MÔ HÌNH HOÁ PHẦN MỀM

TUẦN 3:
CLASS DIAGRAM

GVLT: NGUYỄN THỊ MINH TUYỀN

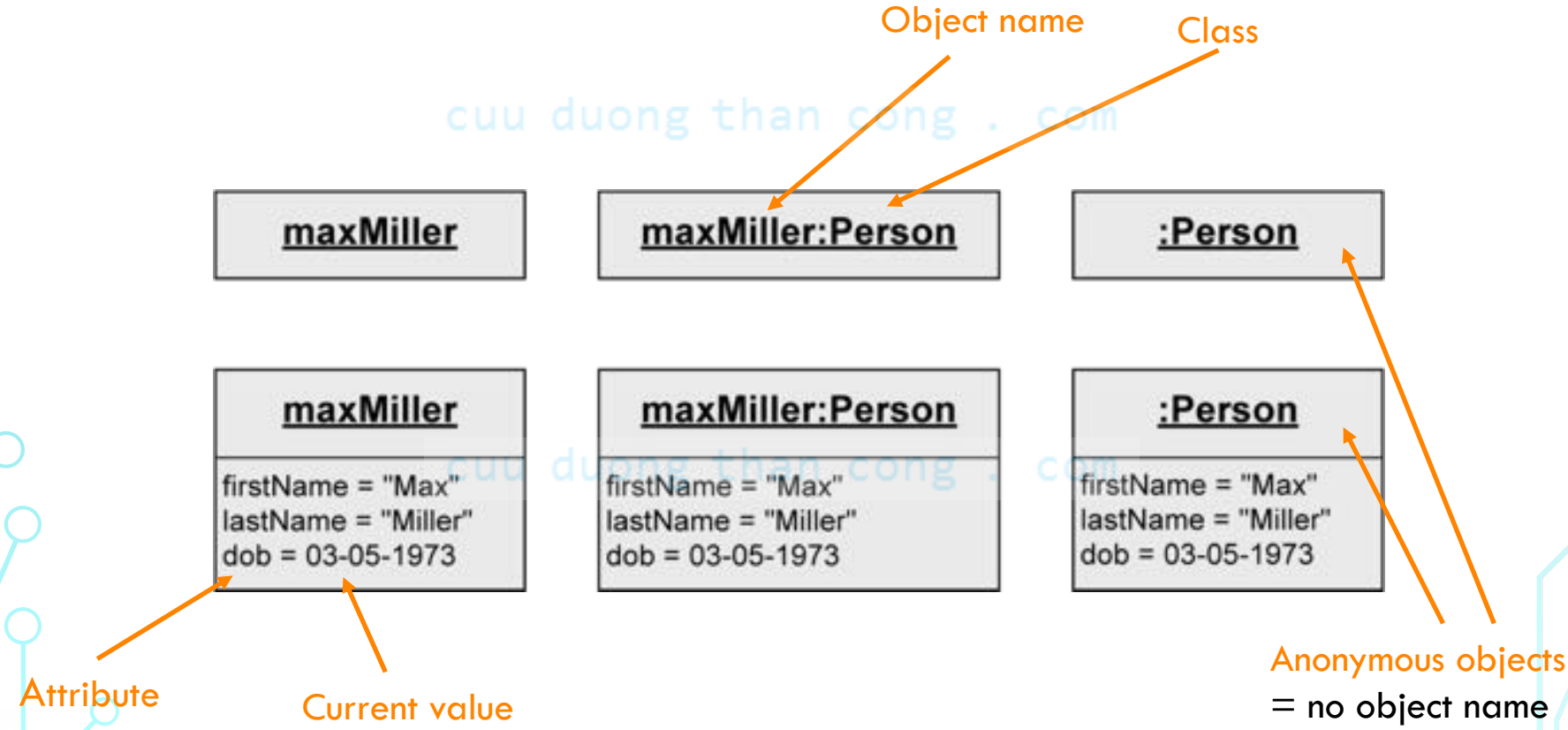
NỘI DUNG

1. Đối tượng
2. Lớp
3. Các thành phần cơ bản
4. Tạo một biểu đồ lớp
5. Phát sinh mã nguồn

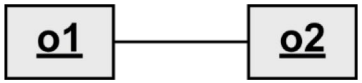
ĐỐI TƯỢNG (OBJECT)

o:C

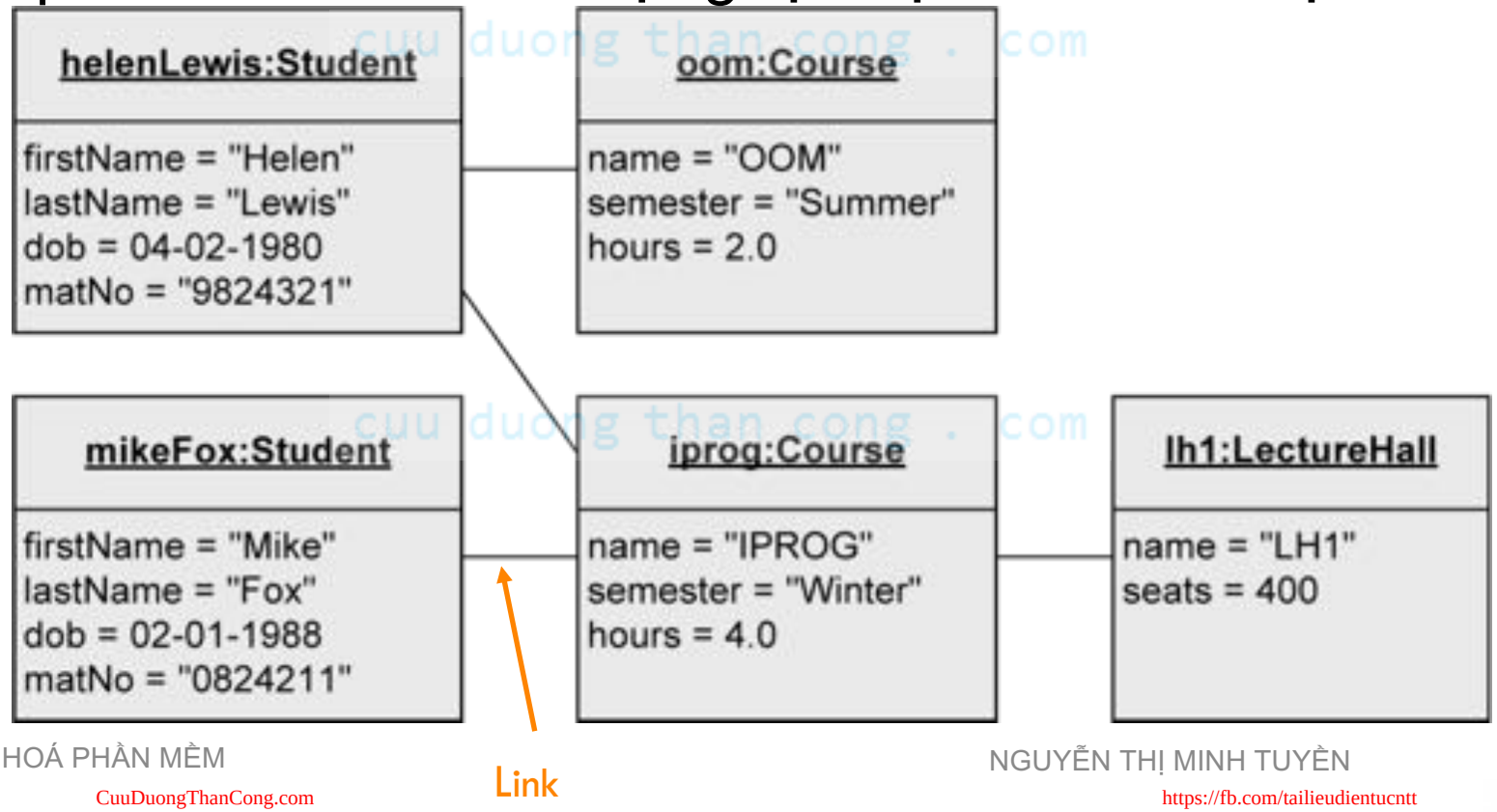
- Các cá thể của một hệ thống
- Các ký hiệu thay thế:



BIỂU ĐỒ ĐỐI TƯỢNG



- Các đối tượng của một hệ thống và quan hệ giữa chúng gọi là liên kết (link)
- Snapshot của các đối tượng tại một thời điểm cụ thể.

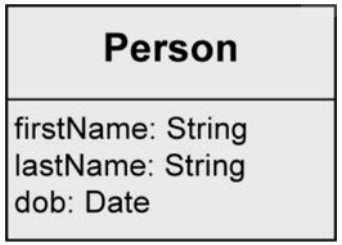


TỪ ĐỐI TƯỢNG ĐẾN LỚP

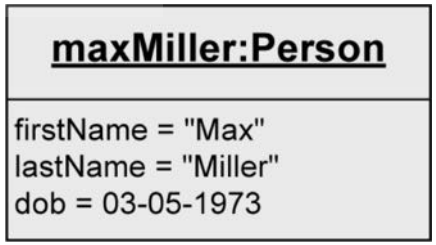
- Nhiều cá thể xuất hiện trong hệ thống có cùng đặc tính và hành vi.
- Nếu mỗi đối tượng được mô hình hoá độc lập nhau → mô hình sẽ trở nên phức tạp và không thể bảo trì được.

➔Sử dụng các lớp cho phép ta mô tả các đối tượng giống nhau mà không phải mô tả chi tiết mỗi một đối tượng một cách riêng lẻ.

