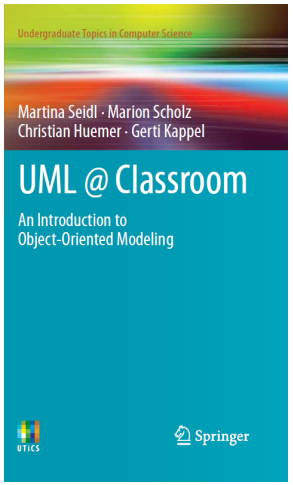
Class diagrams in UML

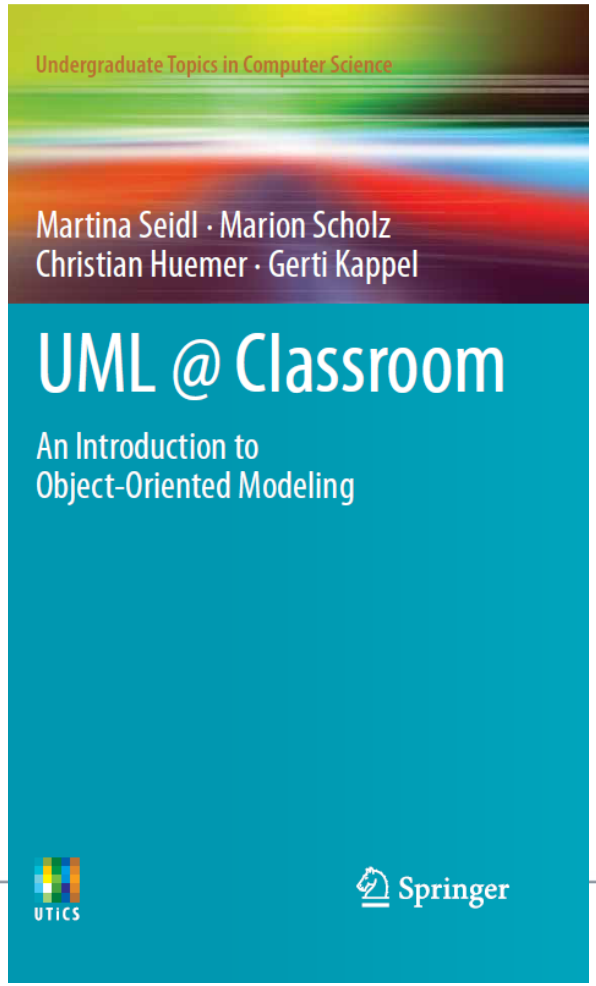


Slides accompanying UML@Classroom
Version 1.0.1

Originally designed by the Business Informatics Group @ TU Wien
Altered by Cor-Paul Bezemer

Literature

- The slides are based on the following book:



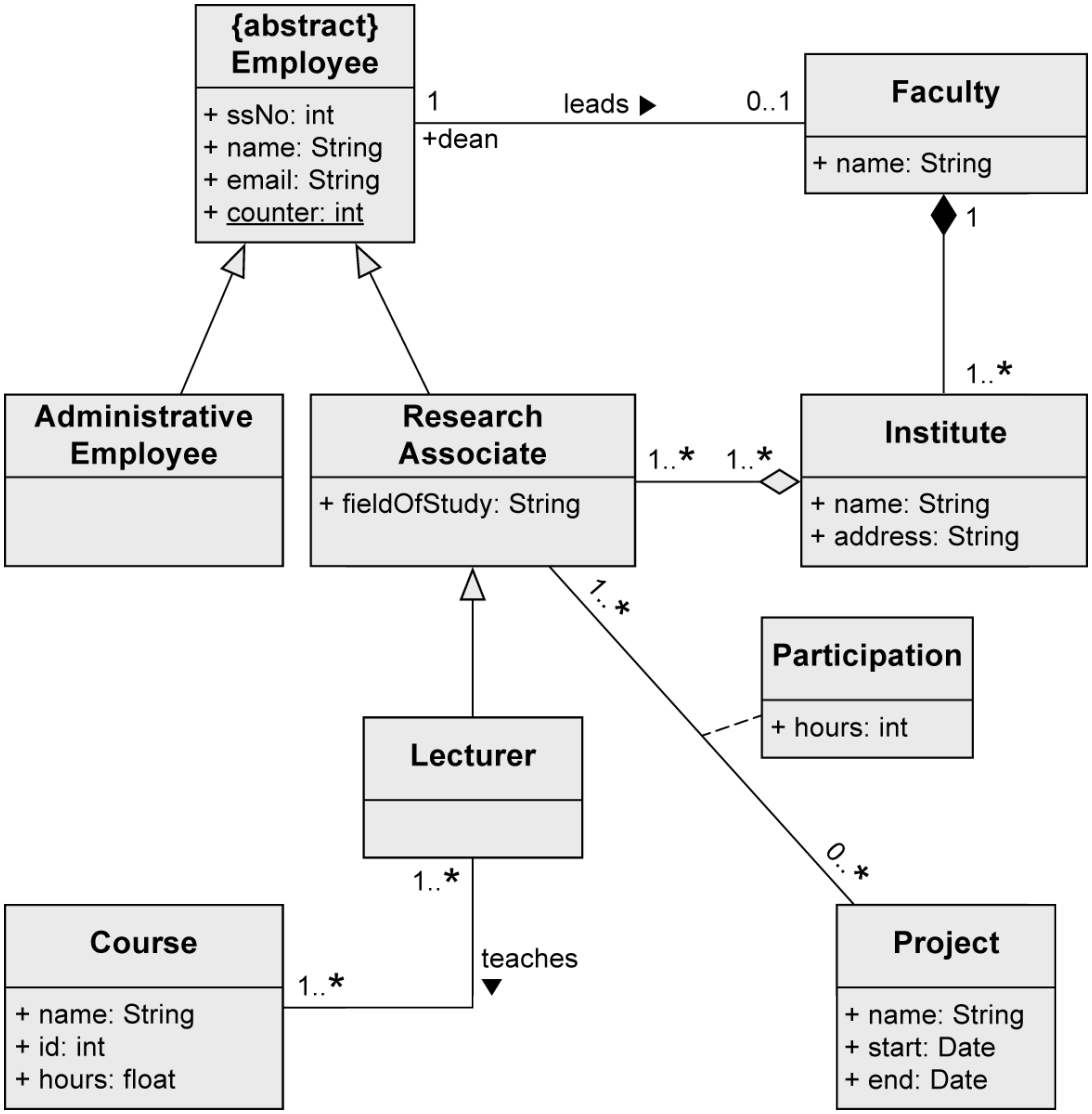
UML @ Classroom:
An Introduction to Object-Oriented Modeling
Martina Seidl, Marion Scholz, Christian Huemer and Gerti Kappel

Springer Publishing, 2015

ISBN 3319127411

The book is available as an eBook in the U of A library
https://www.library.ualberta.ca/

