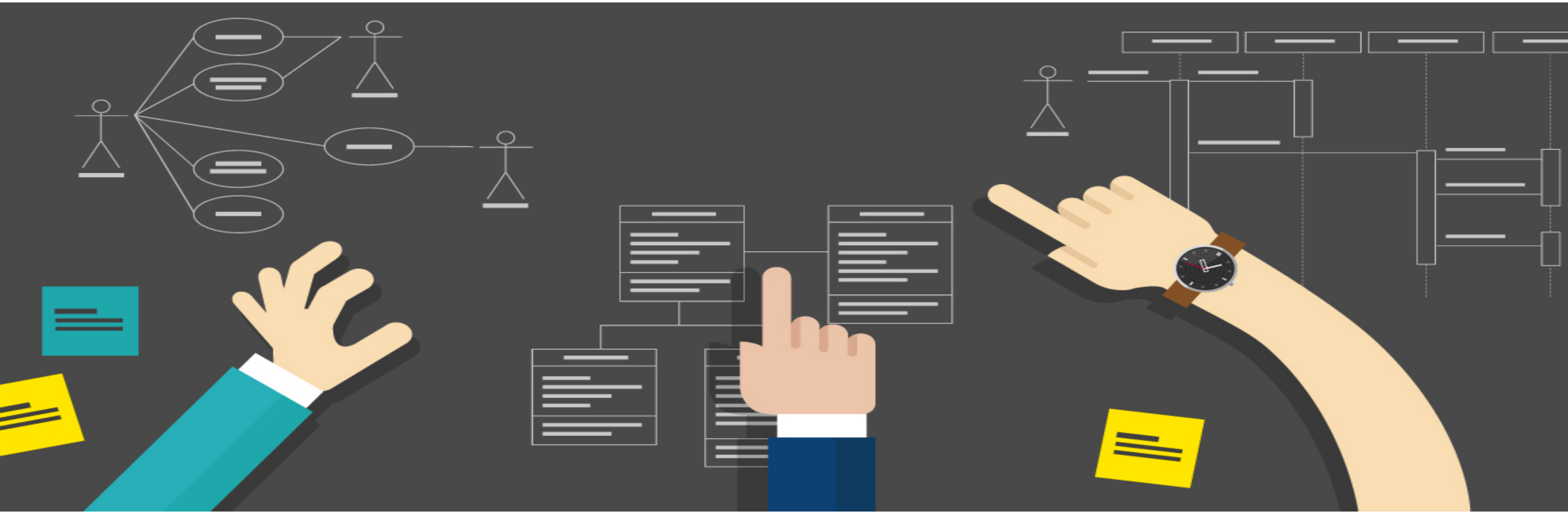
MedTech
MEDITERRANEAN
INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

Software DESGN



Class Diagram

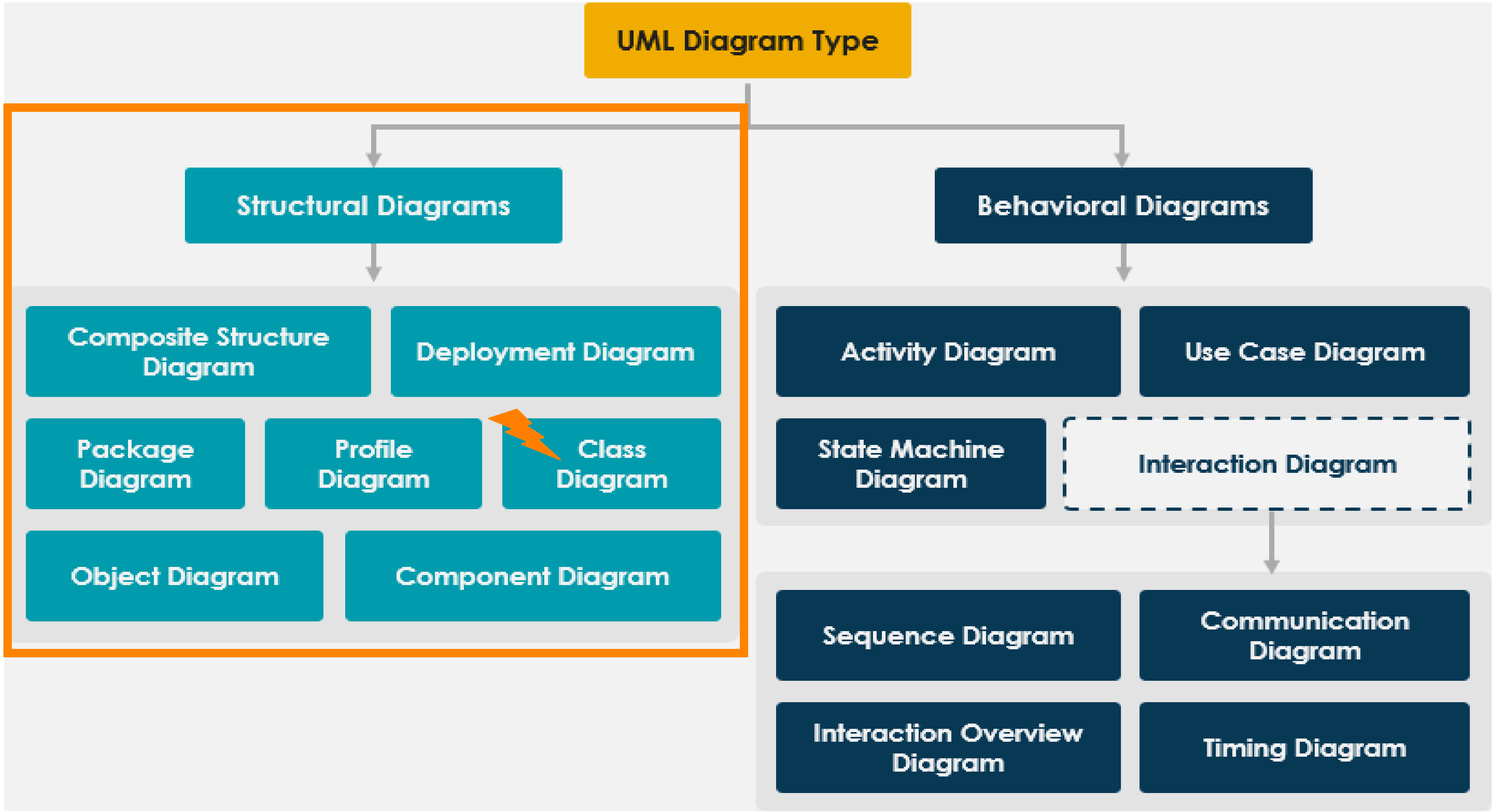
Lecturer: Dr. Ikram SAADAOUI
Credits: Dr. Asma AMDOUNI

System design

- Developing a **system architecture** that satisfies the requirements defined in the requirements specification phase

- The designer chooses **how the system operations should be implemented by the interacting objects** at runtime

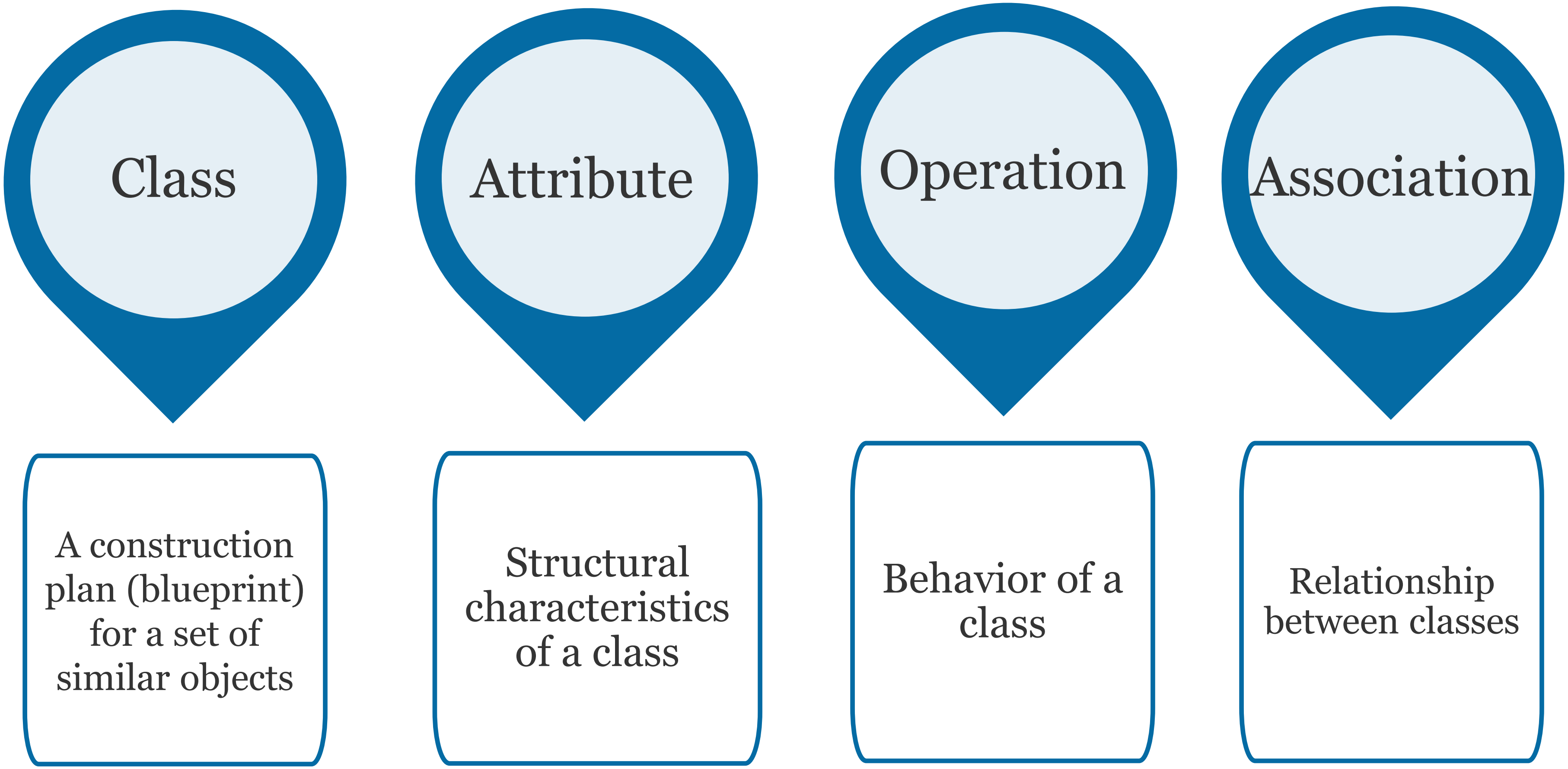
- Provide the foundation for implementation, testing, and maintenance phases