Class



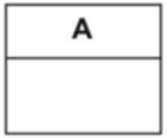
Object of that class



NỘI DUNG

1. Đối tượng
2. **Lớp**
3. Các thành phần cơ bản
4. Tạo một biểu đồ lớp
5. Phát sinh mã nguồn

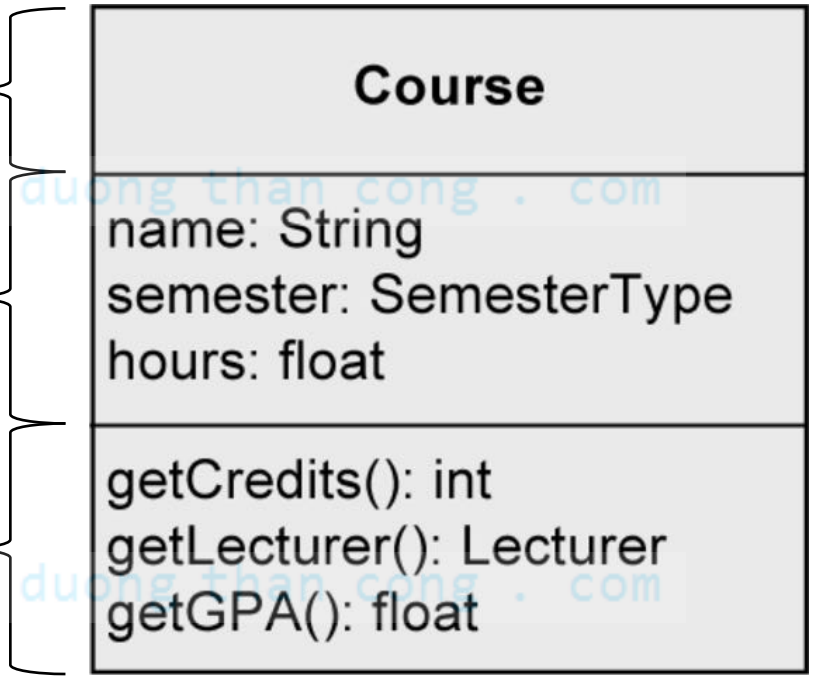
LỚP



Class name

Attributes

Operations

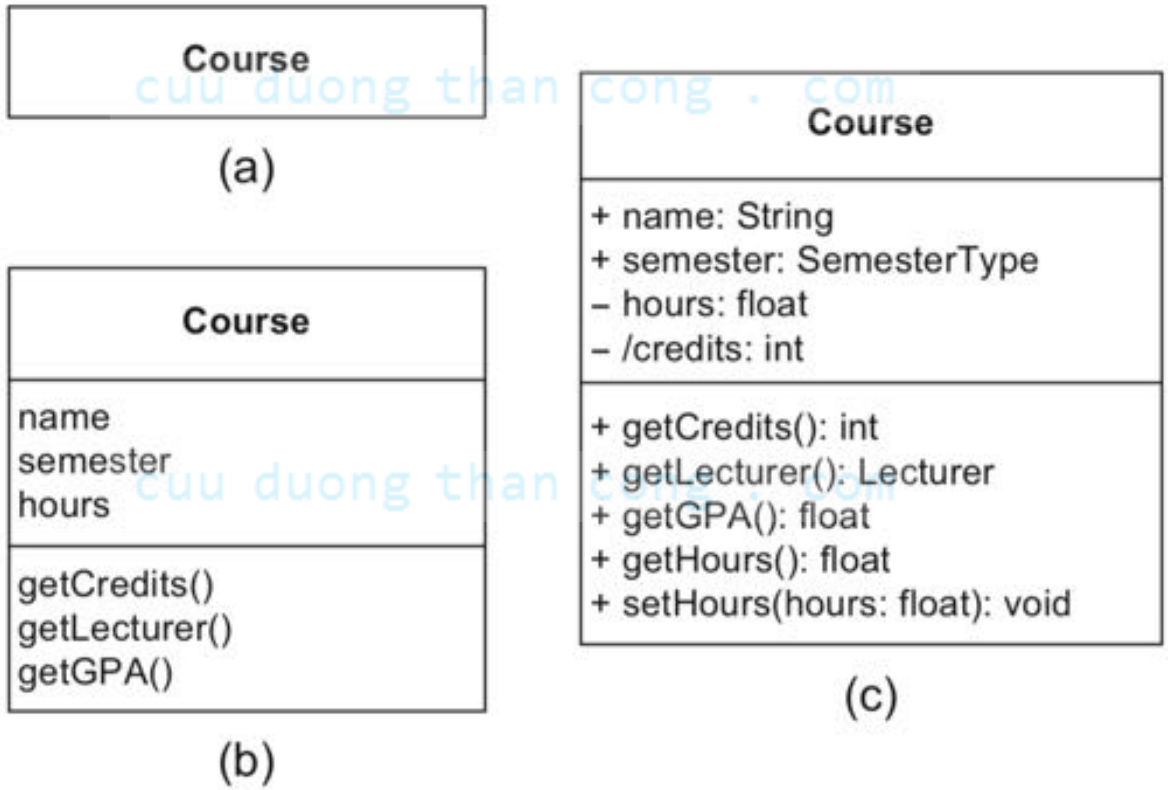


INSTANCE

- Các đối tượng biểu diễn các dạng thức cụ thể của lớp và được gọi là các instance.
- Các đặc tính liên quan của các instance của một lớp được mô tả thông qua định nghĩa các đặc tính cấu trúc (attributes) và hành vi (operations).
  - Các operations cho phép các đối tượng giao tiếp với nhau.

KÝ HIỆU

- Theo quy tắc đặt tên, lớp là các danh từ số ít. Tên lớp nên mô tả lớp sử dụng các từ vựng đặc trưng cho miền ứng dụng.



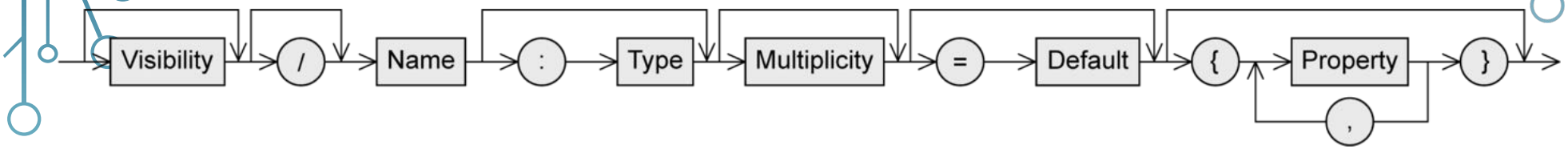
NỘI DUNG

1. Đối tượng
2. Lớp
3. **Các thành phần cơ bản**
    1. **Thuộc tính**
    2. Thao tác
    3. Liên kết
4. Tạo một biểu đồ lớp
5. Phát sinh mã nguồn

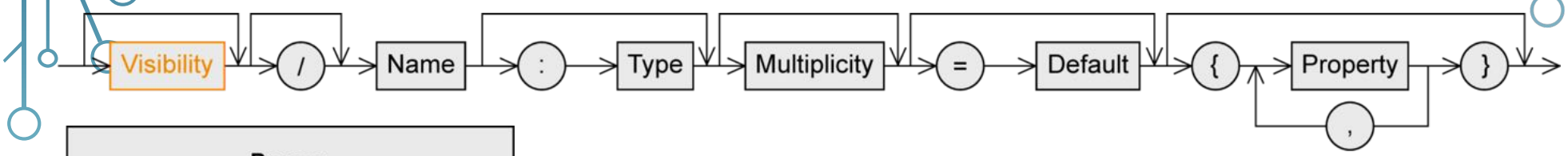
NỘI DUNG

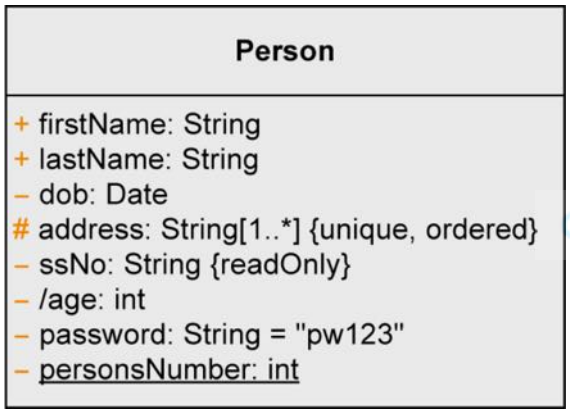
1. Đối tượng
2. Lớp
3. **Các thành phần cơ bản**
    1. **Thuộc tính**
    2. **Thao tác**
    3. **Liên kết**
4. Tạo một biểu đồ lớp
5. Phát sinh mã nguồn

CÚ PHÁP THUỘC TÍNH



VISIBILITY

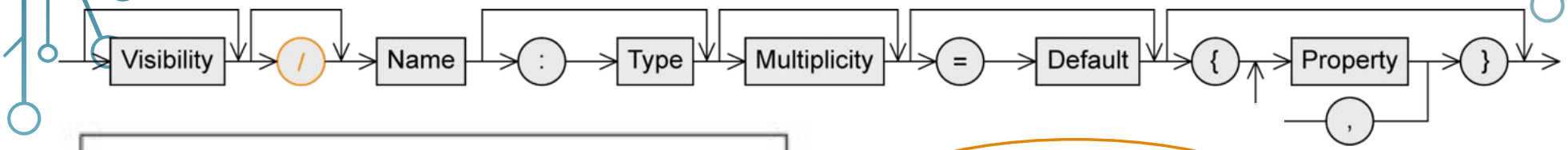


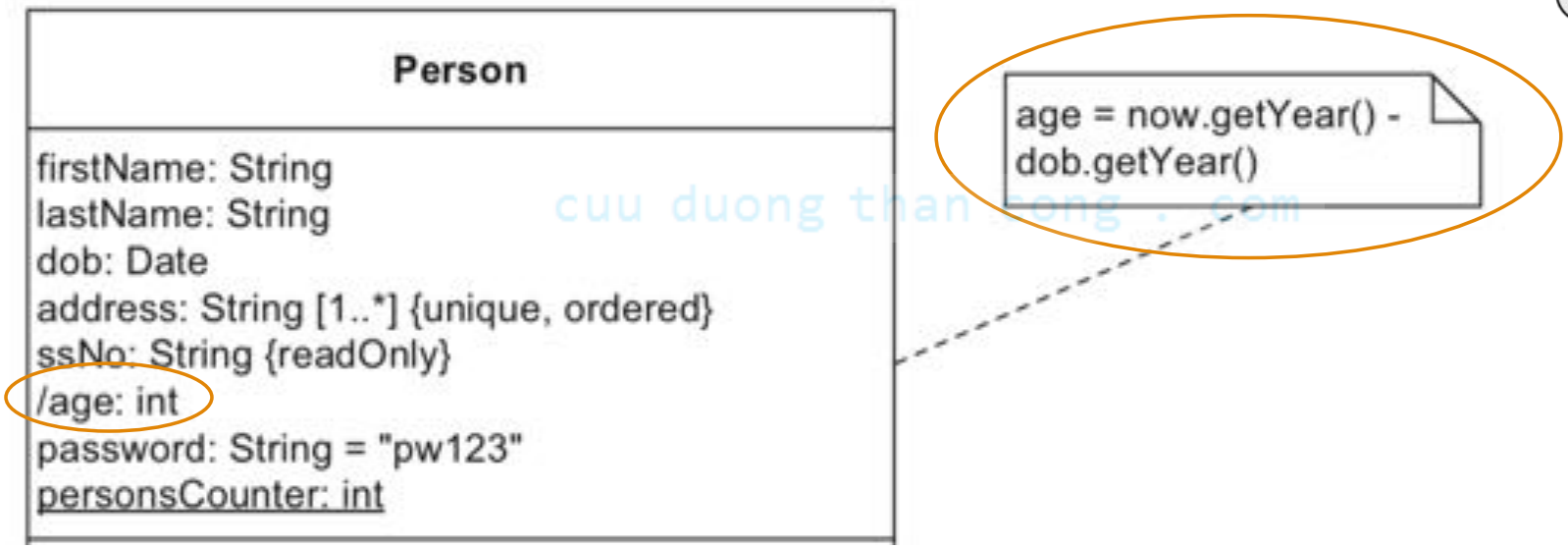


- Ai được phép truy cập vào thuộc tính

+	public	tất cả
-	private	chỉ bản thân đối tượng
#	protected	bản thân lớp và các lớp con
~	package	các lớp nằm trong cùng package