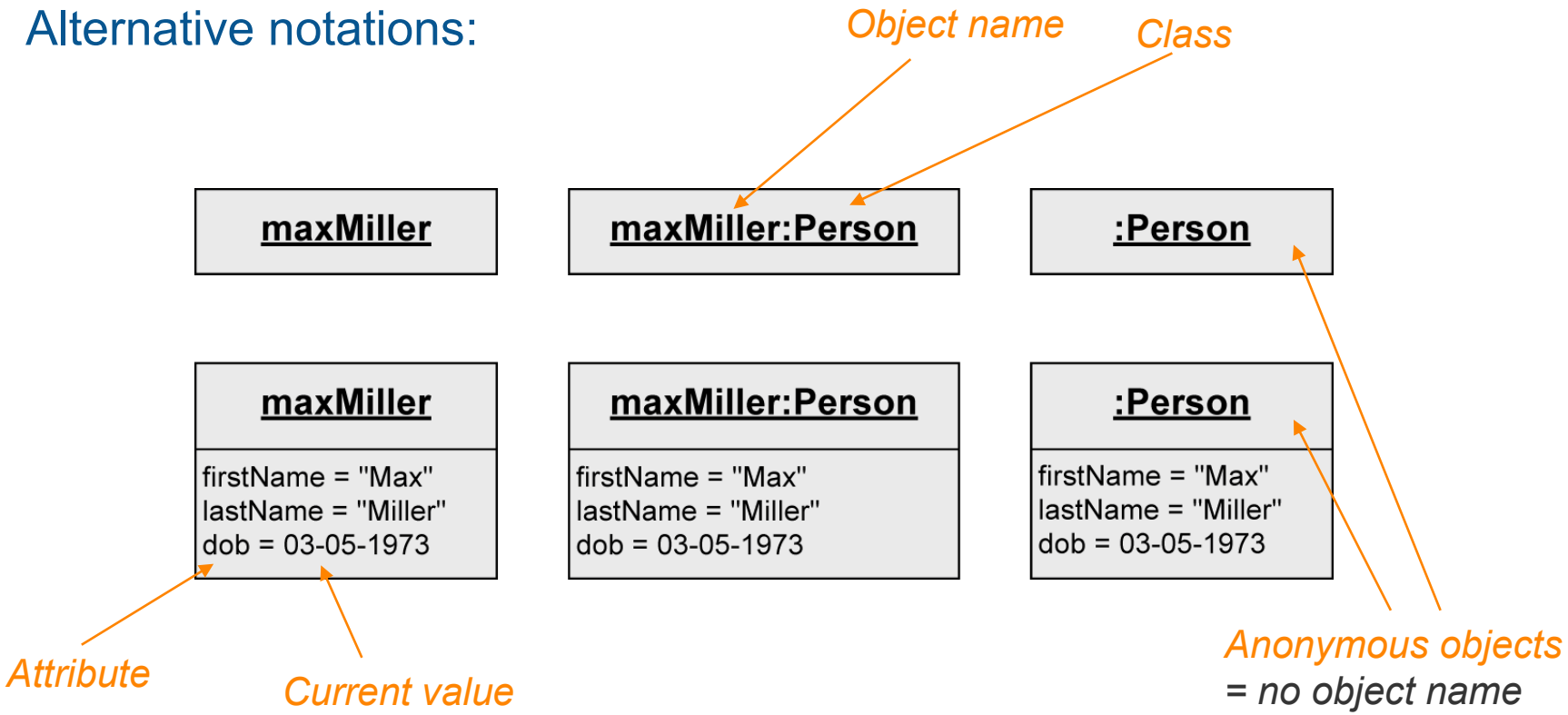
Example – Complete Class Diagram



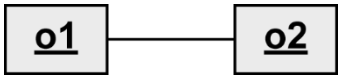
o:C

Object

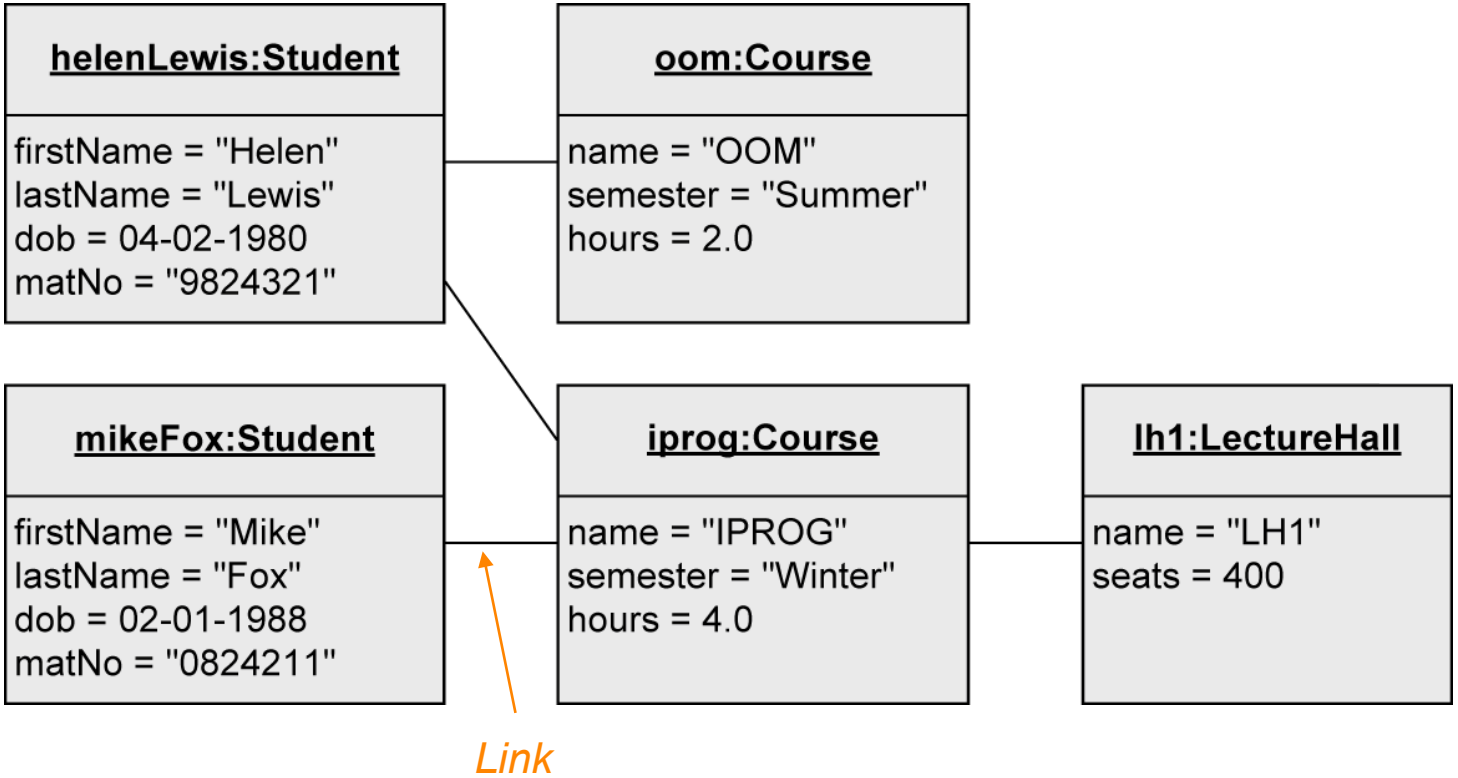
- Individuals of a system
- Alternative notations:



Object Diagram



- Objects of a system and their relationships (links)
- Snapshot of objects at a specific moment in time



From Object to Class

- Individuals of a system often have identical characteristics and behavior
- A class is a construction plan for a set of similar objects of a system
- Objects are instances of classes
- Attributes: structural characteristics of a class
  - Different value for each instance (= object)
- Operations: behavior of a class
  - Identical for all objects of a class
    - not depicted in object diagram

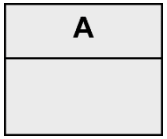
Class

Person
firstName: String lastName: String dob: Date

Object of that class

maxMiller:Person
firstName = "Max" lastName = "Miller" dob = 03-05-1973

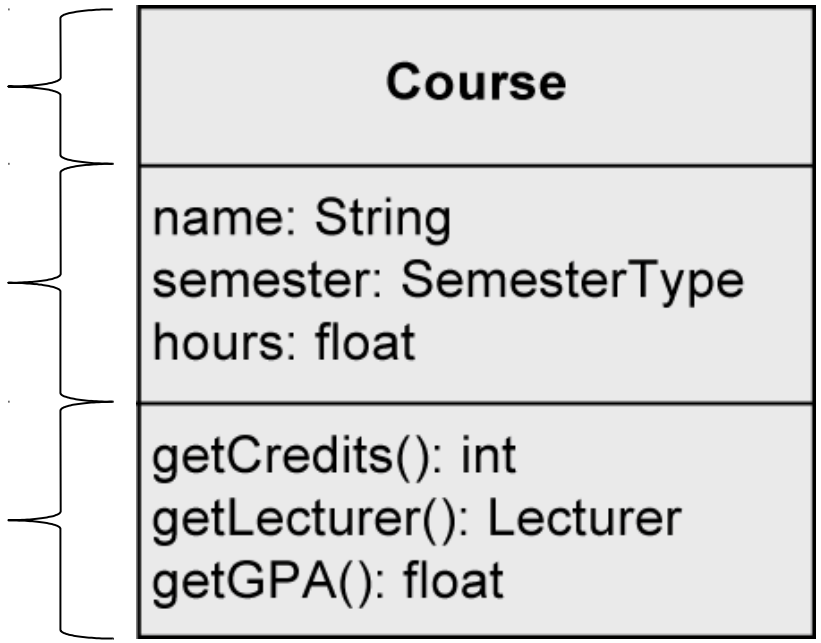
Class



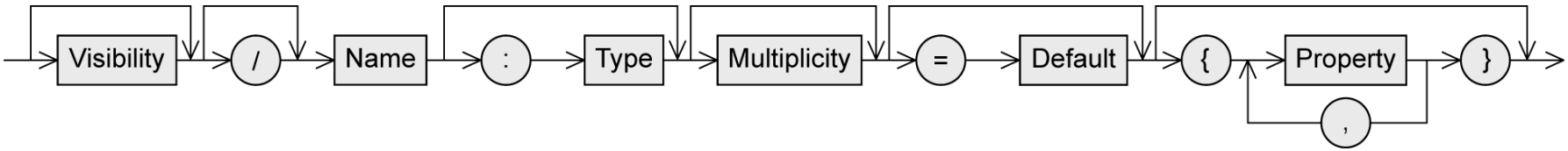
Class name

Attributes

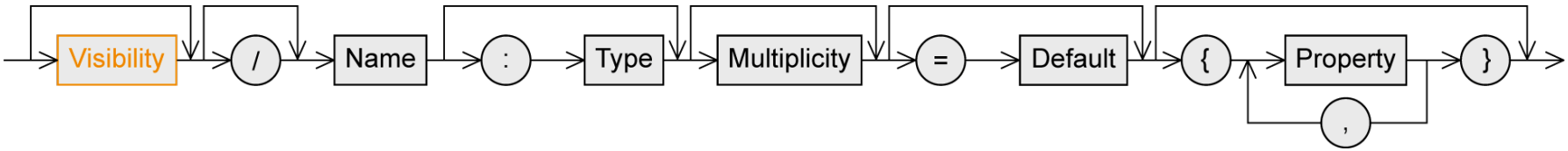
Operations



Attribute Syntax



Attribute Syntax - Visibility

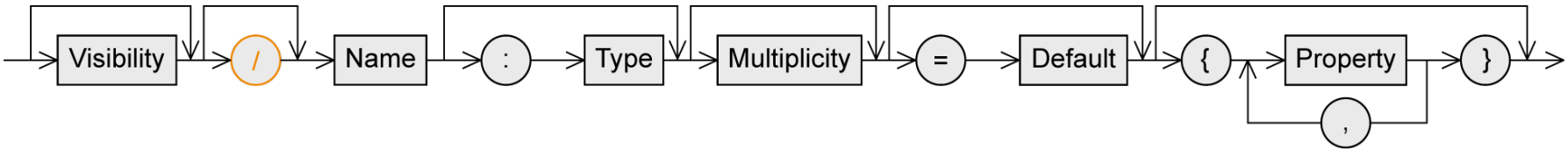


Person

+ firstName: String
+ lastName: String
– dob: Date
address: String[1..*] {unique, ordered}
– ssNo: String {readOnly}
– /age: int
– password: String = "pw123"
– <u>personsNumber: int</u>

- Who is permitted to access the attribute
  - + ... public: everybody
  - - ... private: only the object itself
  - # ... protected: class itself and subclasses
  - ~ ... package: classes that are in the same package

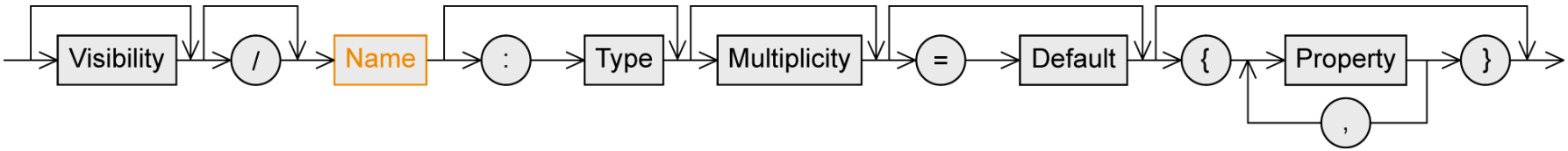
Attribute Syntax - Derived Attribute



Person
firstName: String lastName: String dob: Date address: String[1..*] {unique, ordered} ssNo: String {readOnly} /age: int password: String = "pw123" <u>personsNumber: int</u>

- Attribute value is derived from other attributes
  - `age`: calculated from the date of birth

