Software Engineering

Rogério de Lemos

- Class diagrams – advanced concepts

Lecture Outline

Some of the topics

- Stereotypes
- Dependency
- Generalisation (inheritance)
- Multiple inheritance
- Aggregation
- Derived elements
- Identifying classes and associations

Stereotypes

One can stereotype any UML element

- UML extensibility mechanism
  - gives extra classification to model items
- <<stereotype-name>>
  - e.g., <<interface>> on a class, <<use>> on a dependency

Stereotypes are part of **profiles**

- tailor the language to a particular domain
  - e.g., business modelling

Stereotypes

A

<<interface>>

A

<<enumeration>>

A

<<persistent>>

A

<<table>>

A

Dependencies

A **dependency** exists between two elements if changes in the definition of one (*supplier* or source) affects the other (*client* or target)

Difference between dependency and association

- association represent the fact that objects of the classes are associated
- dependency is between classes and not objects of those classes

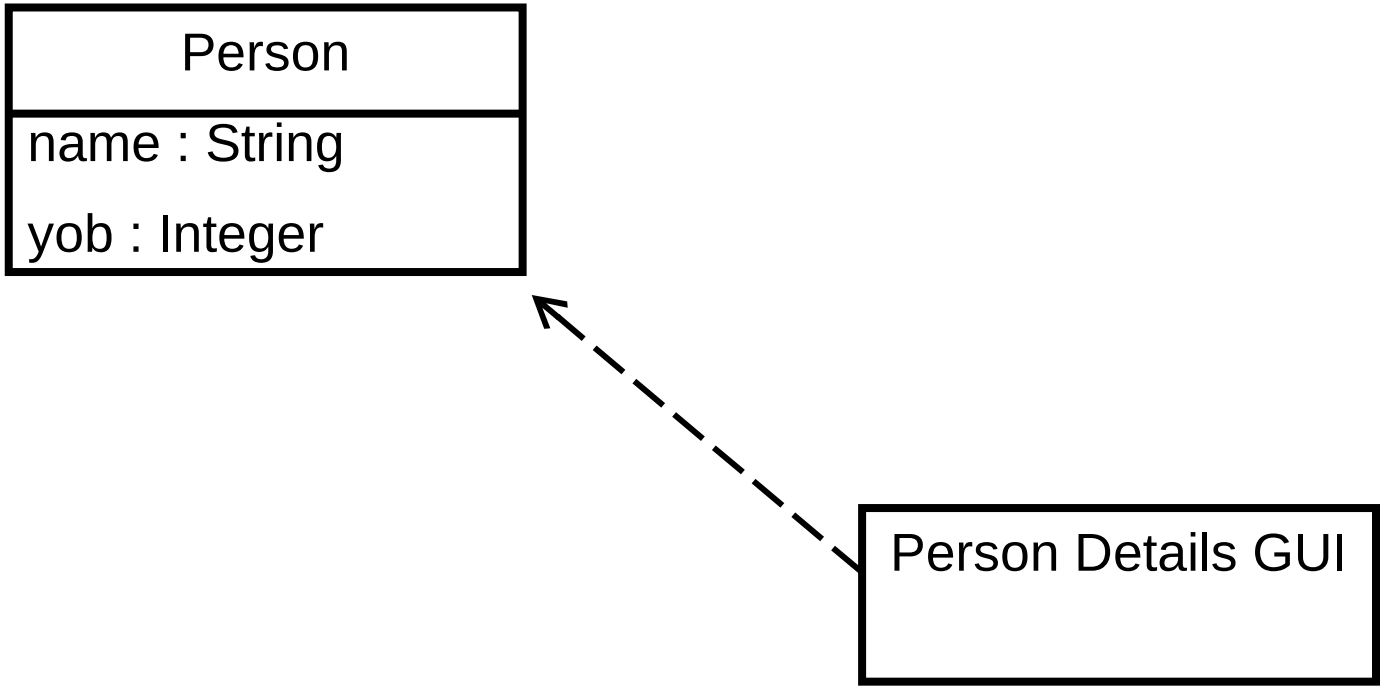
Different kinds of dependency

- a class sends a message to the another
- one class has another as part of its data

Dependencies

As software grows one has to worry about dependencies

- if dependencies get out of control this might be undesirable



Dependencies

Dependencies should be used to show how changes in one element might alter other elements

UML has several kinds of dependencies

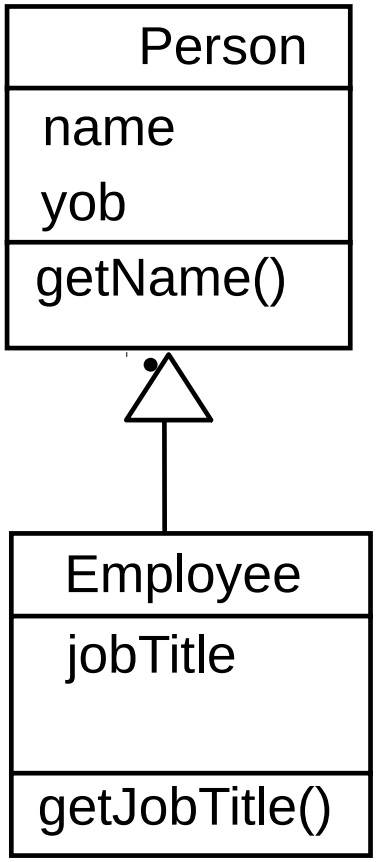
- some dependencies keywords

Keyword	*Meaning*
<<call>>	*The source calls an operation in the target*
<<create>>	*The source creates an instance of the target*
<<instantiate>>	*The source is an instance of the target*
<<use>>	*The source requires the target for its implementation*

Generalisation

Generalisation (can be implemented by inheritance)

- extend and reuse the functionality of a class
- conceptual
  - a subclass “is-a-kind-of “ superclass
- specification
  - subtype that conforms with type
    - subclass inherits all attributes/operations
- implementation
  - ...Employee extends Person ...
- subclass may override operations of the superclass



Generalisation

General conceptual rule

- an object of a superclass can be substituted by an object of subclass, but not the other way

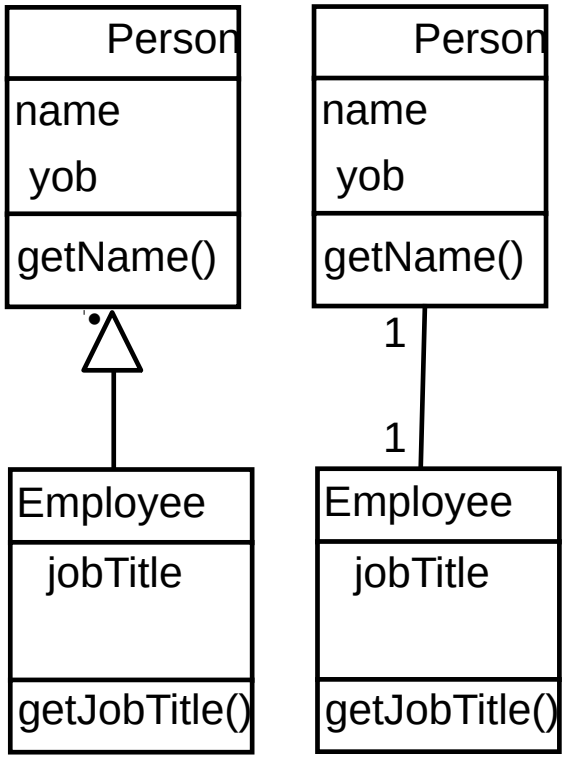
Objects instantiated from the previous diagram

john : Person
name= “John Green” yob = 1948

: Employee
name= “John Green” yob = 1948 jobTitle=“manager”

Generalisation

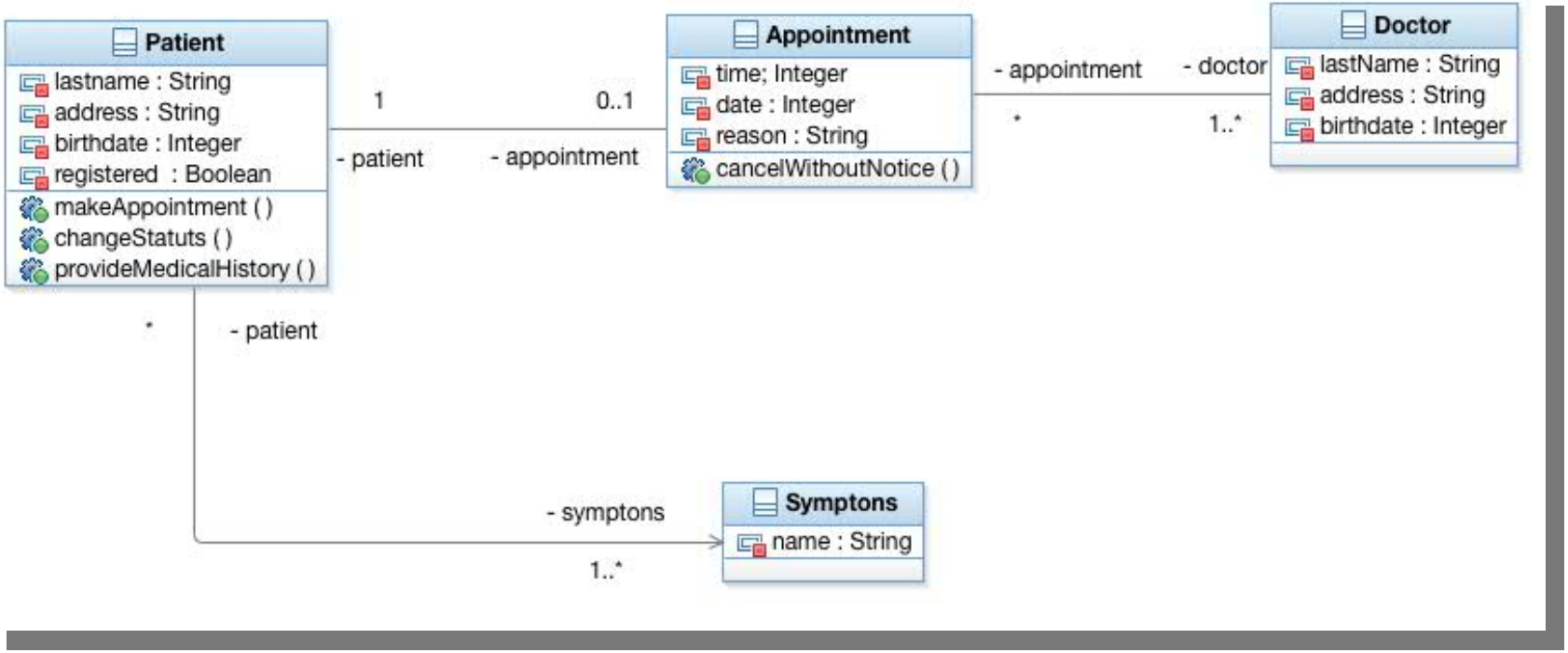
- What is the difference between generalisation and association?
  - conceptually think in terms of instantiation,..