UML Diagram Type
Structural Diagrams
Behavioral Diagrams
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Class Diagram
Object Diagram
Component Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

Content

- Classes
- Attributes
- Operations
- Relationships
  - Binary Association
  - N-ary Association
  - Association Class
  - Aggregation
  - Generalization
- Creating a class diagram

CLASS DIAGRAM



Class

A construction plan (blueprint) for a set of similar objects

Attribute

Structural characteristics of a class

Operation

Behavior of a class

Association

Relationship between classes

Class & Object

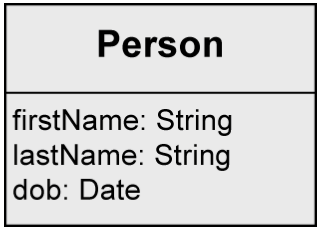
- A class is a blueprint (template) for a set of similar objects of a system

- Objects are instances of classes

- Attributes: structural characteristics (properties) of a class
  - Different value for each instance (= object)

- Operations: behavior of a class
  - Identical for all objects of a class
    → **not depicted in the object**

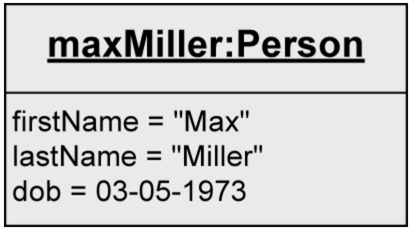
Class



Person

firstName: String
lastName: String
dob: Date

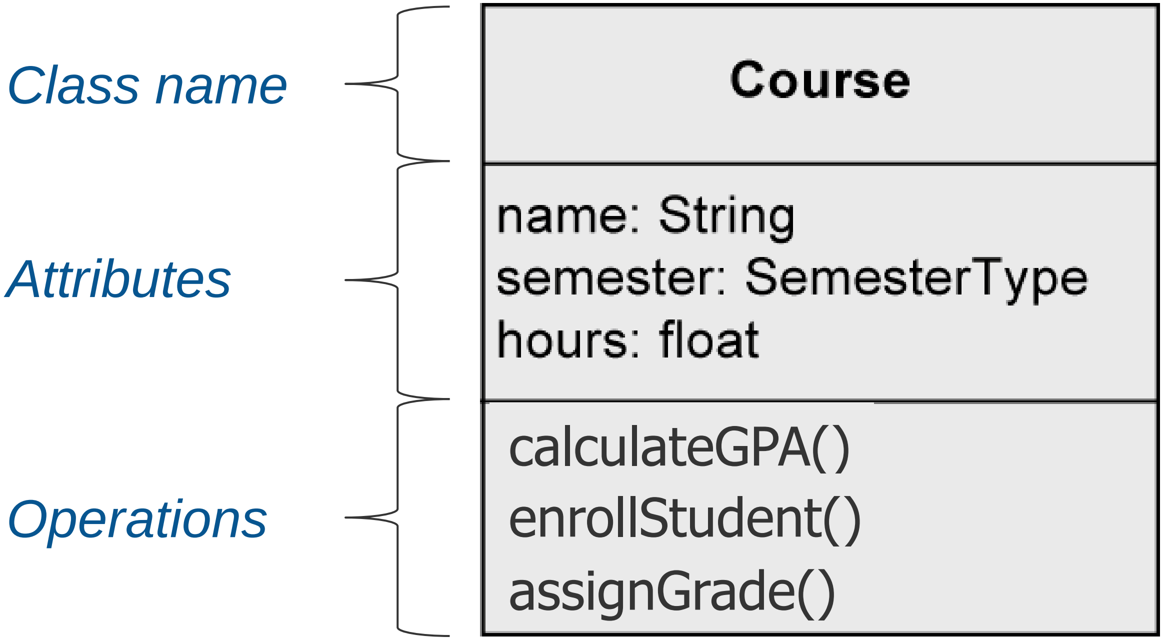
Object of that class



maxMiller:Person

firstName = "Max"
lastName = "Miller"
dob = 03-05-1973

Class



Class name

Attributes

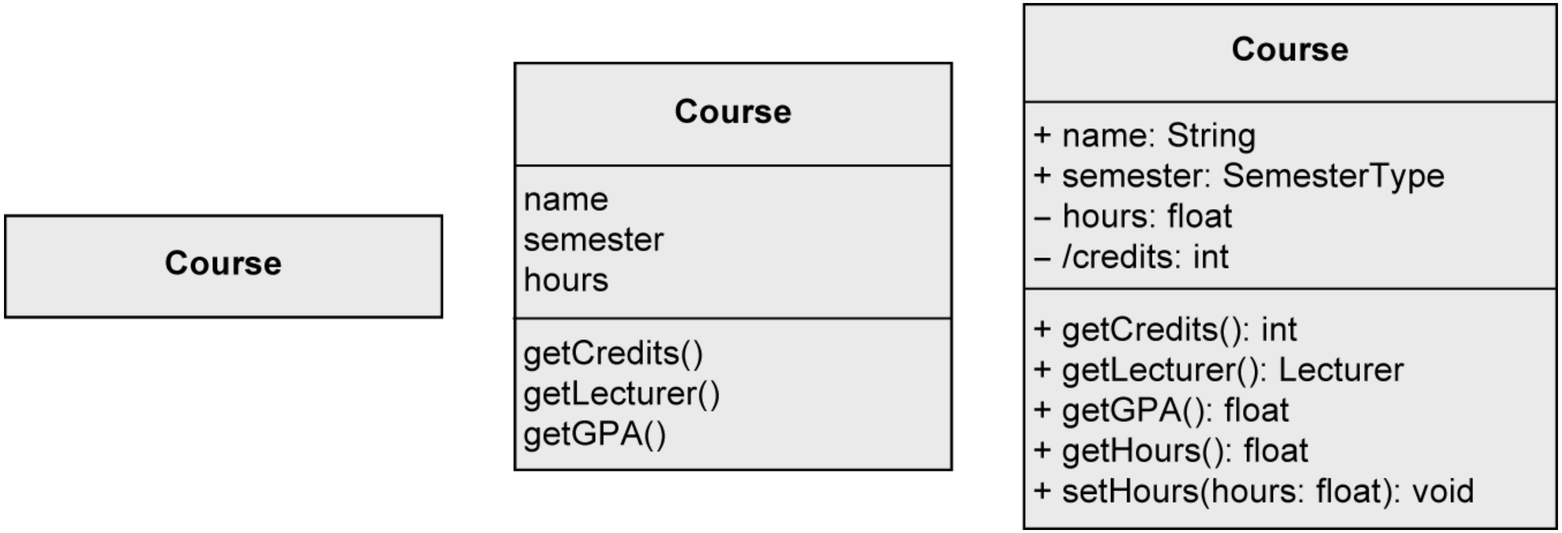
Operations

Course

name: String
semester: SemesterType
hours: float

calculateGPA()
enrollStudent()
assignGrade()

Specification of Classes: Different Levels of Detail



Course

Course

name
semester
hours

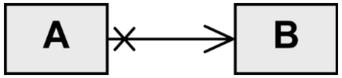
getCredits()
getLecturer()
getGPA()

Course

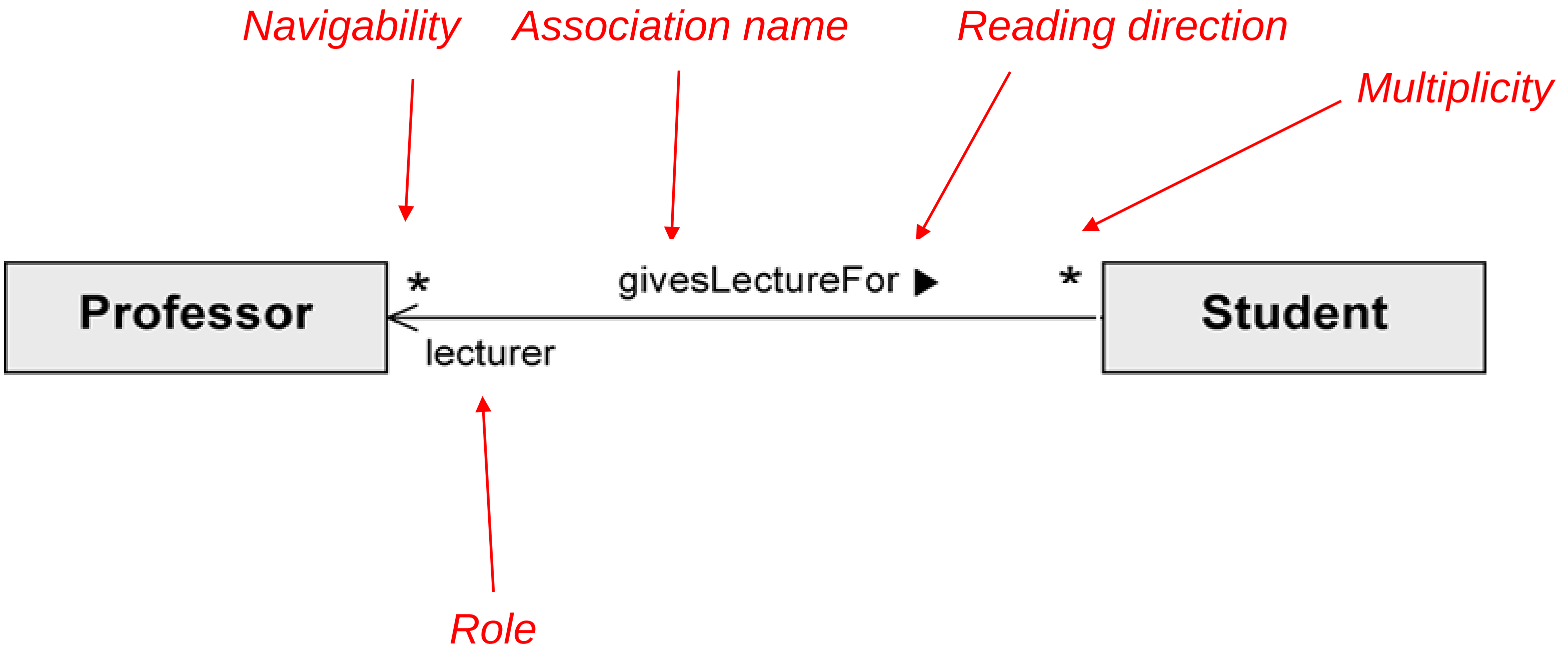
+ name: String
+ semester: SemesterType
– hours: float
– /credits: int

