University of Kent

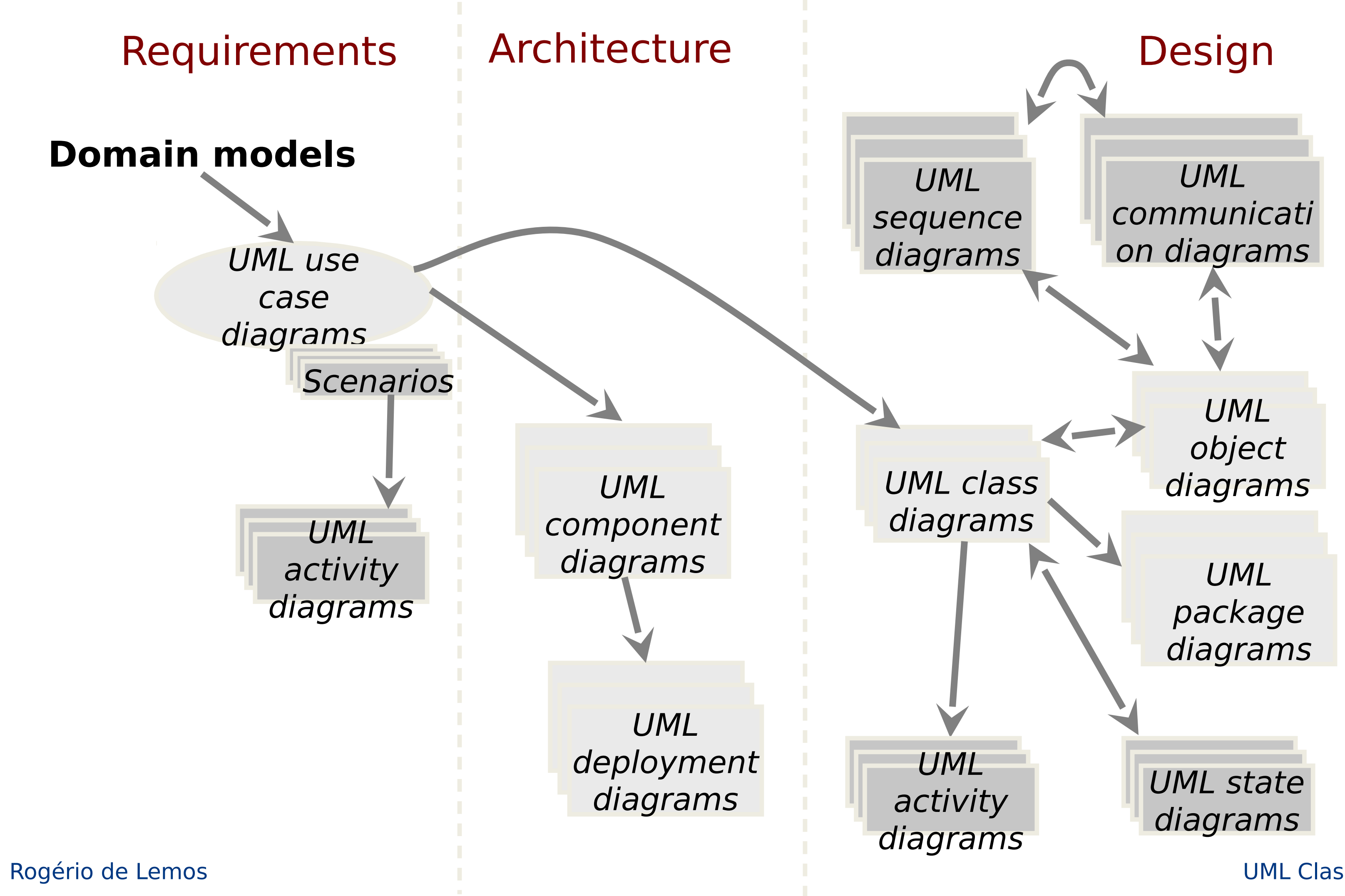
Software Engineering

Rogério de Lemos

- UML Class diagrams

OO Designs in the UML

How UML is used in the development process



Lecture Outline

Some of the topics

- Three perspectives
- UML Class diagrams
- Classes, attributes, operations
- Multiplicities, navigation
- An implementation in Java

Perspectives of Class Diagrams

There are three interpretations for class diagrams

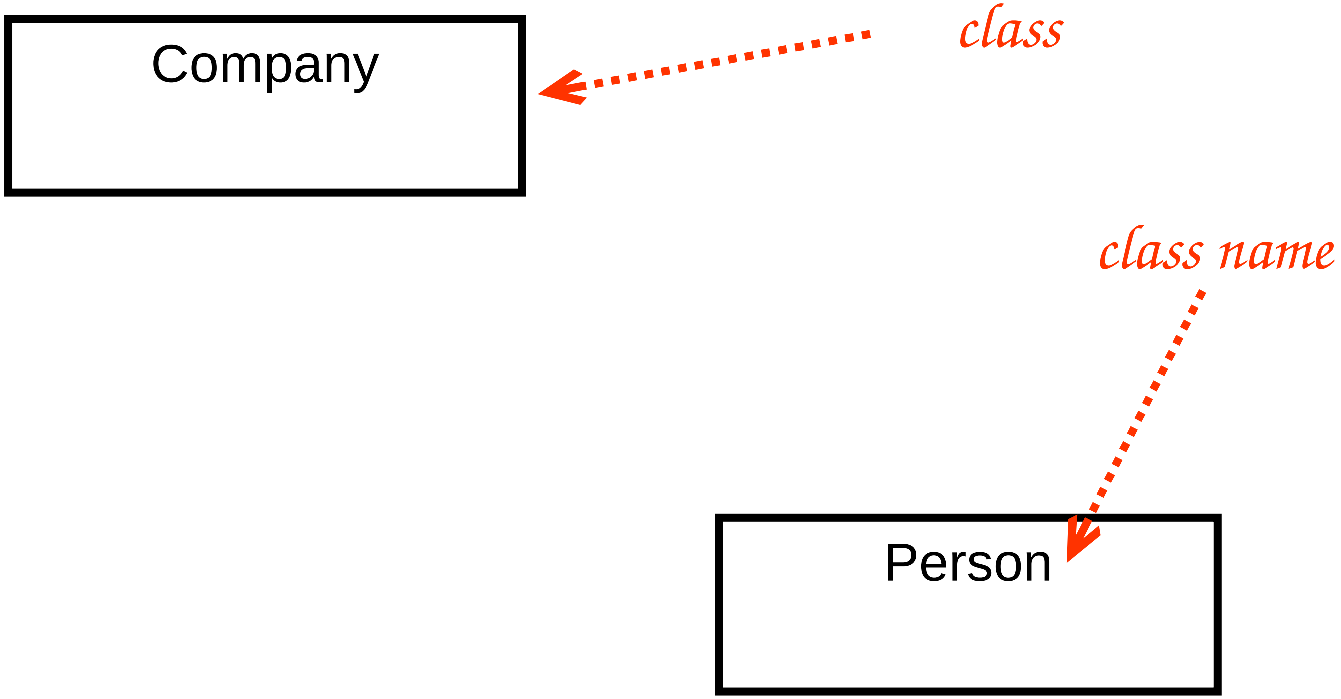
- conceptual (key domain abstraction)
  - classes represent concepts in the domain under study
- specification (interfaces of the software : not impl…)
  - classes define types in the system
- implementation (the code)
  - classes are mapped to Java classes in the code

Class Diagrams

Class diagrams

- describe the **types of objects** in the system, and the **static relationships** that exist among them
  - show type of structures for the objects
- show the **properties** and **operations** of a class
  - feature is a term that covers properties and operations of a class
- show the **constraints** that apply to the way objects are connected