Instantiate from both diagrams and notice the difference

Example: UML Class Diagram

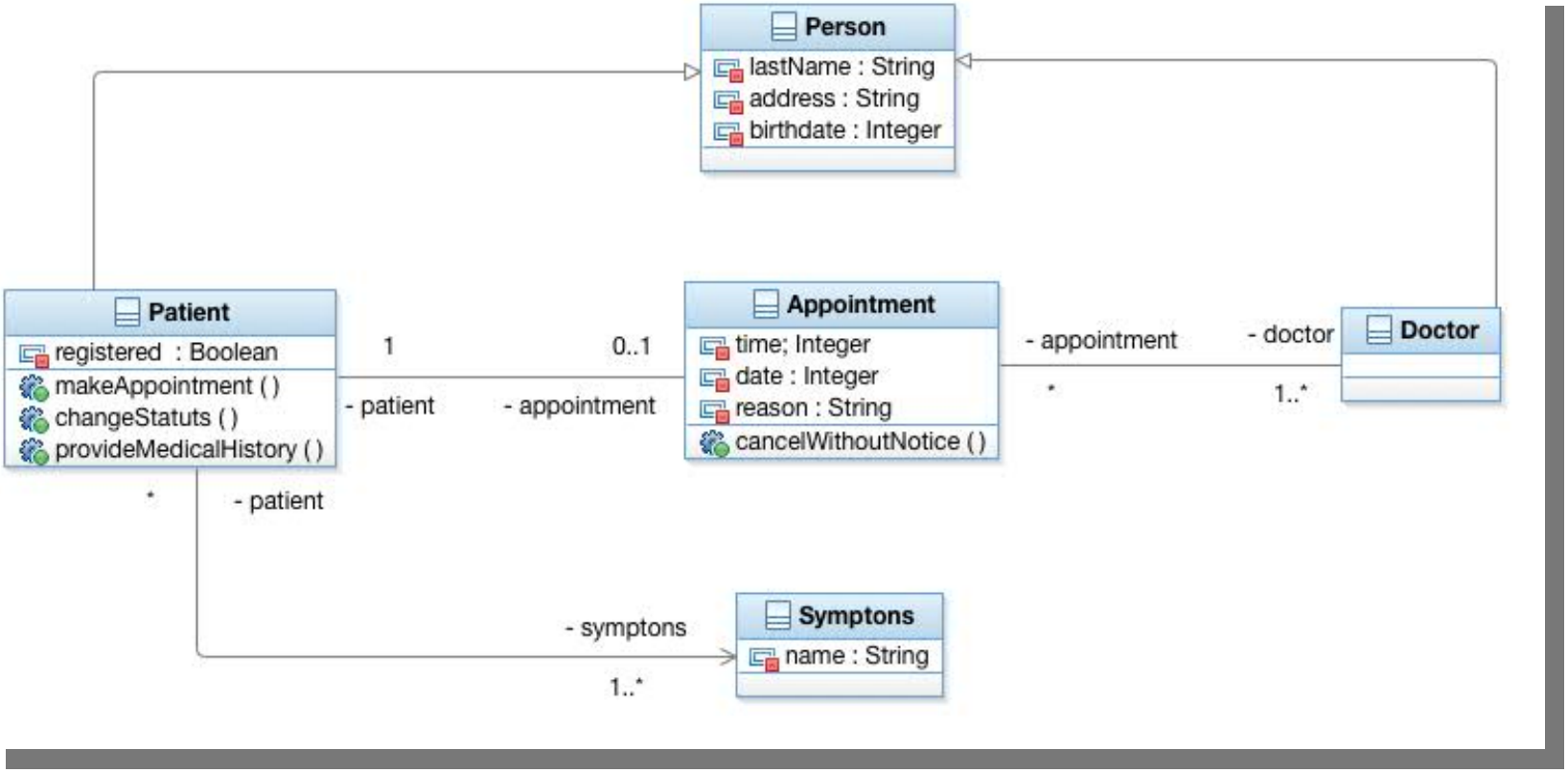
Again, lets look at this class diagram and see how could be improved?



Example: UML Class Diagram

How to improve the diagram?

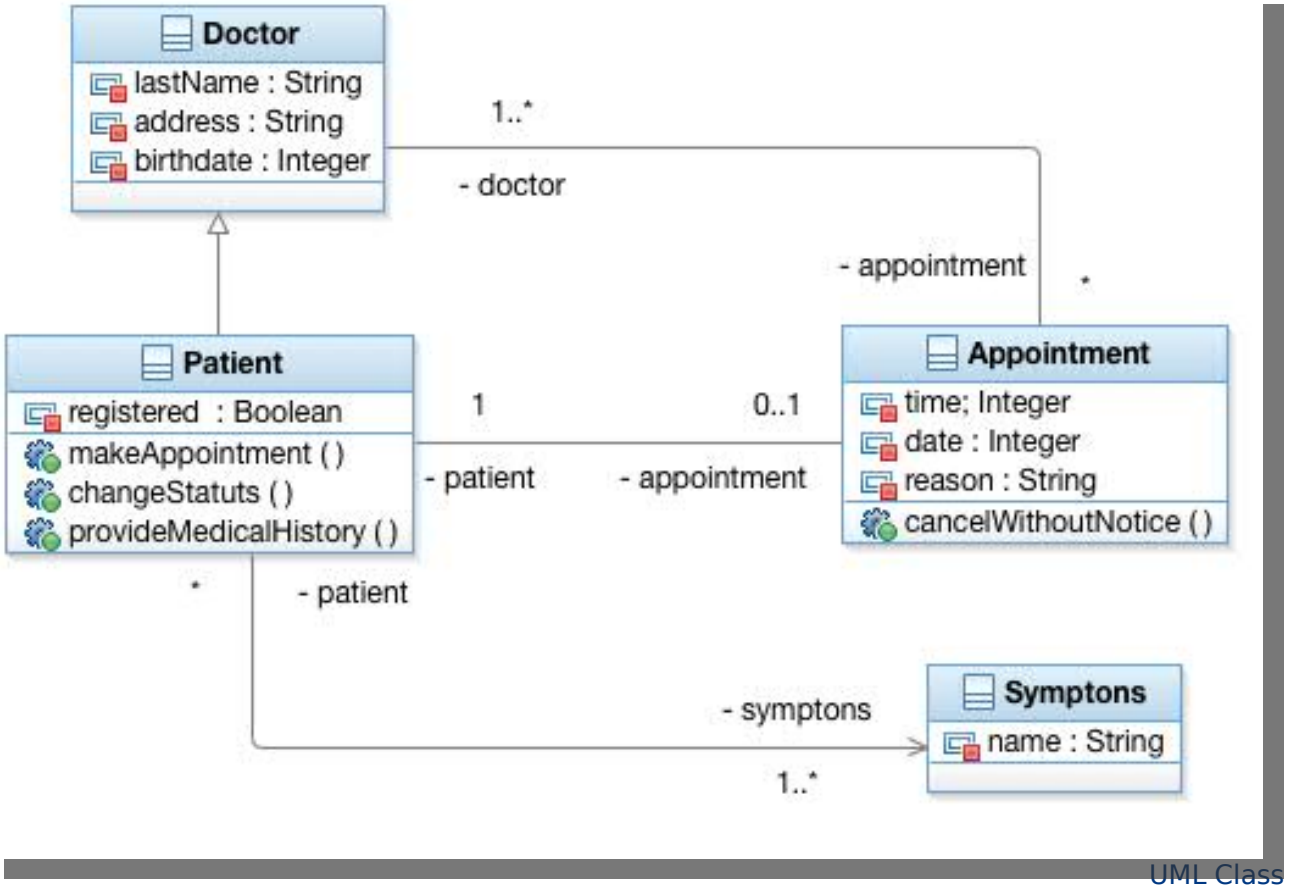
Solution 1 – what about this one?



Example: UML Class Diagram

How to improve the diagram?

Solution 2 – is this a meaningful one?



Generalisation and Classification

- *What is a class, and what is an object*

Replacing “a” by “every” helps to identify the difference

Generalisation

- Border Collie is a subtype of the type Dog
- generalisation symbol ( )

How to interpret the ***is a*** relationship?