+ getCredits(): int
+ getLecturer(): Lecturer
+ getGPA(): float
+ getHours(): float
+ setHours(hours: float): void

Binary Association

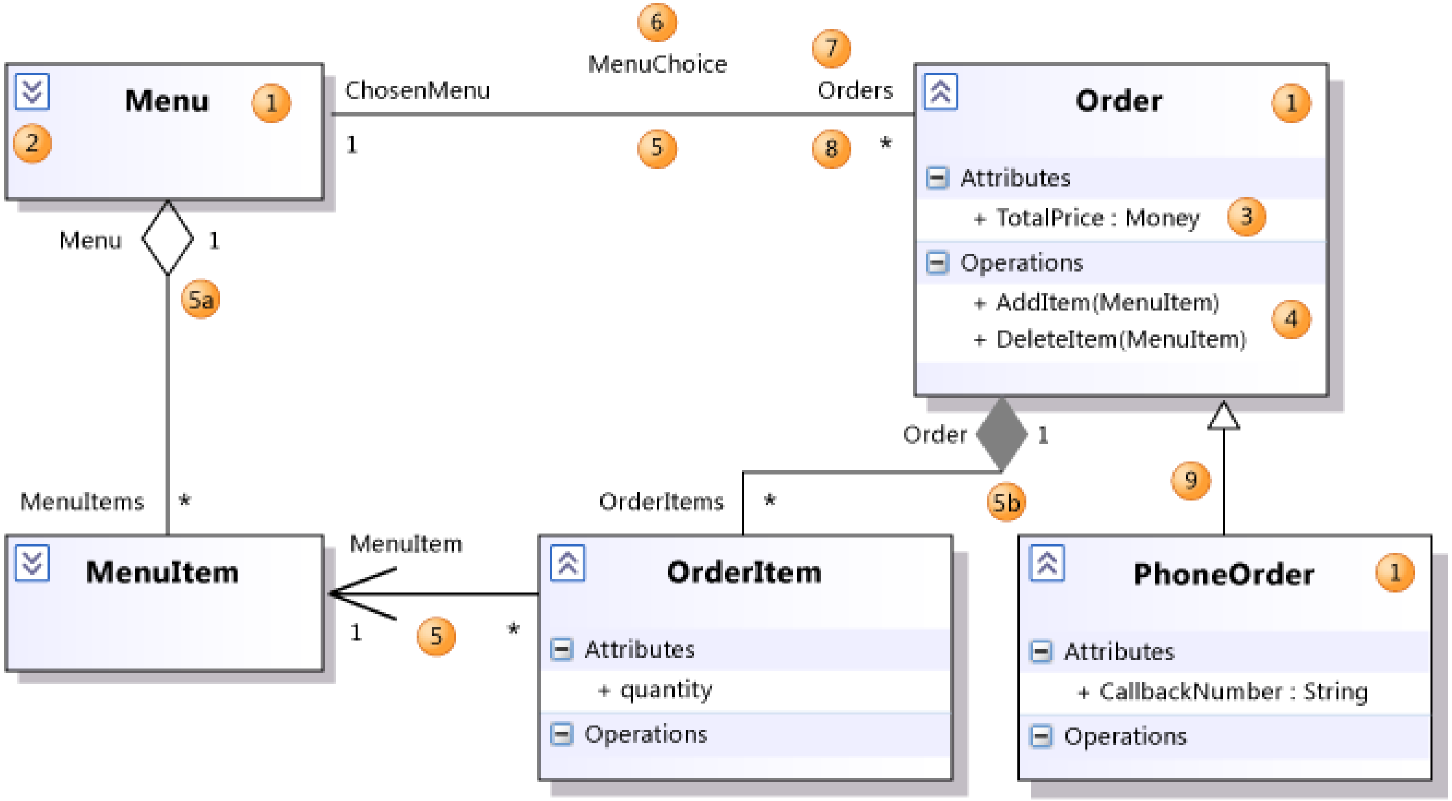


- **Association** is a relationship between classes
- Connects instances of two classes with one another



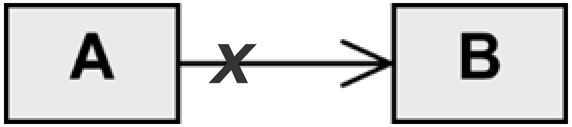
Binary Association - Navigability

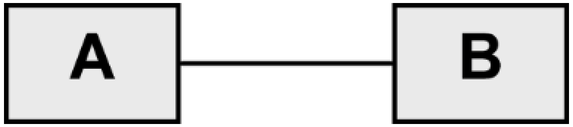
- Navigability: an object knows its partner objects and can therefore access their visible attributes and operations
  - Indicated by open arrow head



Binary Association - Navigability

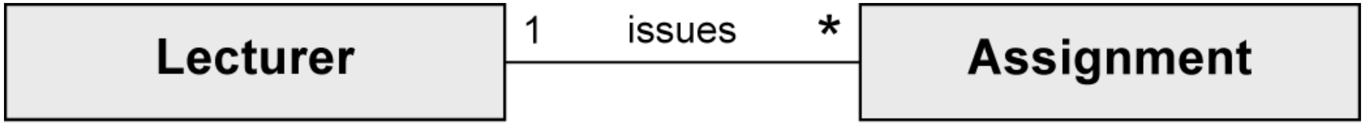
- Non-navigability
  - Indicated by cross
- Example:
  - **A** can access the visible attributes and operations of **B**
  - **B** cannot access any attributes and operations of **A**
- Navigability undefined
  - Bidirectional navigability is assumed

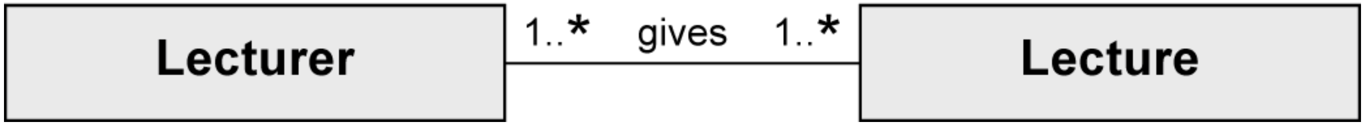




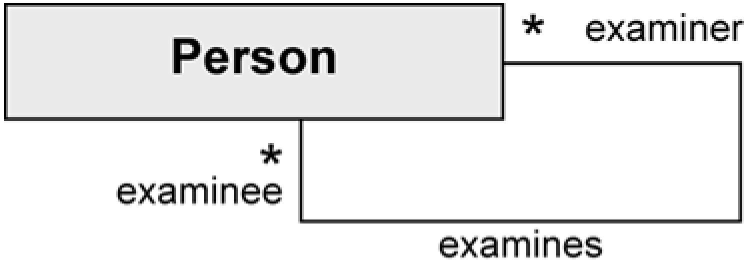
Binary Association – Multiplicity and Role

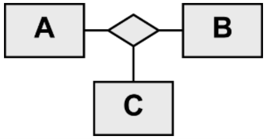
- Multiplicity: Number of objects that may be associated with exactly one object of the opposite side





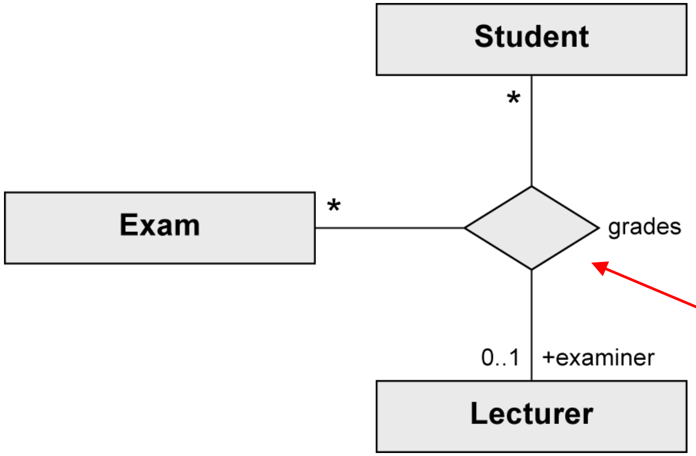
- Role: describes the way in which an object is involved in an association relationship





n-ary Association (1/2)