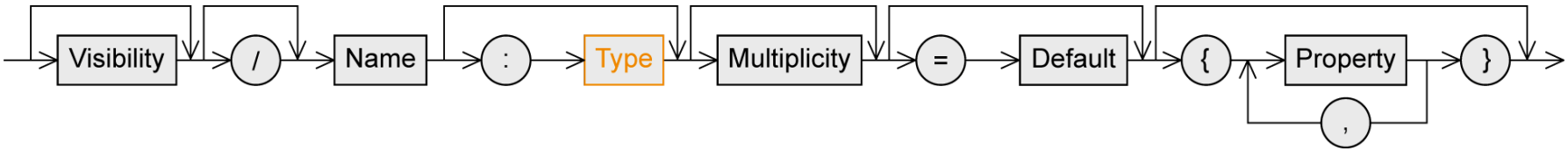
Attribute Syntax - Name



Person
firstName: String lastName: String dob: Date address: String[1..*] {unique, ordered} ssNo: String {readOnly} /age: int password: String = "pw123" personsNumber: int

- Name of the attribute

Attribute Syntax - Type



Person
firstName: String lastName: String dob: Date address: String[1..*] {unique, ordered} ssNo: String {readOnly} /age: int password: String = "pw123" personsNumber: int

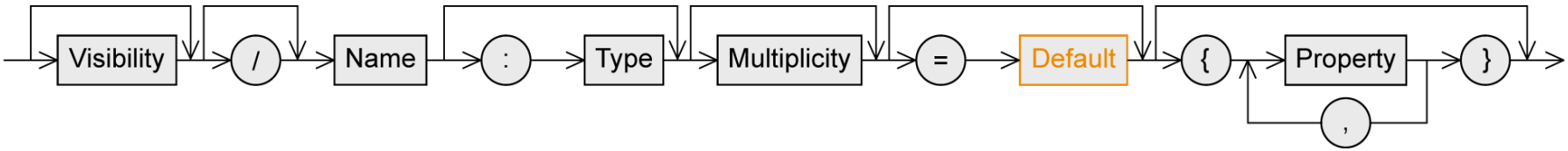
- Type

Attribute Syntax - Multiplicity

Person
firstName: String lastName: String dob: Date address: String[1..*] {unique, ordered} ssNo: String {readOnly} /age: int password: String = "pw123" <u>personsNumber: int</u>

- Number of values an attribute may contain
- Default value: 1
- Notation: **[min..max]**
  - no upper limit: `[*]` or `[0..*]`

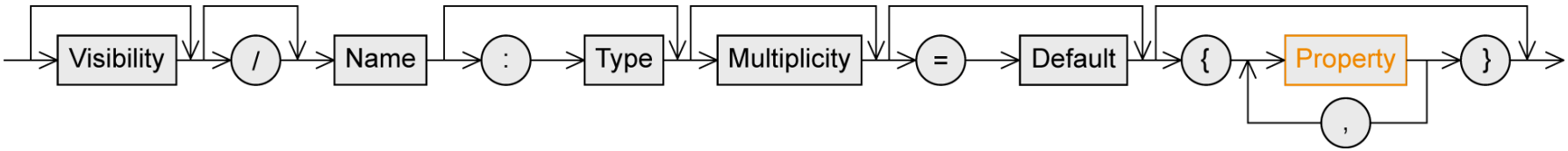
Attribute Syntax – Default Value



Person
firstName: String lastName: String dob: Date address: String[1..*] {unique, ordered} ssNo: String {readOnly} /age: int password: String = "pw123" <u>personsNumber: int</u>

- Default value
  - Used if the attribute value is not set explicitly by the user

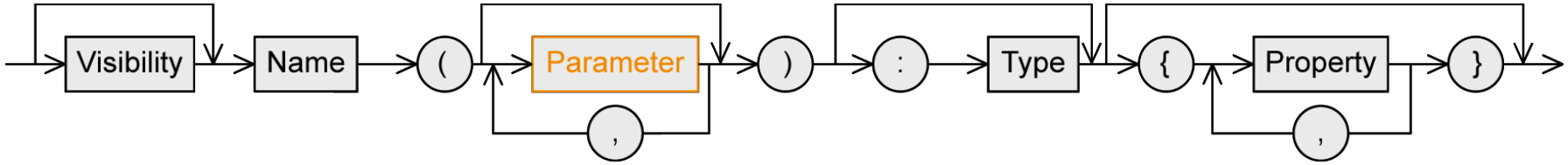
Attribute Syntax – Properties



Person
firstName: String lastName: String dob: Date address: String[1..*] {unique, ordered} ssNo: String {readOnly} /age: int password: String = "pw123" <u>personsNumber: int</u>

- Pre-defined properties
  - `{readOnly}` … value cannot be changed
  - `{unique}` … no duplicates permitted
  - `{non-unique}` … duplicates permitted
  - `{ordered}` … fixed order of the values
  - `{unordered}` … no fixed order of the values

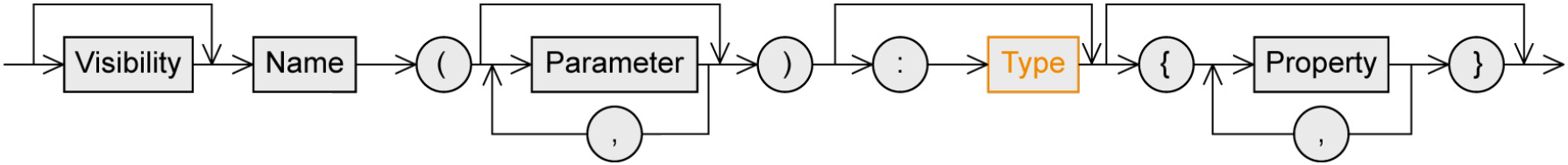
Operation Syntax - Parameters



Person
...
+ getName(out fn: String, out ln: String): void + updateLastName(newName: String): boolean + getPersonsNumber(): int

- Notation similar to attributes
- Direction of the parameter
  - `in` … input parameter
    - When the operation is used, a value is expected from this parameter
  - `out` … output parameter
    - After the execution of the operation, the parameter has adopted a new value
  - `inout` : combined input/output parameter

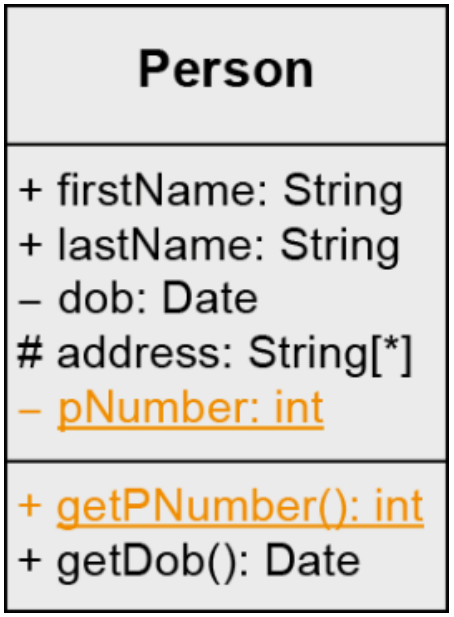
Operation Syntax - Type



Person
...
getName(out fn: String, out ln: String): void updateLastName(newName: String): boolean getPersonsNumber(): int

- Type of the return value

Can you convert this class into Java source code?



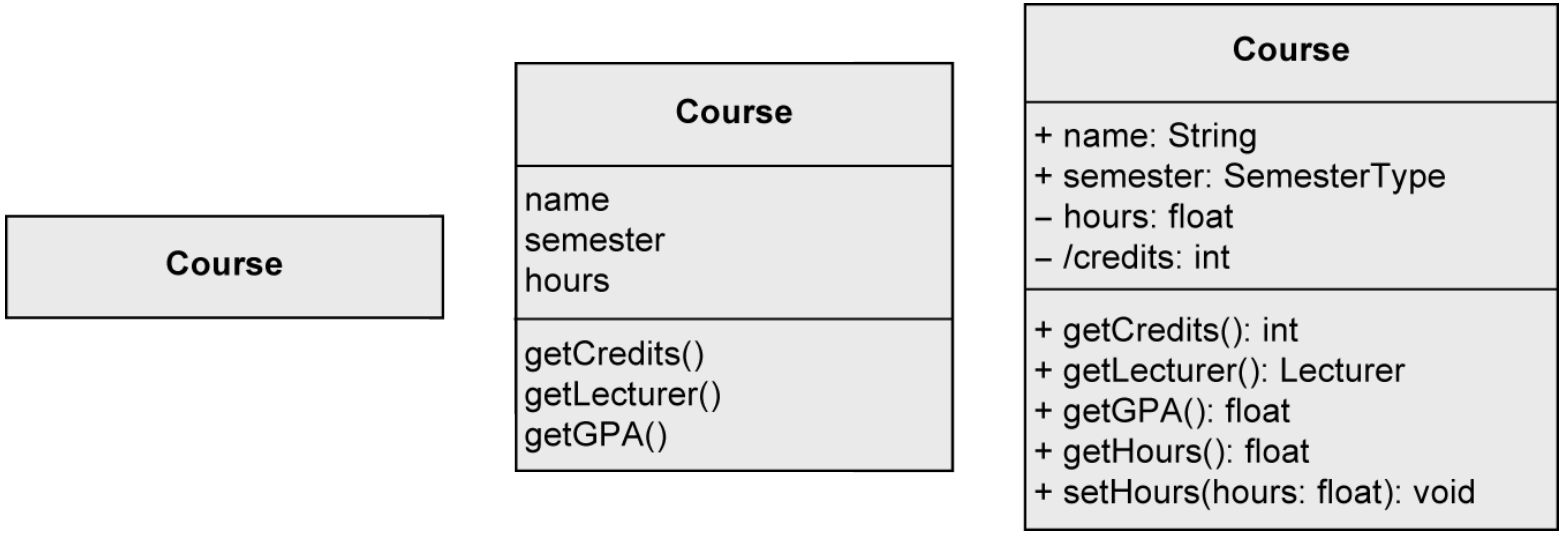

```
class Person {

    public String firstName;
    public String lastName;
    private Date dob;
    protected String[] address;
    private static int pNumber;
    public static int getPNumber() {…}
    public Date getDob() {…}
}
```

(underline in a class diagram = `static`)

Specification of Classes: Different Levels of Detail

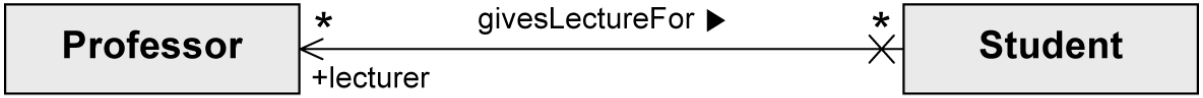
coarse-grained ⟷ *fine-grained*



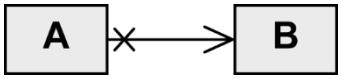
Depending on where you are in the requirements process, these levels are very useful!