- it can lead to inappropriate use of subclassing and confused responsibilities

Classification

- an object Shep is an instance of a type Border Collie
- dependency with <<instantiate>> keyword

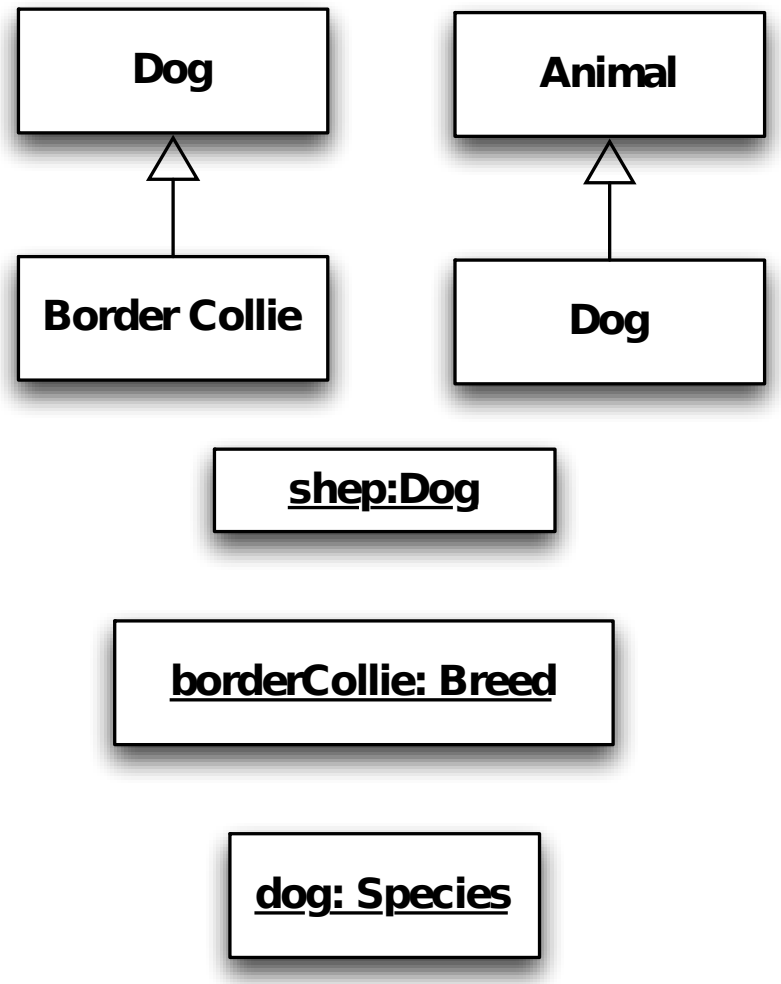
Generalisation and Classification

Is it a classification or a generalisat

1. Shep is a Border Collie (c)
2. a Border Collie is a Dog (g)
3. a Dog is an Animal (g)
4. Border Collie is a Breed (c)
5. Dog is a Species (c)

Example

- Dogs are kinds of Animals
- Every instance of a Border C



Generalisation and Classification

Is a classification a generalisation?

- 1 & 2 – Sheep is a Dog
- 2 & 3 – Border Collies are Animals
- 1 & 2 & 3 – Sheep is an Animal
- 1 & 4 – Sheep is a Breed
- 1 & 4 are classifications
- 2 & 5 – A Border Collie is a Species
  - 5 is a classification

1. Sheep is a Border Collie (c)
2. a Border Collie is a Dog (g)
3. a Dog is an Animal (g)
4. Border Collie is a Breed (c)
5. Dog is a Species (c)

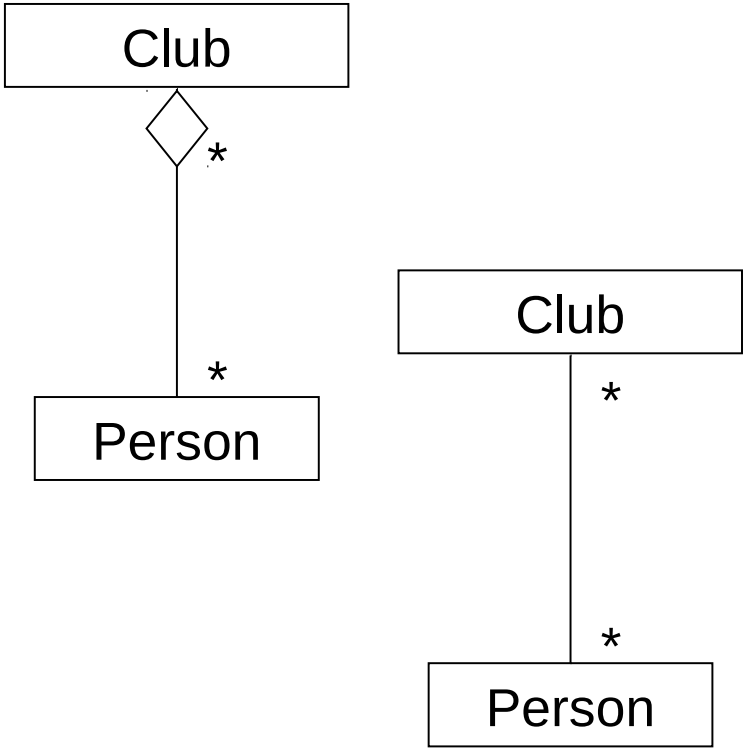
Aggregation and Composition

Aggregation and **composition** are kinds of association

- they both record that an object of a class ***is part of*** another class

Aggregation

- hollow diamond
- some form of part-whole relationship
  - diamond goes on whole
- no major difference between aggregation and association
  - it doesn't give any more formal information
- strictly meaningless !



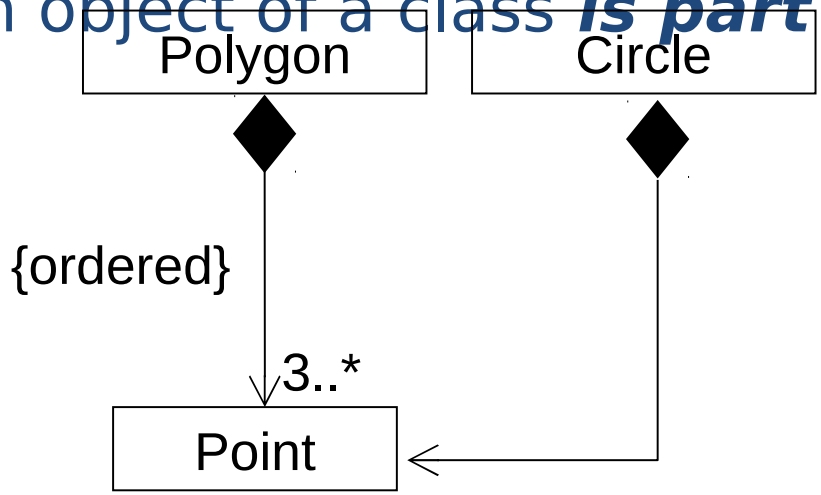
Aggregation and Composition

Aggregation and **composition** are kinds of association

- they both record that an object of a class ***is part of*** another class

Composition

- solid diamond
- lifetime of part is linked to the whole
  - the whole strongly owns parts
- no sharing (part can only belong to one whole)
- owned by value



- either an instance Point is part of a polygon or the centre of a circle, not both

- although a class may be an element of many other classes, any instance must be an element of only one owner

Aggregation and Composition

Aggregation and **composition** are kinds of association

- they both record that an object of a class ***is part of*** another class

Aggregation

- hollow diamond
- some form of part-whole relationship
  - diamond goes on whole
- no major difference between aggregation and association
  - it doesn't give any more formal information
- strictly meaningless !

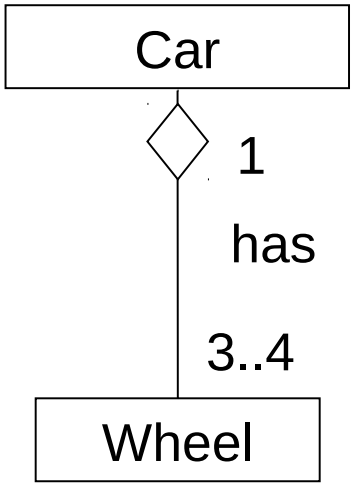
Composition

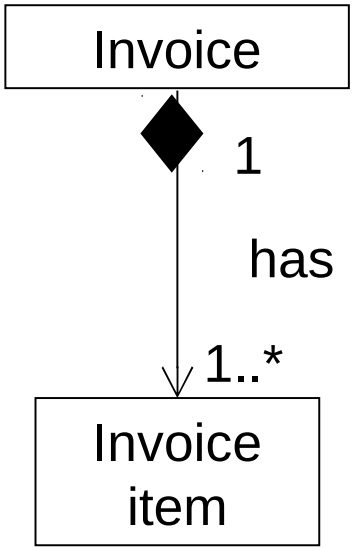
- solid diamond
- lifetime of part is linked to the whole
  - the whole strongly owns parts
- no sharing (part can only belong to one whole)
- owned by value

Aggregation and Composition

Alternatively, a special case of aggregation is composition

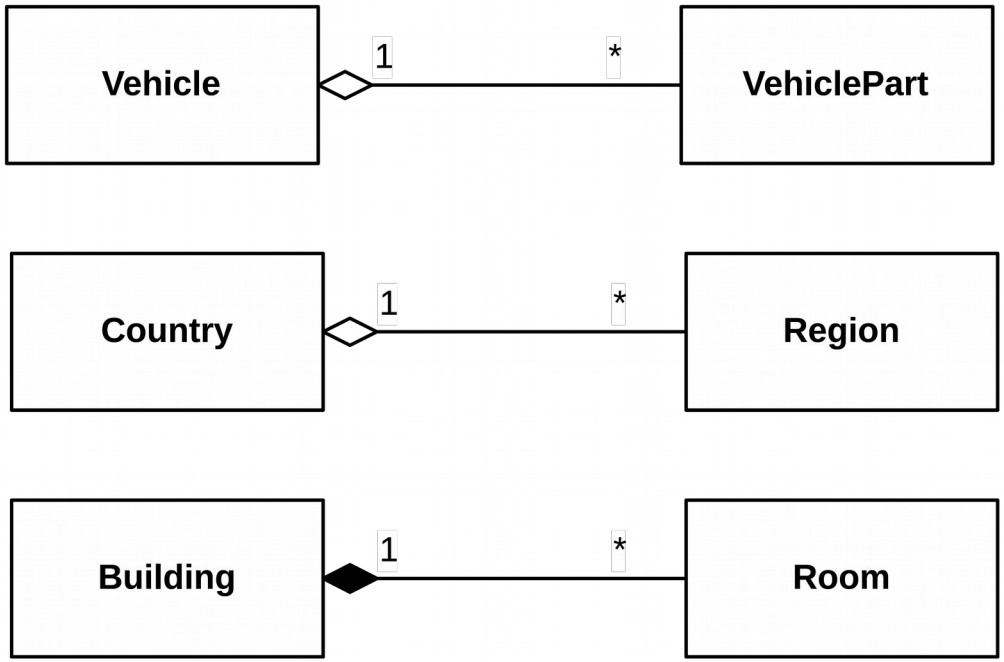
- the individual parts depend on the whole for their existence