THUỘC TÍNH CÓ NGUỒN GỐC



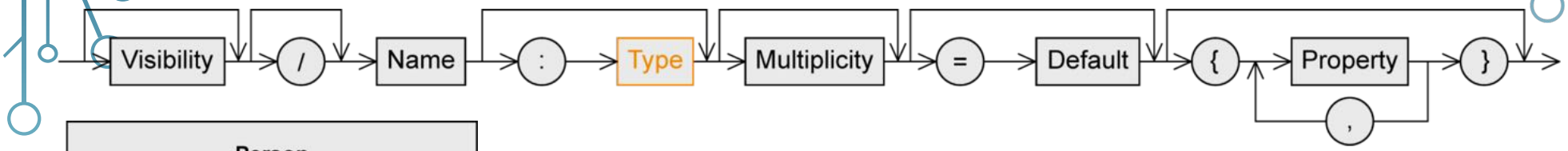


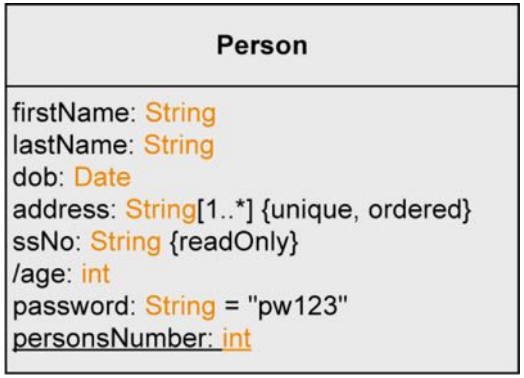
- Giá trị thuộc tính bắt nguồn từ các thuộc tính khác
  - **age**: được tính từ the date of birth

TÊN THUỘC TÍNH

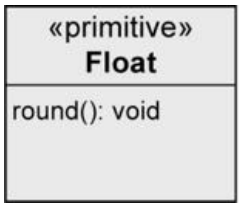
Person
firstName: String lastName: String dob: Date address: String[1..*] {unique, ordered} ssNo: String {readOnly} /age: int password: String = "pw123" personsNumber: int

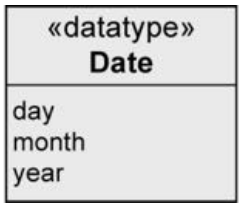
KIỂU

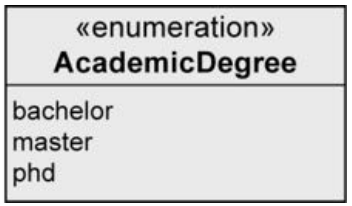




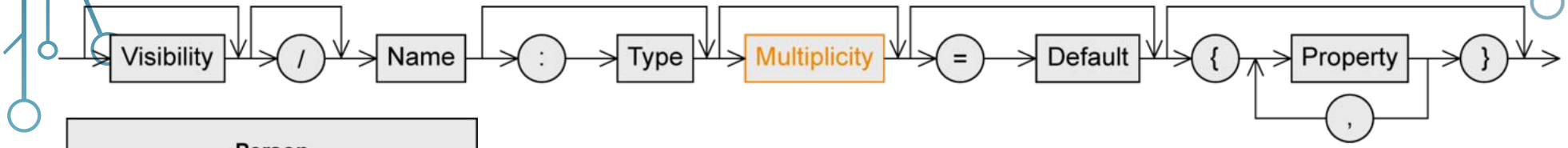
- Các lớp được định nghĩa bởi người dùng
- Kiểu dữ liệu
  - Primitive data type
    - Pre-defined: Boolean, Integer, UnlimitedNatural, String
    - User-defined: «**primitive**»
    - Composite data type: «**datatype**»
  - Enumerations: «**enumeration**»

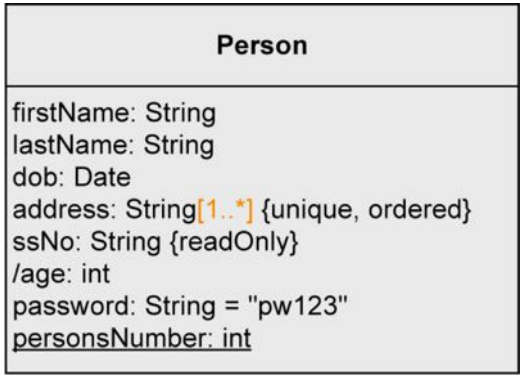




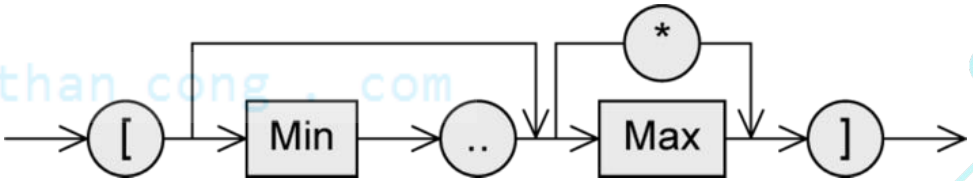


MULTIPLICITY





- Số giá trị một thuộc tính có thể chứa
- Giá trị mặc định: 1
- Ký hiệu: **[min..max]**
  - Không có giới hạn trên: `[*]` or `[0..*]`

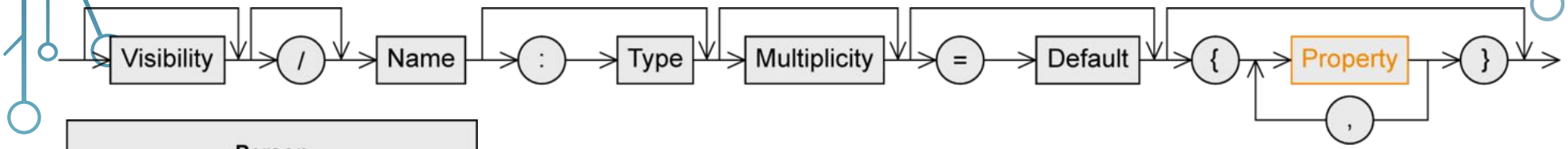


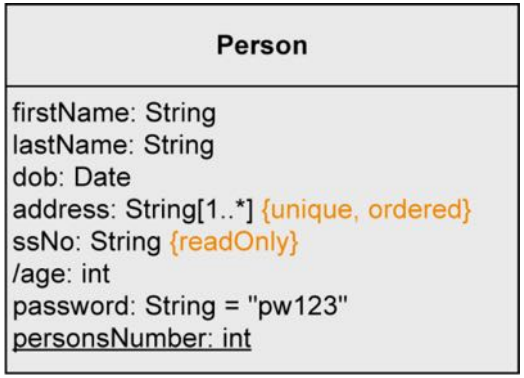
GIÁ TRỊ MẶC ĐỊNH

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

- Được dùng nếu giá trị thuộc tính không được thiết lập rõ ràng bởi người dùng

CÁC THUỘC TÍNH





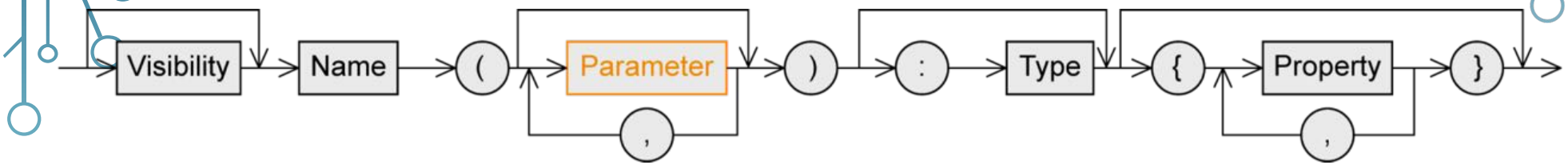
Các thuộc tính định nghĩa trước	
{readOnly}	Giá trị không thay đổi
{unique}	Giá trị là duy nhất
{non-unique}	Cho phép lặp lại giá trị
{ordered}	Cố định thứ tự của các giá trị
{unordered}	không cố định thứ tự của các giá trị

Đặc tả thuộc tính	
{unordered, unique}	Set
{unordered, non-unique}	Multi-set
{ordered, unique}	Ordered set
{ordered, non-unique}	List

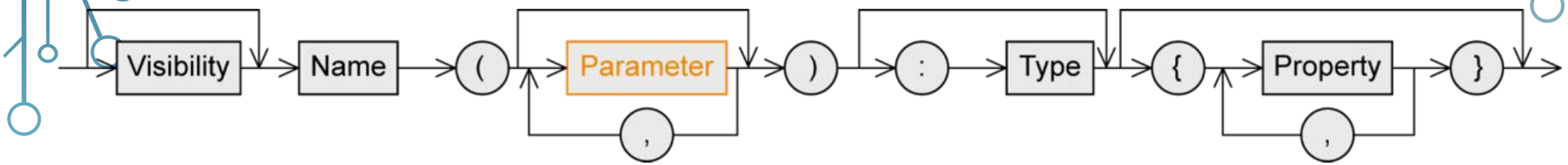
NỘI DUNG

1. Đối tượng
2. Lớp
3. **Các thành phần cơ bản**
    1. Thuộc tính
    2. **Thao tác**
    3. Liên kết
4. Tạo một biểu đồ lớp
5. Phát sinh mã nguồn

CÚ PHÁP THAO TÁC

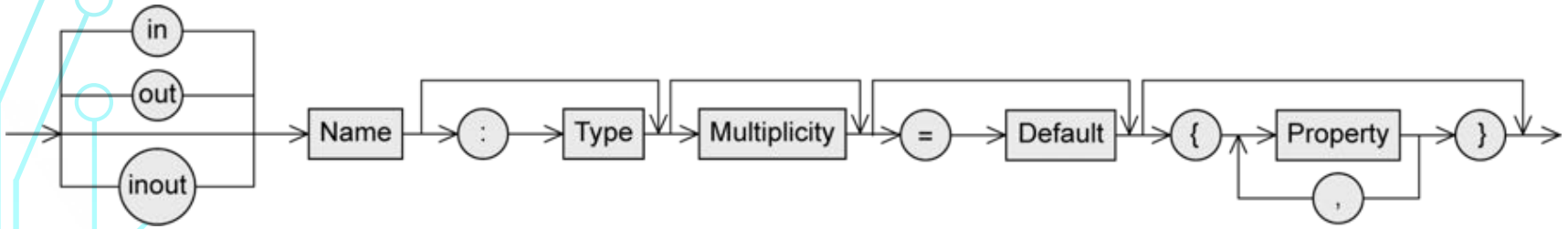


THAM SỐ

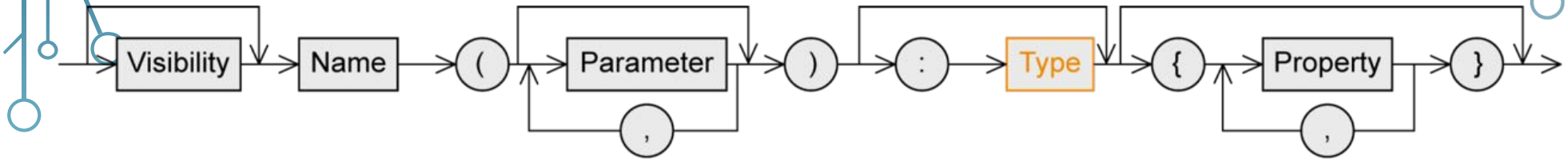


Person
...
+ getName(out fn: String, out ln: String): void + updateLastName(newName: String): boolean + getPersonsNumber(): int