Association

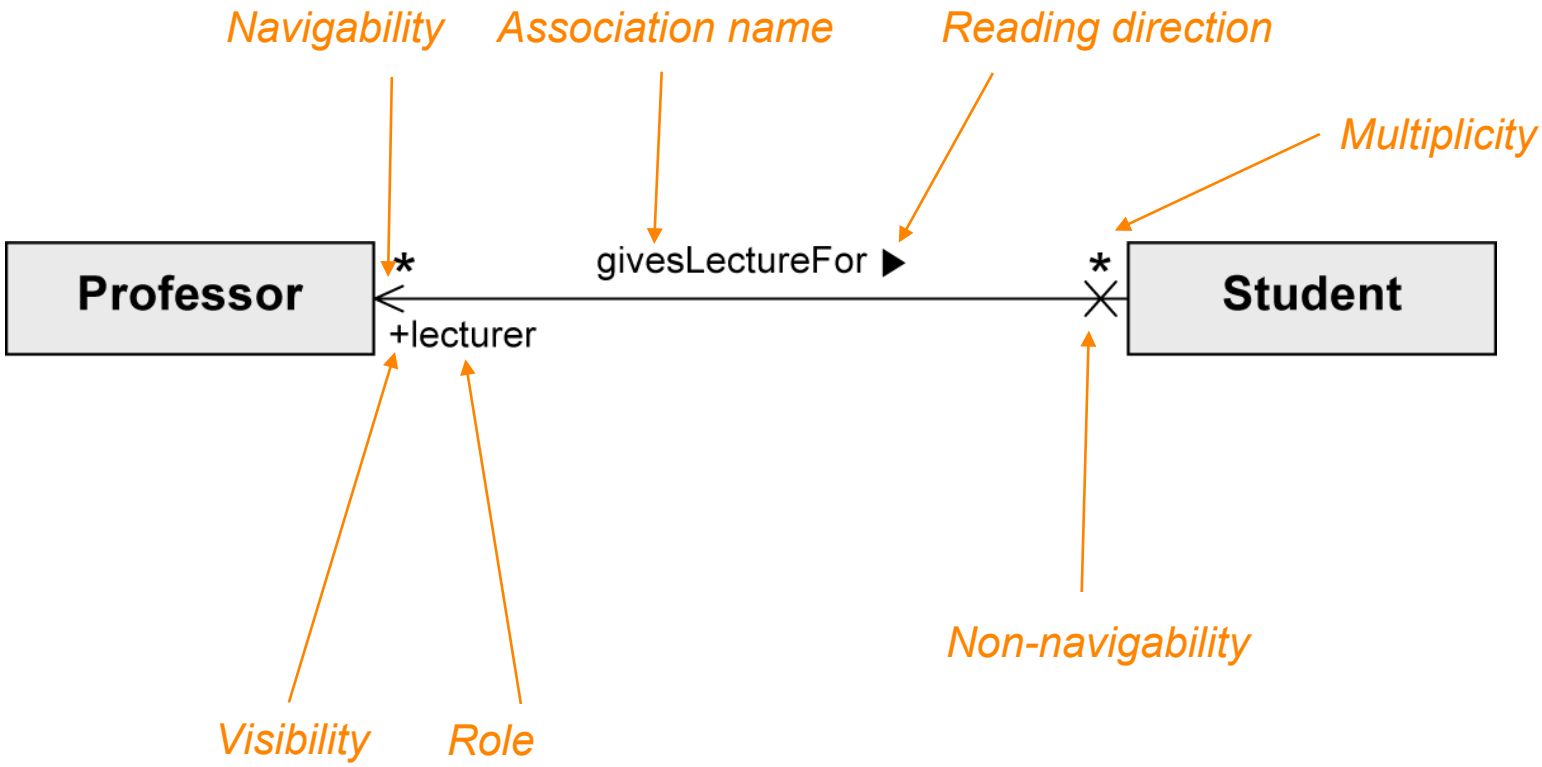
- Models possible relationships between instances of classes



Binary Association

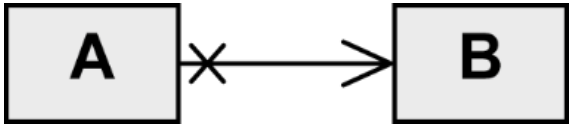


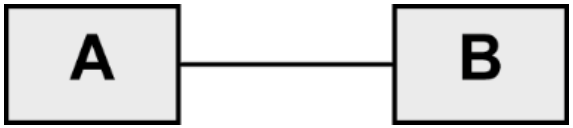
- Connects instances of two classes with one another



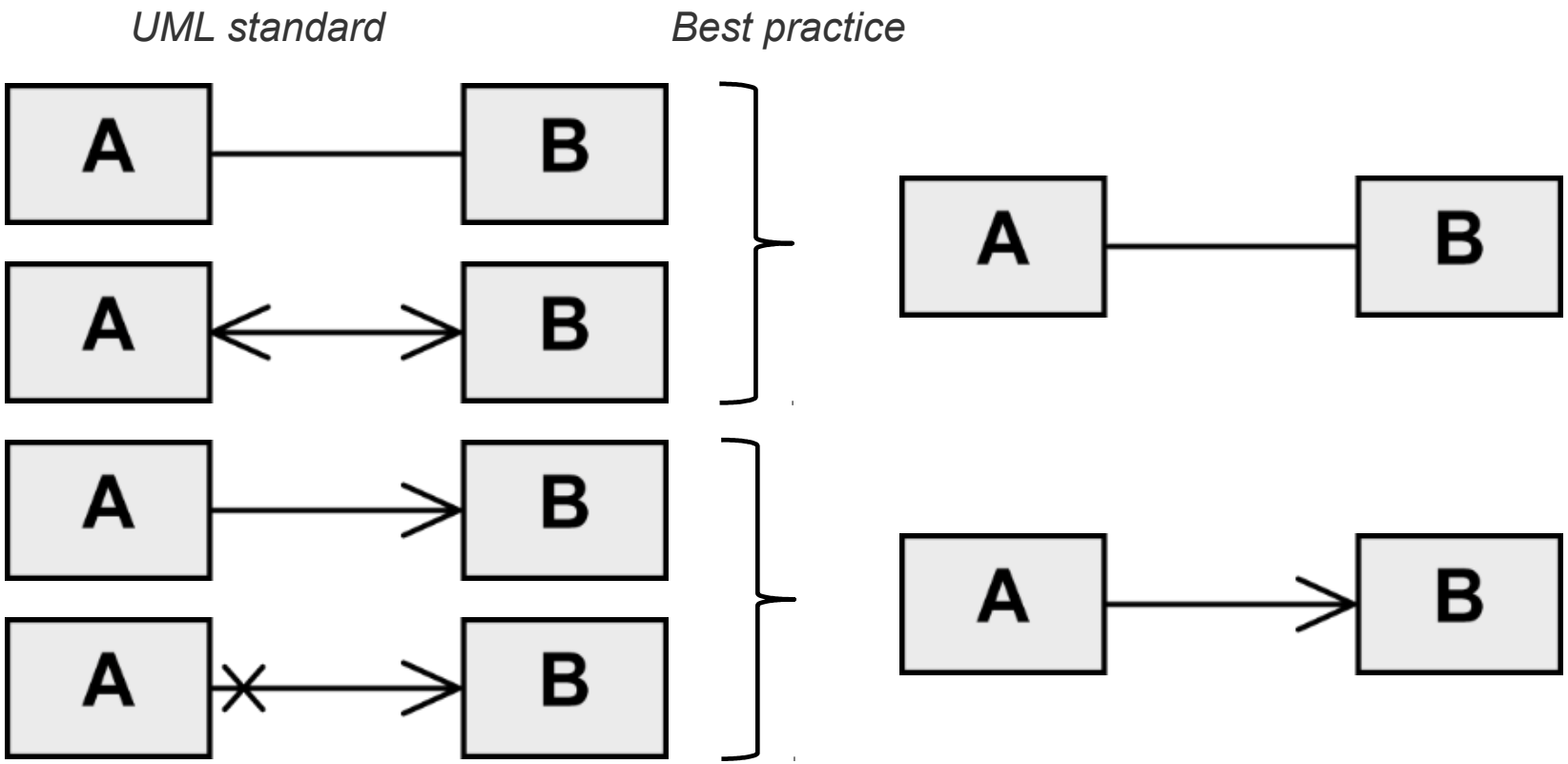
Binary Association - Navigability

- Navigability: an object knows its partner objects and can therefore access their visible attributes and operations
  - Indicated by open arrow head
- Non-navigability
  - Indicated by cross
- Example:
  - **A** can access the visible attributes and operations of **B**
  - **B** cannot access any attributes and operations of **A**
- Navigability undefined
  - Bidirectional navigability is assumed



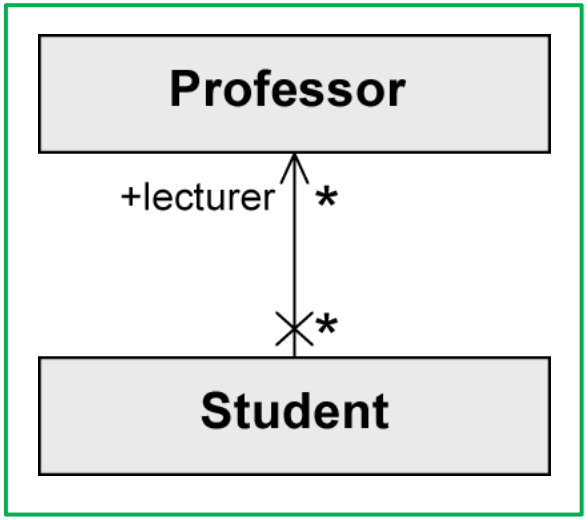


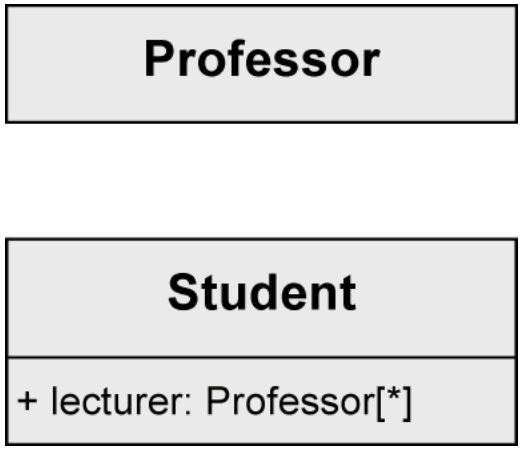
Navigability – UML Standard vs. Best Practice



