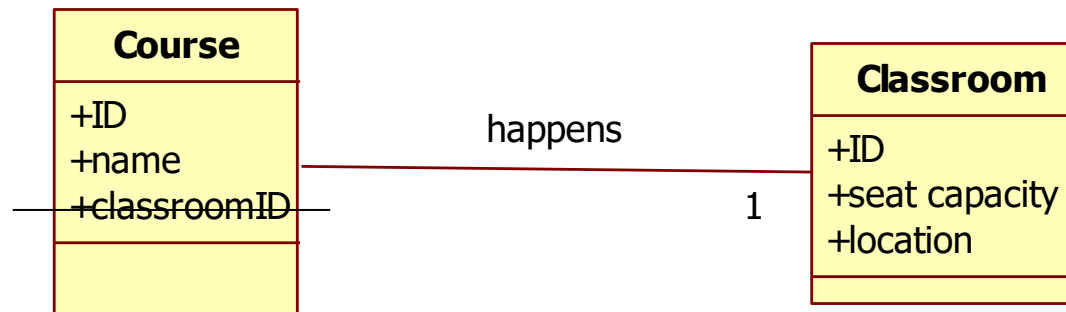


‘Enters’, ‘exits’

- Correspond to events (student enters a classroom)
- Avoid them
 - ◆ They clutter the diagrams
 - ◆ They are better represented in scenarios, or activity diagrams, or BPMN (all dynamic models)
 - ◆ even if the information is needed (ex application to trace Covid19 contacts), the association may not be enough
 - Only one link possible student-x classroom-y
 - Then use class ‘Entrance log’

DO NOT in class diagrams

- Repeat as an attribute of a class a relationship starting from the class

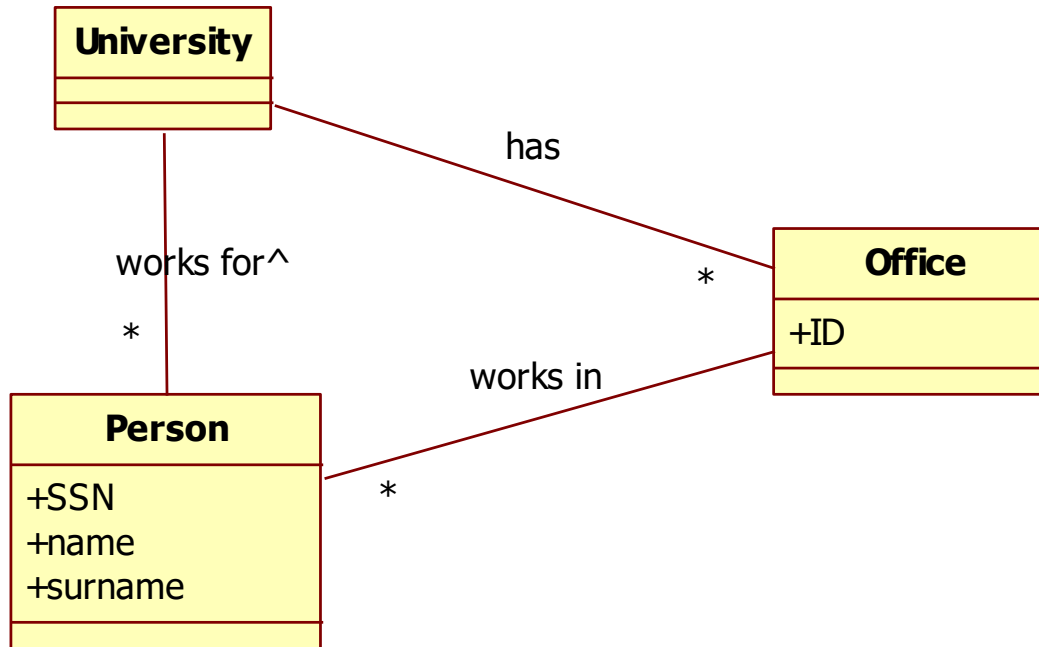


classroomID in Course is redundant (and part of sw design), the association 'happens' already conveys the information