- Khái niệm tương tự thuộc tính
- Hướng của tham số
  - `in` … tham số đầu vào
  - `out` … tham số đầu ra
  - `inout` : Kết hợp vào /ra



KIỂU

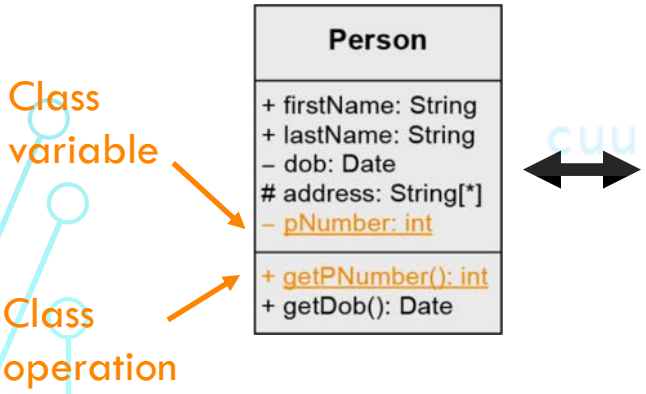


Person
...
getName(out fn: String, out ln: String): void updateLastName(newName: String): boolean getPersonsNumber(): int

- Kiểu của giá trị trả về

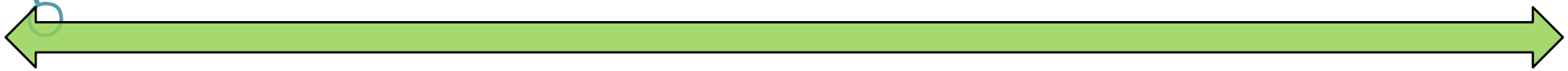
CLASS VARIABLE VÀ CLASS OPERATION

- Instance variable (= instance attribute): các thuộc tính được định nghĩa ở mức instance
- Class variable (= class attribute, static attribute)
  - Được định nghĩa một lần trong lớp, được chia sẻ bởi tất cả các instance của lớp
- Class operation (= static operation)
  - Có thể được sử dụng nếu không có instance của lớp tương ứng được tạo ra
- Ký hiệu: gạch dưới tên của class variable hoặc class operation



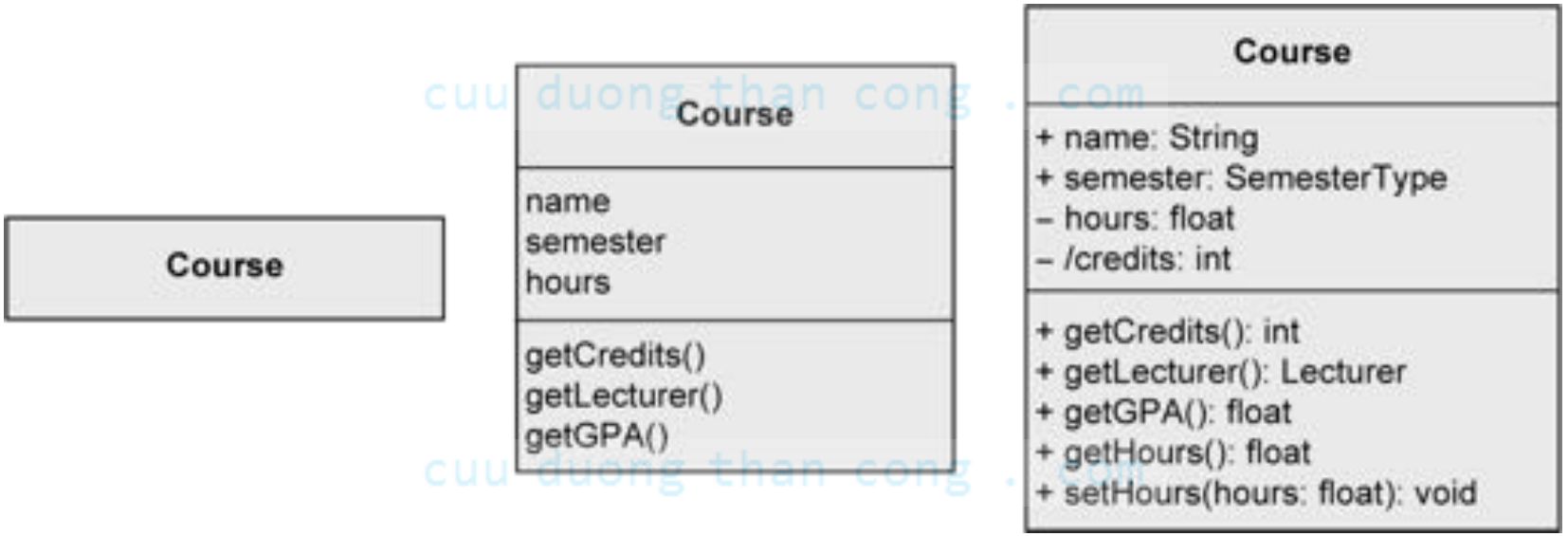

```
class Person {
   public String firstName;
   public String lastName;
   private Date dob;
   protected String[] address;
   private static int pNumber;
   public static int getPNumber() {…}
   public Date getDob() {…}
}
```

ĐẶC TẢ CÁC LỚP: CÁC MỨC CHI TIẾT KHÁC NHAU



coarse-grained

fine-grained

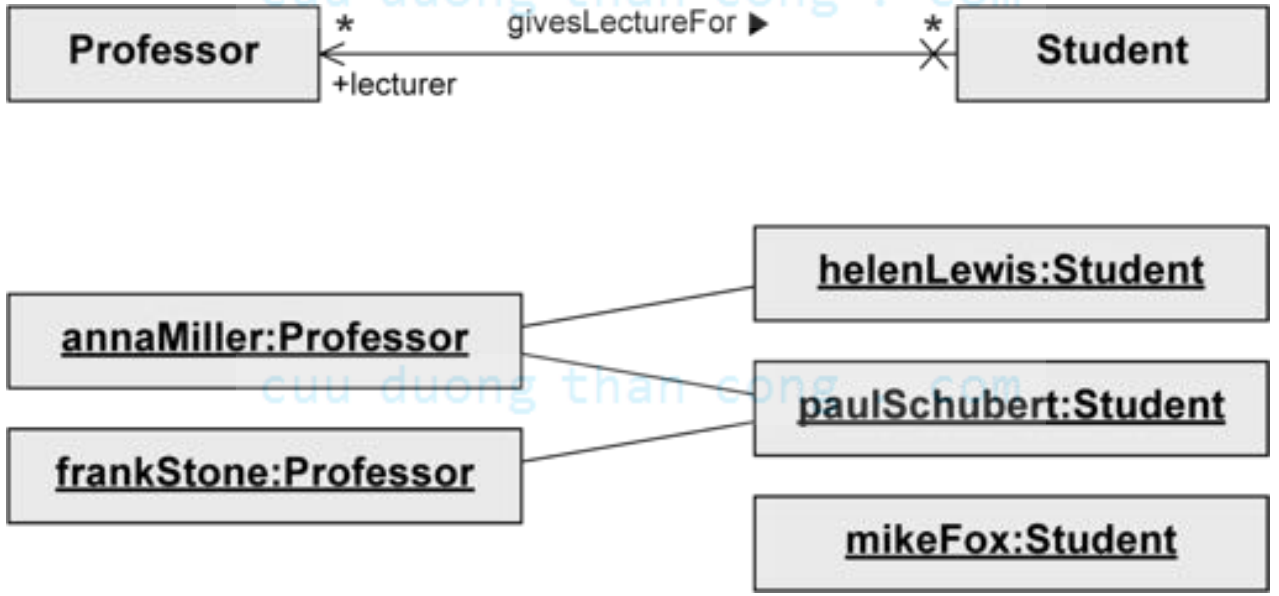


NỘI DUNG

1. Đối tượng
2. Lớp
3. **Các thành phần cơ bản**
    1. Thuộc tính
    2. Thao tác
    3. **Liên kết**
4. Tạo một biểu đồ lớp
5. Phát sinh mã nguồn

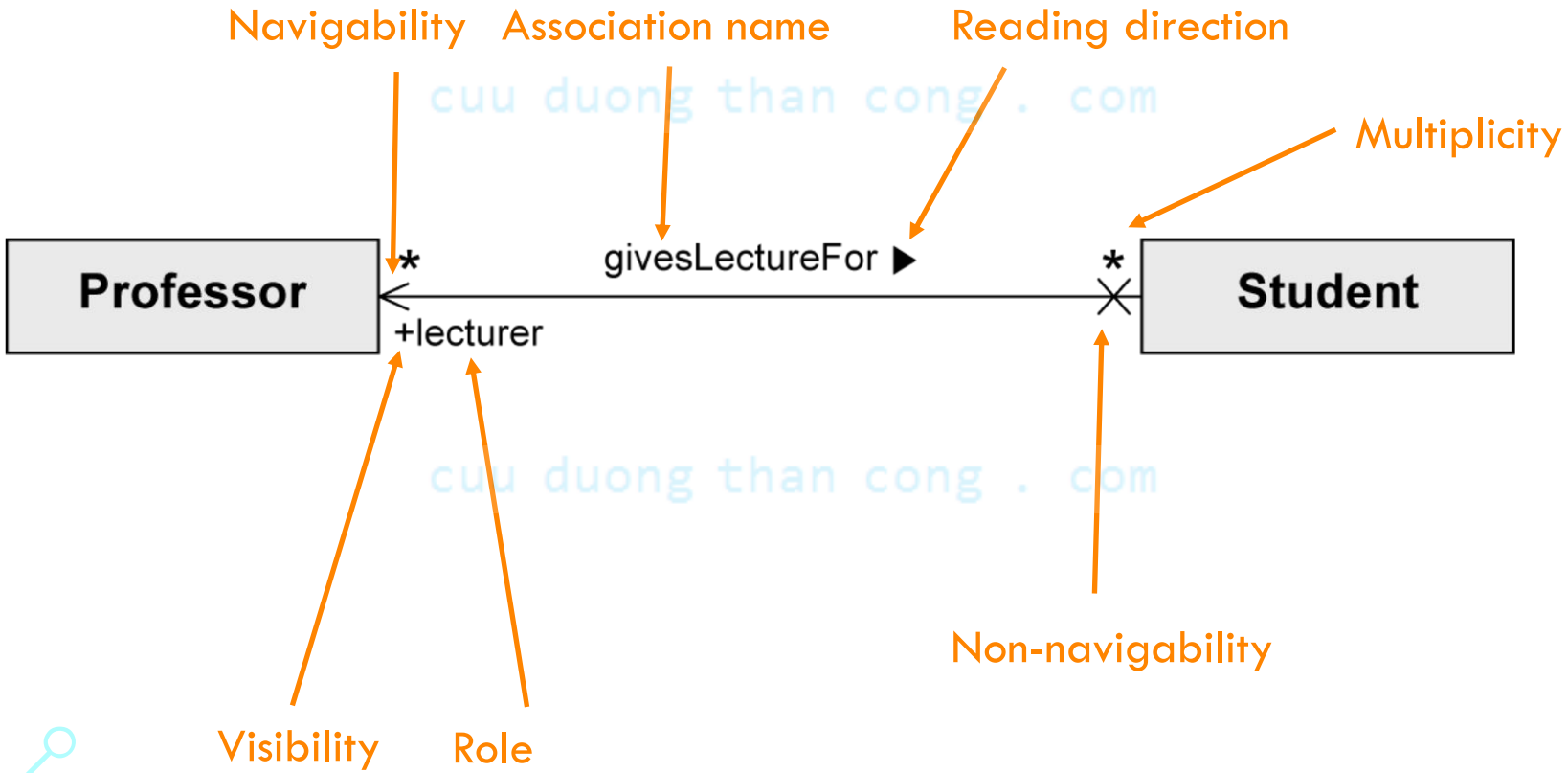
LIÊN KẾT

- Association, link
- Mô hình các mối quan hệ có thể giữa các instance của các lớp