Class Diagram Notation



Properties

Properties

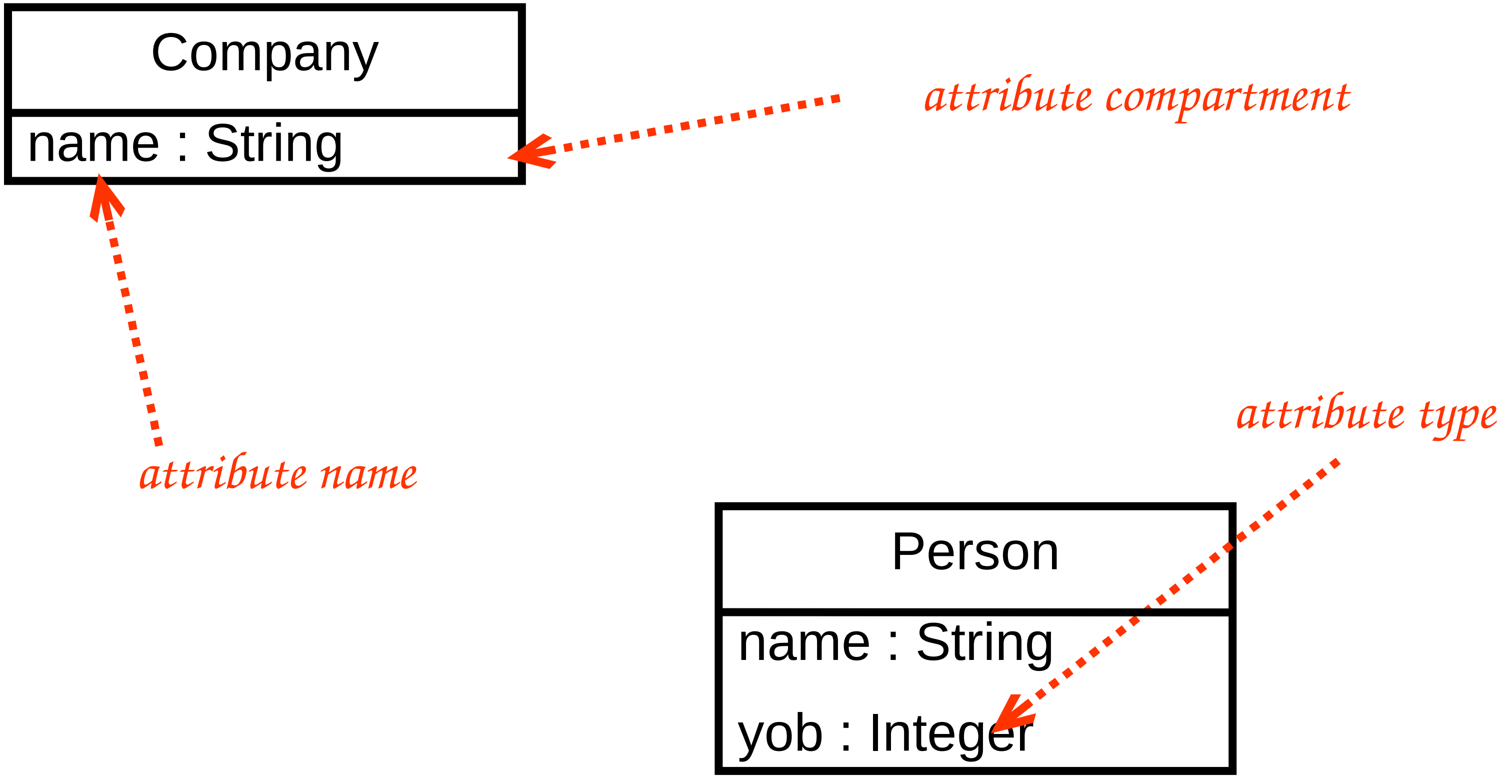
- represent structural features of a class
- properties correspond to fields in the class
  - ***attributes*** and ***associations***
    - they are quite the same thing

Attributes

An attribute

- describes a property as a line of text within the class box itself
  - visibility name : type multiplicity = default {property-string}
- an example
  - - title: String [1] = “Untitled” {readonly}

Class Diagram Notation

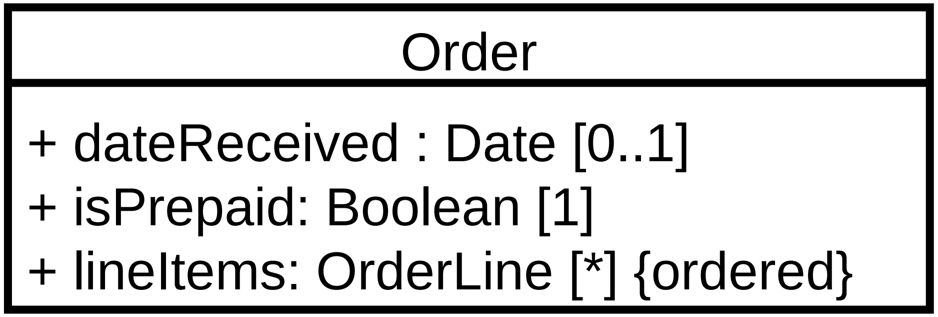


Associations

Associations

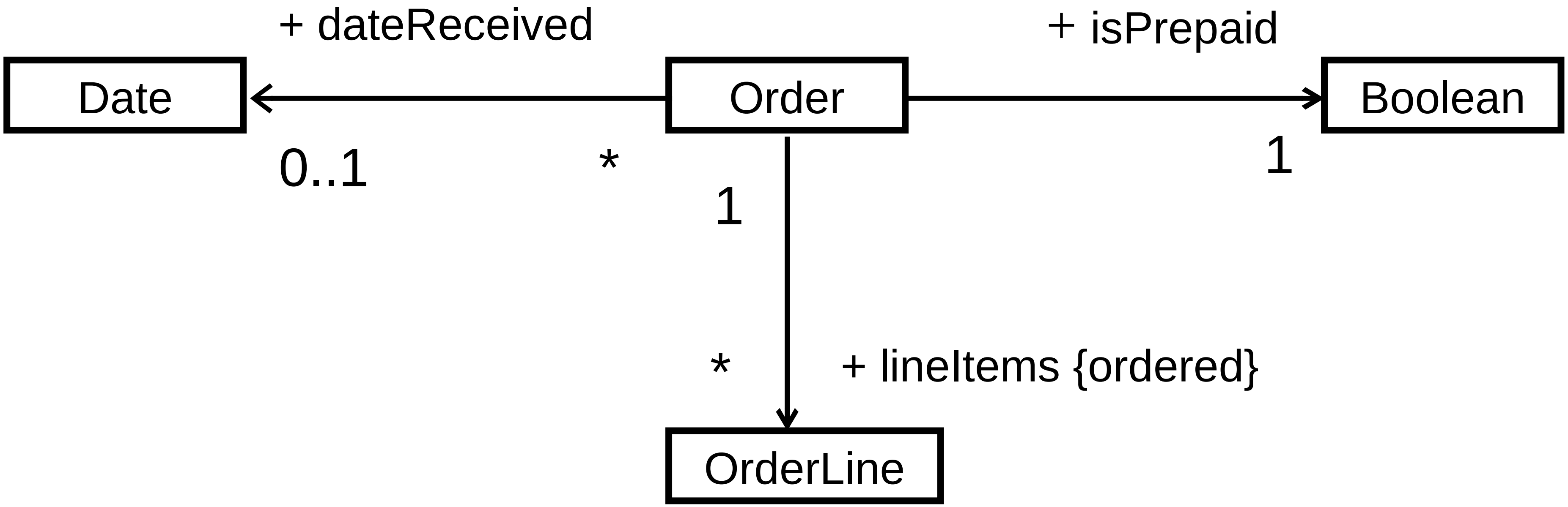
- an alternative way to notate a property
  - the same information that appears on an attribute can be shown on an association (see next slide)
  - attributes are used for small things (e.g. Booleans and dates)
  - associations to significant classes
- a solid line between two classes, directed from the source class to the target class
  - the name of property goes on the target end
- associations can show multiplicity at both ends

Associations

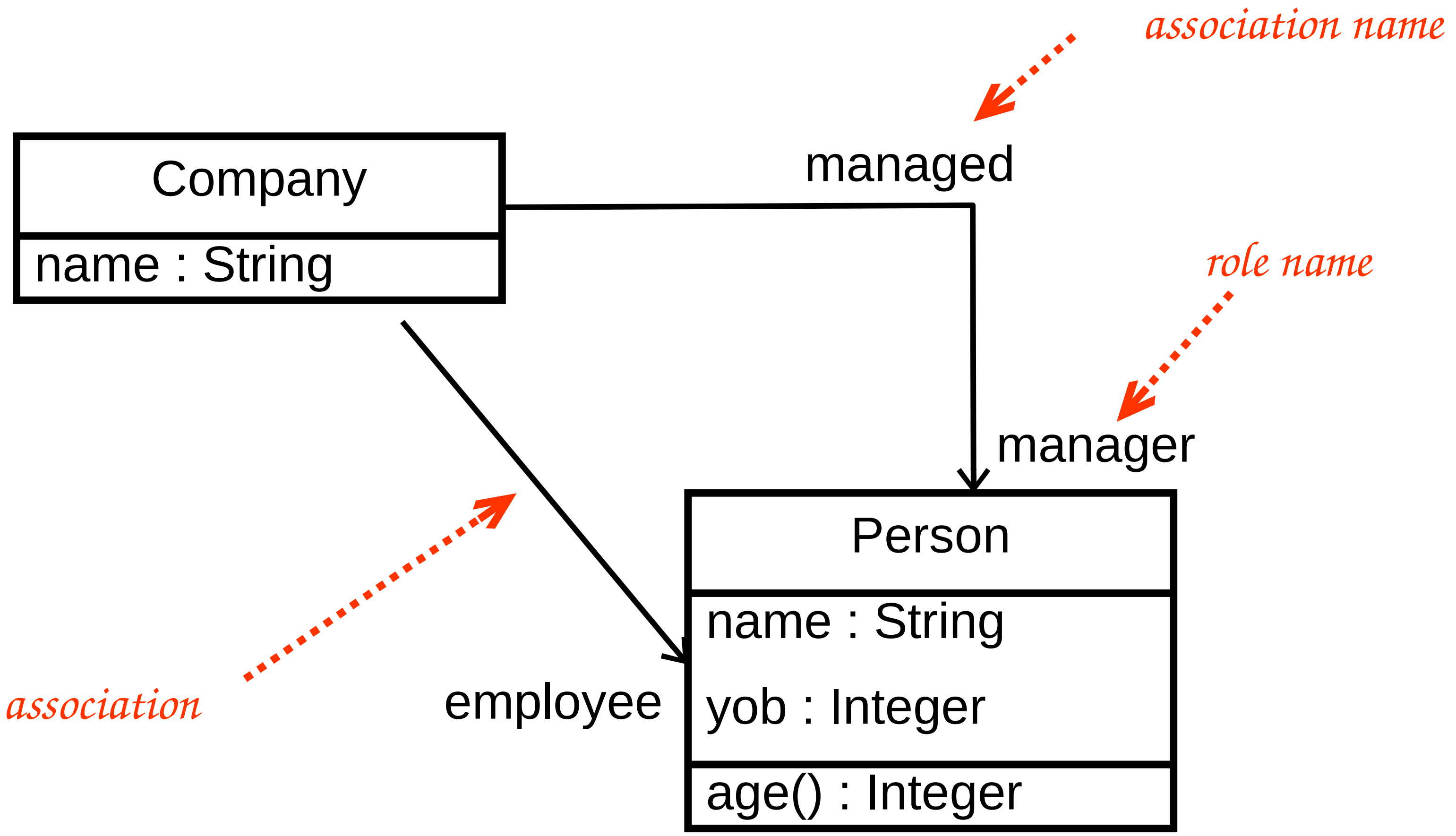


Order

+ dateReceived : Date [0..1]
+ isPrepaid: Boolean [1]
+ lineItems: OrderLine [*] {ordered}