DO NOT in class diagrams

- Use loops in relationships (normally avoid them unless the information they represent is different over different paths)

Loops



Loop university – office – person

Ok, because 'works in' identifies the specific office where a person is, while 'has' identifies a larger set of offices

DO NOT in class diagrams

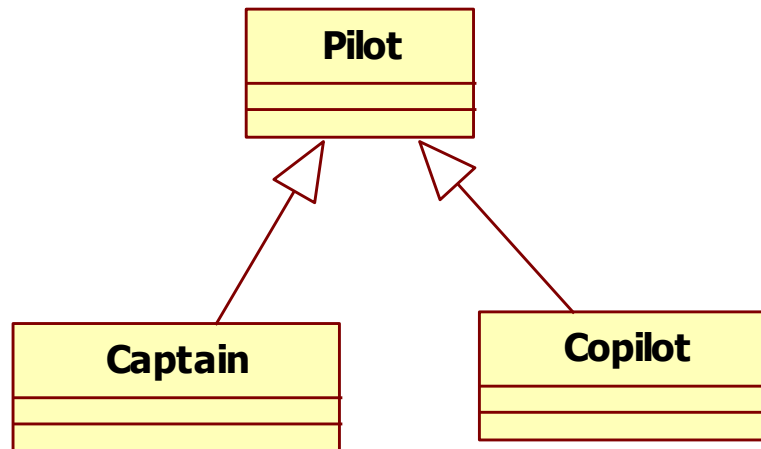
- Confound system design, software design, glossary /conceptual model
 - ♦ DO decide goal of diagram

DO NOT in conceptual model

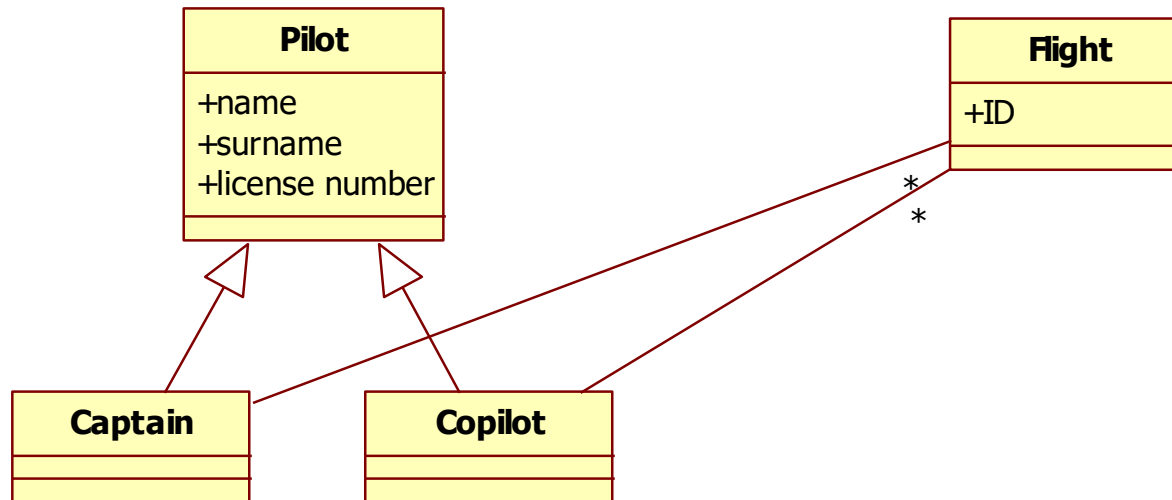
- Dont model classes that belong to software design
- Collections
 - ♦ Linkedlist
 - ♦ Array
- GUI classes
 - ♦ Window
 - ♦ Button
- Dont model classes that belong to hardware / network design
 - ♦ Will appear in technological model (UML deployment diagram)