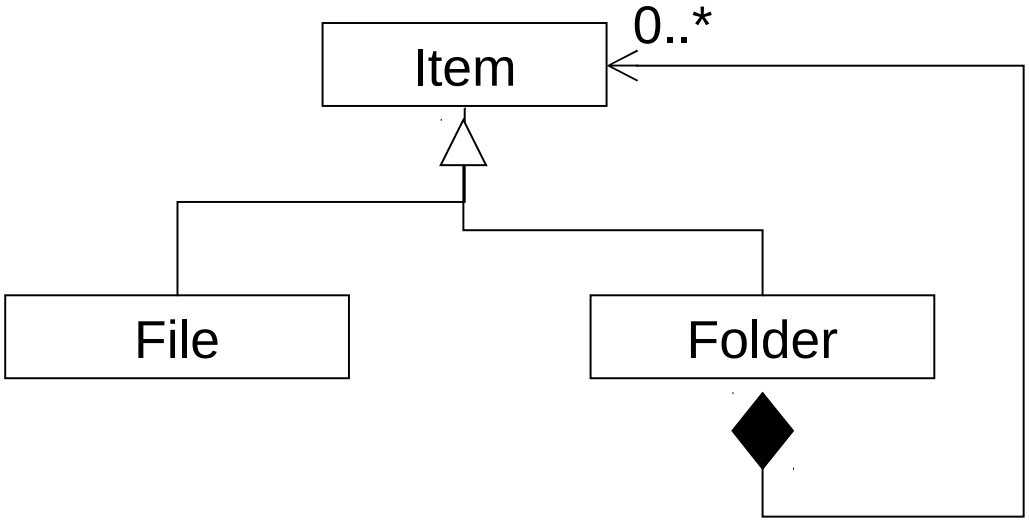


Aggregation and Composition

Other examples

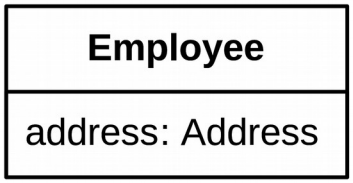


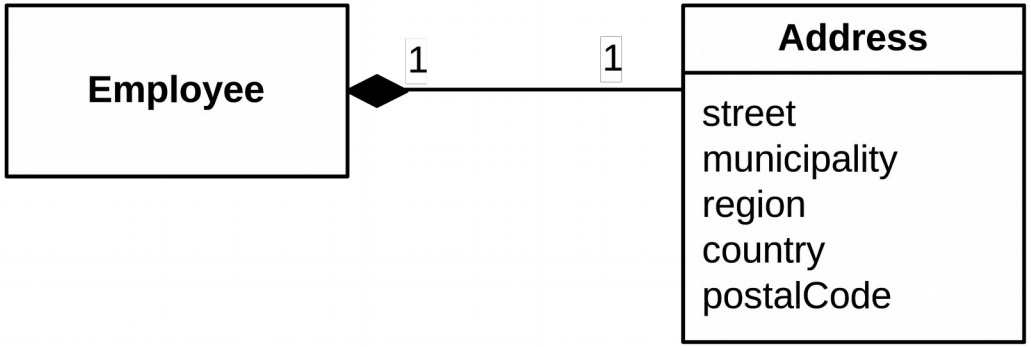
Another Composition Example



Another Composition Example

The address of an employee can be represented as an attribute or as a composition

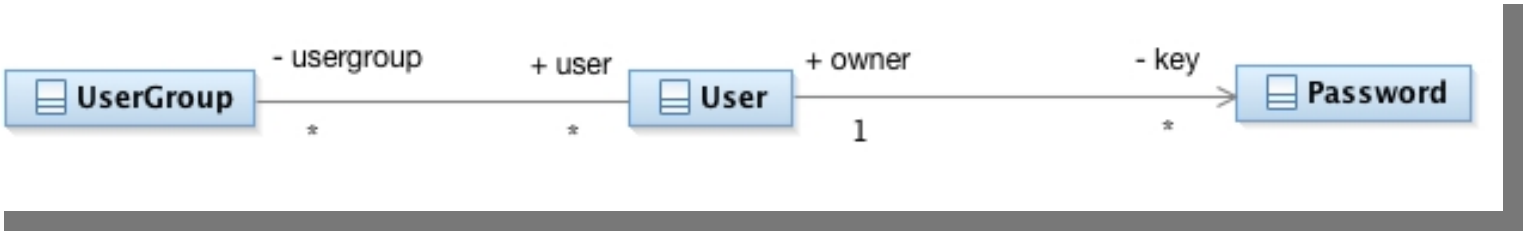




Visibility

One can specify three levels of visibility

- public (+) – any one can access the objects
- private (-) – objects at the end of a link are not accessible to any object outside the association
- protected (#) – objects at the end of a link are not accessible to any object outside the association, except for the descendants of the other end



More about Classes

There three different interpretations on how to classify objects

- **<<interface>>** – list of operations
  - there are no implementations associated with operations
  - it does not specify anything about state
    - no attributes, and associations navigable from an interface
- **<<type>>** - is an interface with state
  - it specifies attributes and operations
  - it doesn't define any implementation

More about Classes

- **<<implementation class>>** - defines the physical implementation of its operations and attributes
  - it can realise a type

An object has exactly one implementation class, though it can have several types and match several interfaces

Interfaces

An interface specifies some operations that are visible from outside the class

- a class can match several interfaces

Classes have two kind of relationships with interfaces

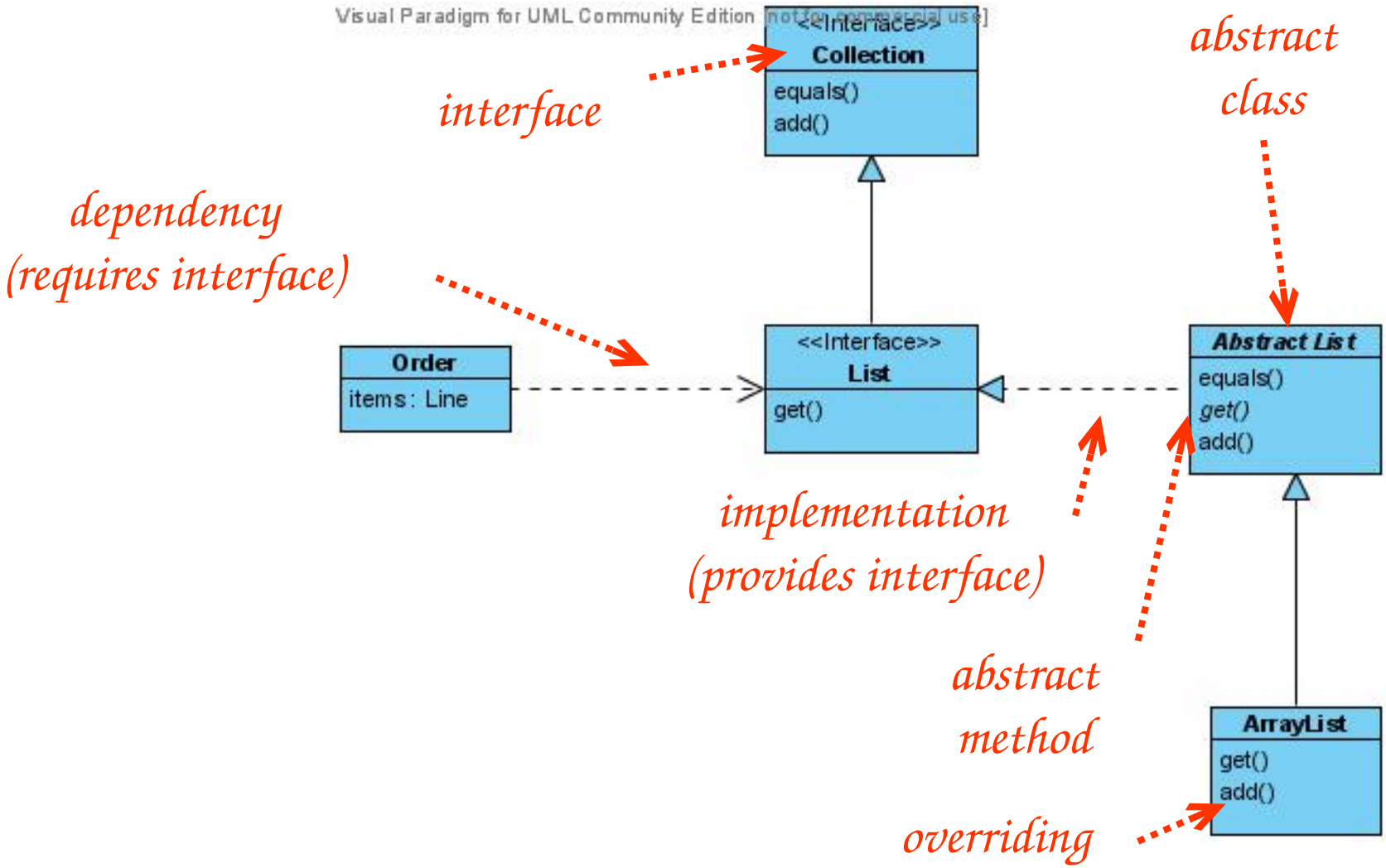
- a class provides an interface if it is substitutable for the interface
- a class requires an interface if it needs an instance of that interface
  - it has a dependency on the interface