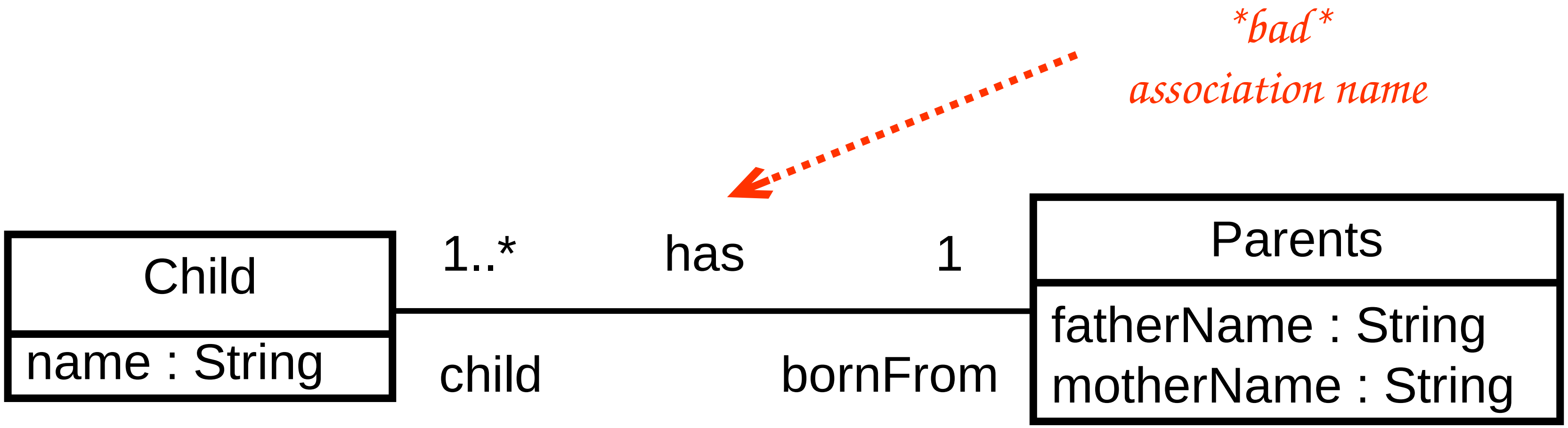
Class Diagram Notation



Roles

Instead of using an association name identify roles

- it’s more readable to have roles that objects play in the association
- association name **and** roles are not advisable

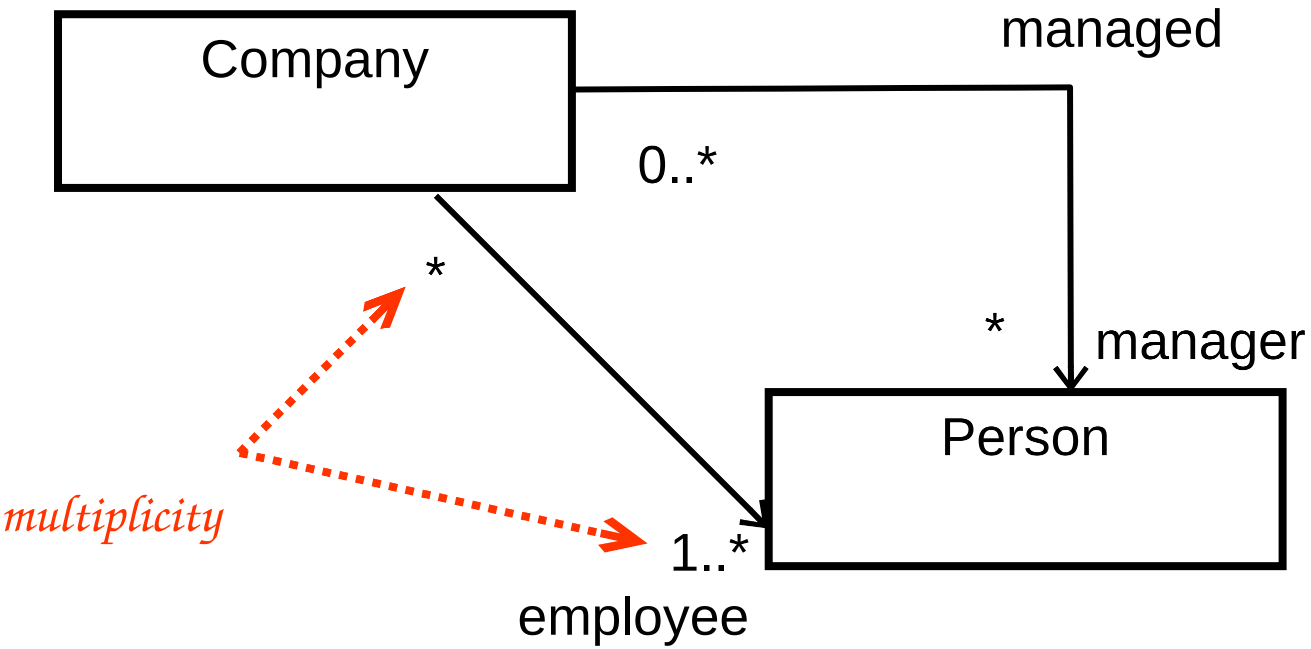


Multiplicity

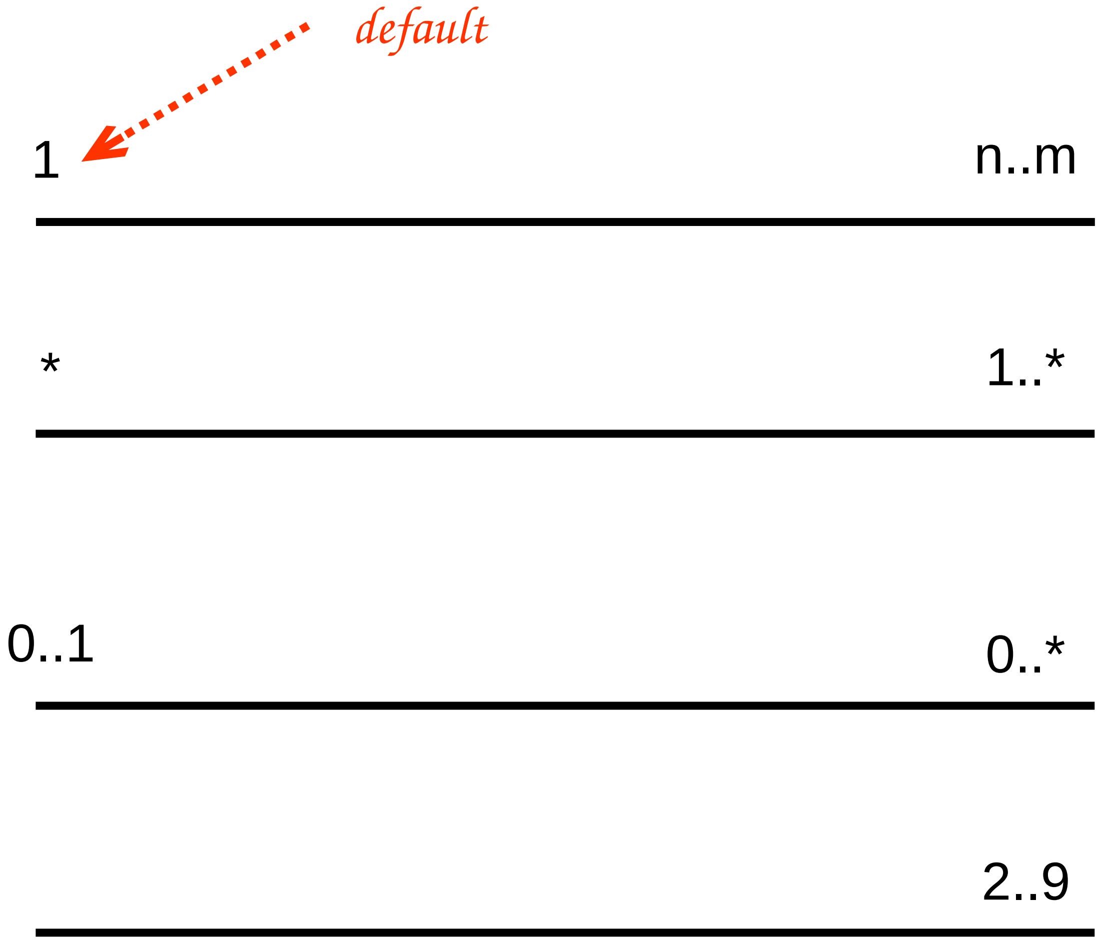
Multiplicity

- how many objects may fill the property
  - an exact number
    - **1** – an order must have exactly one customer
  - a range of numbers
    - **0..1** – a corporate customer may or may not have a single sales representative
  - an arbitrary, unspecified number
    - ***** - a customer needs not to place an Order and there is no upper limit to the number of Orders a Customer may place – zero or more orders