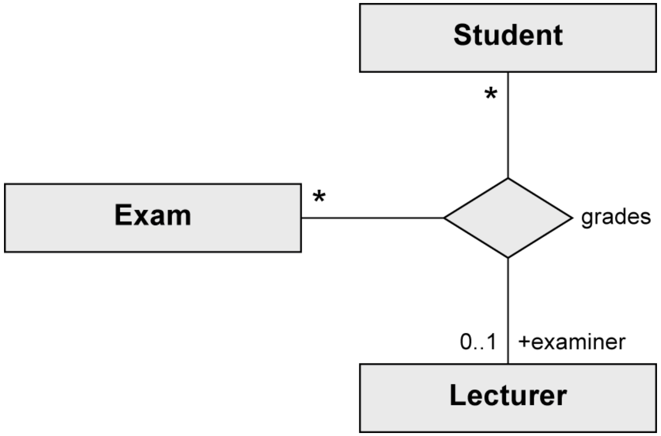
- More than two partner objects are involved in the relationship.
- No navigation directions



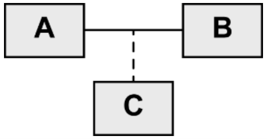
Ternary association

n-ary Association (2/2)

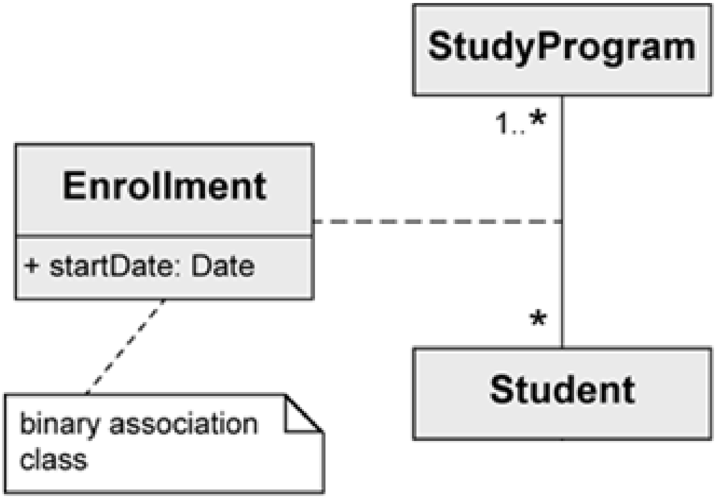
- Example
  - (**`Student`**, **`Exam`**) → (**`Lecturer`**)
    - One student takes one exam with one or no lecturer
  - (**`Exam`**, **`Lecturer`**) → (**`Student`**)
    - One exam with one lecturer can be taken by any number of students
  - (**`Student`**, **`Lecturer`**) → (**`Exam`**)
    - One student can be graded by one **`Lecturer`** for any number of exams



Association Class

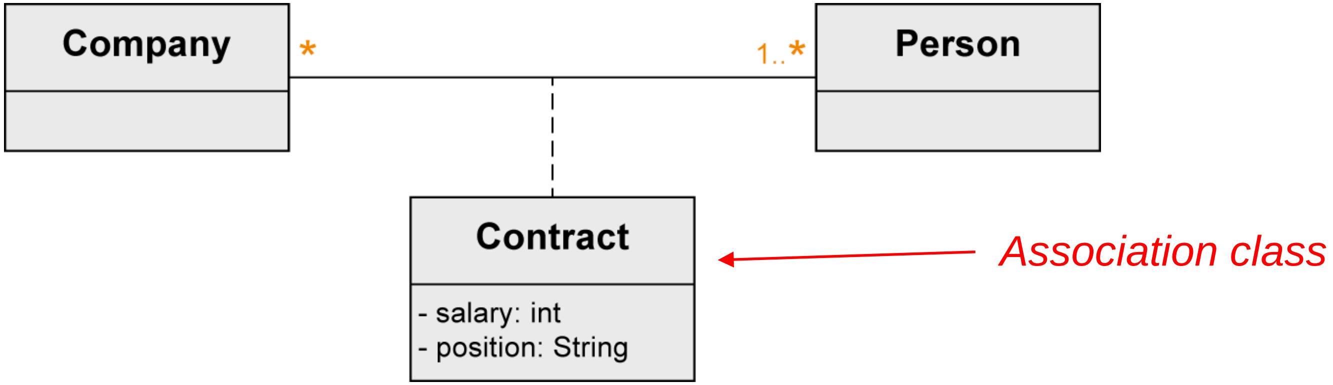


- An Association Class is a class that is part of an association relationship between two other classes.
- An association class is used when **an attribute goes with the association rather than with any of the connected classes**.
- Often used with many-to-many associations.

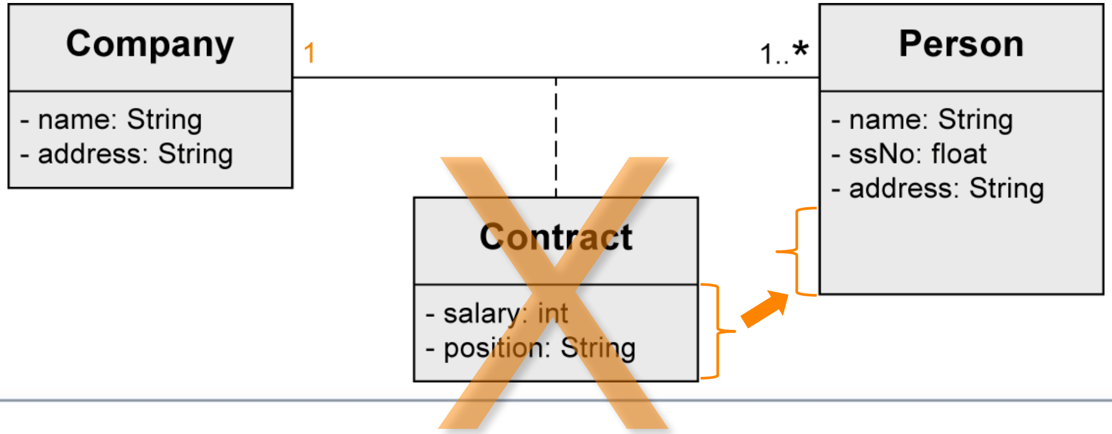


Association Class

- Necessary when modeling n:m Associations

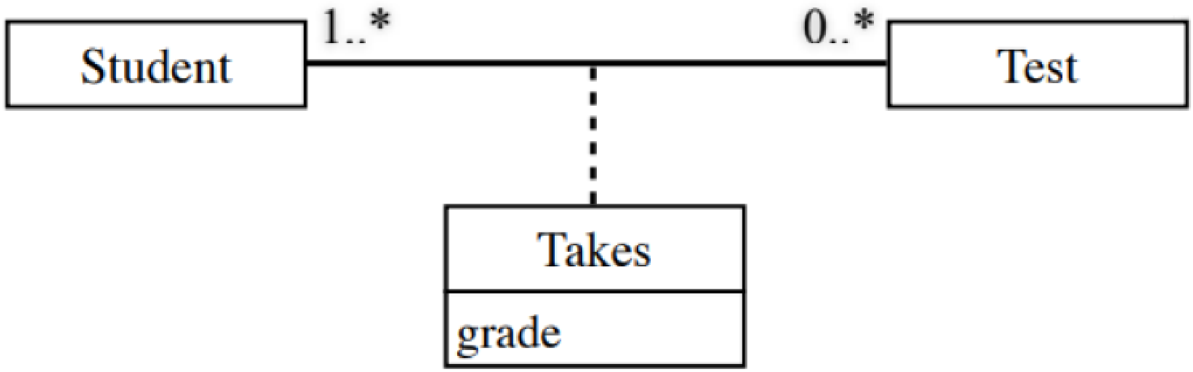


- With 1:1 or 1:n possible but not necessary

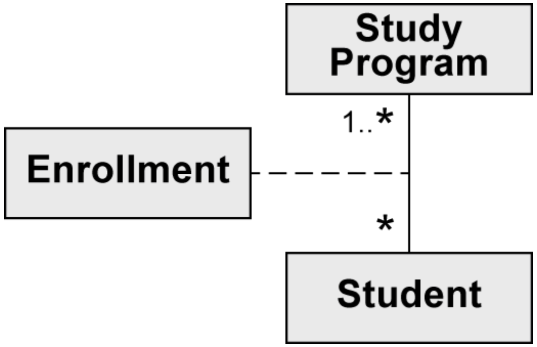


Association Class

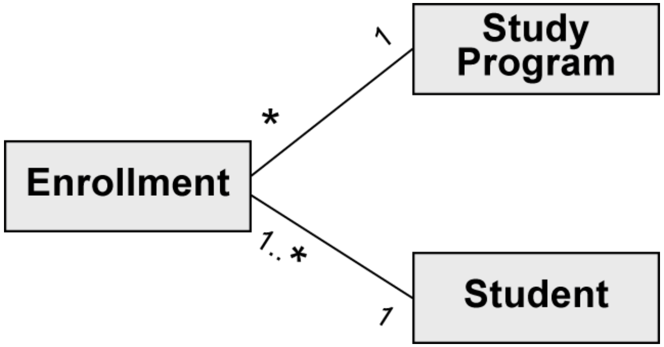
- We want to keep track of the grade a student has got for a test.
- The attribute does not belong to the student, because a student may take many tests.
- Neither does it belong to a test, because many students take the same test.