Abstract Classes

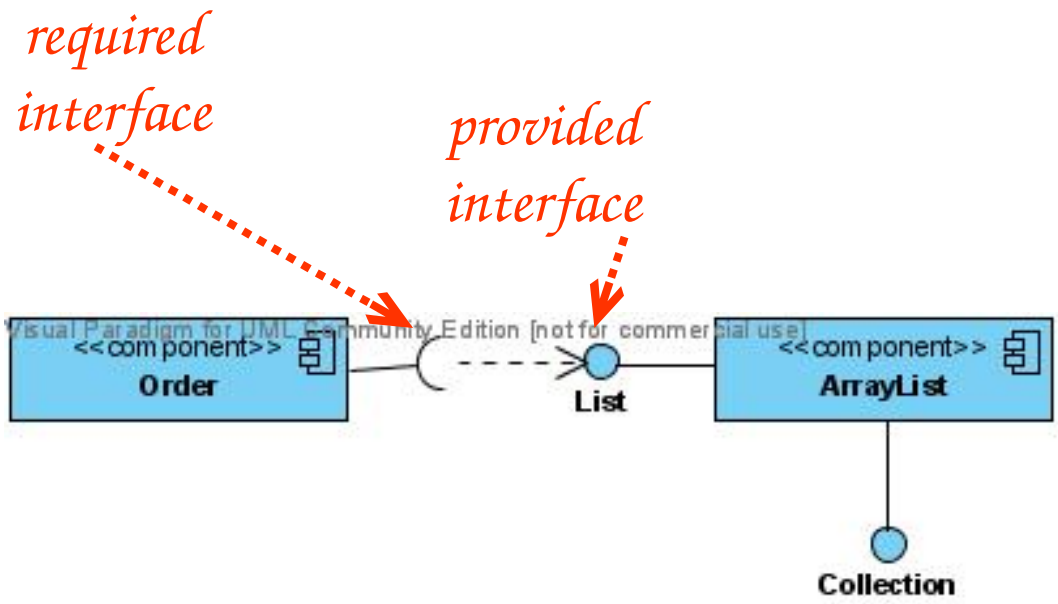
An **abstract class** is a class that cannot be directly instantiated

- instead, one instantiates an instance of a subclass
- it contains one or more operations that are abstract
  - no implementation

Interfaces and Abstract Classes

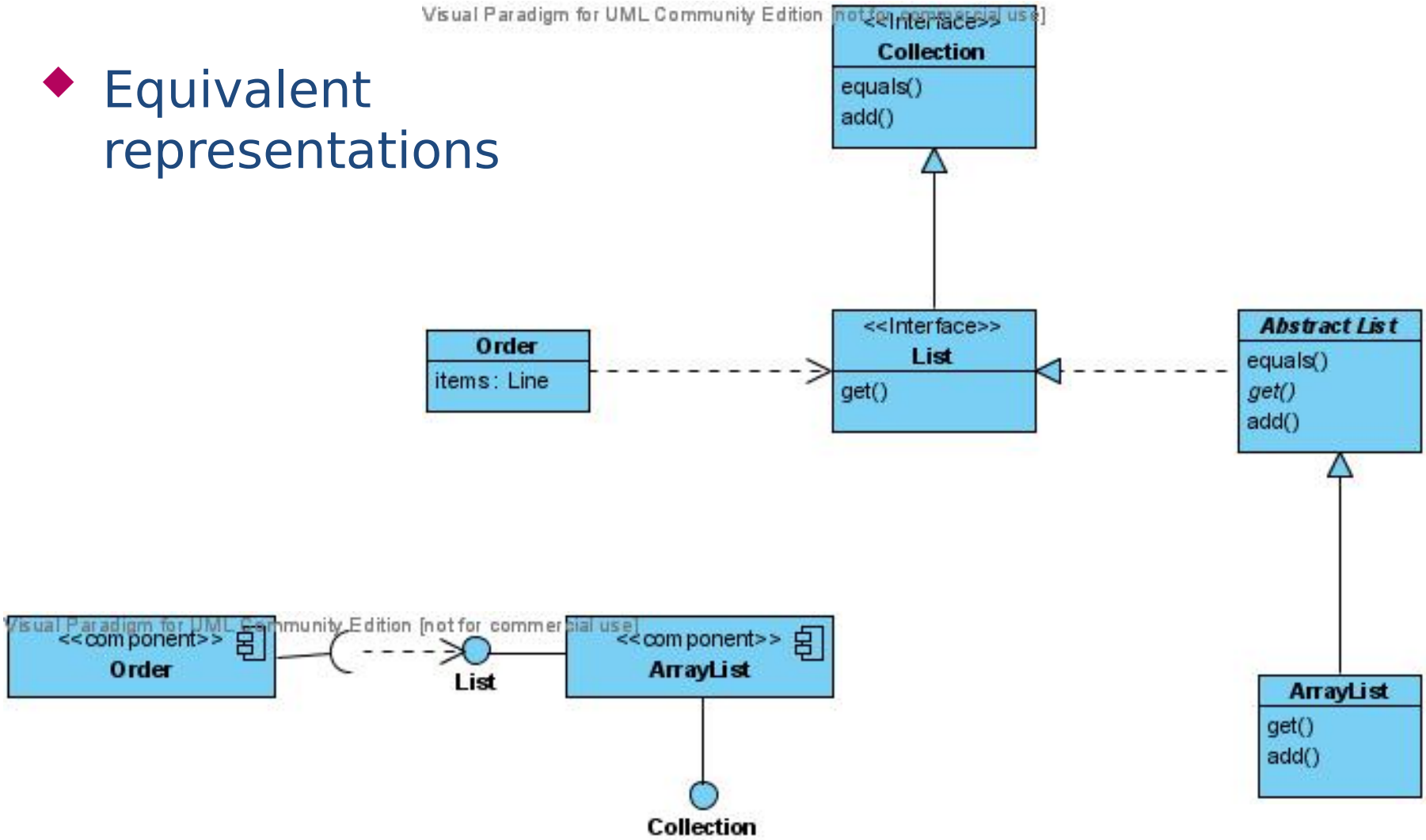


Interfaces and Abstract Classes



Interfaces and Abstract Classes

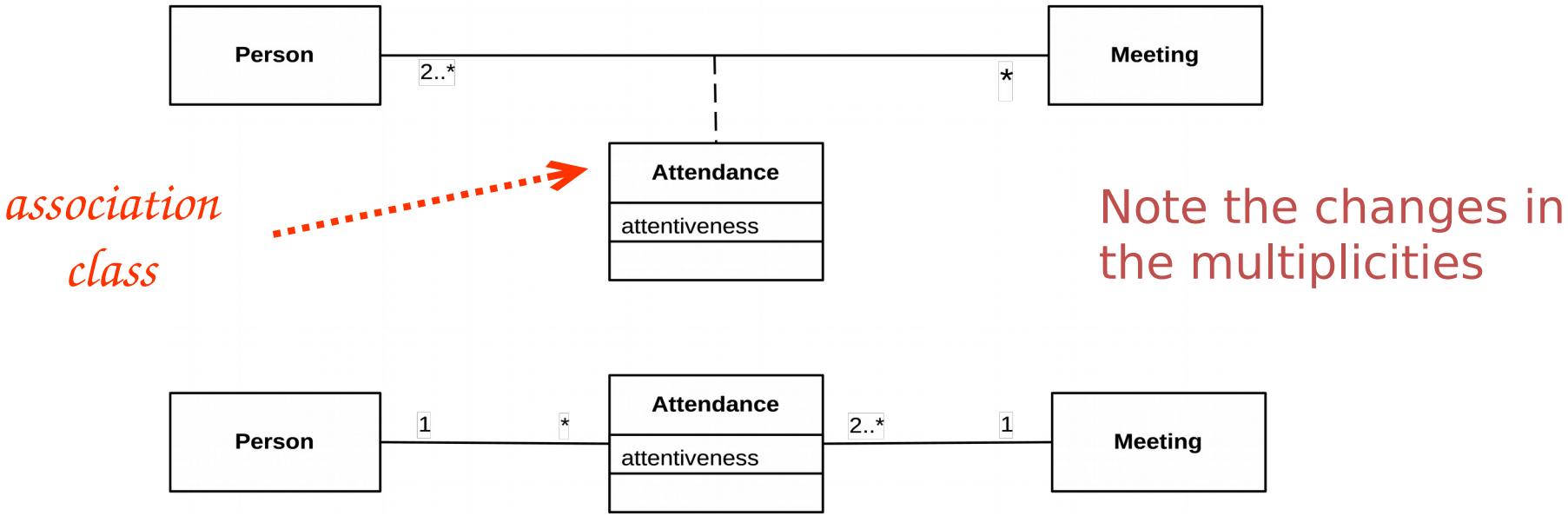
- Equivalent representations



Association Class

Associations classes allow to add attributes, operations, and other features to associations