Multiplicity



Multiplicity



default

1 n..m

* 1..*

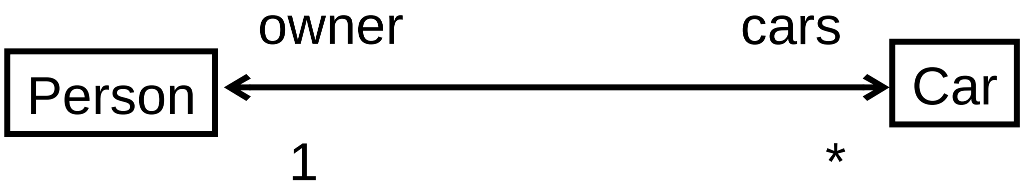
0..1 0..*

2..9

Navigation

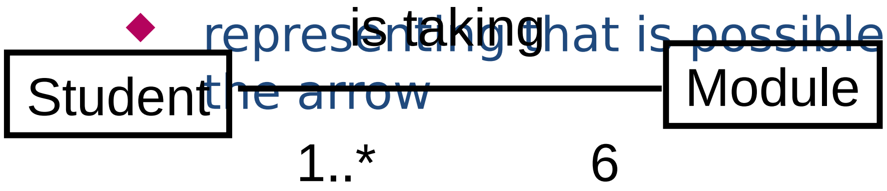
In addition to unidirectional associations there are also **bidirectional associations**

- a pair of properties that are linked together as inverses;
  - the Car class has property owner : Person [1], and the Person class has a property cars : Car [*];



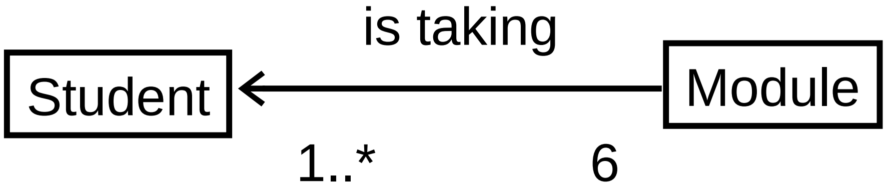
Navigation

- for each object of class Student there are six objects of class Module that are associated with Student
- for each object of class Module there are some Student objects that are associated with Module
- it doesn’t record the direction of the association
  - representing that is possible to send messages in the direction of the arrow

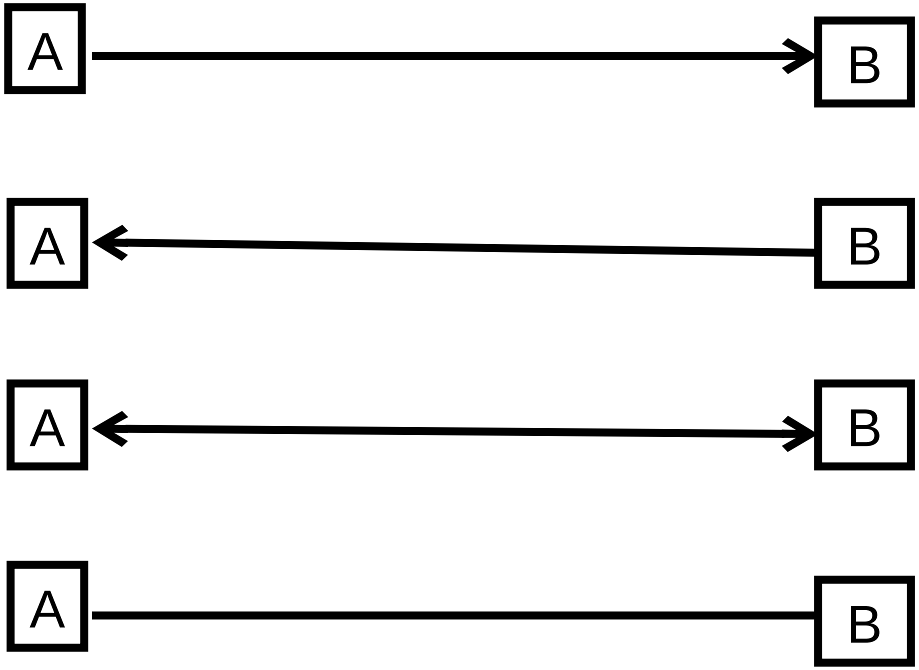


Navigation

- objects type Module can send messages to the objects type Students
  - not vice-versa
- Module knows about Student
  - not vice-versa



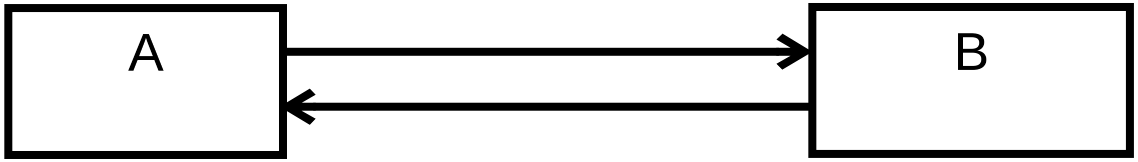
Navigation

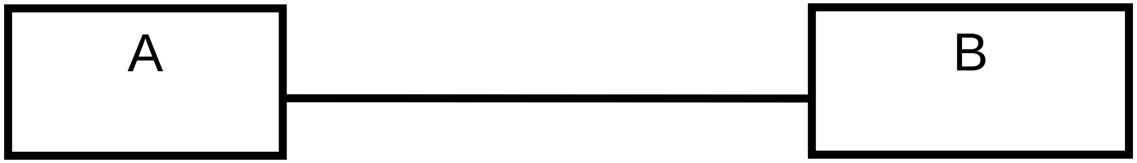


context a:A a.b	**context** b:B b.a
✓	✗
✗	✓
✓	✓
✓	✓

Navigation

- What’s the difference ?





Instantiation

The state of an object-oriented system

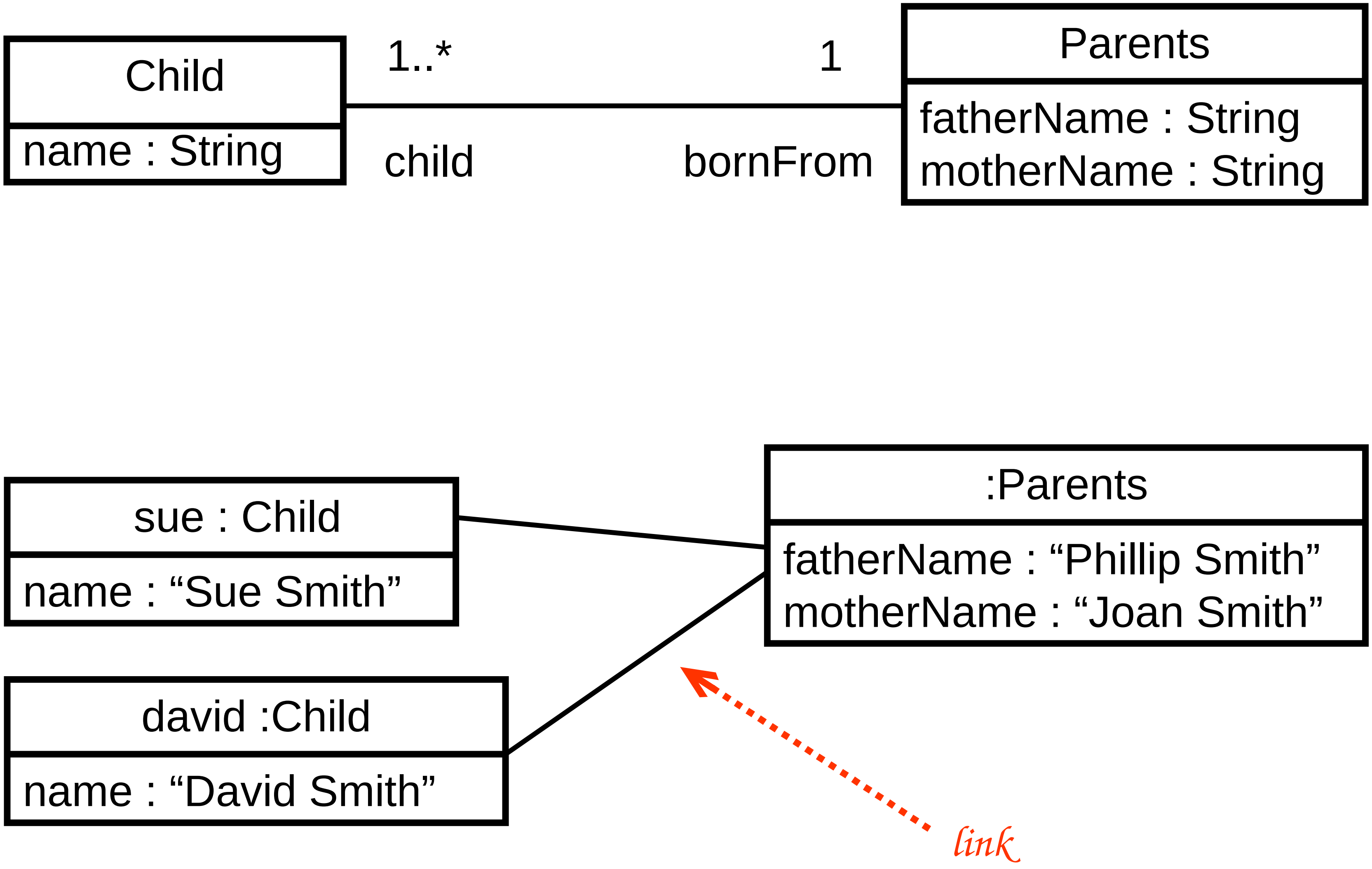
- a collection of objects, with values for their attributes
- a collection of links between objects