Association Class vs. Regular Class



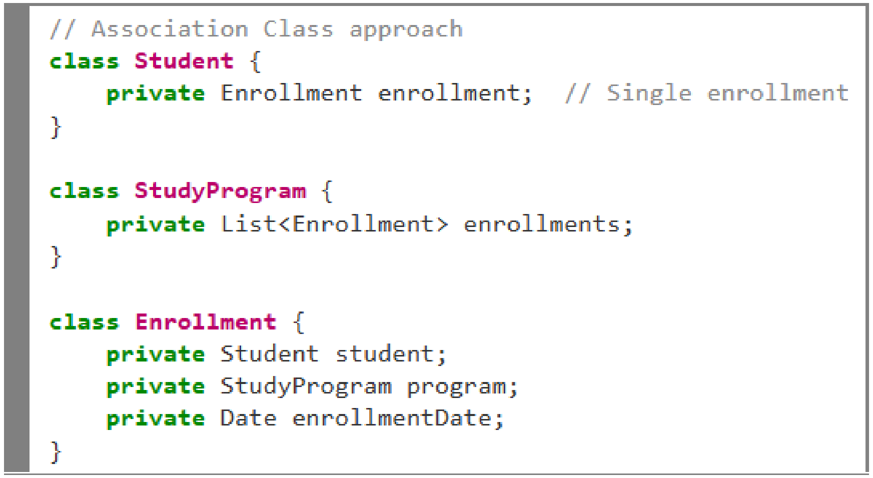
≠



A `Student` can enroll for one particular `StudyProgram` only once

A `Student` can have mutiple `Enrollment`s for one and the same `StudyProgram`

Association Class vs. Regular Class



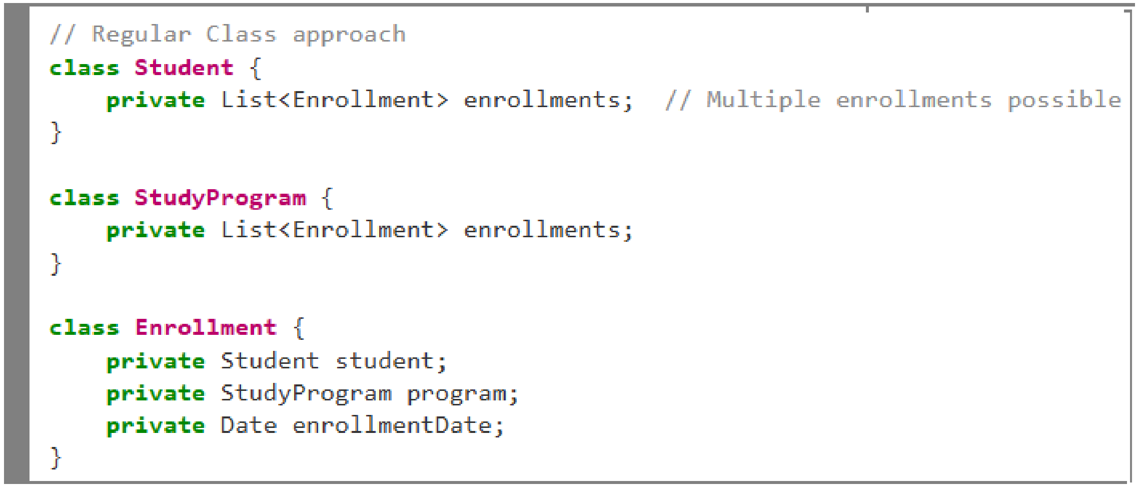
```
// Association Class approach
class Student {
    private Enrollment enrollment;  // Single enrollment
}

class StudyProgram {
    private List<Enrollment> enrollments;
}

class Enrollment {
    private Student student;
    private StudyProgram program;
    private Date enrollmentDate;
}
```

Use Association Class when the relationship itself has attributes but represents a strict one-time connection

Use Regular Class when you need to support multiple relationships between the same entities

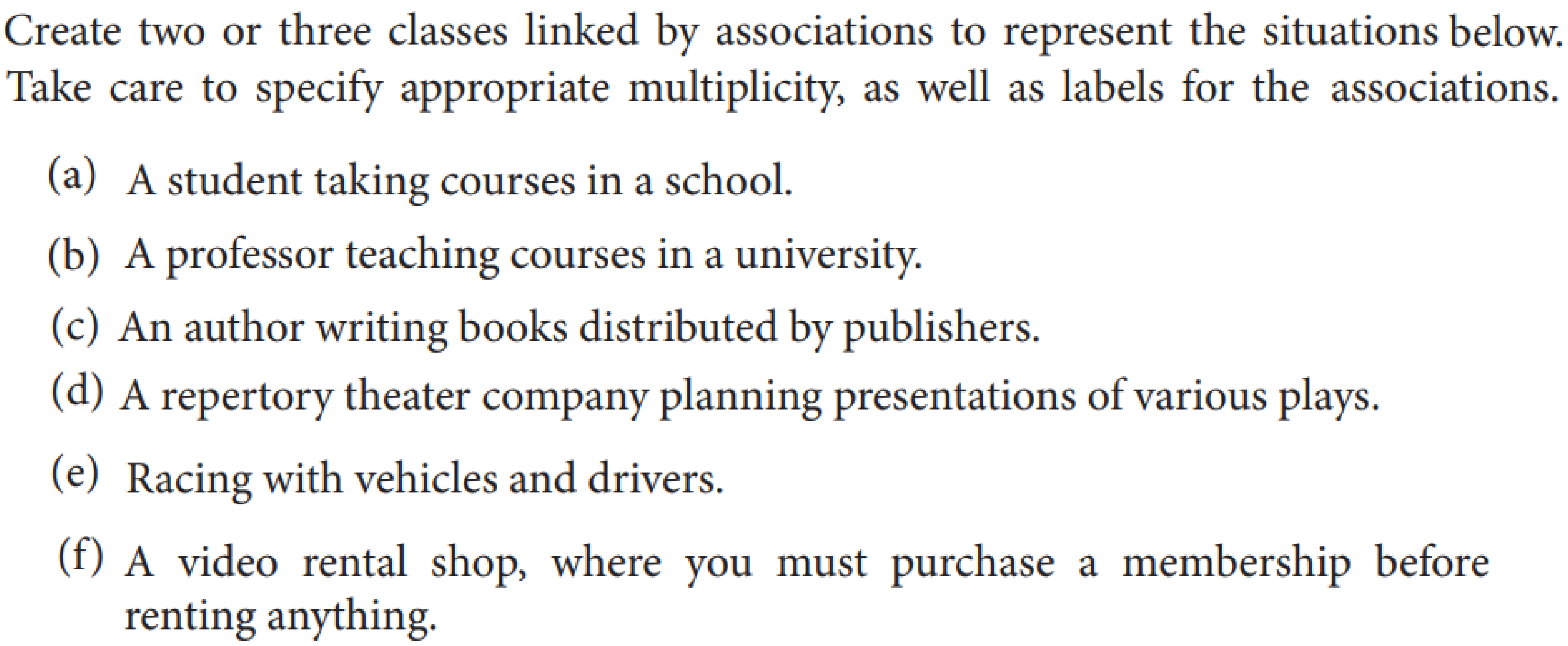


```
// Regular Class approach
class Student {
    private List<Enrollment> enrollments;  // Multiple enrollments possible
}

class StudyProgram {
    private List<Enrollment> enrollments;
}

class Enrollment {
    private Student student;
    private StudyProgram program;
    private Date enrollmentDate;
}
```

Exercise



Create two or three classes linked by associations to represent the situations below. Take care to specify appropriate multiplicity, as well as labels for the associations.

(a) A student taking courses in a school.

(b) A professor teaching courses in a university.

(c) An author writing books distributed by publishers.

(d) A repertory theater company planning presentations of various plays.

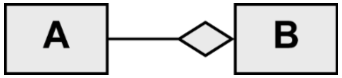
(e) Racing with vehicles and drivers.

(f) A video rental shop, where you must purchase a membership before renting anything.

Aggregation and Composition

- Special form of association
- Used to express that a class **(constituent or part)** is part of another class **(whole or aggregate)**
- Properties of the aggregation/composition association:
  - **Transitive**: if **C** is part of **B** and **B** is part of **A** , **C** is also part of **A**
  - **Asymmetric:** If **A** is part of **B**, then **B** cannot be part of **A**

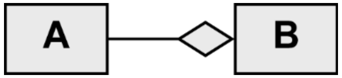
Aggregation



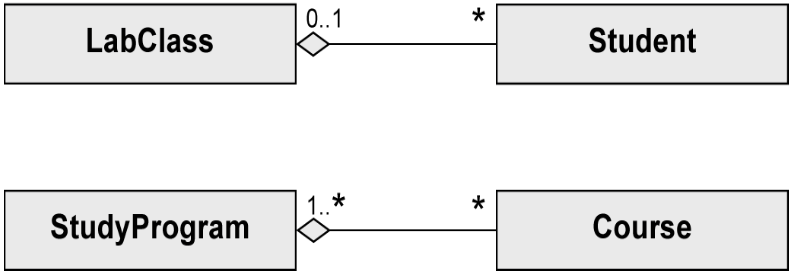
- Aggregation ("has-a" relationship):
- Weaker relationship between objects
- Objects can exist independently: part can exist without aggregate
- part can be shared among multiple aggregates

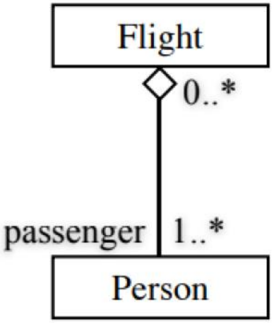
- One element can be part of multiple other elements simultaneously. Example :
  - A path is an ordered set of segments.
  - A segment can belong to several paths. The path "needs" its segments.