Binary Association as Attribute

Preferable





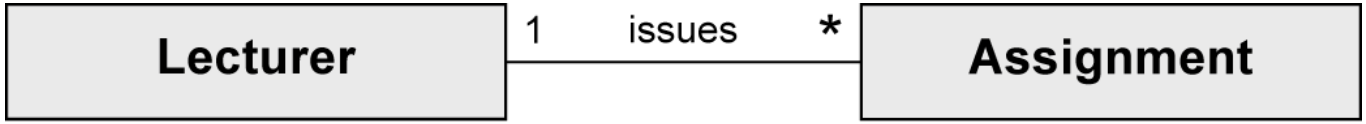
- Java-like notation:

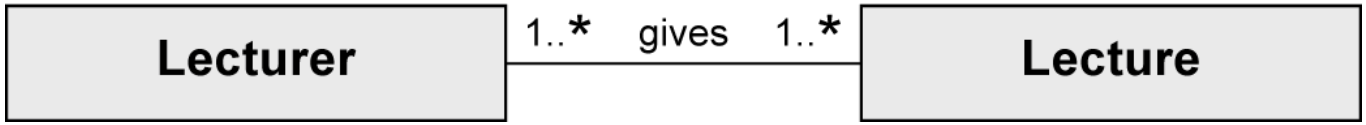
```
class Professor {…}

class Student{
    public Professor[] lecturer;
    …
}
```

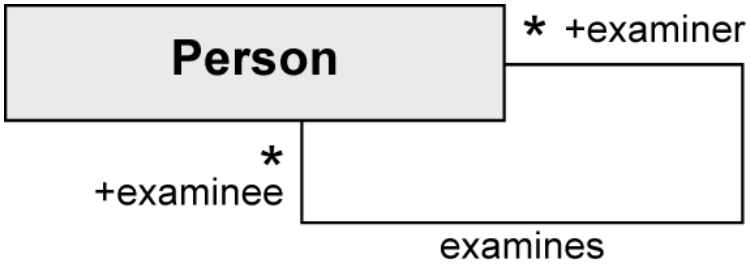
Binary Association – Multiplicity and Role

- Multiplicity: Number of objects that may be associated with exactly one object of the opposite side





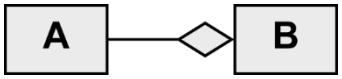
- Role: describes the way in which an object is involved in an association relationship



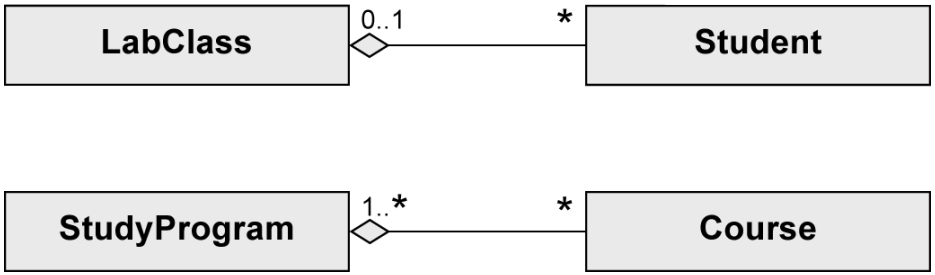
Aggregation

- Special form of association
- Used to express that a class is part of another class
- Two types:
  - Shared aggregation
  - Composition

Shared Aggregation



- Expresses a weak belonging of the parts to a whole
  = Parts also exist independently of the whole
- Multiplicity at the aggregating end may be >1
  = One element can be part of multiple other elements simultaneously
- Syntax: Hollow diamond at the aggregating end
- Example:
  - `Student` is part of `LabClass`
  - `Course` is part of `StudyProgram`



Composition

- Existence dependency between the composite object and its parts
- One part can only be contained in at most one composite object at one specific point in time

  Multiplicity at the aggregating end max. 1

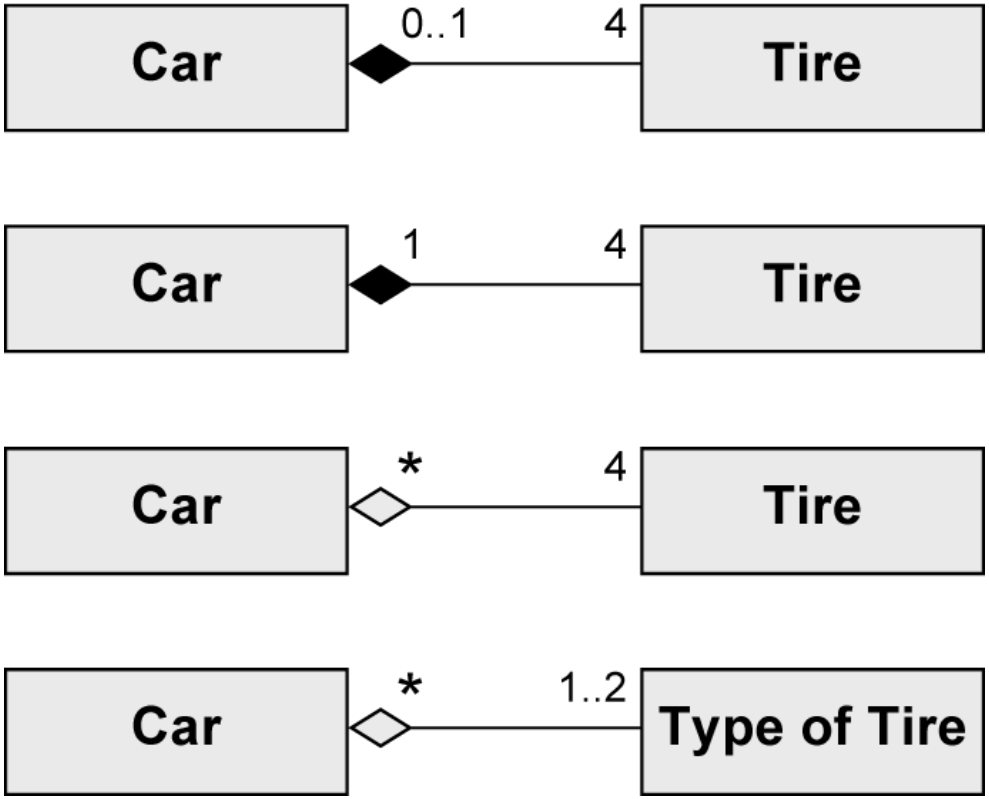
  -> The composite objects form a tree
- If the composite object is deleted, its parts are also deleted
- Syntax: Solid diamond at the aggregating end
- Example: **`Beamer`** is part of **`LectureHall`** is part of **`Building`**

If the `Building` is deleted, the `LectureHall` is also deleted

The `Beamer` can exist without the `LectureHall`, but if it is contained in the `LectureHall` while it is deleted, the `Beamer` is also deleted

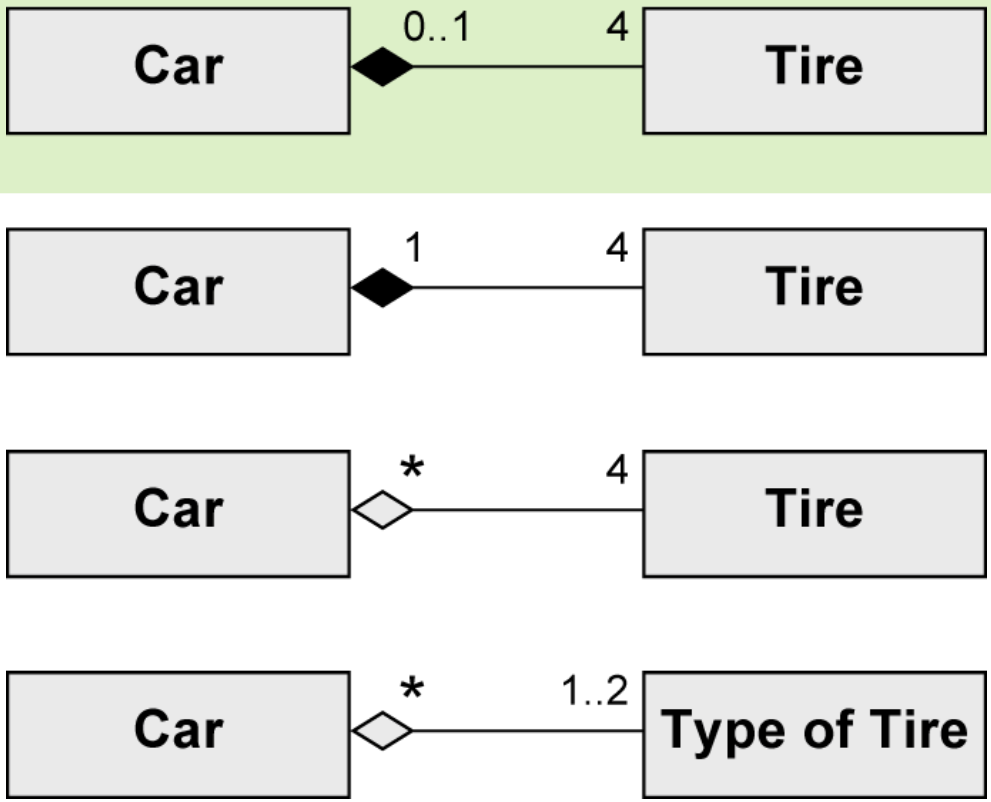
Shared Aggregation and Composition

- Which model applies?



Shared Aggregation and Composition

- Which model applies?