Associations

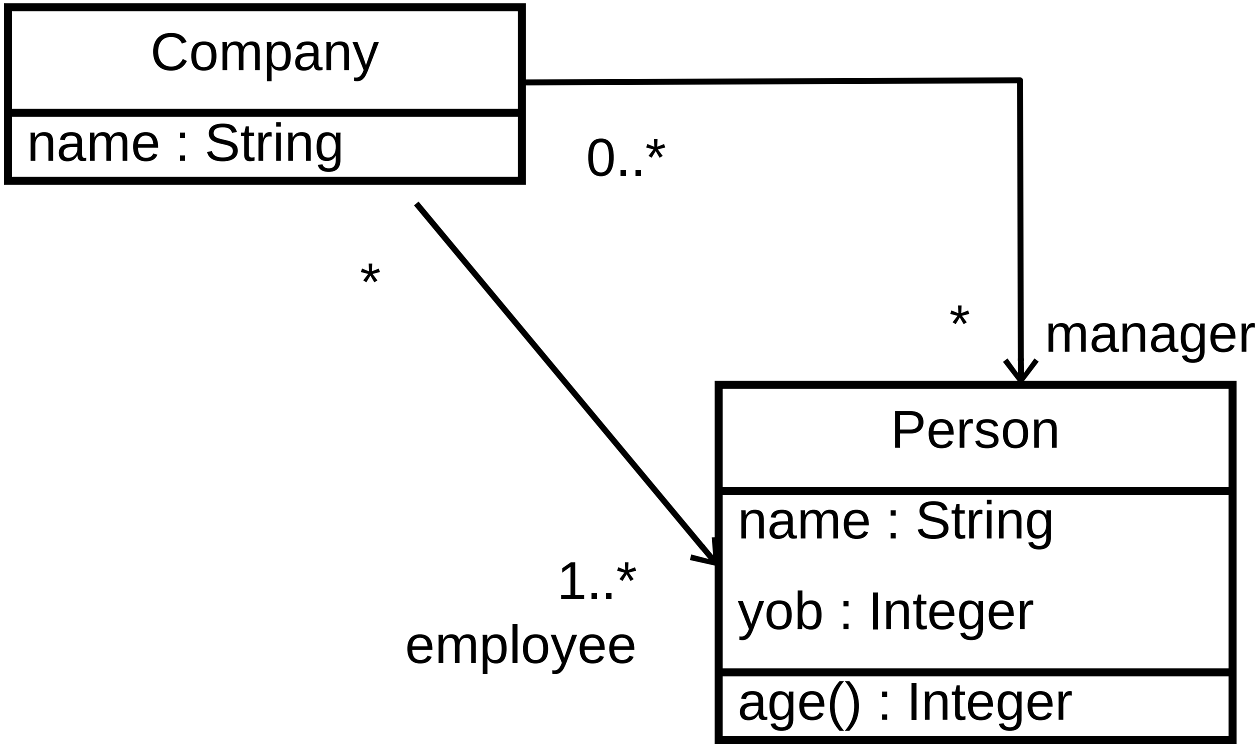
In terms of object diagrams

- associations are the type construct for links
  - objects -> classes
  - links -> associations
- show relationship types between objects
- define navigation paths

Instantiation: Example 1



Instantiation: Example 2



Company

name : String

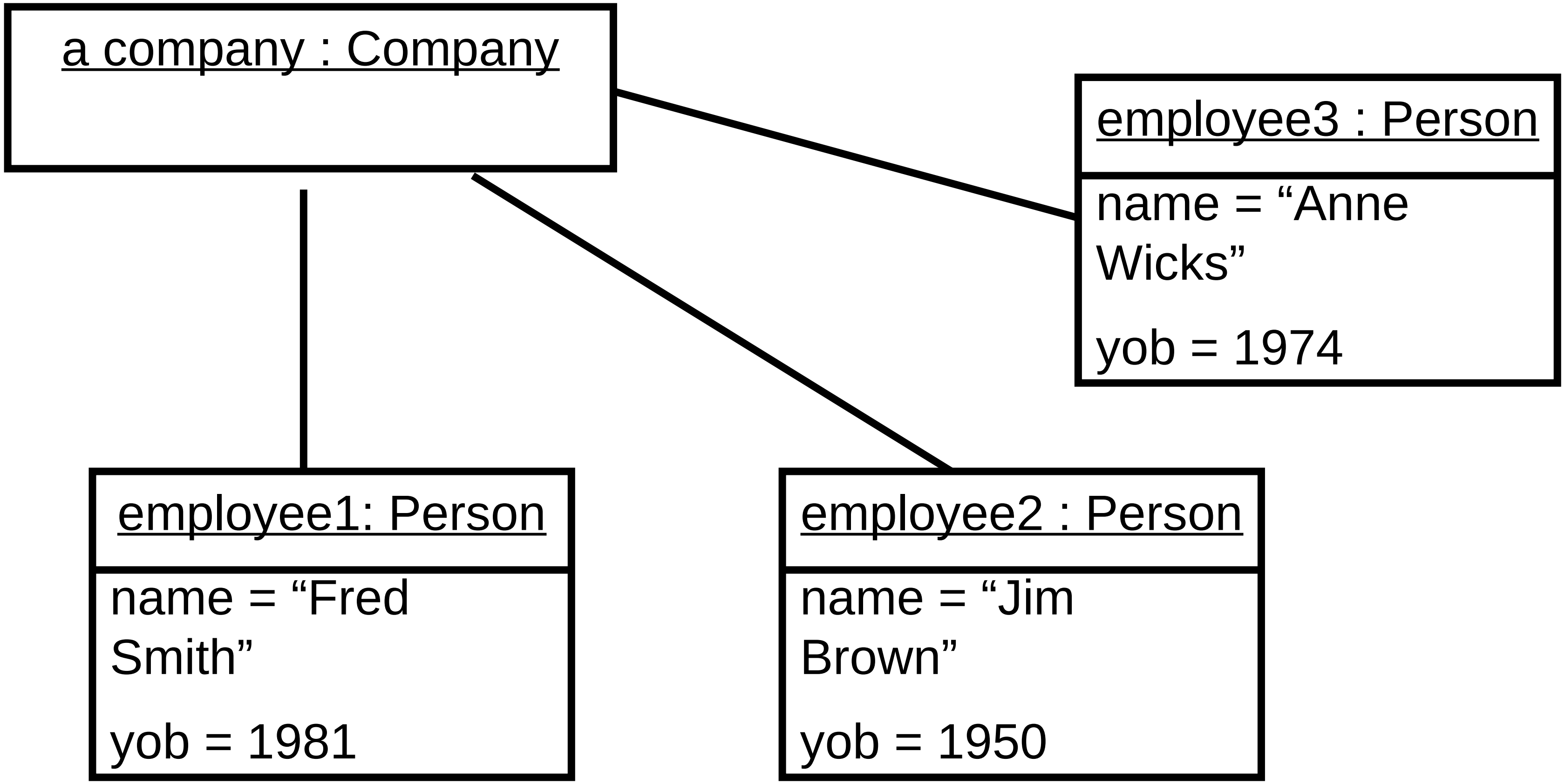
Person

name : String

yob : Integer

age() : Integer

Instantiation: Example 2



a company : Company

employee3 : Person

name = “Anne Wicks”

yob = 1974

employee1: Person

name = “Fred Smith”

yob = 1981

employee2 : Person

name = “Jim Brown”

yob = 1950

Operations

An **operation** is a service that can be requested from an object (instance of a class)

- correspond to methods of the class

An operation has a signature, which describes the actual parameters that are possible (including possible return values)

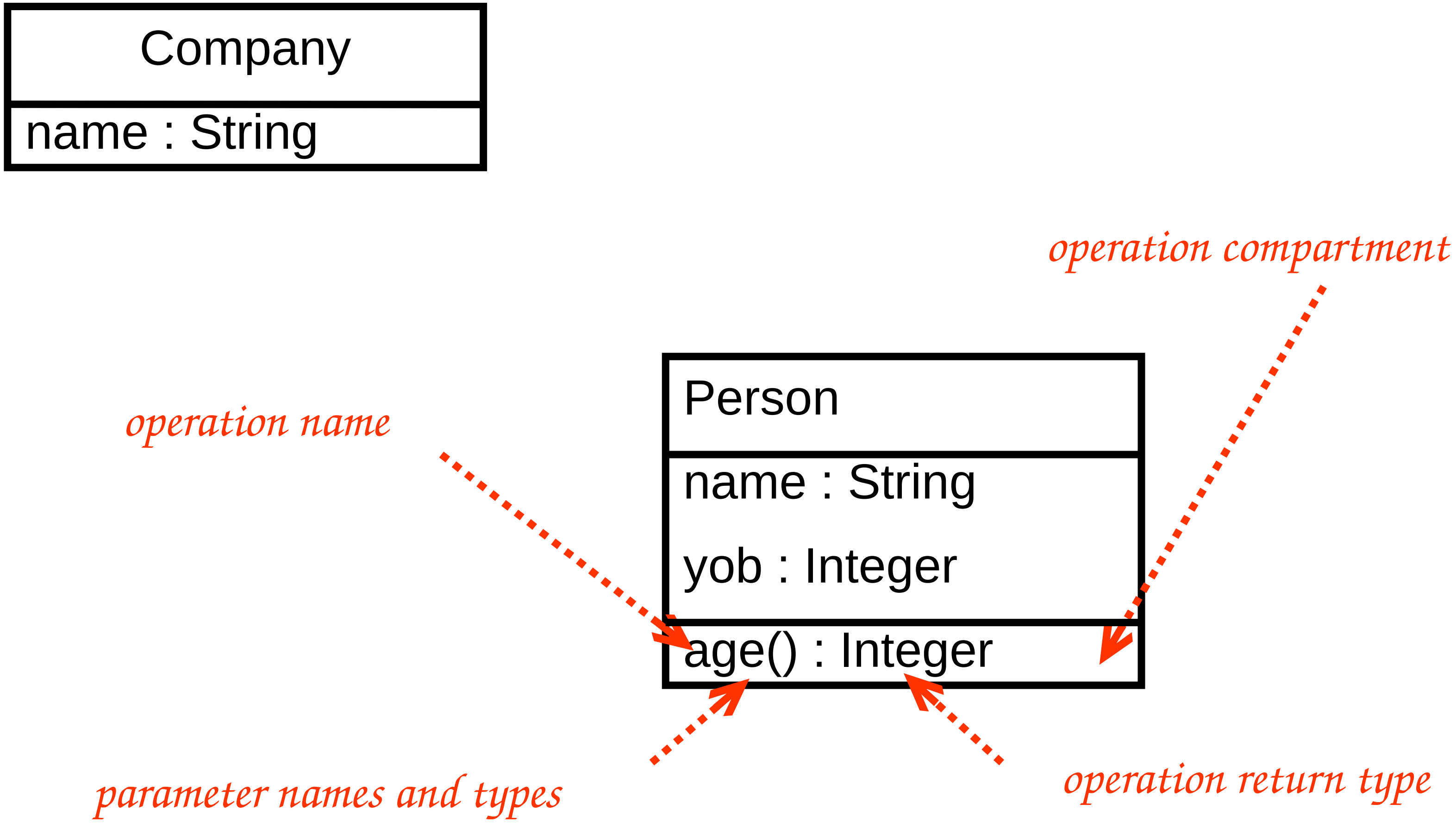
- UML syntax for operations
  - visibility name (par_list) : return_type {property-string}
- example
  - + setName(n: String) : void