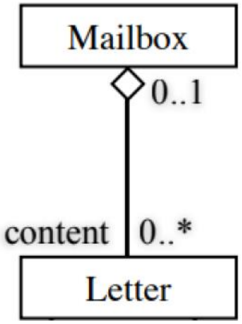
Aggregation



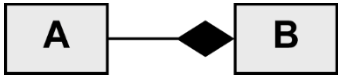
- Syntax: Hollow diamond at the association end of the aggregate.
- Example:
  - `Student` is part of `LabClass`
  - `Course` is part of `StudyProgram`





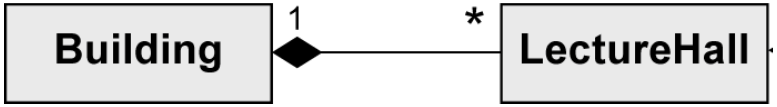


Composition



- Existence dependency between the **composite** object and its **parts**
- If the composite object is deleted, its parts are also deleted.
- 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
- Syntax: Solid diamond at the aggregating end
- Example: **`LectureHall`** is part of **`Building`**



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

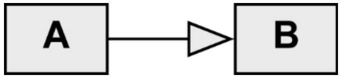
Real-world Examples

- : Aggregation:
  - School - Student
  - Library - Book
  - Company - Employee
  - Department - Professor
- Composition:
  - House - Room
  - Computer - CPU
  - Book - Page
  - Car - Engine
  - Car - Wheel

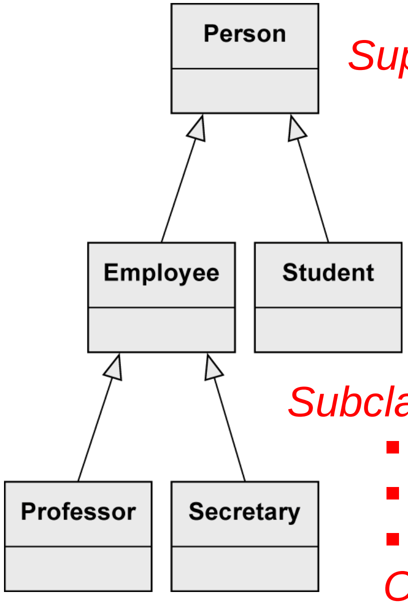
Tips for identifying aggregation and composition

- If the whole object is deleted, do the part objects still exist?
- Check the multiplicity of the whole side
  - 0..*, 1..*: aggregation
  - 1..1: composition
- Use Aggregation when:
  - Parts need to exist independently
  - Parts can be shared among multiple wholes
  - Relationship is temporary or optional usually (0..* on the whole side)
- Use Composition when:
  - Parts are fundamental to the whole
  - Parts shouldn't exist independently
  - Parts are exclusively owned by one whole
  - Relationship is permanent and mandatory usually (1..1 on the whole side)

Generalization



- Every instance of a subclass is simultaneously an indirect instance of the superclass.
- Subclass inherits all characteristics, associations, and aggregations of the superclass except private ones.
- Subclass may have further characteristics, associations, and aggregations.
- Generalizations are transitive.



Superclass

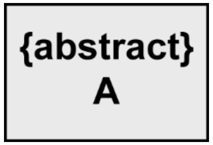
Subclasses inherit

- *characteristics,*
- *associations, and*
- *aggregations*

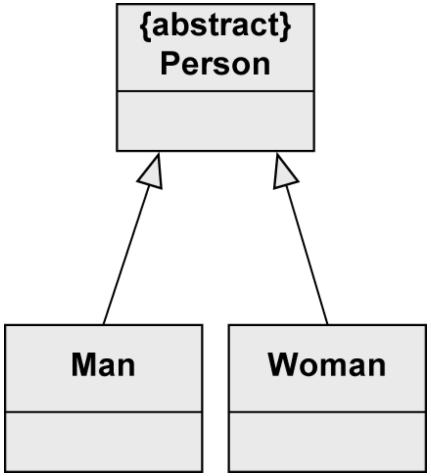
Of superclass

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

Generalization – Abstract Class



- Used to highlight common characteristics of their subclasses.
- Used to ensure that there are no direct instances of the superclass.
- Only its non-abstract subclasses can be instantiated.
- Useful in the context of generalization relationships.
- Notation: keyword **`{abstract}`** or class name in italic font.



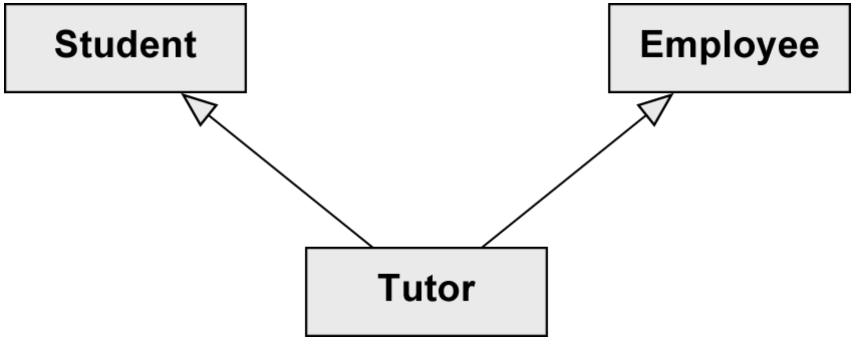
No `Person`-object possible

Two types of `Person`: `Man` and `Woman`

Generalization – Multiple Inheritance

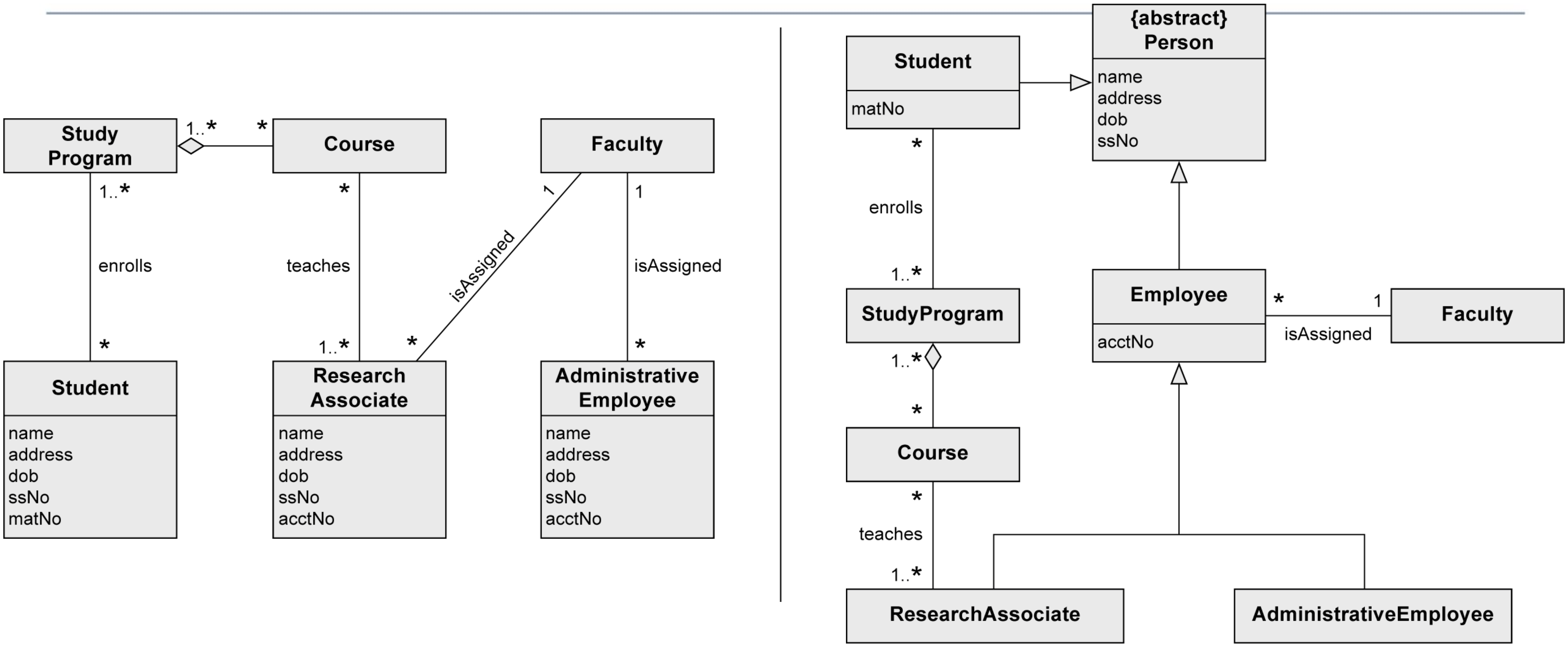
- UML allows multiple inheritance.
- A class may have multiple super-classes.

- Example:



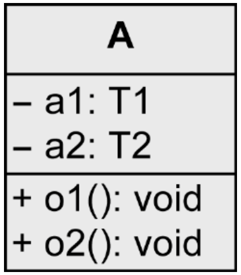
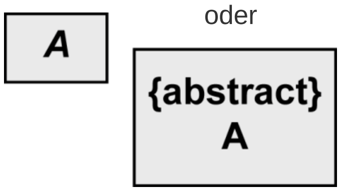
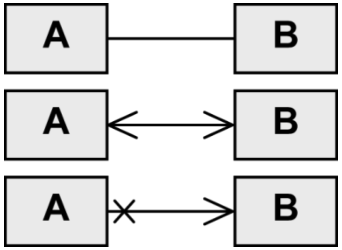
A `Tutor` is both an `Employee` and a `Student`

With and Without Generalization

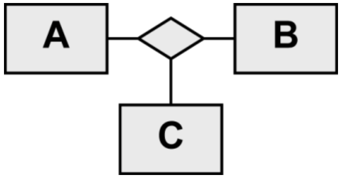
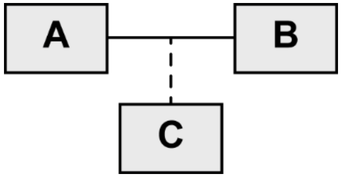


Simplification: Abstraction simplifies complex systems by focusing on essential features and hiding unnecessary details.
Reusability: Abstraction encourages code reuse. Once you've defined abstract classes, you can create multiple concrete implementations that adhere to the same abstraction.