Be careful

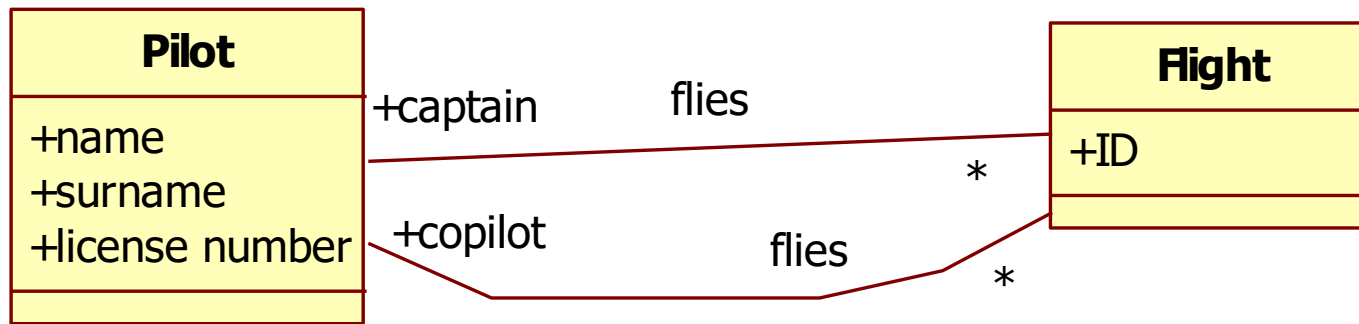
- Instance of a subclass cannot become instance of another subclass
 - ◆ Ex: captain cannot be copilot and copilot cannot be captain



- Flights must have a captain, and a copilot



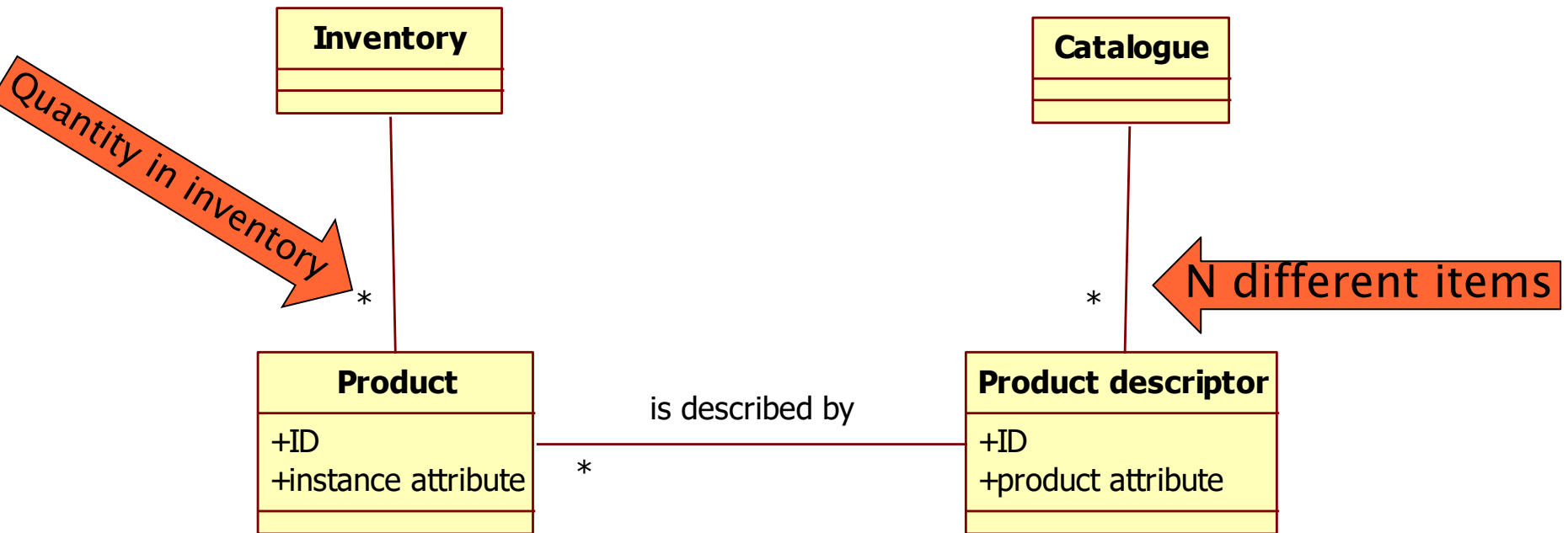
- Captain John Smith is always captain, cannot be copilot (copilot X Y is always copilot)