Operations

The most important operations

- those which define how a class interacts
  - avoid to show operations that manipulate properties

Class Diagram Notation



Company

name : String

operation compartment

Person

name : String

yob : Integer

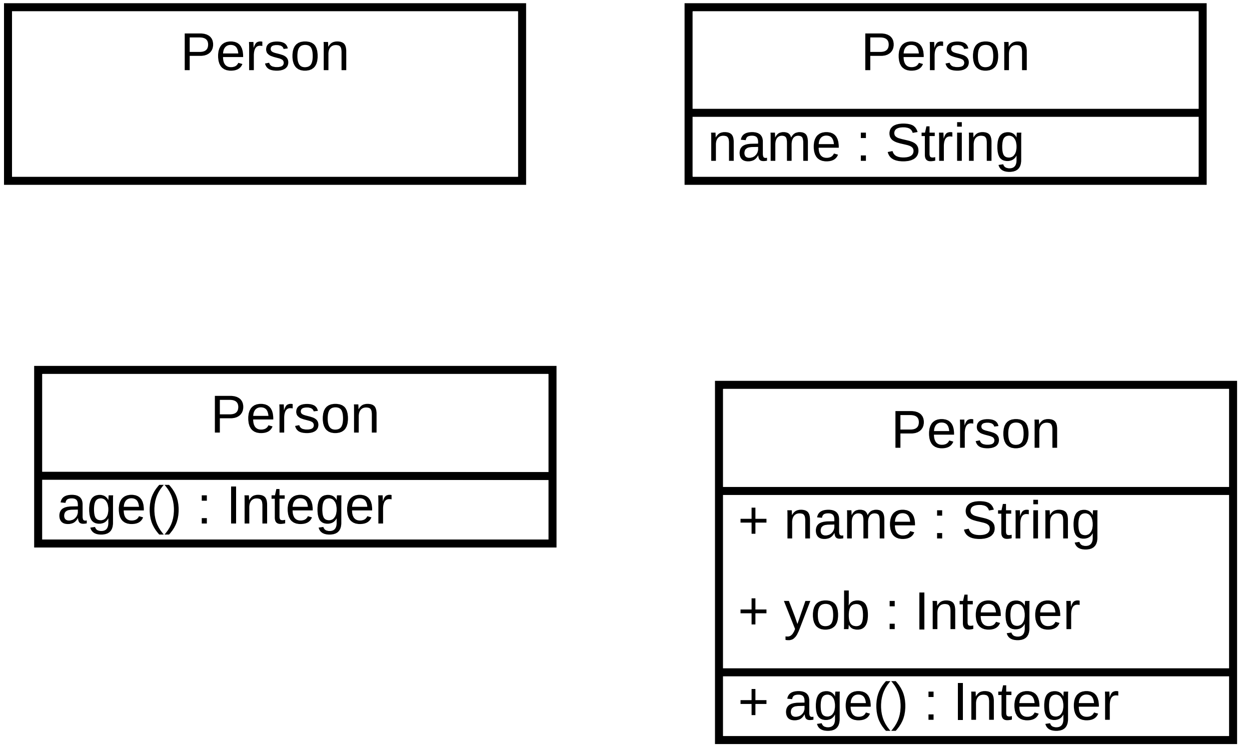
age() : Integer

operation name

parameter names and types

operation return type

Class Diagram Notation



Person

Person

name : String

Person

age() : Integer

Person

+ name : String

+ yob : Integer

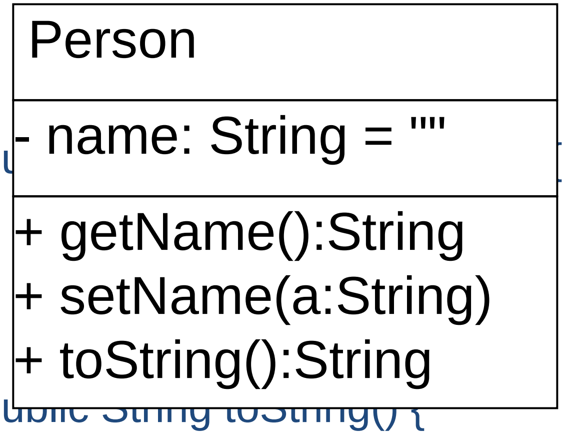
+ age() : Integer

Simple Implementation in Java

```
public class Person {
	private String name="";
	public String getName() {
		return this.name;
	}

	pu                                  {

	}
	public String toString() {
		String detail = ...
		return detail;
	}
}
```



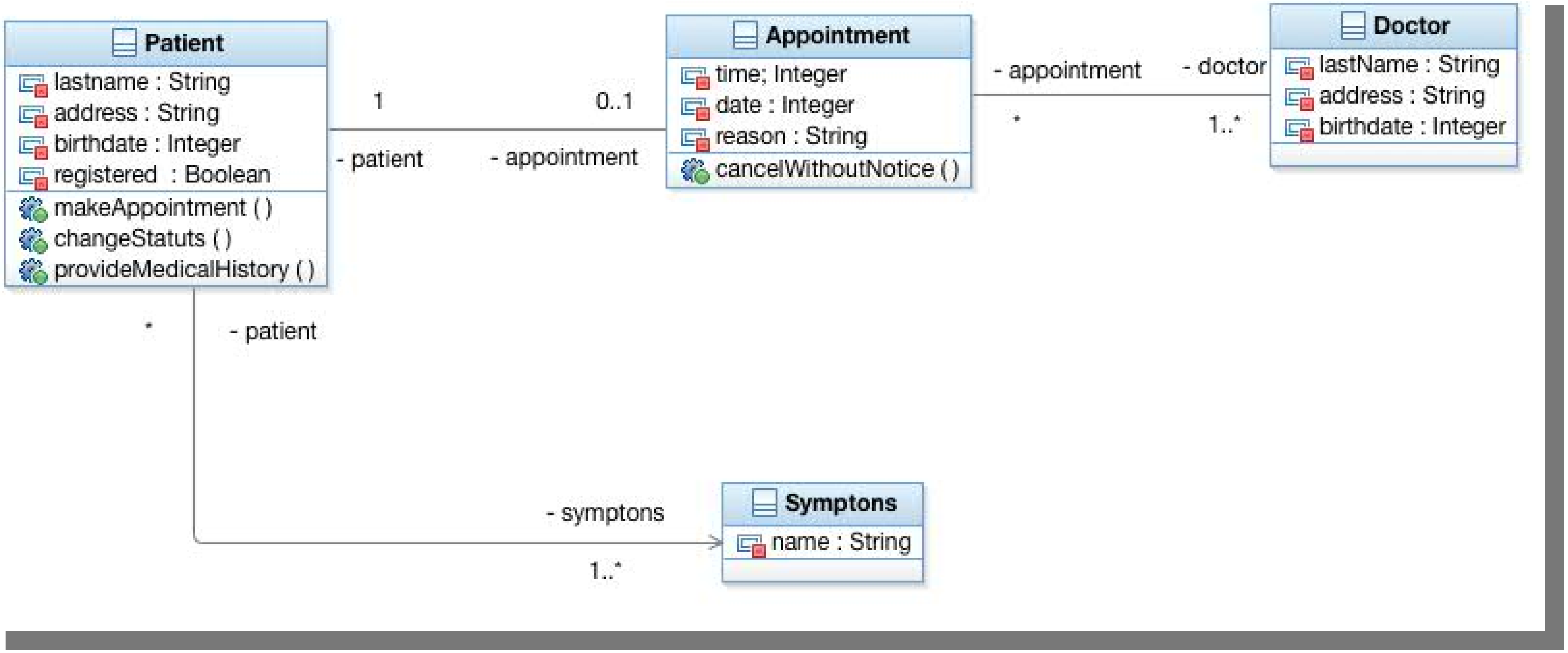
Person

- name: String = ""

+ getName():String
+ setName(a:String)
+ toString():String

Example: UML Class Diagram

Try to understand what the diagram below represents by instantiating it into objects