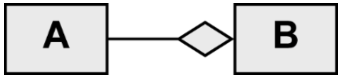
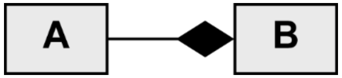
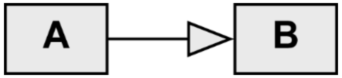
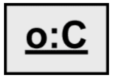
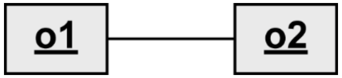
Notation Elements (1/3)

Name	Notation	Description
Class		Description of the structure and behavior of a set of objects
Abstract class		Class that cannot be instantiated
Association		Relationship between classes: navigability unspecified, navigable in both directions, not navigable in one direction

Notation Elements (2/3)

Name	Notation	Description
n-ary association		Relationship between n (here 3) classes
Association class		More detailed description of an association

Notation Elements (3/3)

Name	Notation	Description
Shared aggregation		Parts-whole relationship (**A** is part of **B**)
Strong aggregation = composition		Existence-dependent parts-whole relationship (**A** is part of **B**)
Generalization		Inheritance relationship (**A** inherits from **B**)
Object		Instance of a class
Link		Relationship between objects

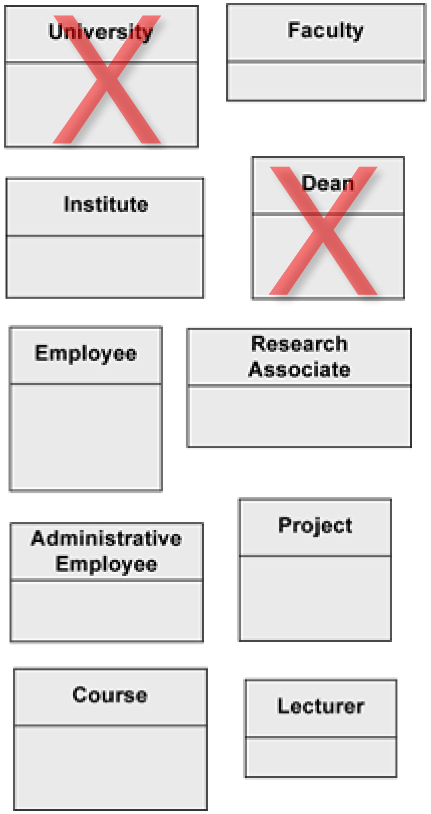
Example – University Information System

- A university consists of multiple faculties. Faculty is 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. Faculty is 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.

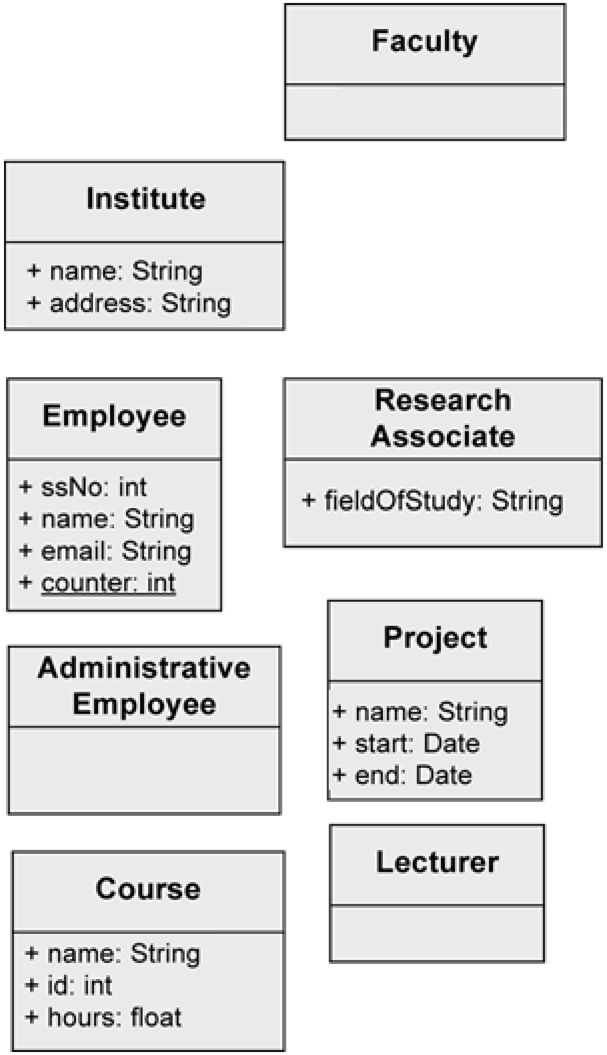
We model the system University



Dean has no further attributes than any other employee

Example – Step 2: Identifying the Attributes

- A university consists of multiple faculties. Faculty is 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.



Faculty

Institute
+ name: String
+ address: String

Employee
+ ssNo: int
+ name: String
+ email: String
+ counter: int

Research Associate
+ fieldOfStudy: String

Administrative Employee

Project
+ name: String
+ start: Date
+ end: Date

Course
+ name: String
+ id: int
+ hours: float

Lecturer

Example – Step 3: Identifying Relationships (1/7)

Binary Association

N-ary Association

Association Class

Aggregation

Composition

Generalization

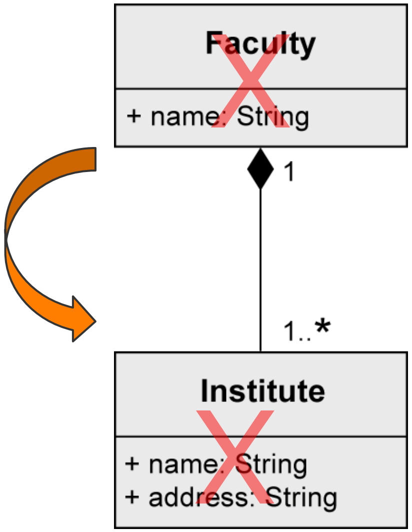
Strong Dep

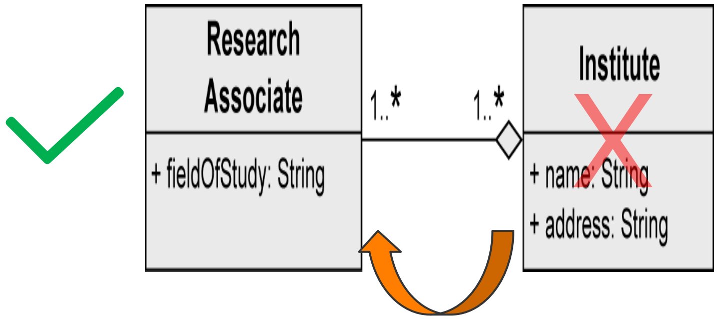
Weak Dep

Composition

Aggregation

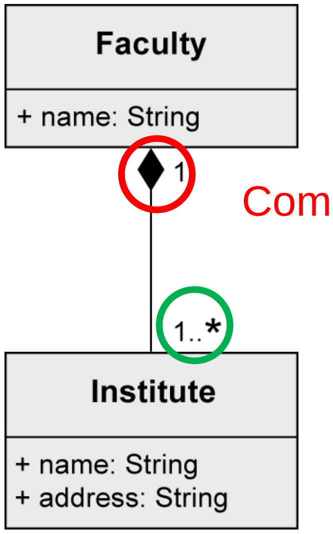
To show
dependency between Classes





Example – Step 3: Identifying Relationships (2/7)

- *“Faculty is composed of various institutes.”*

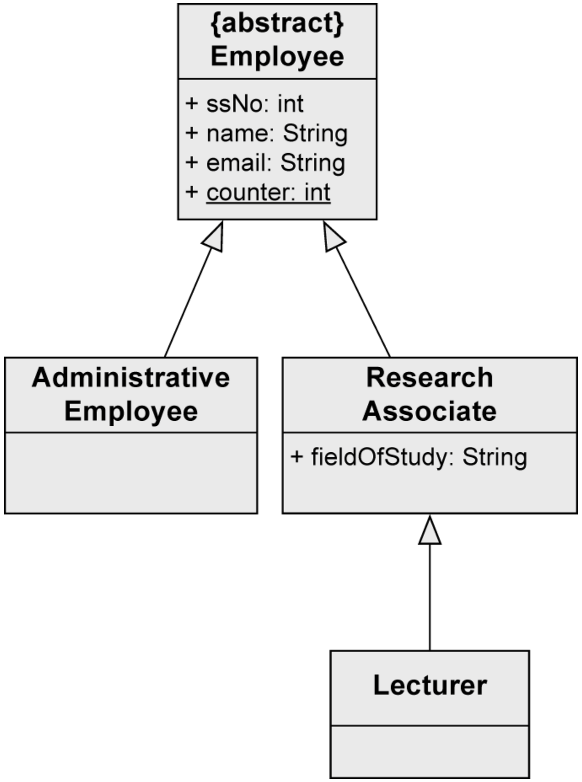


Composition to show existence dependency

Example – Step 3: Identifying Relationships (3/7)

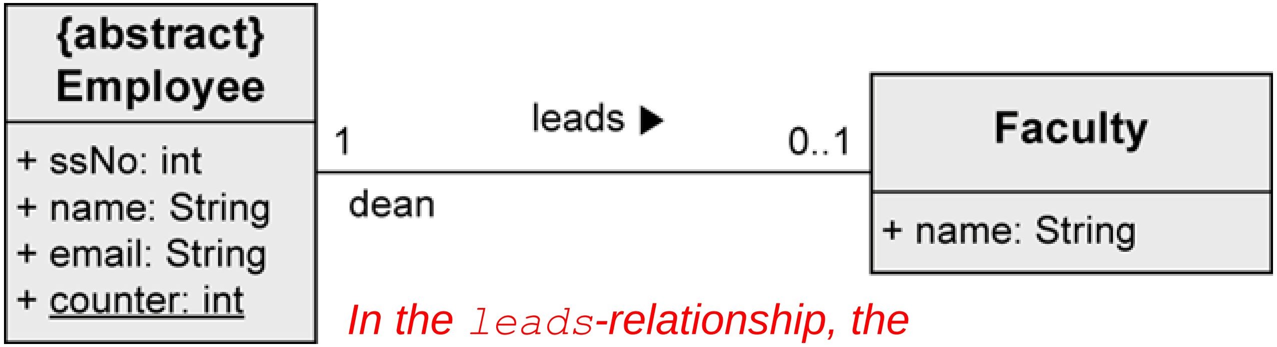
- Indication of a generalization
- *“Employees have a social security number, a name, and an email address. There is a distinction between research and administrative personnel.”*
- *“Some research associates hold courses. Then they are called lecturers.”*