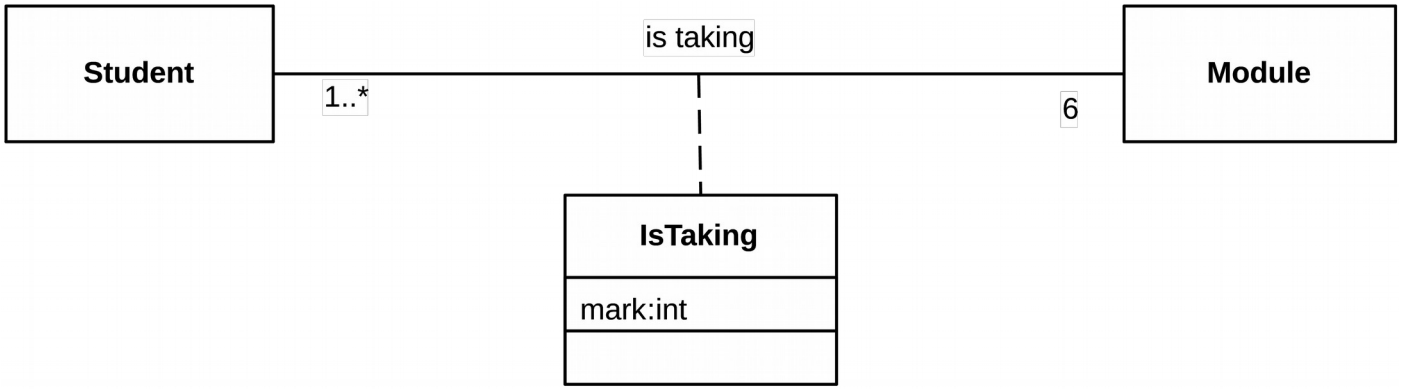
- only one instance of the association class is allowed between two participating objects

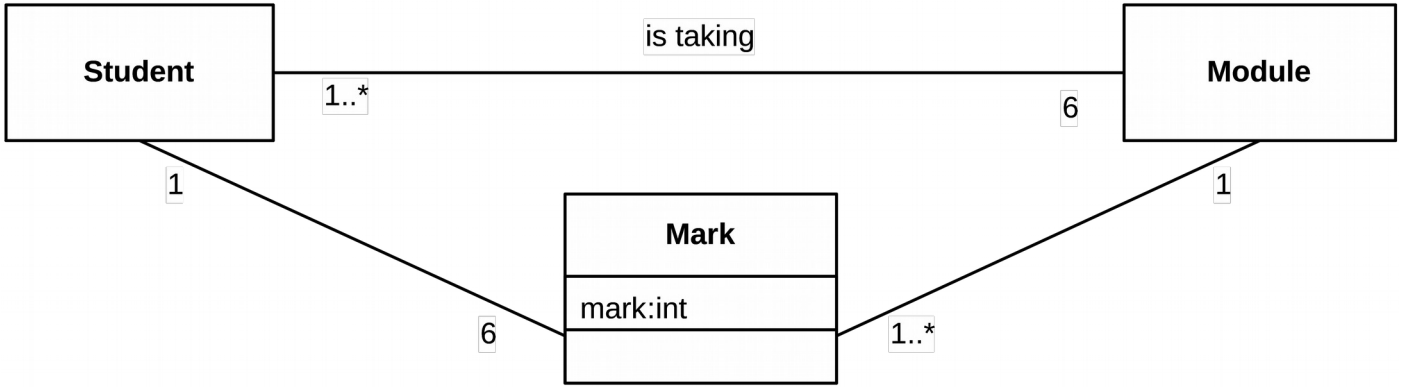


Association Class

Another example

- marks in the association between Student and Module





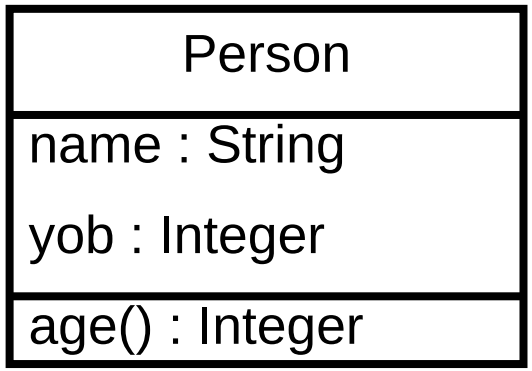
Derived Elements

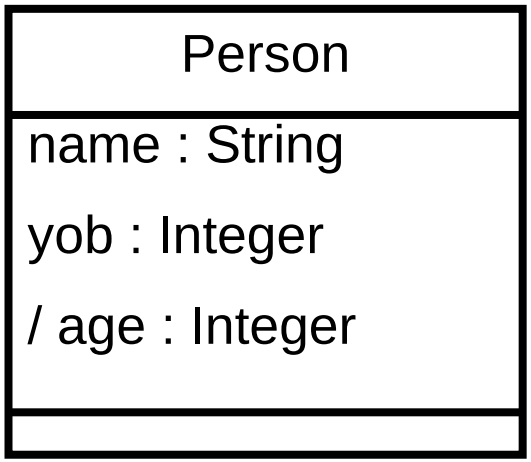
Anything in UML can be marked as “derived”

- it is applied to classes, associations or properties

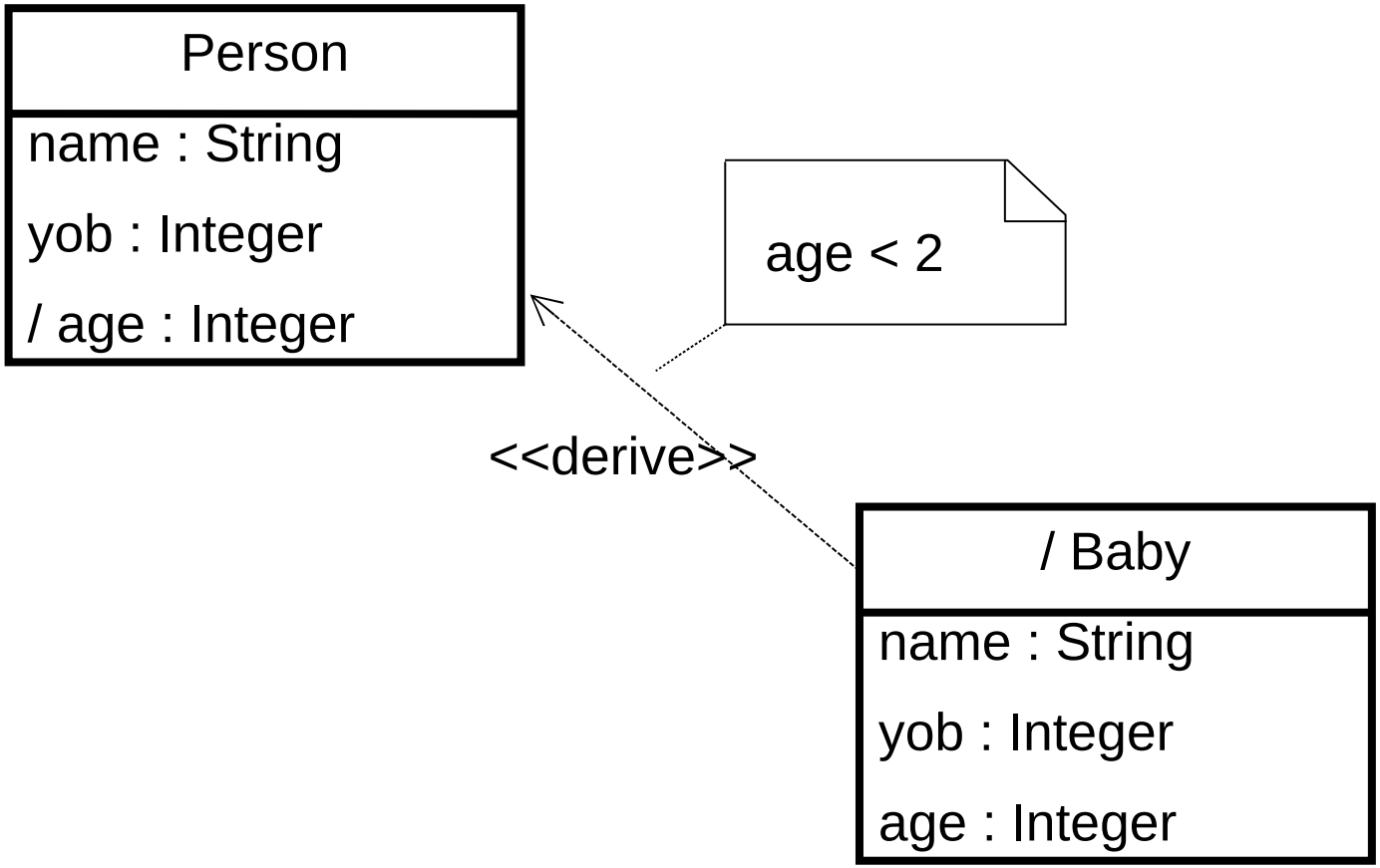
It is calculated rather than stored as a value

It shows there is a constraint between values





Derived Class



Identifying Classes

In analysis, we try to capture the key domain abstractions

- domain - the application we are working with
  - e.g., library
- abstraction (instead of class) to emphasize the aspects of the domain that are important to the application
- e.g. looking for features and facts about the library, which matter for the system we are building

Identifying Classes

One technique for identifying classes is noun identification technique

- underline nouns and noun phrases from requirements
  - identifying words and phrases that denote things
  - gives a list of candidate classes
    - can be eliminated, amalgamated or modified

Identifying Classes

Books and journals. The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All the other books might be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time. Only members of staff may borrow journals.

Borrowing. The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.

Identifying Classes

Books and journals. The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All the other books might be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time. Only members of staff may borrow journals.

Borrowing. The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.