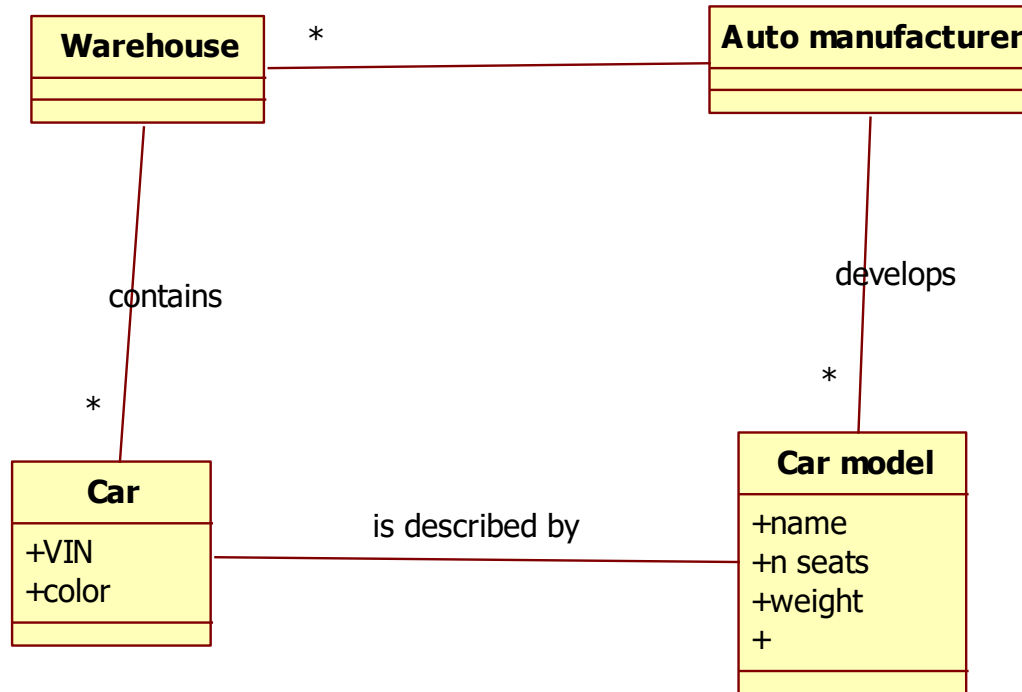
- Captain and copilot as roles. Pilot john smith can be captain on one flight and copilot on another