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



Example – Step 3: Identifying Relationships (4/7)

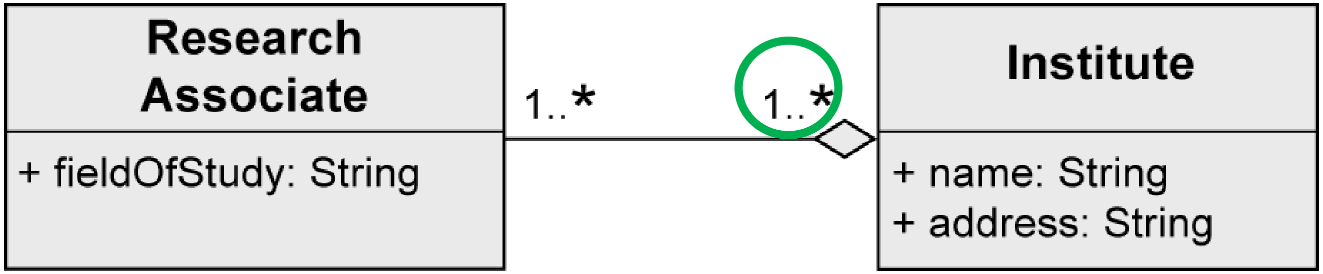
- *“Each faculty is led by a dean, who is an employee of the university”*



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

Example – Step 3: Identifying Relationships (5/7)

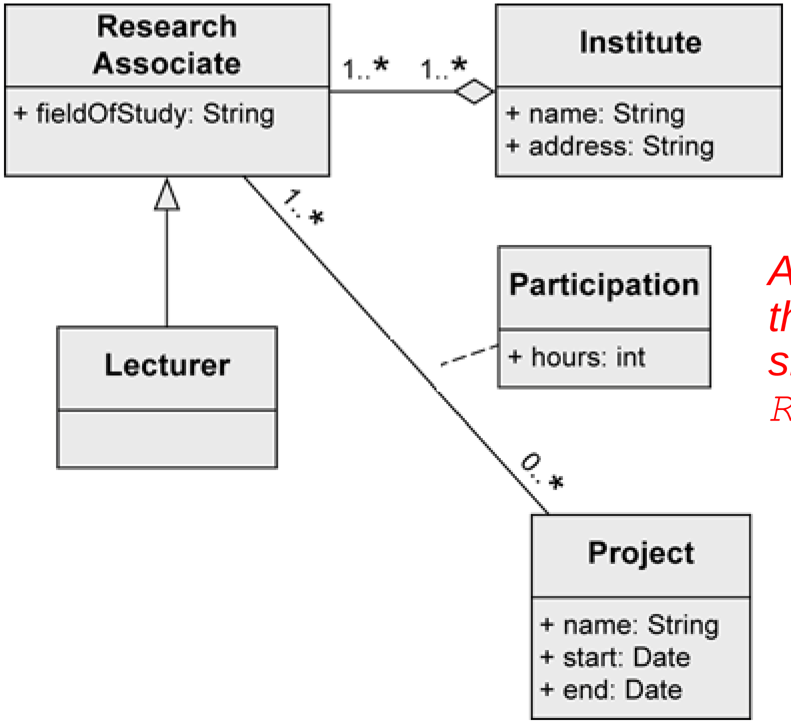
- “Research associates are assigned to at least one institute.”



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

Example – Step 3: Identifying Relationships (6/7)

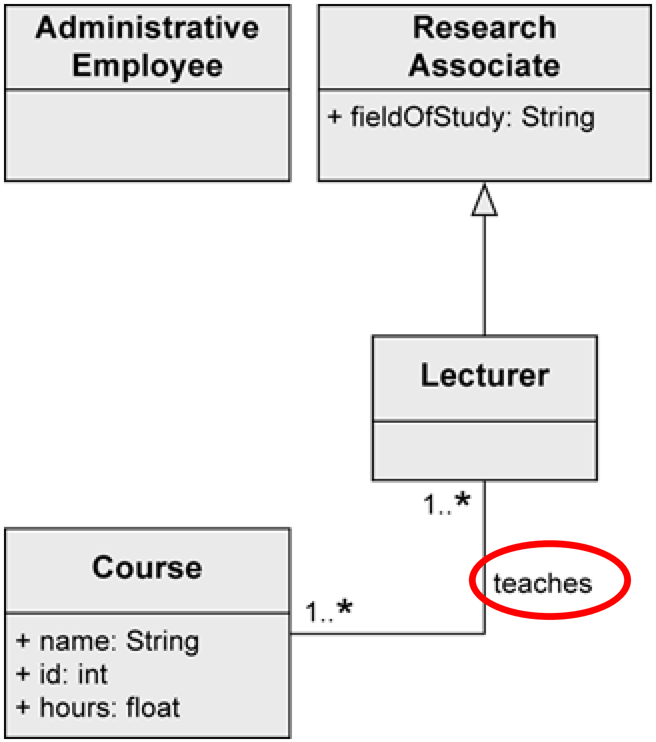
- *“Furthermore, research associates can be involved in projects for a certain number of hours.”*



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

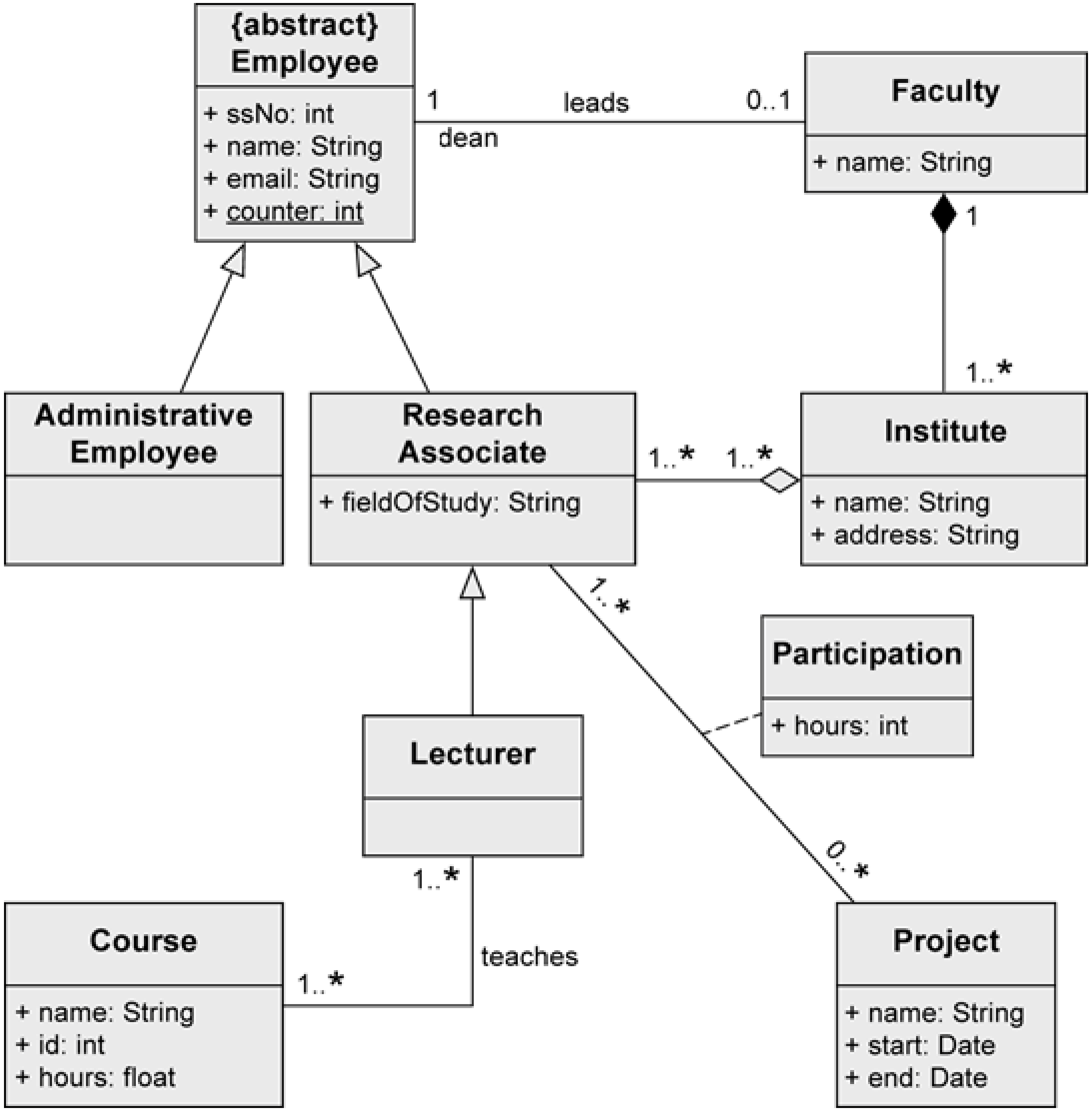
Example – Step 3: Identifying Relationships (7/7)

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



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

Example – Complete Class Diagram


{abstract}
Employee
+ ssNo: int
+ name: String
+ email: String
+ counter: int
1
leads
0..1
dean
Faculty
+ name: String
1
1..*
Administrative Employee
Research Associate
+ fieldOfStudy: String
1..*
1..*
Institute
+ name: String
+ address: String
1..*
Participation
+ hours: int
Lecturer
0..*
1..*
teaches
Course
+ name: String
+ id: int
+ hours: float
1..*
Project
+ name: String
+ start: Date
+ end: Date