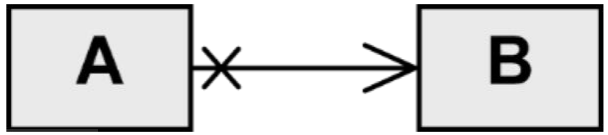
LIÊN KẾT NHỊ PHÂN

- Binary Association
- Liên kết các instance của hai lớp với nhau

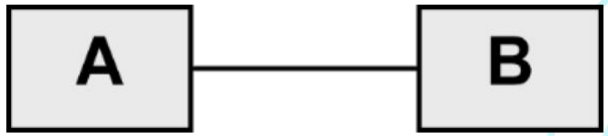


BINARY ASSOCIATION - NAVIGABILITY

- Navigability: một đối tượng biết đối tượng kia và vì vậy có thể truy cập vào các thuộc tính và thao tác thấy được.
  - Ký hiệu bằng đầu mũi tên mở

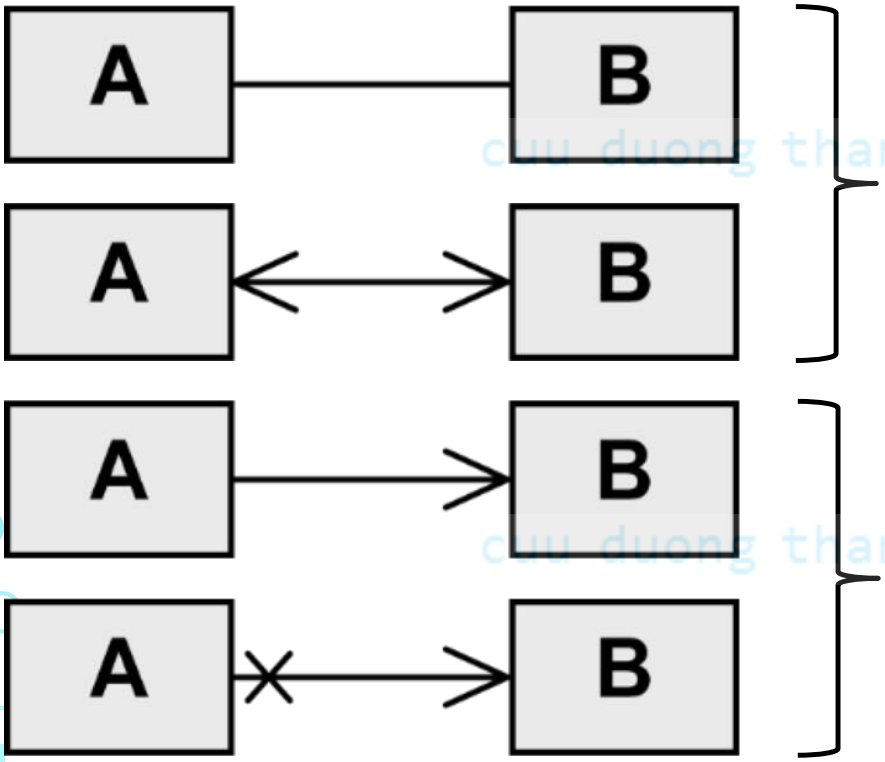


- Non-navigability
  - Ký hiệu bằng dấu x
- Ví dụ:
  - **A** có thể truy cập các thuộc tính và thao tác thấy được của B
  - **B** không thể truy cập vào các thuộc tính và thao tác thấy được của A
- Navigability undefined
  - Giả định điều hướng cả hai hướng

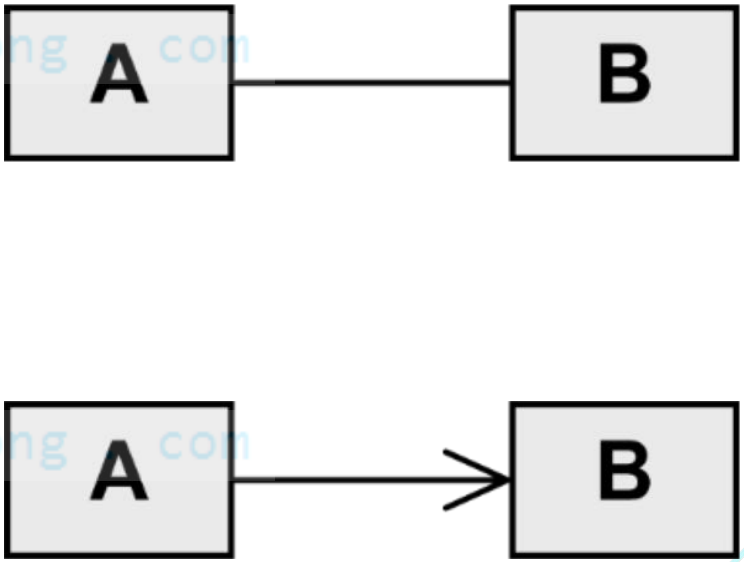


NAVIGABILITY – UML STANDARD VS. BEST PRACTICE

UML standard

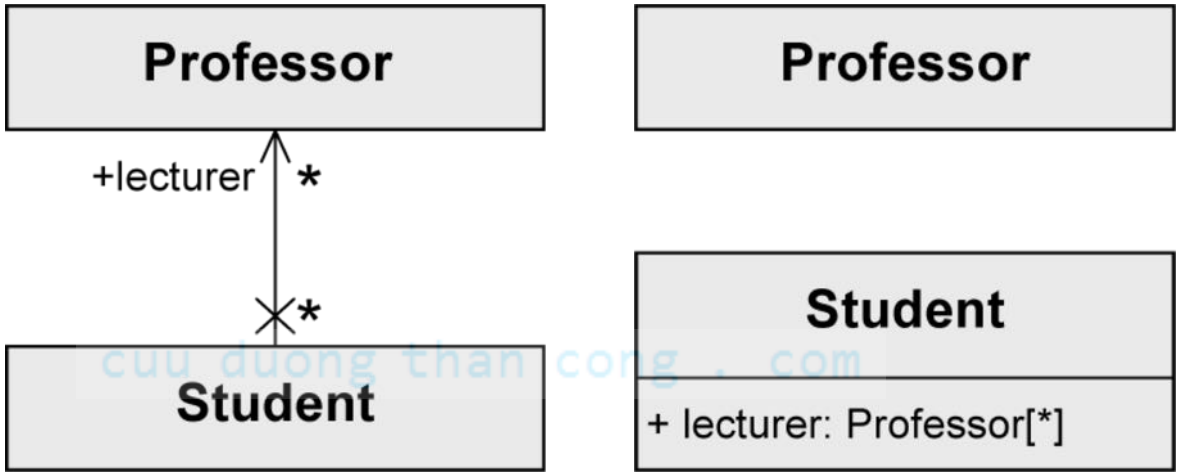


Best practice



LIÊN KẾT NHỊ PHÂN DƯỚI DẠNG THUỘC TÍNH

Preferable



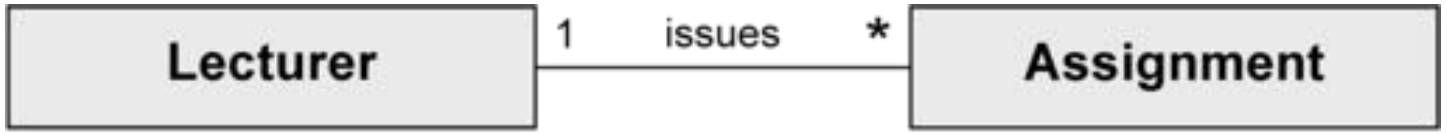
- Mã nguồn Java :