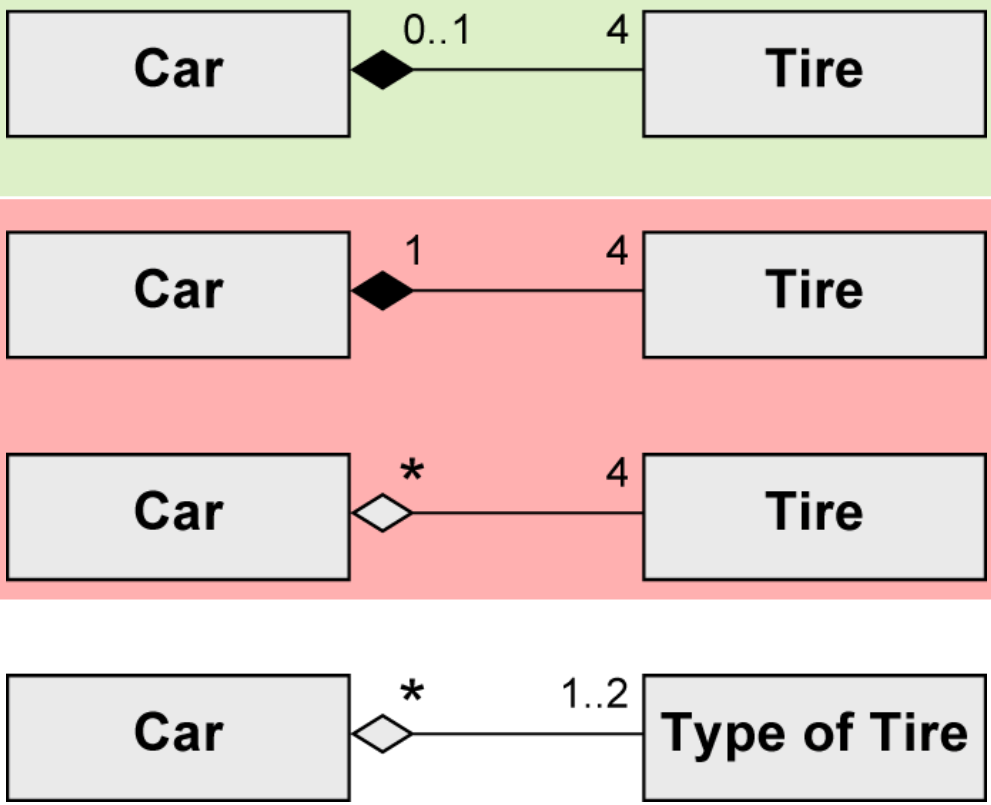


A `Tire` can exist without a `Car`. A `Tire` belongs to one `Car` at most.

Yes

Shared Aggregation and Composition

- Which model applies?



A `Tire` can exist without a `Car`. A `Tire` belongs to one `Car` at most.

A `Tire` cannot exist without a `Car`.

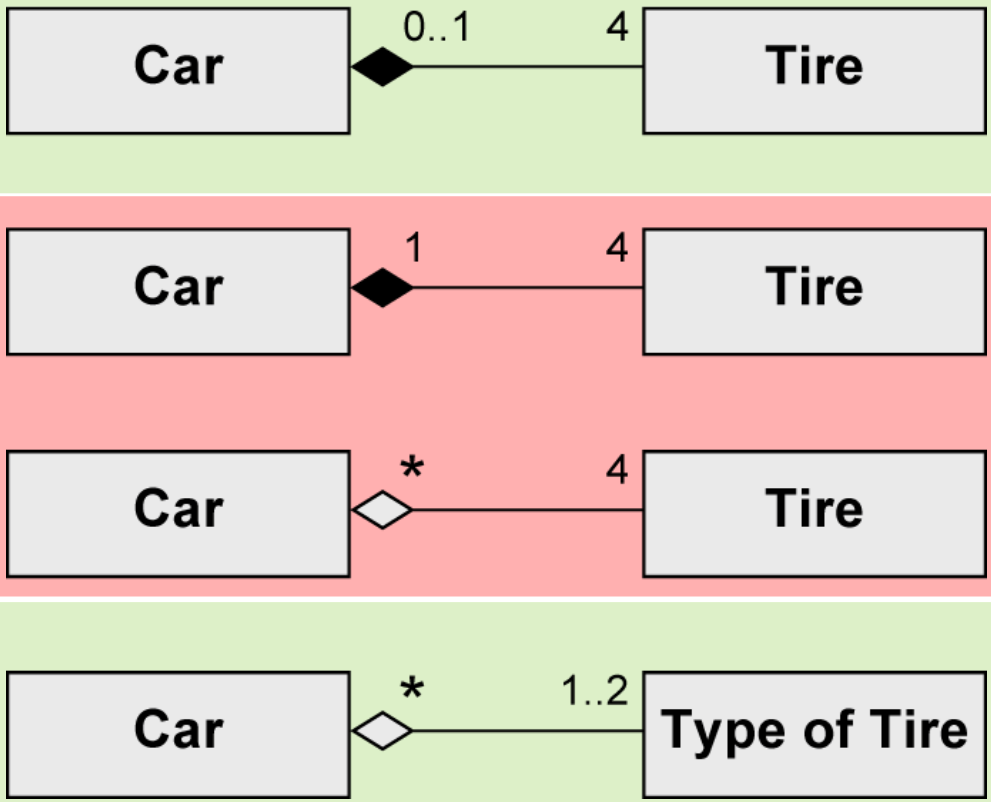
A `Tire` can belong to multiple `Cars`

Yes

No

Shared Aggregation and Composition

- Which model applies?



A `Tire` can exist without a `Car`. A `Tire` belongs to one `Car` at most.

A `Tire` cannot exist without a `Car`.

A `Tire` can belong to multiple `Car`s

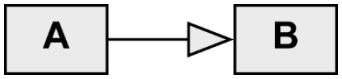
A `Car` has one or two types of `Tire`s. Several `Car`s may have the same Type of `Tire`s.

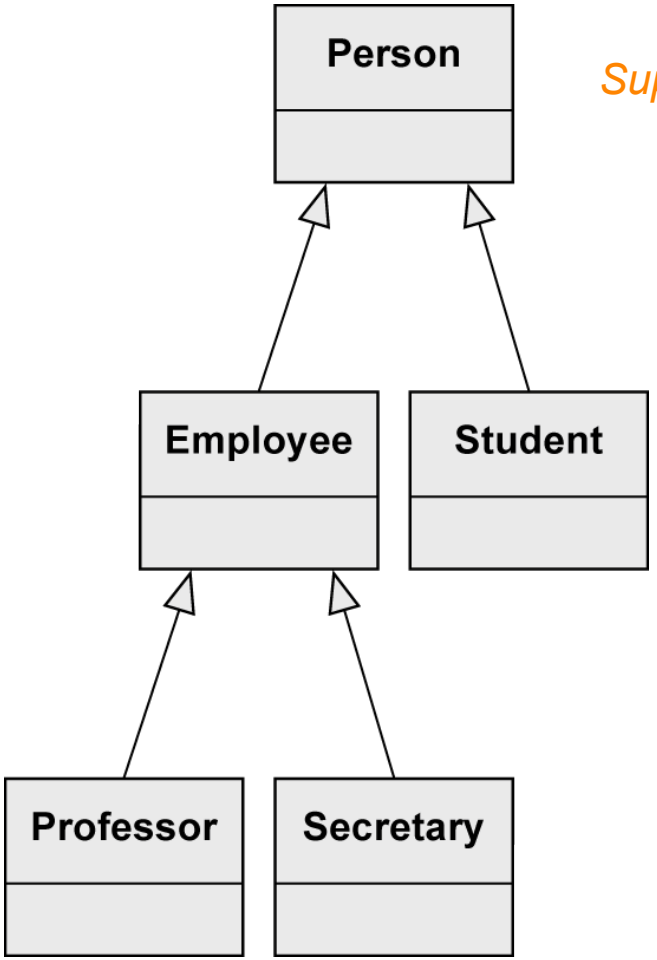
Yes

No

Yes

Generalization





Superclass

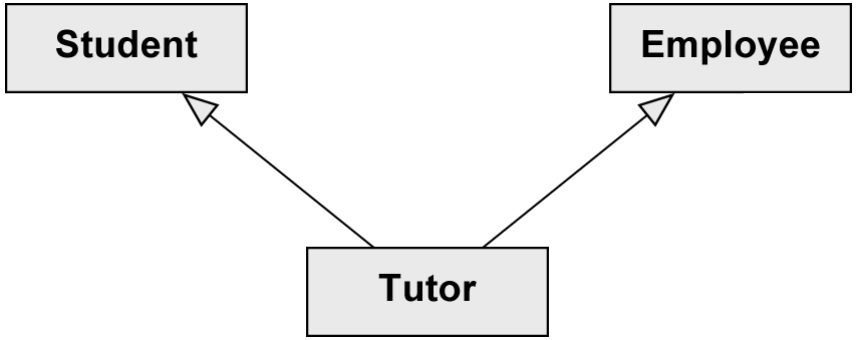
Subclasses
…inherit characteristics, associations, and aggregations

A `Secretary` *is an* `Employee` *and a* `Person`

Generalization – Multiple Inheritance

- UML allows multiple inheritance.
- A class may have multiple superclasses.

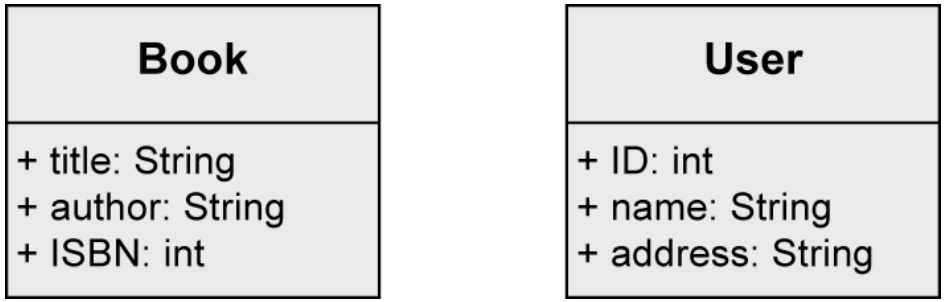
- Example:



A `Tutor` is both an `Employee` and a `Student`

Creating a Class Diagram

- Not possible to completely extract classes, attributes and associations from a natural language text automatically.
- Guidelines
  - Nouns often indicate classes
  - Adjectives indicate attribute values
  - Verbs indicate operations
- Example: The library management system stores users with their unique ID, name and address as well as books with their title, author and ISBN number. Ann Foster wants to use the library.