Patterns in IS – descriptor

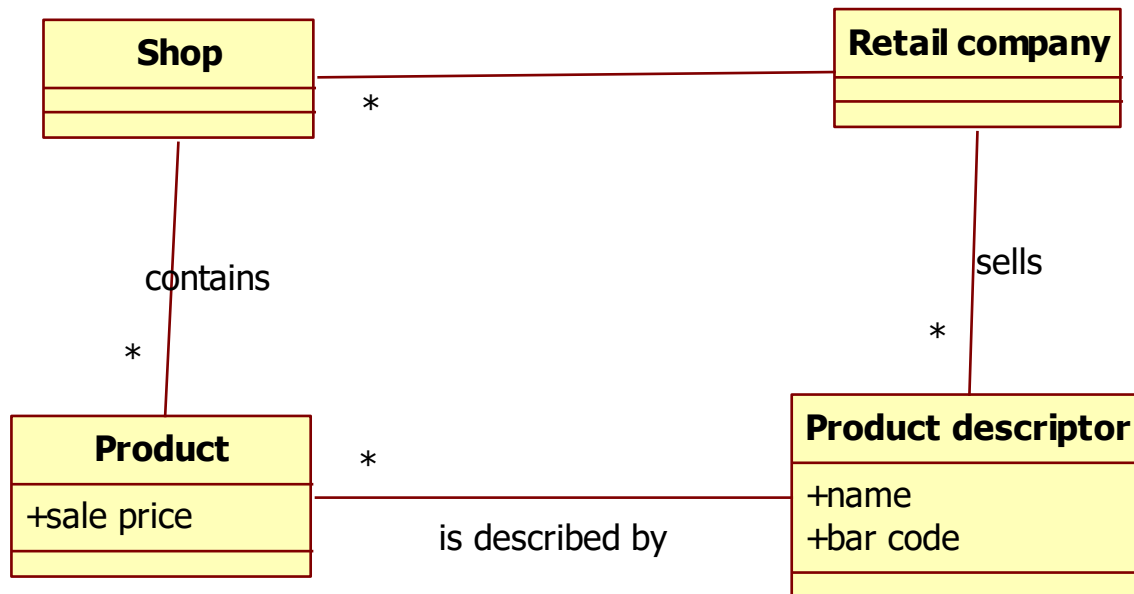
- Catalogue vs inventory



Ex automotive

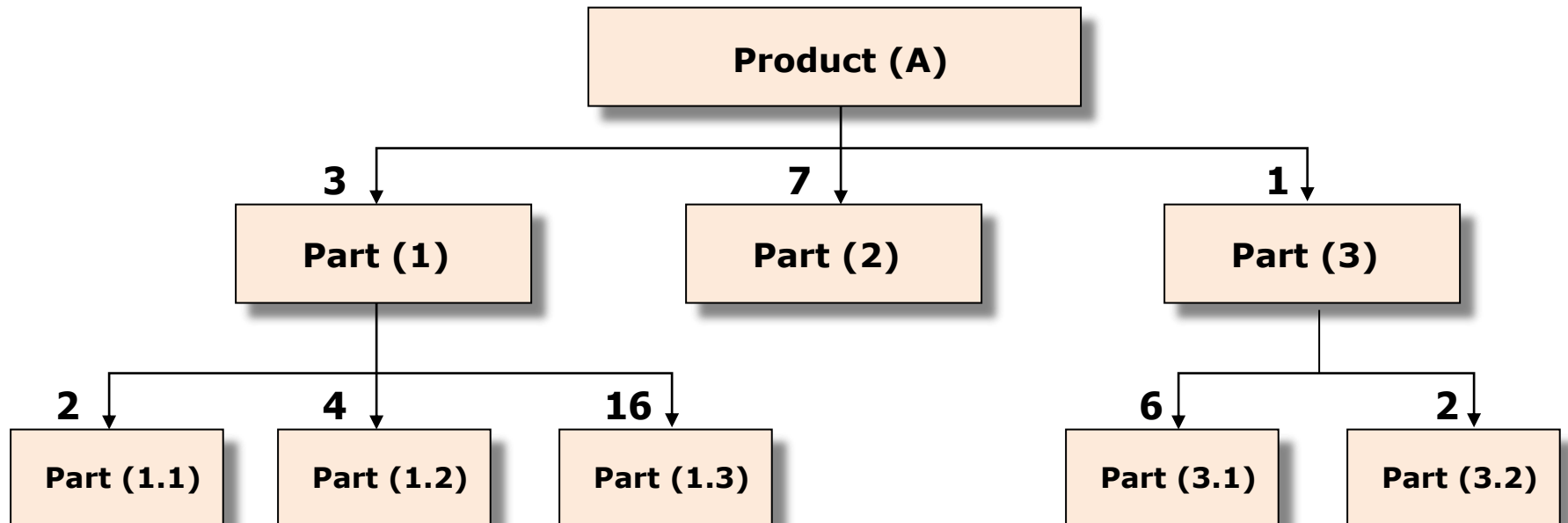


Ex retail



Patterns in IS – composite (BOM)

Bill of materials (BOM)



Composite pattern – BOM