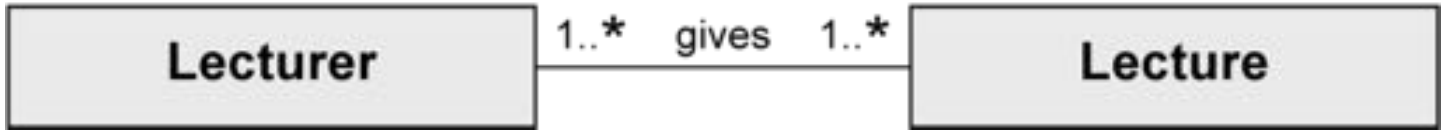
```
class Professor {…}

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

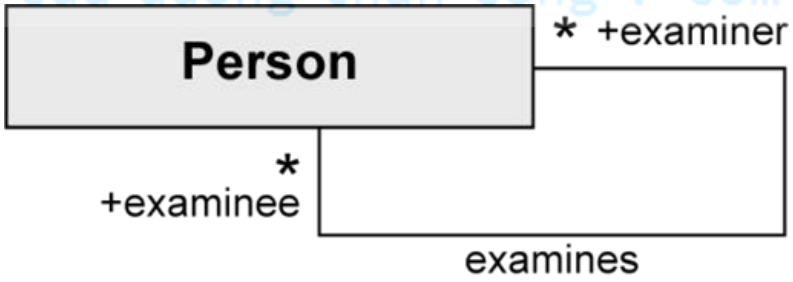
BINARY ASSOCIATION – MULTIPLICITY VÀ ROLE

- Multiplicity: Số lượng đối tượng có thể được liên kết với ít nhất một đối tượng của phía đối diện

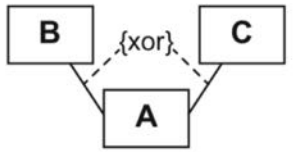




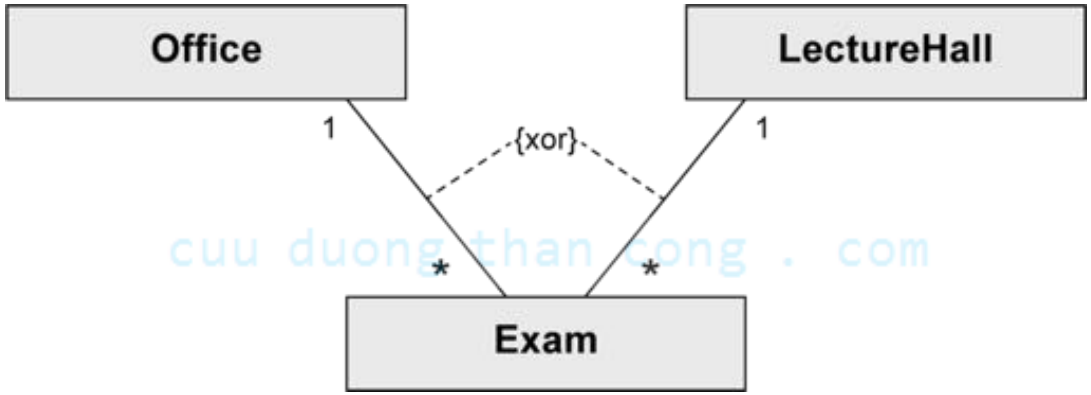
- Role: mô tả vai trò của một đối tượng khi tham gia vào một quan hệ liên kết



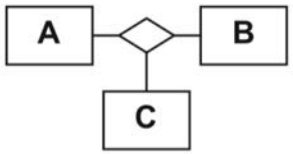
BINARY ASSOCIATION – RÀNG BUỘC `xor`



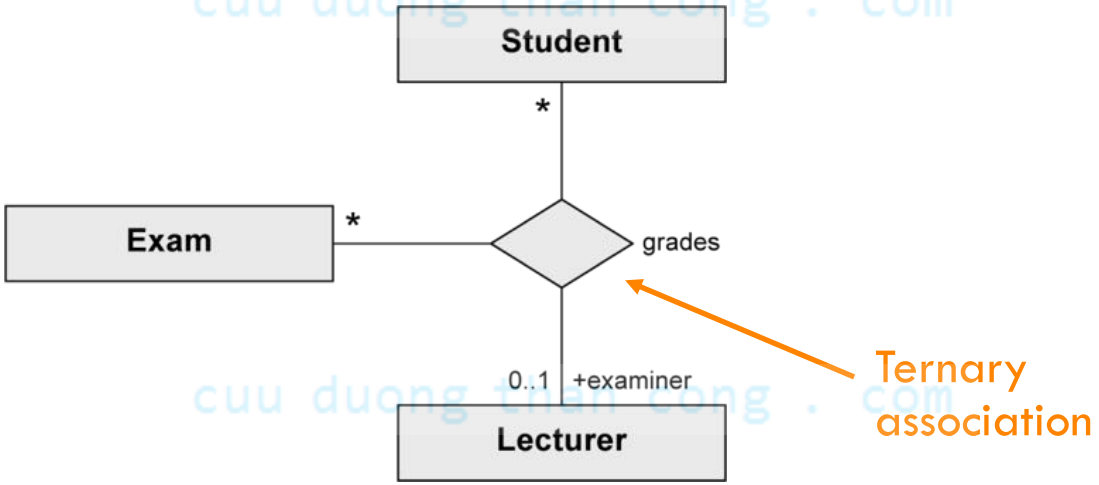
- Ràng buộc “exclusive or”
- Một đối tượng của lớp A được liên kết với một đối tượng của lớp B hoặc một đối tượng của lớp C nhưng không đồng thời.



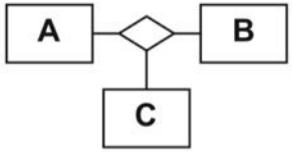
LIÊN KẾT BẬC N [1]



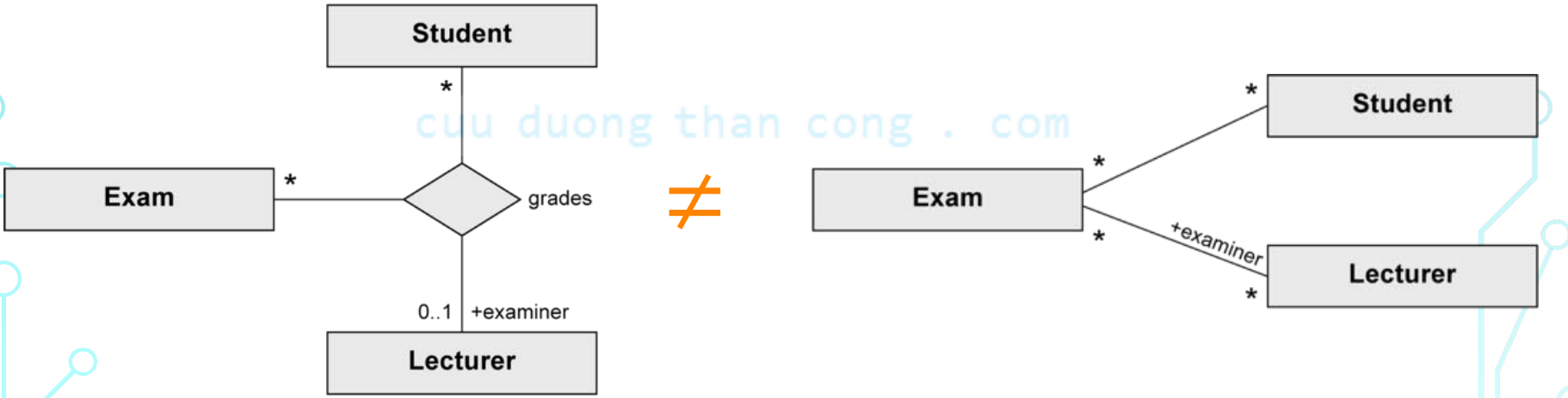
- n-ary Association
- Nhiều hơn hai đối tượng tham gia vào quan hệ.
- Cách cạnh không có mũi tên chỉ hướng.



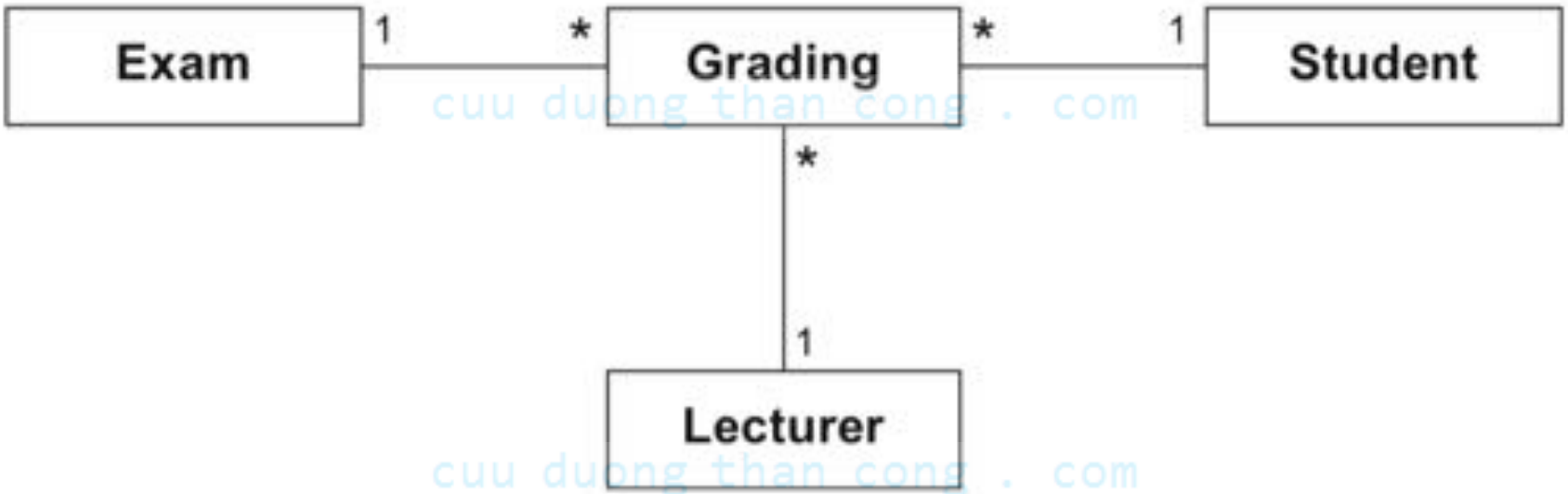
LIÊN KẾT BẬC N [2]



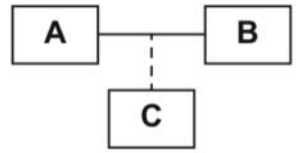
- (**Student**, **Exam**) → (**Lecturer**)
  - One student takes one exam with one or no lecturer
- (**Exam**, **Lecturer**) → (**Student**)
  - One exam with one lecturer can be taken by any number of students
- (**Student**, **Lecturer**) → (**Exam**)
  - One student can be graded by one **Lecturer**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



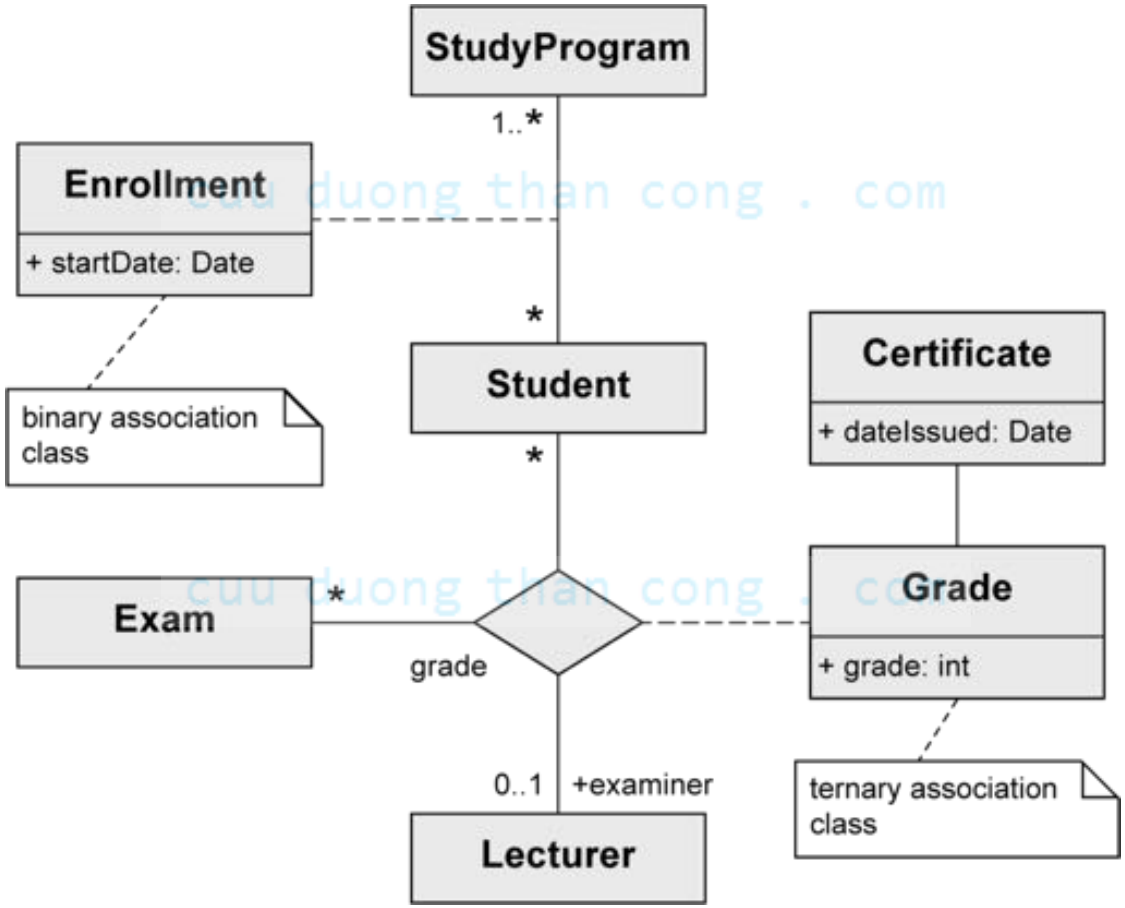
LIÊN KẾT BẬC N [3]



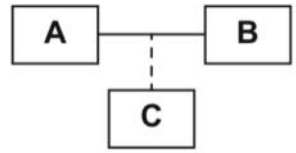
LỚP LIÊN KẾT [1]



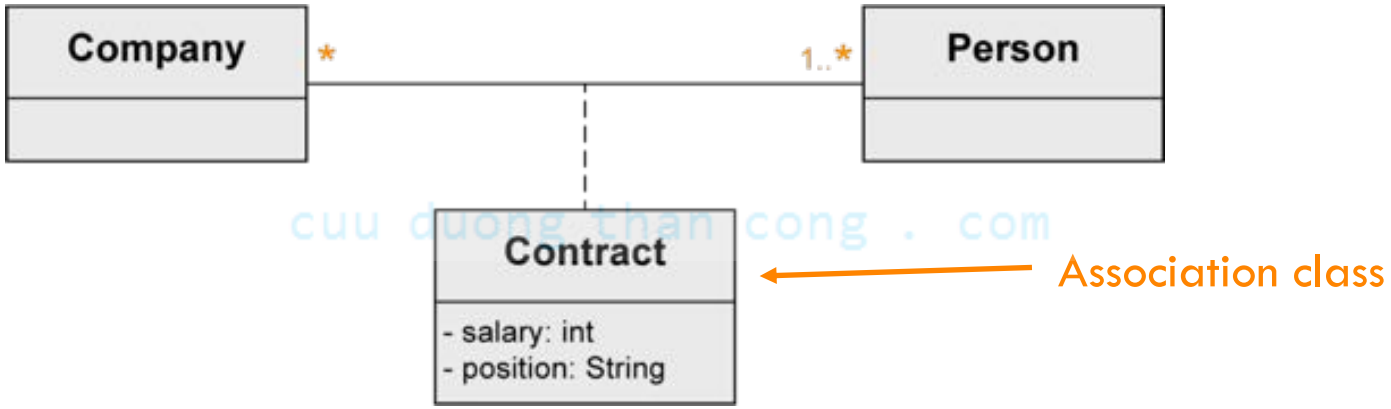
- Association Class
- Gán các thuộc tính vào quan hệ giữa các lớp hơn là vào chính lớp đó



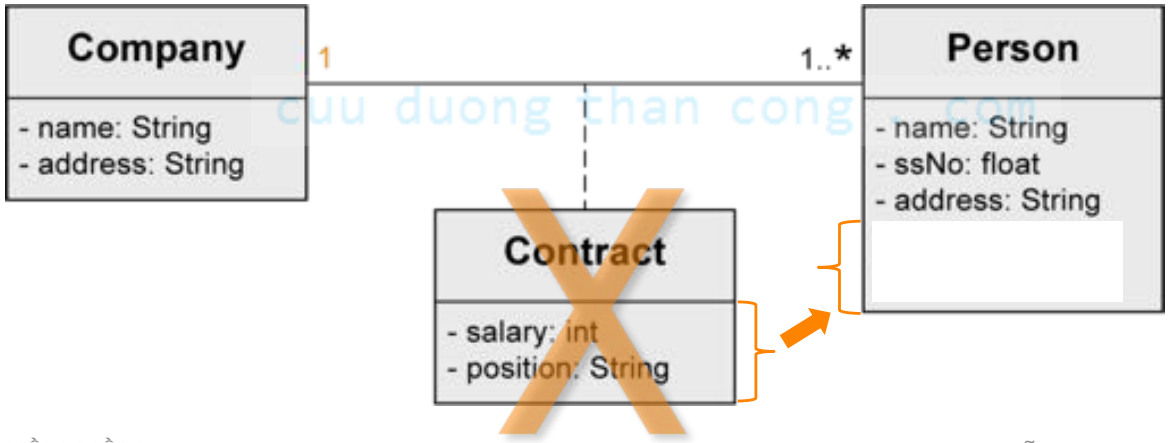