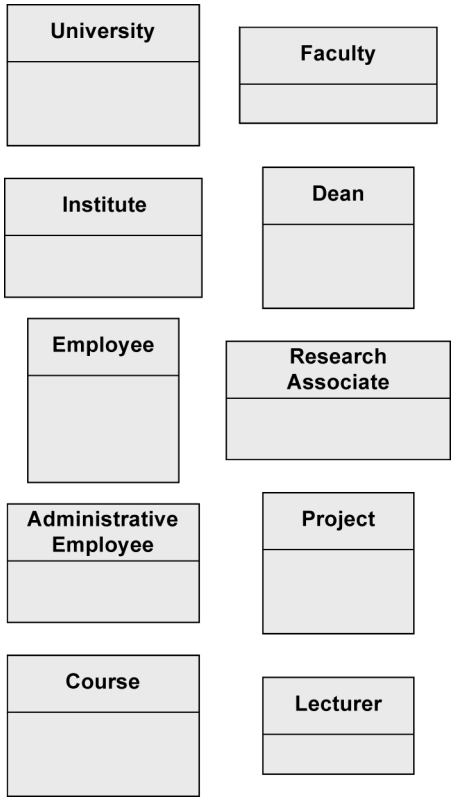
Question: What about Ann Foster?

Example – University Information System

- A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.
- Each faculty is led by a dean, who is an employee of the university.
- The total number of employees is known. Employees have a social security number, a name, and an email address. There is a distinction between research and administrative personnel.
- Research associates are assigned to at least one institute. The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates hold courses. Then they are called lecturers.
- Courses have a unique number (ID), a name, and a weekly duration in hours.

Example – Step 1: Identifying Classes

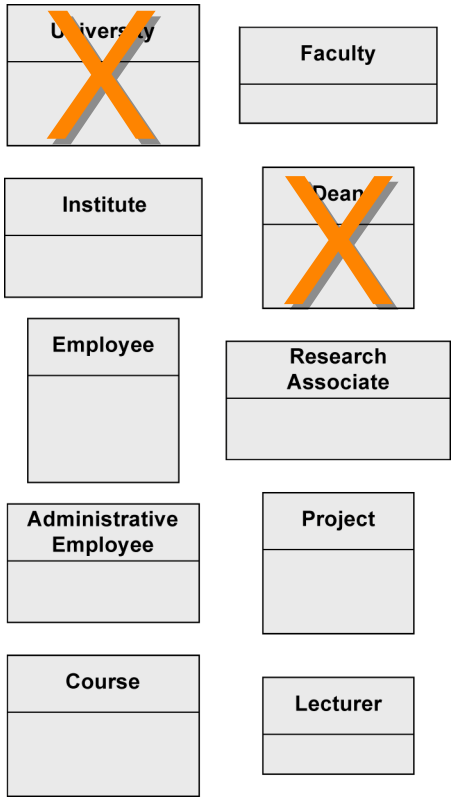
- A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.
- Each faculty is led by a dean, who is an employee of the university.
- The total number of employees is known. Employees have a social security number, a name, and an email address. There is a distinction between research and administrative personnel.
- Research associates are assigned to at least one institute. The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates hold courses. Then they are called lecturers.
- Courses have a unique number (ID), a name, and a weekly duration in hours.



Example – Step 1: Identifying Classes

- A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.
- Each faculty is led by a dean, who is an employee of the university.
- The total number of employees is known. Employees have a social security number, a name, and an email address. There is a distinction between research and administrative personnel.
- Research associates are assigned to at least one institute. The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates hold courses. Then they are called lecturers.
- Courses have a unique number (ID), a name, and a weekly duration in hours.

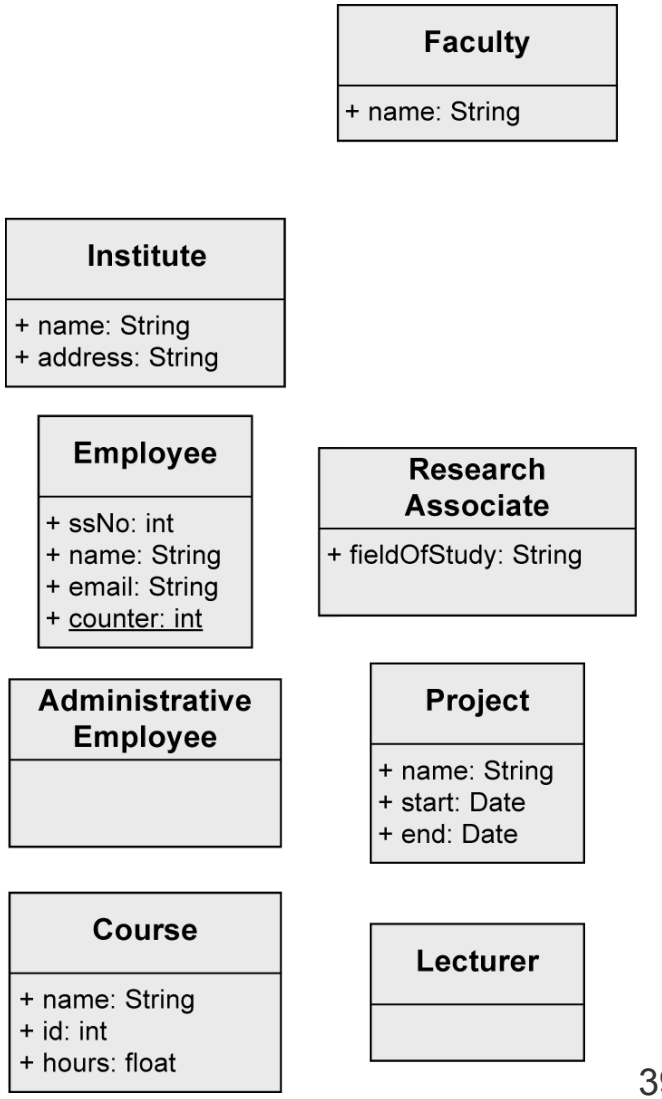
We model the system "University"



Dean has no further attributes than any other employee

Example – Step 2: Identifying the Attributes

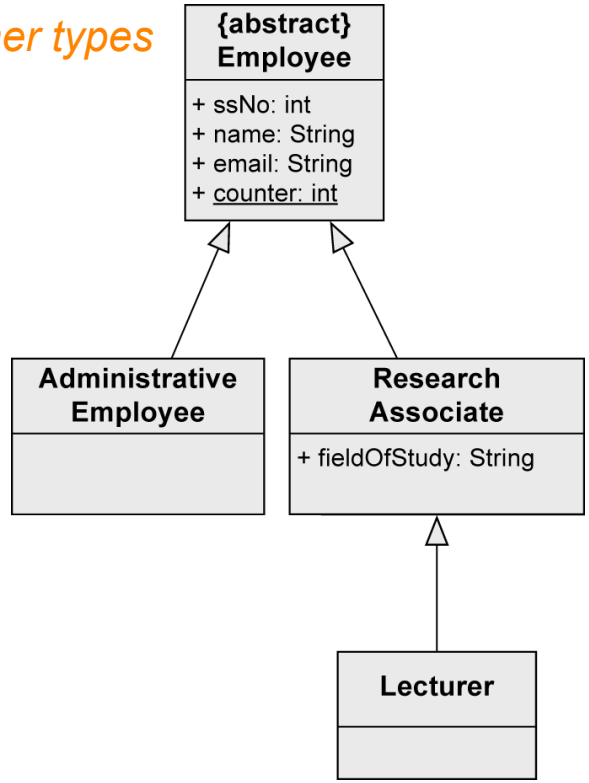
- A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.
- Each faculty is led by a dean, who is an employee of the university.
- The total number of employees is known. Employees have a social security number, a name, and an email address. There is a distinction between research and administrative personnel.
- Research associates are assigned to at least one institute. The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates hold courses. Then they are called lecturers.
- Courses have a unique number (ID), a name, and a weekly duration in hours.



Example – Step 2: Identifying Relationships (1/6)

- Three kinds of relationships:
  - Association
  - Generalization
  - Aggregation

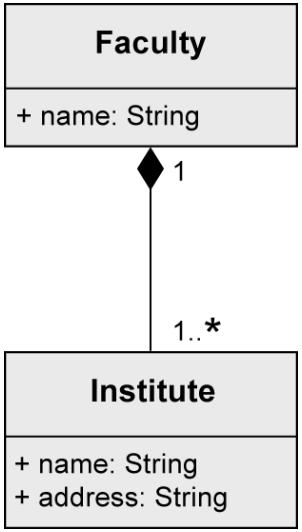
Abstract, i.e., no other types of employees



- Indication of a generalization
- *"There is a distinction between research and administrative personnel."*
- *"Some research associates hold courses. Then they are called lecturers."*

Example – Step 2: Identifying Relationships (2/6)

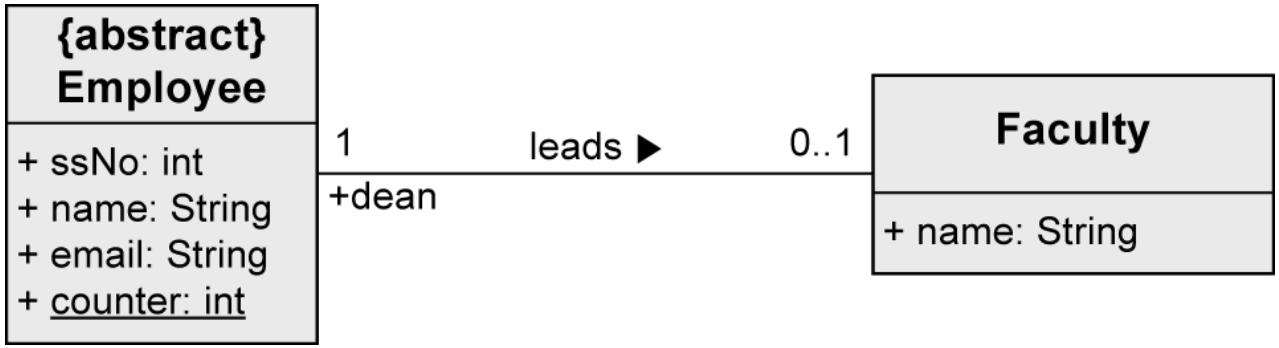
- *"A university consists of multiple faculties which are composed of various institutes."*



Composition to show existence dependency

Example – Step 2: Identifying Relationships (3/6)

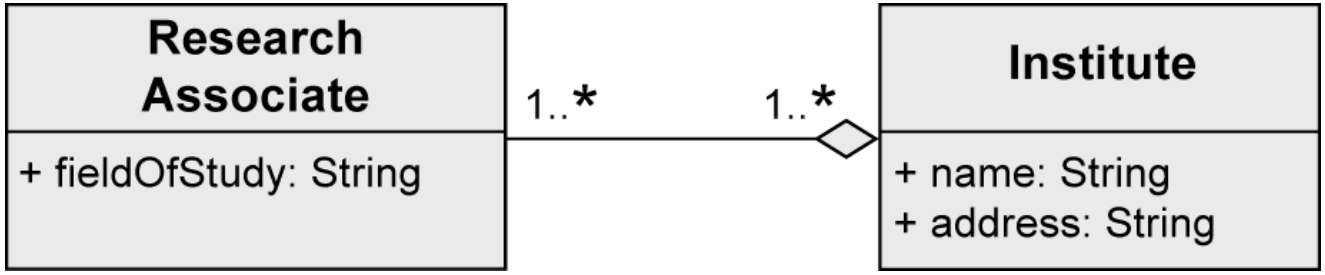
- *"Each faculty is led by a dean, who is an employee of the university"*



In the `leads`-relationship, the `Employee` takes the role of a `dean`.