Patient

- lastname : String
- address : String
- birthdate : Integer
- registered : Boolean
- makeAppointment ( )
- changeStatuts ( )
- provideMedicalHistory ( )

Appointment

- time; Integer
- date : Integer
- reason : String
- cancelWithoutNotice ( )

Doctor

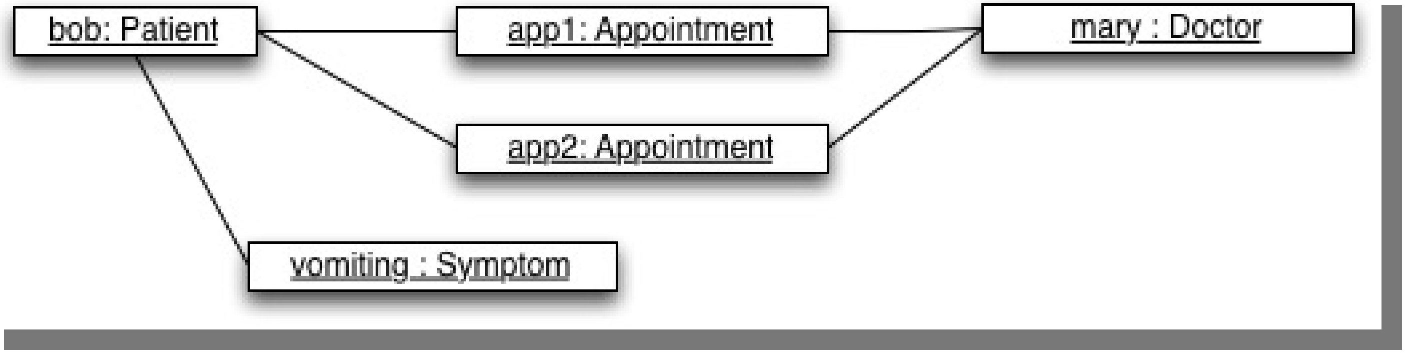
- lastName : String
- address : String
- birthdate : Integer

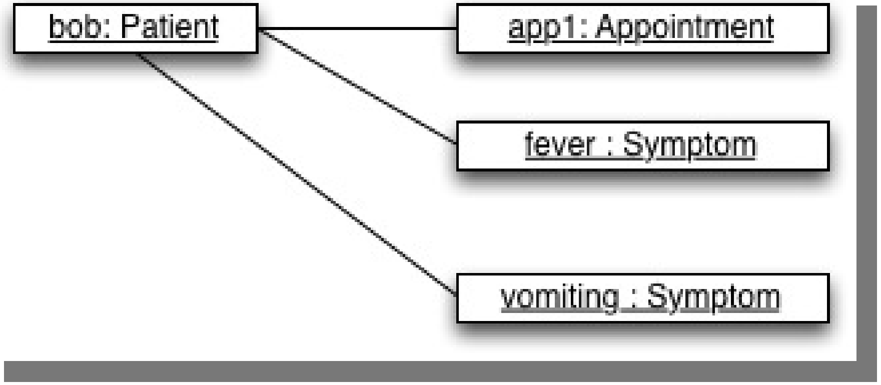
Symptons

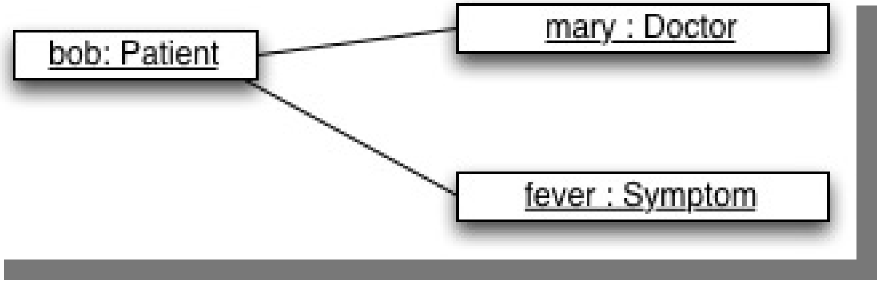
- name : String

Example: UML Class Diagram

Are these valid diagrams?