LỚP LIÊN KẾT [2]



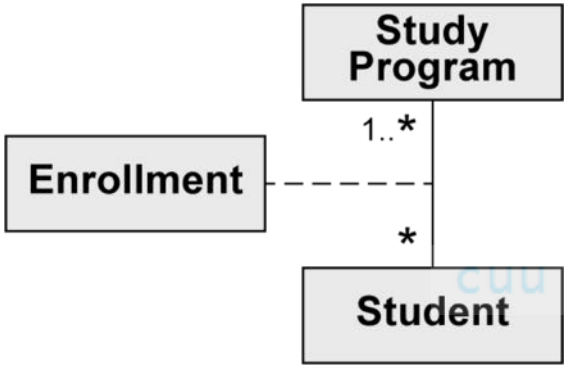
- Cần thiết khi mô hình hoá liên kết n:m



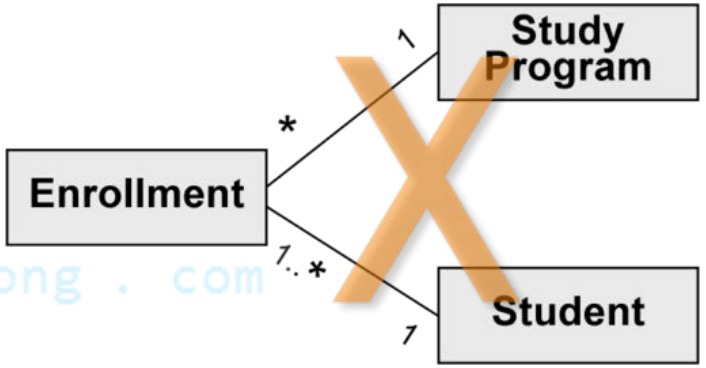
- Với liên kết 1:1 hoặc 1:n có thể dùng nhưng không cần thiết



LỚP LIÊN KẾT VS. LỚP THÔNG THƯỜNG



≠

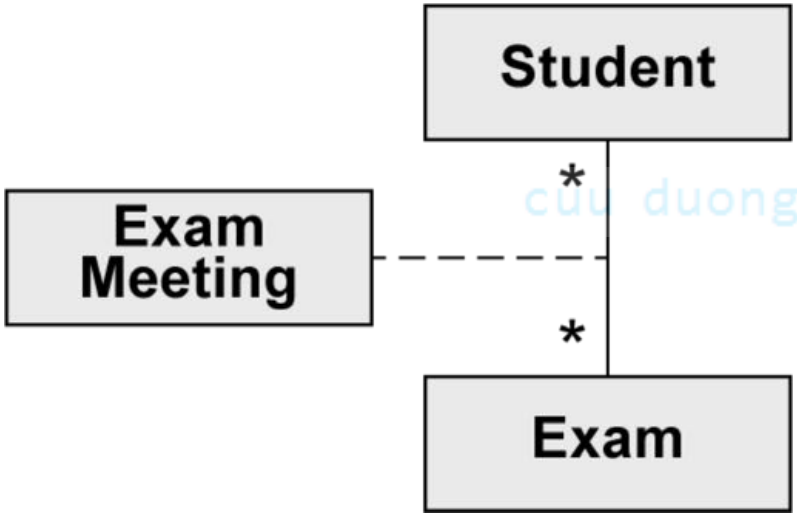


A `Student` can enroll for one particular `StudyProgram` only once

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

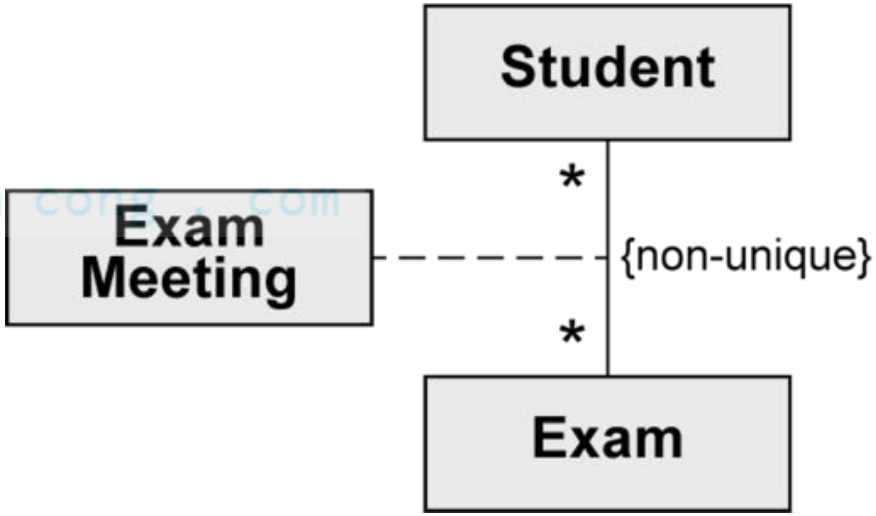
LỚP LIÊN KẾT – unique/non-unique [1]

- Default: no duplicates



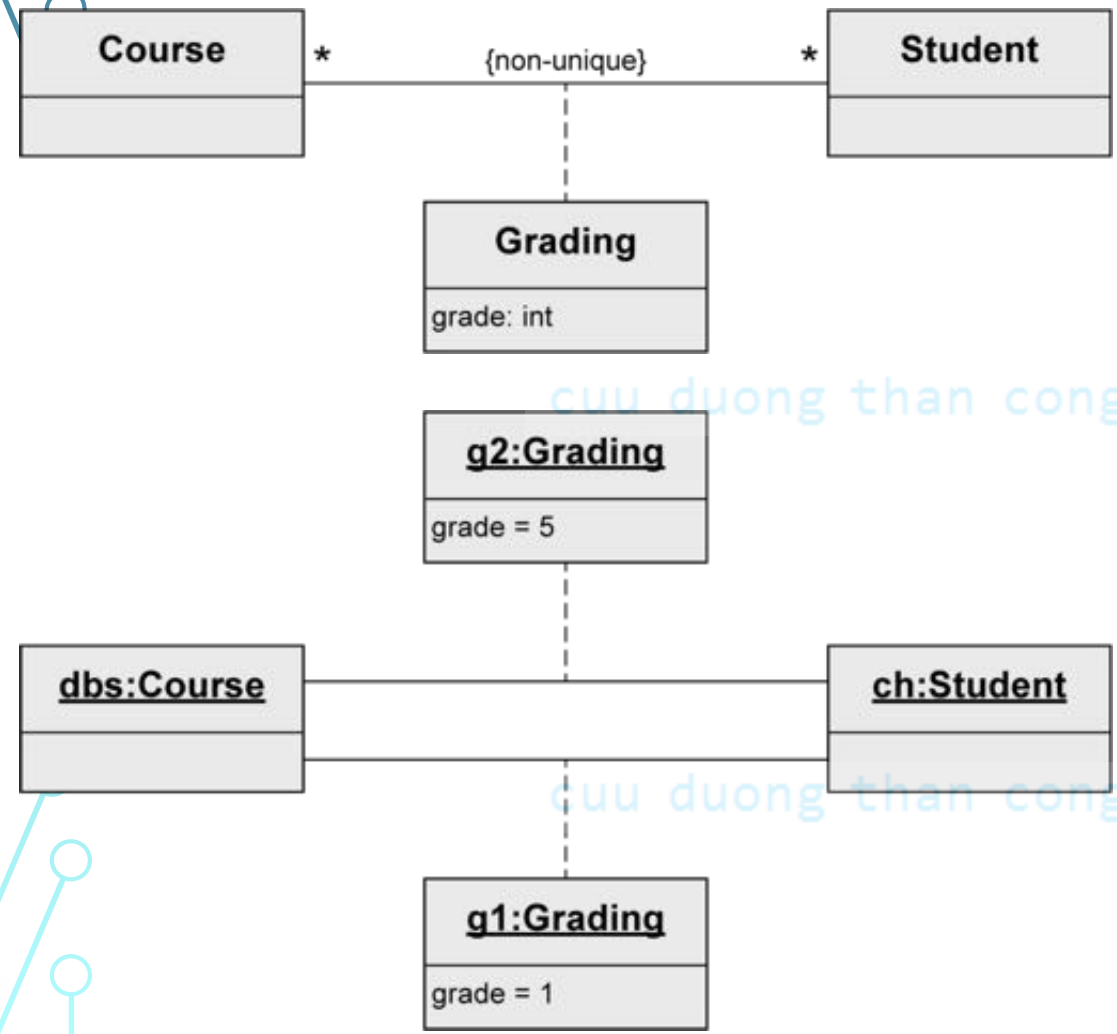
A student can only be granted an exam meeting for a specific exam once.

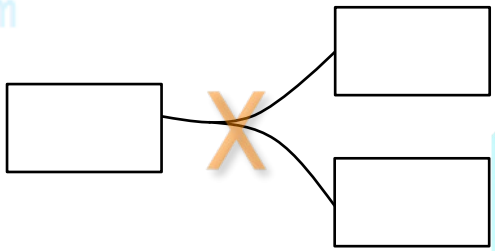
- non-unique: duplicates allowed



A student can have more than one exam meetings for a specific exam.

LỚP LIÊN KẾT – UNIQUE/NON-UNIQUE [2]





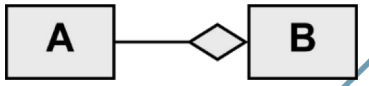
AGGREGATION

AGGREGATION

- Là một dạng đặc biệt của liên kết
- Dùng để biểu diễn một lớp là một phần của lớp khác
- Các thuộc tính của liên kết aggregation:
  - **Transitive**: Nếu **`B`** là một phần của **`A`** và **`C`** là một phần của **`B`**, **`C`** cũng là một phần của **`A`**
  - **Asymmetric**: A không thể là một phần của B đồng thời B là một phần của A.

- Có hai loại :
  - Shared aggregation
  - Composition

SHARED AGGREGATIONS



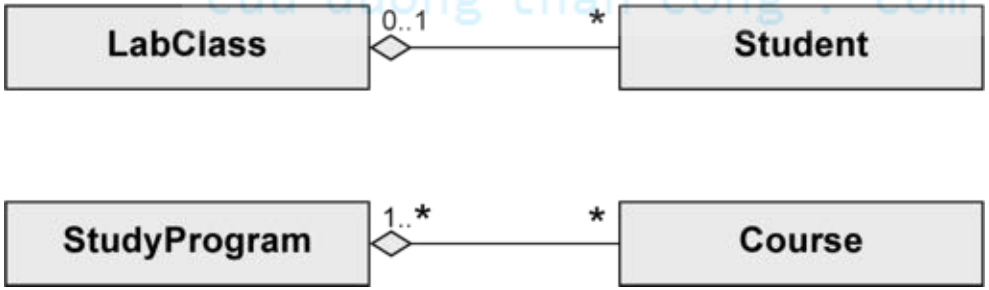
- Biểu diễn một phụ thuộc "yếu" của các phần đối với toàn bộ

  = Các phần cũng có thể tồn tại độc lập với toàn bộ

- Multiplicity tại đầu tổng hợp có thể >1

  = Một phần tử có thể đồng thời là một phần của nhiều phần tử khác nhau

- Cú pháp: Hình thoi tại đầu tổng hợp
- Ví dụ:
  - **`Student`** là một phần của **`LabClass`**