Identifying Classes

Discard those that are not good candidates

- library – outside the scope of the system
- short term loans – loan is an event (lending a book)
- member of library – redundant to library member
- week – measure of time not a thing
- time – outside the scope of system
- system and rule – not part of the domain, part of the language of requirements

Identifying Classes

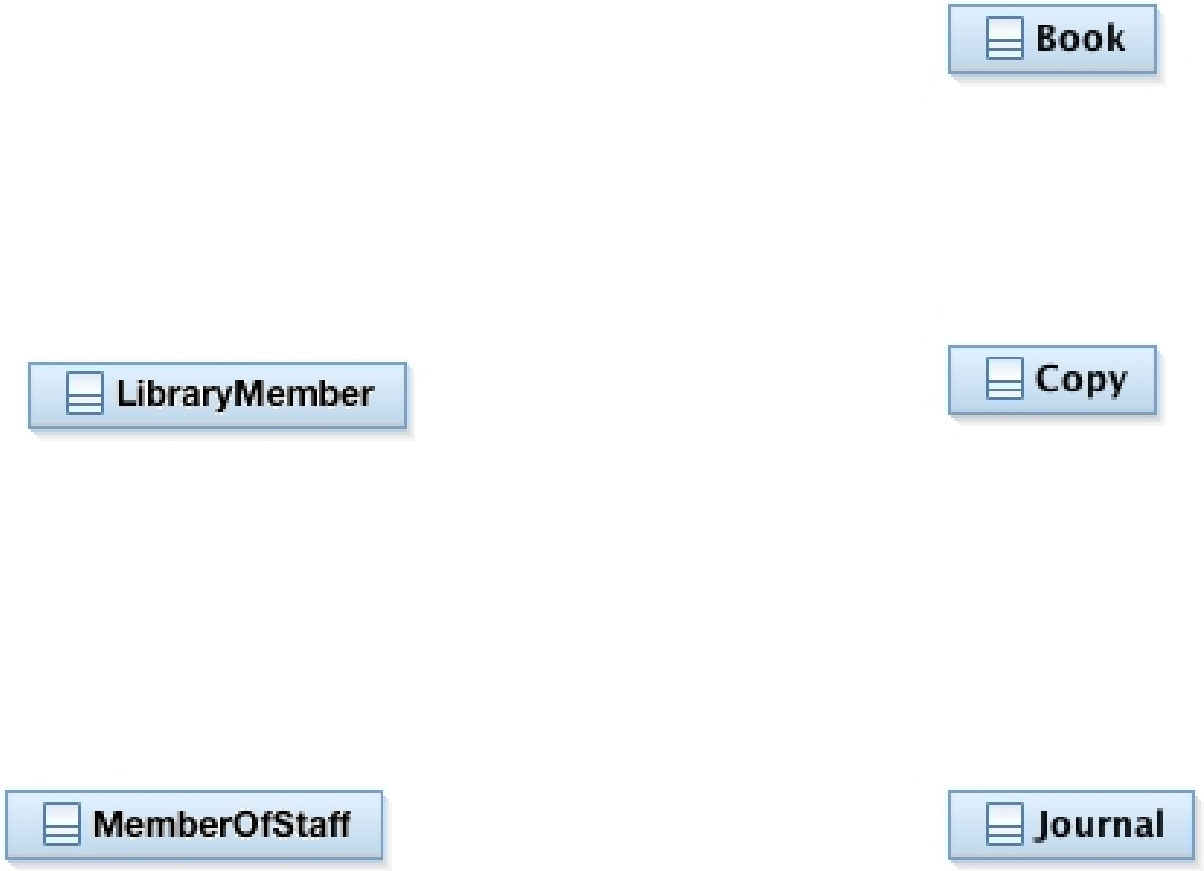
Candidate classes

- book
- journal
- copy (of book)
- library member
- member of staff

Record the provisional responsibilities and identify improvements

Library Systems

Classes of the library system



Relations between Classes

Identify important relationships between classes

- clarify our understanding of the domain
  - describe how objects work together
- we are following good design principle
  - e.g., low coupling

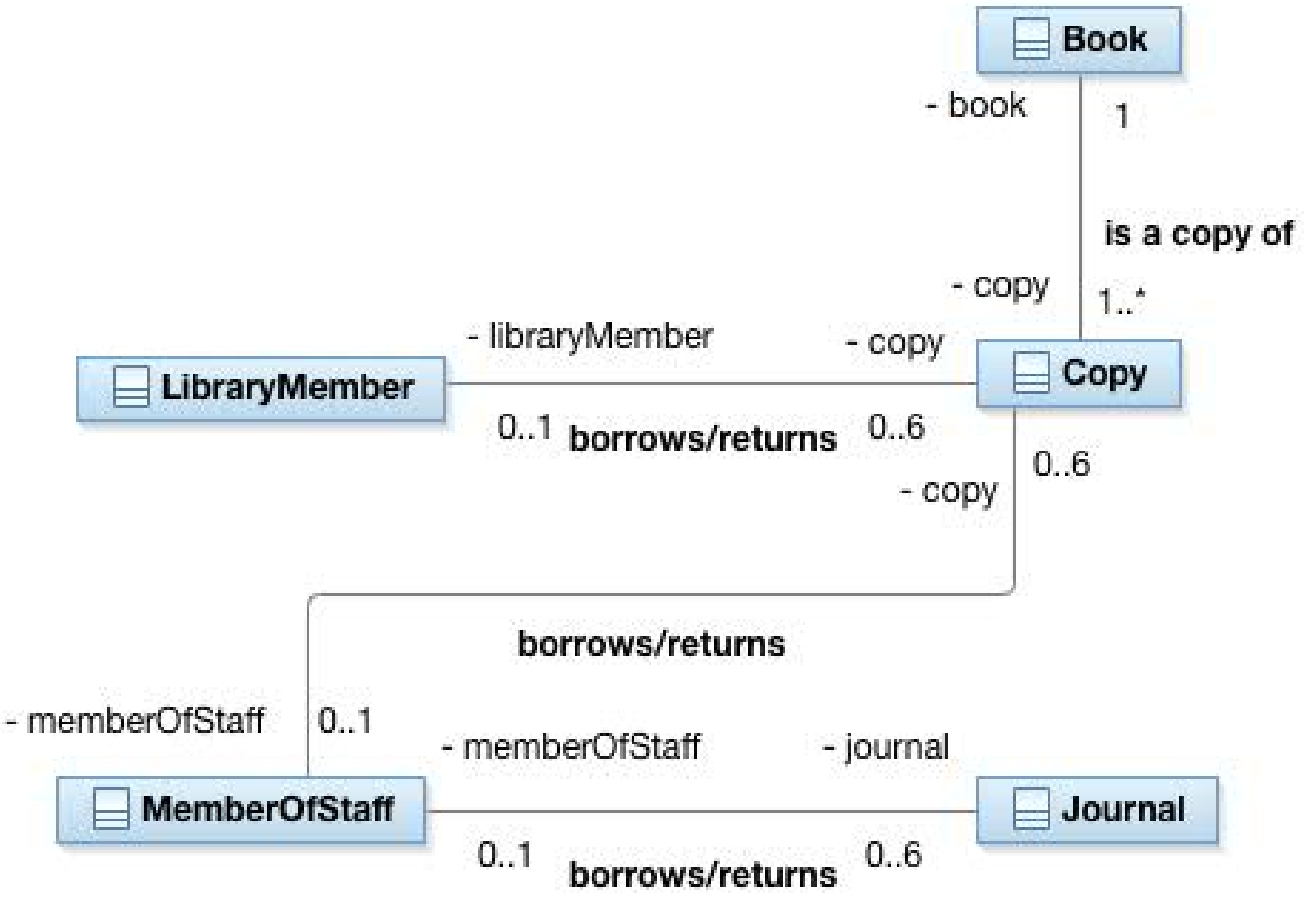
Relations between Classes

Main relations

- a copy is a copy of a book
- a library member borrows/returns a copy
- a member of staff borrows/returns a copy
- a member of staff borrows/returns a journal