Example – Step 2: Identifying Relationships (4/6)

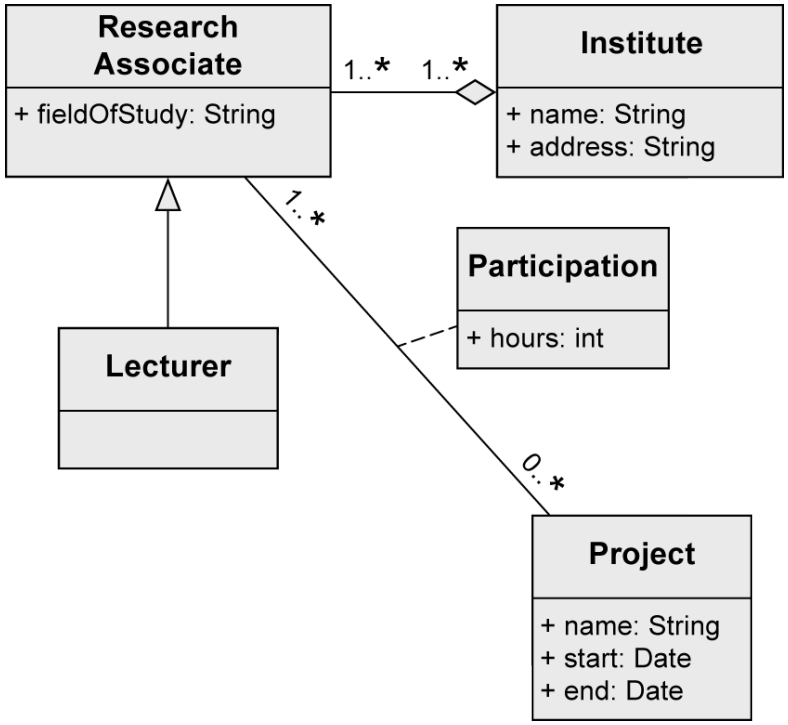
- "Research associates are assigned to at least one institute."



Shared aggregation to show that `ResearchAssociates` *are part of an* `Institute`*, but there is no existence dependency*

Example – Step 2: Identifying Relationships (5/6)

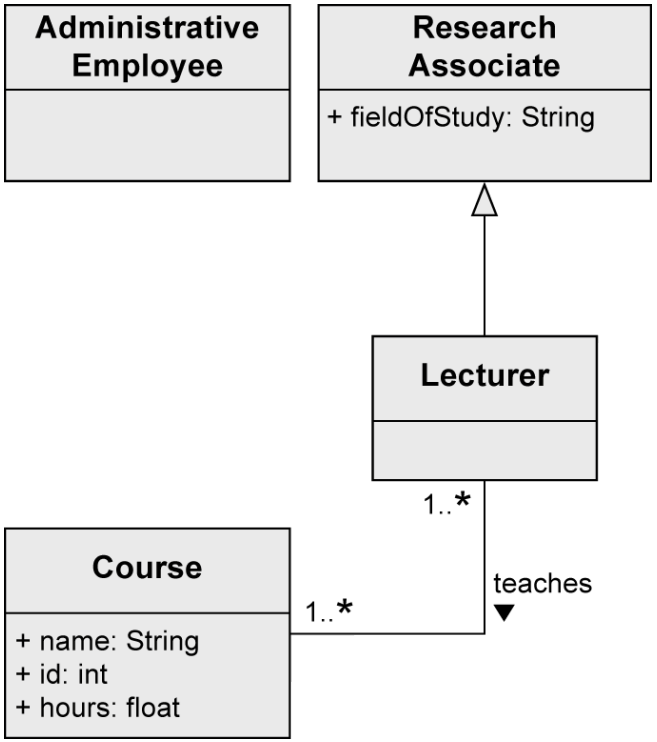
- *"Furthermore, research associates can be involved in projects for a certain number of hours."*



Association class enables to store the number of hours for every single `Project` *of every single* `ResearchAssociate`

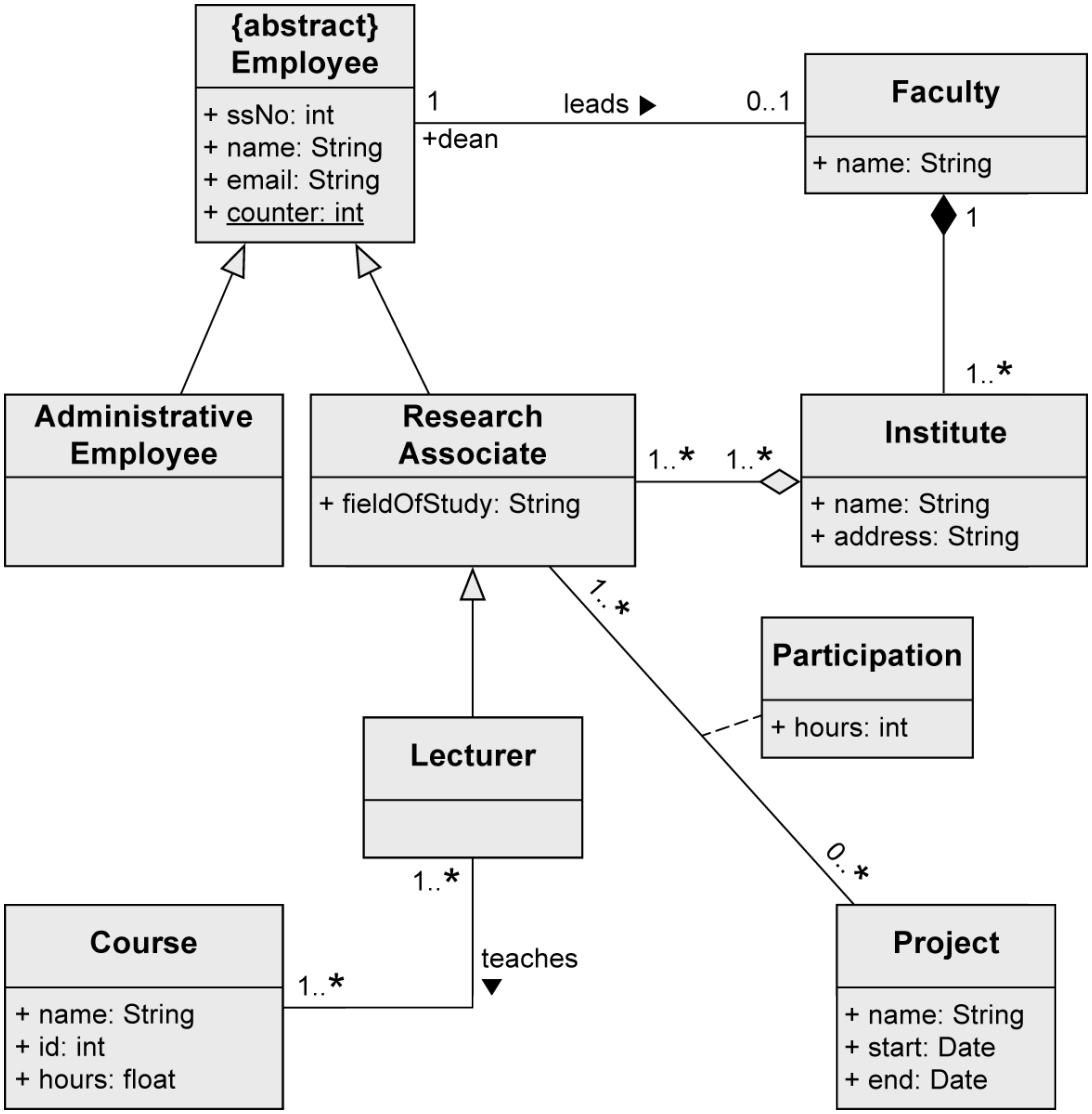
Example – Step 2: Identifying Relationships (6/6)

- *"Some research associates hold courses. Then they are called lecturers."*



`Lecturer` inherits all characteristics, associations, and aggregations from `ResearchAssociate`.
In addition, a `Lecturer` has an association `teaches` to `Course`.

Example – Complete Class Diagram



Notation Elements (1/2)

Name	Notation	Description
Class	A – a1: T1 – a2: T2 + o1(): void + o2(): void	Description of the structure and behavior of a set of objects
Abstract class	*A* oder {abstract} A	Class that cannot be instantiated
Association	A — B A ↔ B A → B	Relationship between classes: navigability unspecified, navigable in both directions, not navigable in one direction

Notation Elements (2/2)

Name	Notation	Description
Shared aggregation	A ◇ B	Parts-whole relationship (`A` is part of `B`)
Strong aggregation = composition	A ◆ B	Existence-dependent parts-whole relationship (`A` is part of `B`)
Generalization	A ▷ B	Inheritance relationship (`A` inherits from `B`)
Object	o:C	Instance of a class
Link	o1 — o2	Relationship between objects