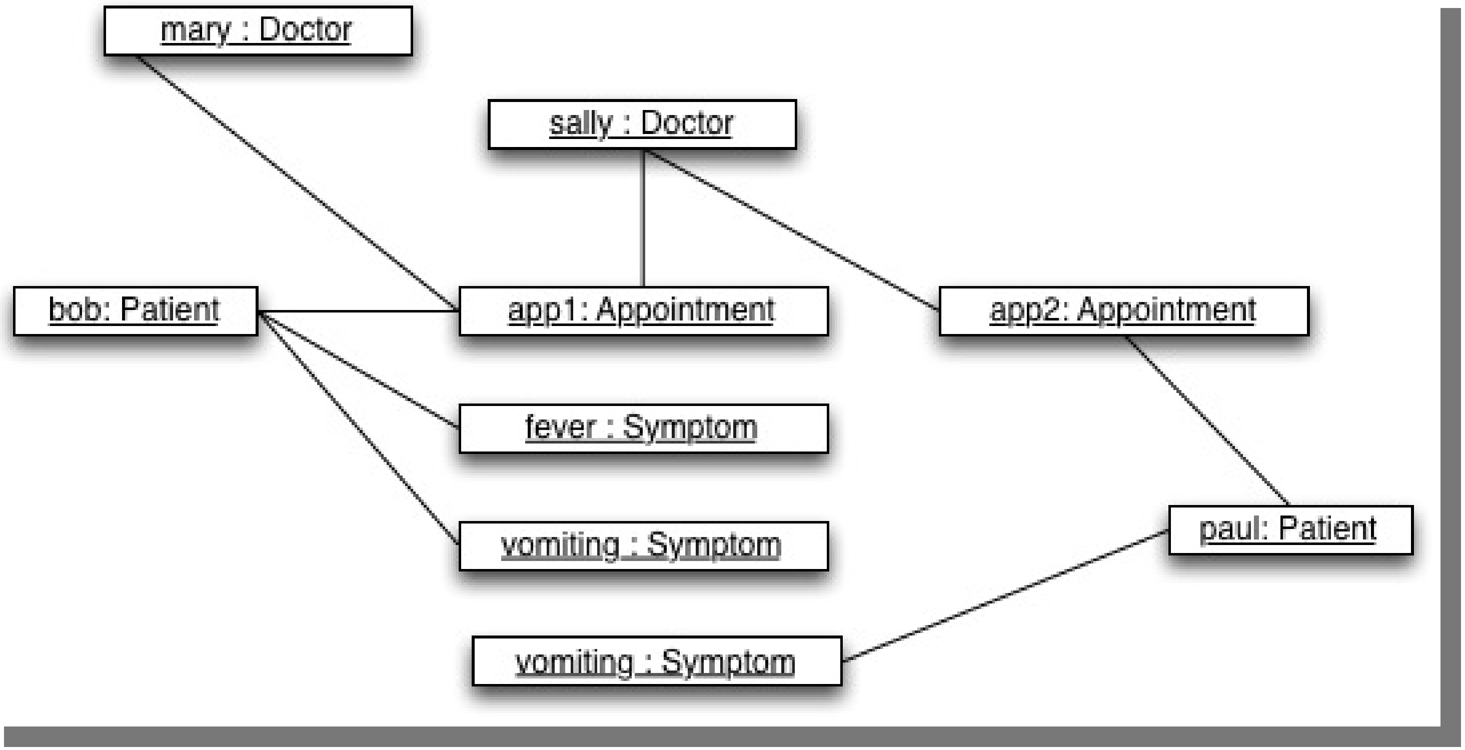






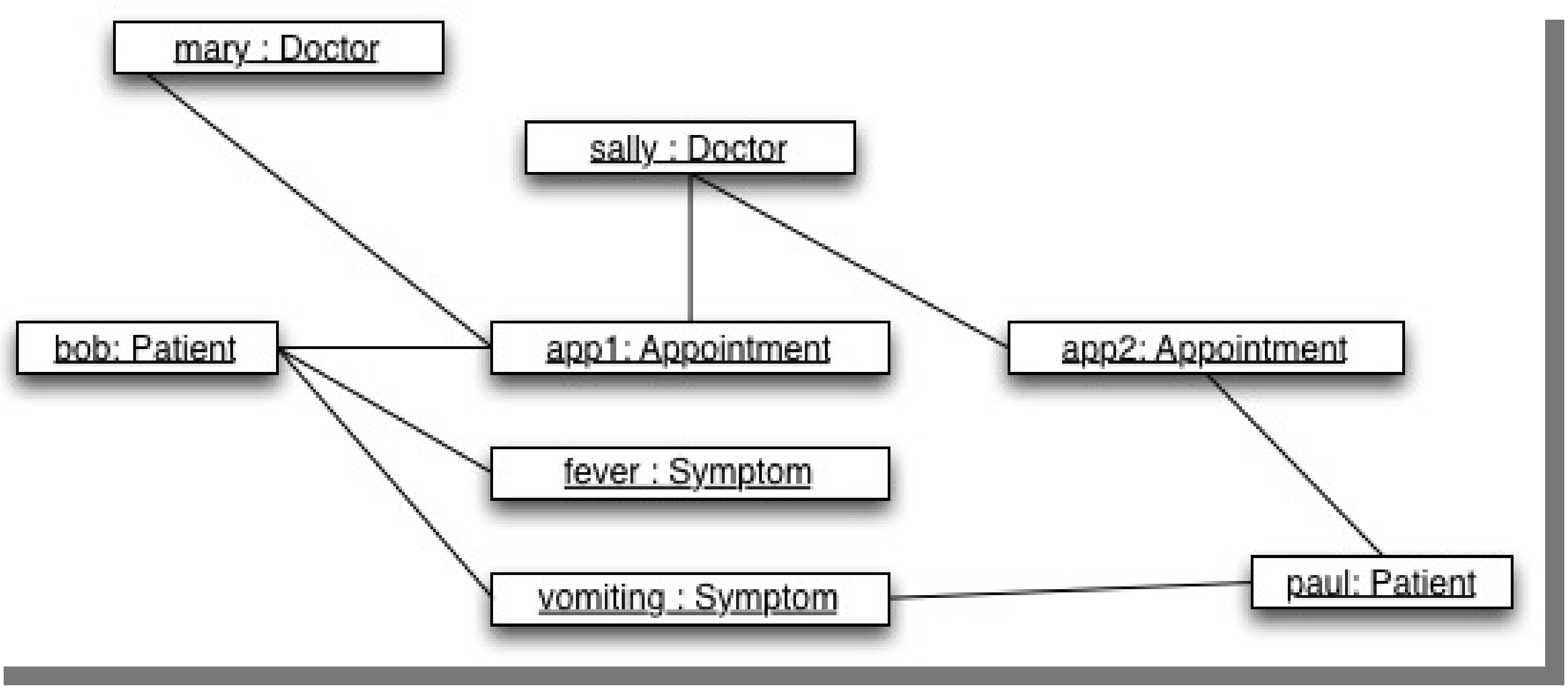
Example: UML Class Diagram

Is this a valid diagram?



Example: UML Class Diagram

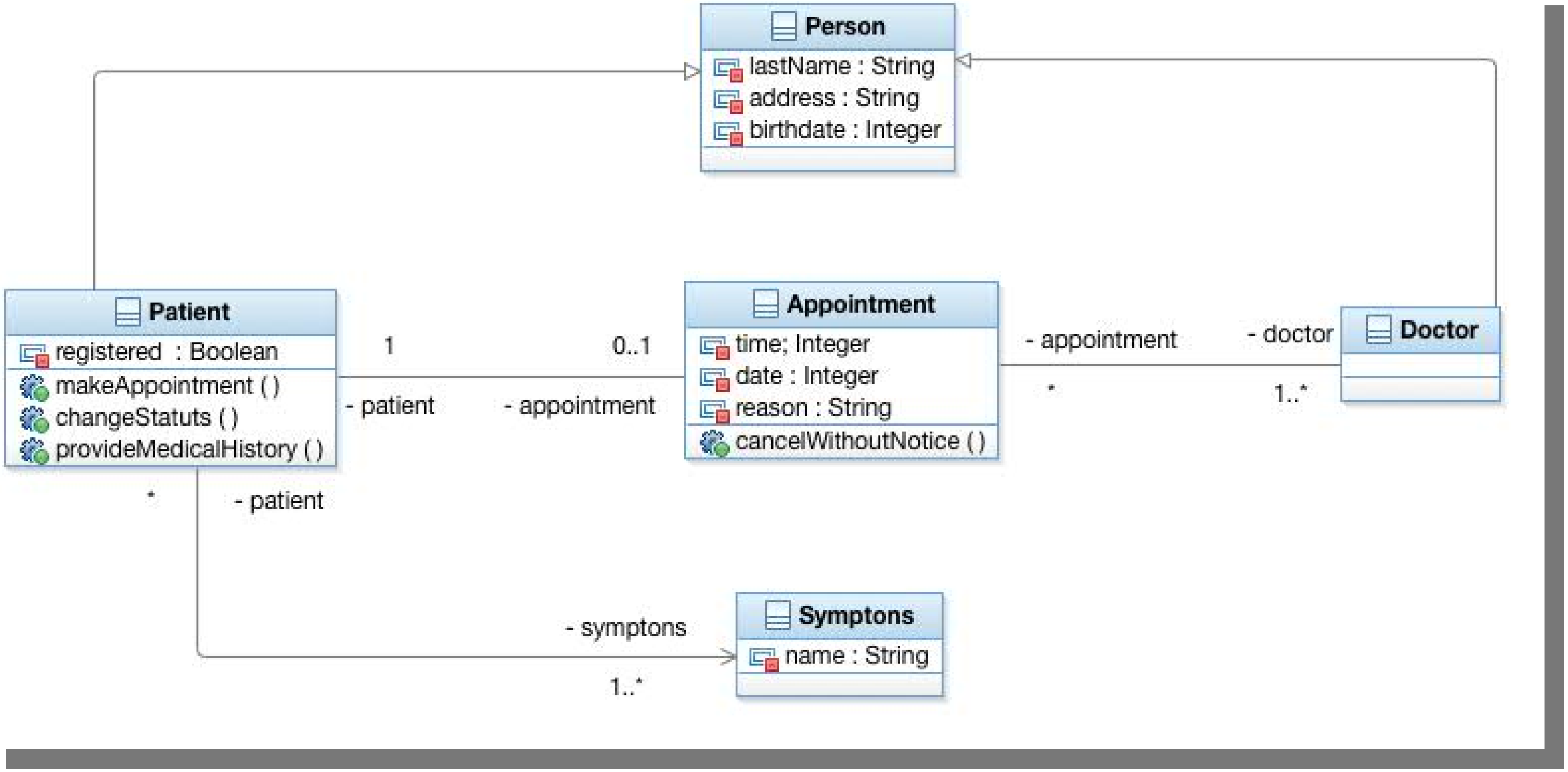
What about this, is it a valid diagram?



Example: UML Class Diagram

How to improve the diagram?

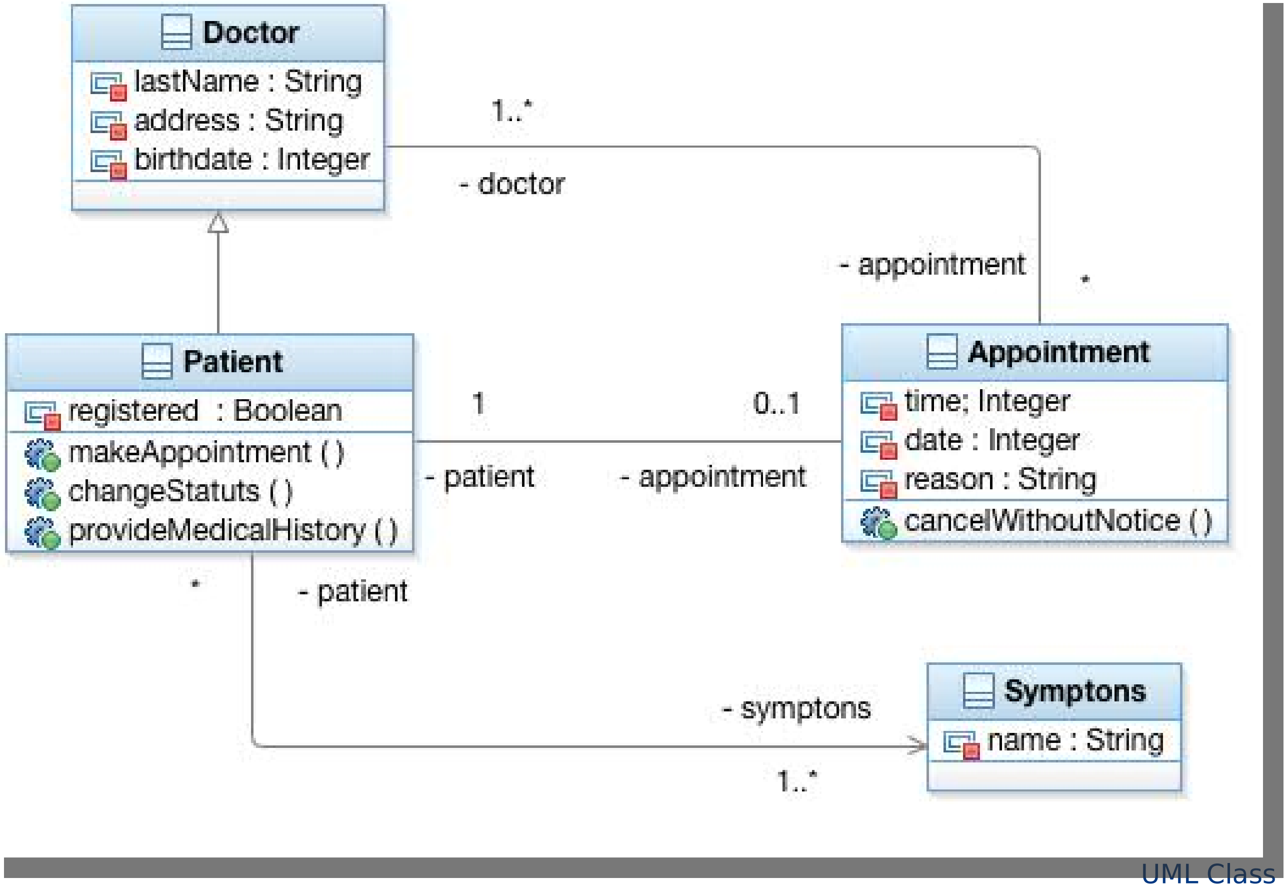
Solution 1 – what about this one?



Example: UML Class Diagram

How to improve the diagram?

Solution 2 – is this a meaningful one?


Doctor
lastName : String
address : String
birthdate : Integer
1..*
- doctor
- appointment
*
Patient
registered : Boolean
makeAppointment ( )
changeStatuts ( )
provideMedicalHistory ( )
1
0..1
- patient
- appointment
Appointment
time; Integer
date : Integer
reason : String
cancelWithoutNotice ( )
*
- patient
- symptons
Symptons
name : String
1..*