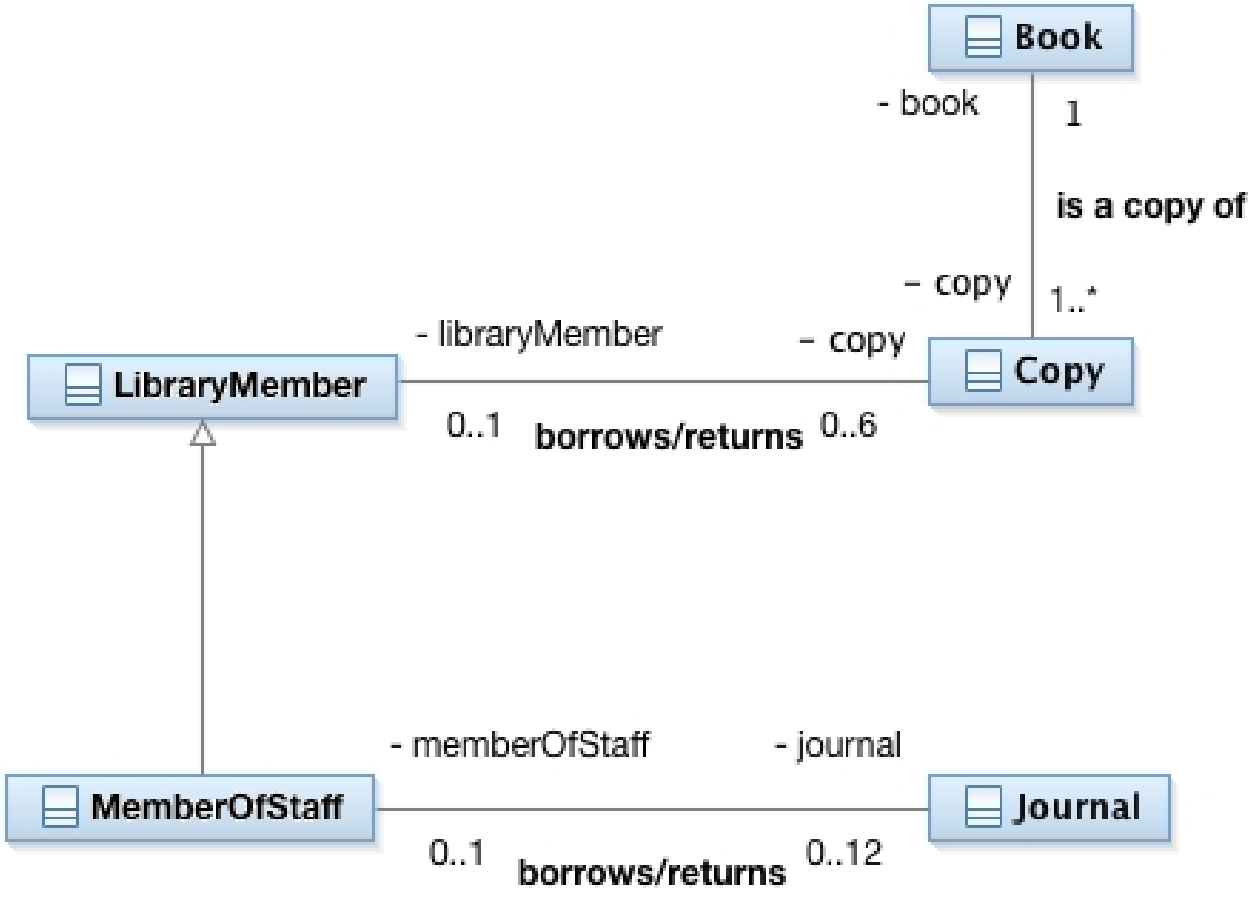
Relations between Classes

Initial class model of the library



Relations between Classes

Revised class model of the library



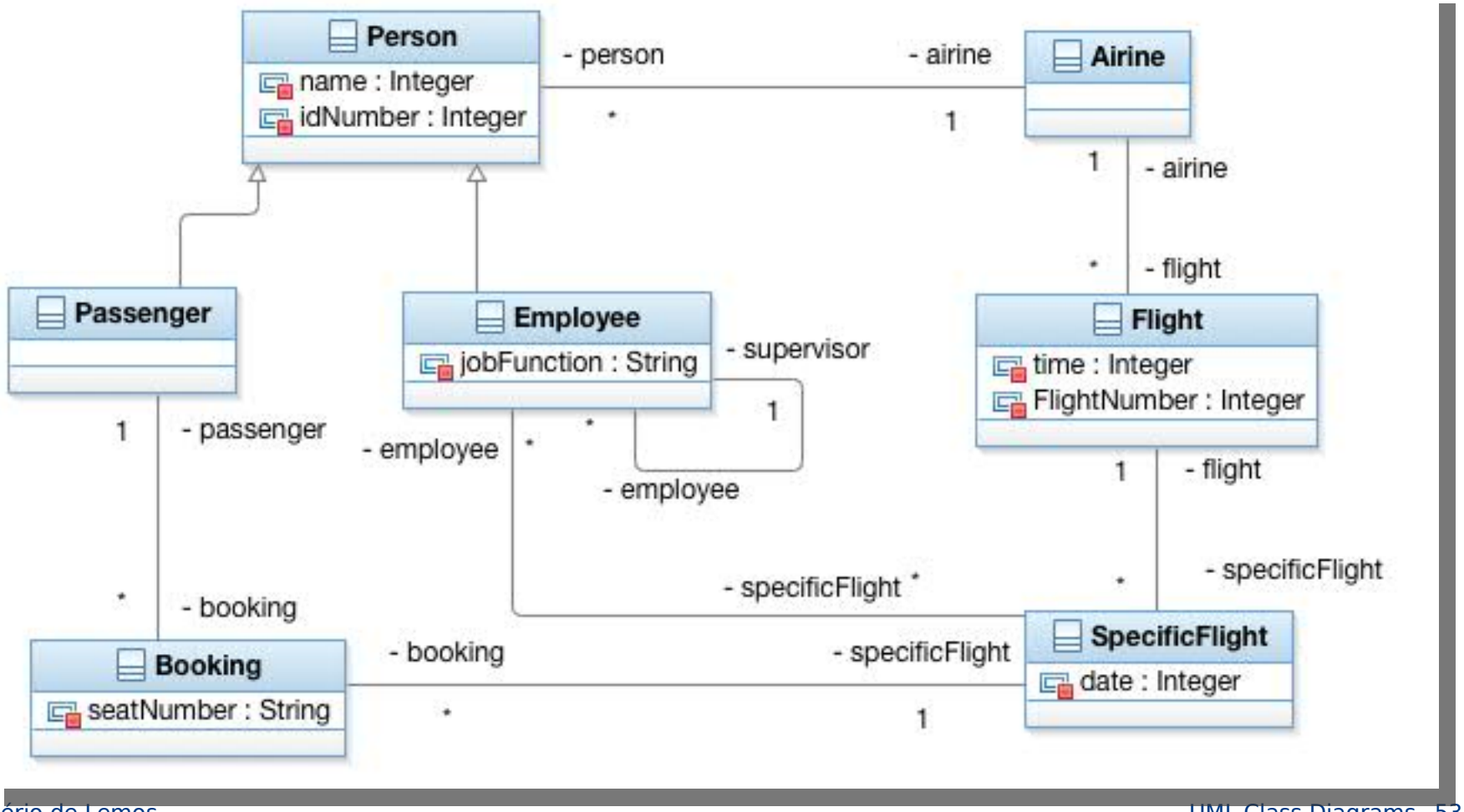
Example Domain Modelling

Airline reservation system

- Koffee Airlines runs a sightseeing flights from Java Valley, the capital of Koffee. The reservation system keeps track of passengers who will be flying in specific seats on various flights, as well as employees who will form the crew. For the crew, the system needs to track what everyone does, who supervises whom, and the passangers' bookings. Koffee Airlines runs several daily numbered flights on a regular schedule. Koffee Airlines expects to expand in the future, therefore the system needs to be flexible; in particular will be adding a frequent-flier program.

Example Domain Modelling

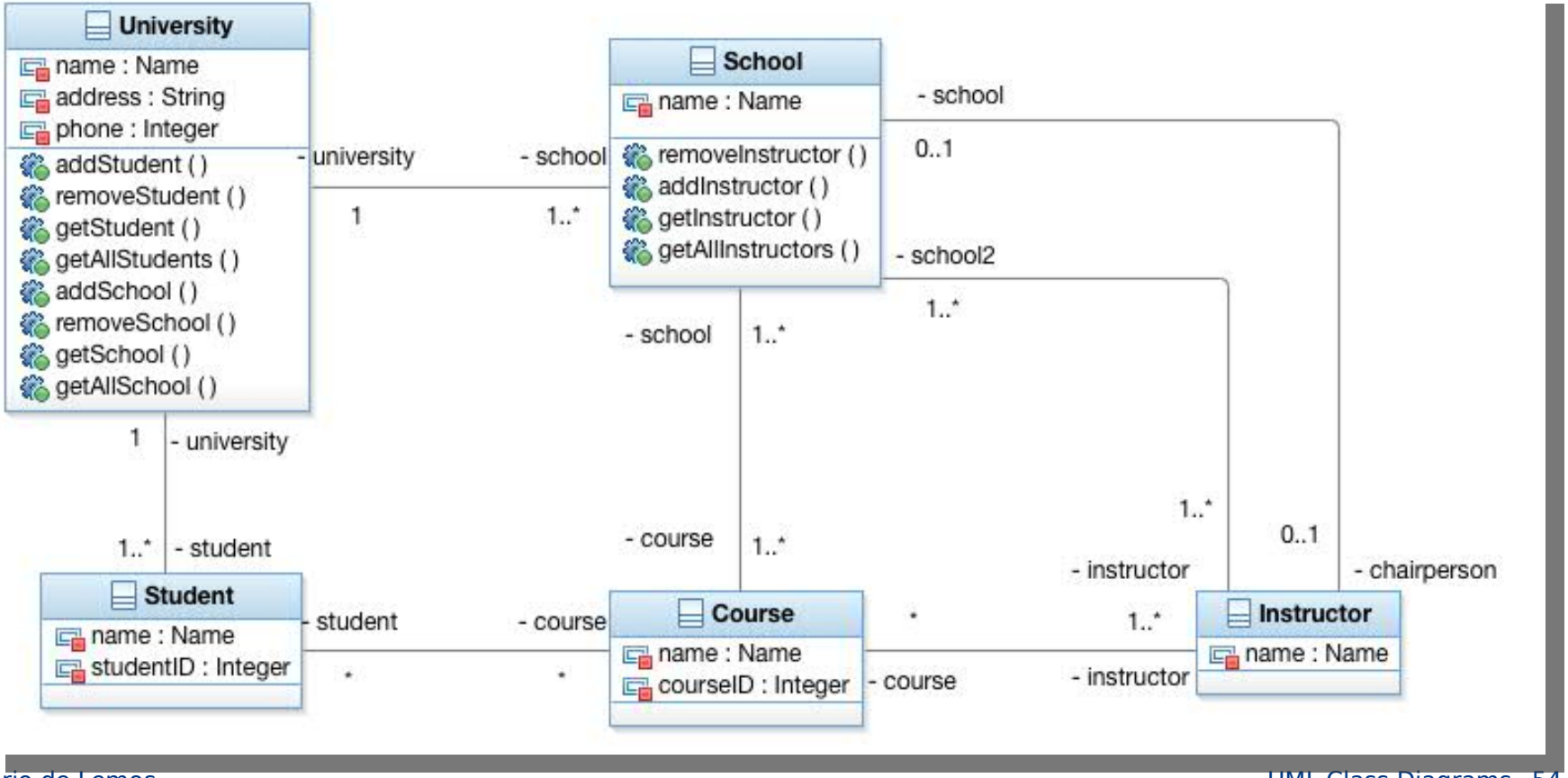
Airline reservation system



Example Domain Modelling

University

- describe the following UML class diagram



UML Class Diagrams: Hints and Tips

Every class diagram is a graphical representation of the static design view of a system

A well structured class diagram

- focus on communicating one aspect of the system's static design view
- contains only elements that are essential to understand that aspect
- provides detail consistent with its level of abstraction

UML Class Diagrams: Hints and Tips

When drawing a class diagram

- give it a name that communicates purpose
- lay out its elements to minimize lines that cross
- organise its elements spatially so that things that are semantically close are laid out physically close
- use notes to draw attention to some important features
- try not to show too many kinds of relationships