  - **`Course`** là một phần của **`StudyProgram`**



COMPOSITION

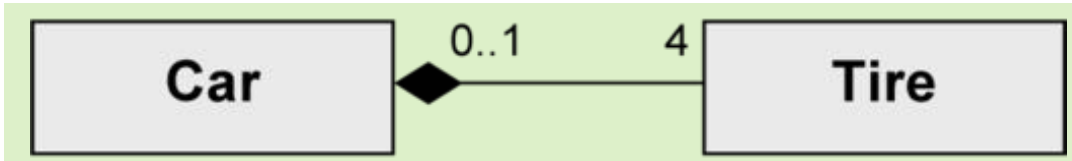
- Phụ thuộc tồn tại giữa composite object và các phần của nó
- Một phần chỉ có thể được chứa trong nhiều nhất một composite object tại một thời điểm cụ thể.
  - Multiplicity tại đầu tổng hợp tối đa là 1 → các composite object hình thành một cây
- Nếu composite object bị xoá, các phần của nó cũng bị xoá.
- Cú pháp: Hình thoi màu đen tại đầu tổng hợp
- Ví dụ: **`Beamer`** là một phần của **`LectureHall`** là một phần của **`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

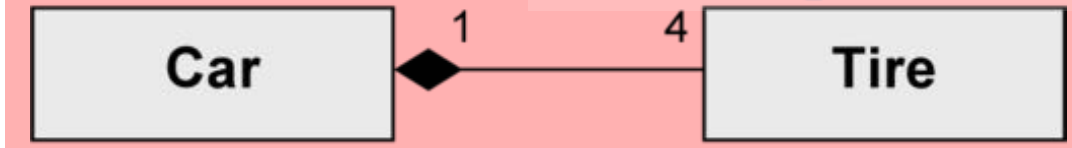
SHARED AGGREGATION VÀ COMPOSITION

- Mô hình nào áp dụng được?

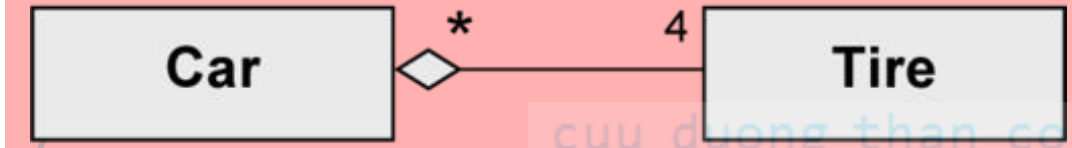


A `Tire` can exist without a `Car`. A `Tire` belongs to one `Car` at most.

Yes

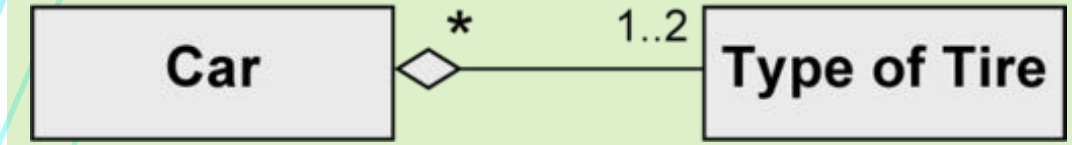


A `Tire` cannot exist without a `Car`.



A `Tire` can belong to multiple `Cars`

No



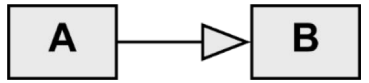
A `Car` has one or two types of `Tires`. Several `Cars` may have the same Type of `Tires`.

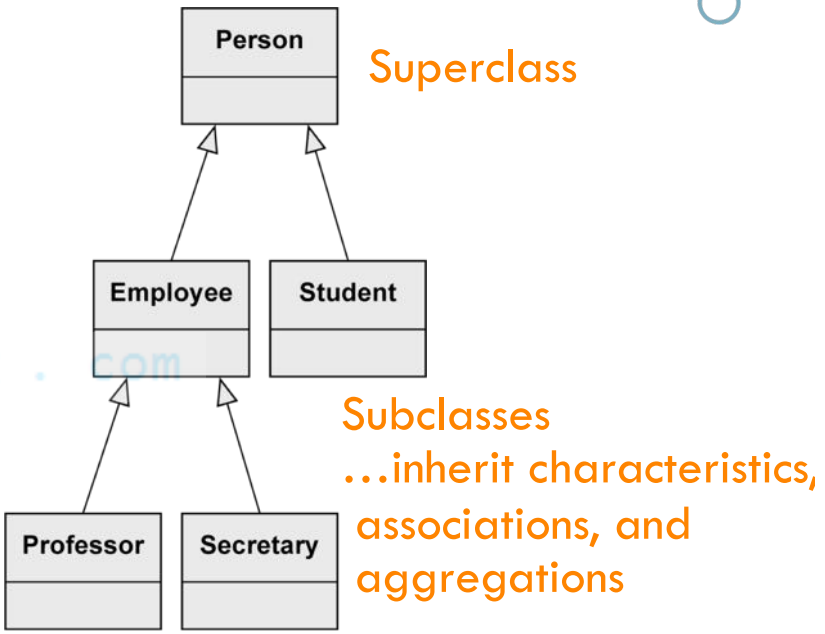
Yes

TỔNG QUÁT HOÁ

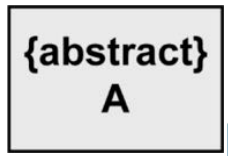
TỔNG QUÁT HOÁ

- Các đặc tính (các thuộc tính và các thao tác), các liên kết, và các aggregation được đặc tả cho một lớp chung (superclass) sẽ được truyền cho lớp con của nó.
- Mỗi instance của lớp con đồng thời là một instance gián tiếp của lớp cha.
- Lớp con kết thừa tất cả đặc tính, liên kết và aggregations của lớp cha ngoại trừ các thành phần private.
- Lớp con có thể có thêm các đặc tính, liên kết và aggregation mới.
- Tổng quát hoá là chuyển tiếp.

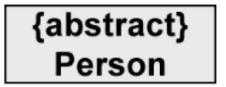


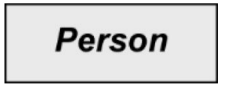


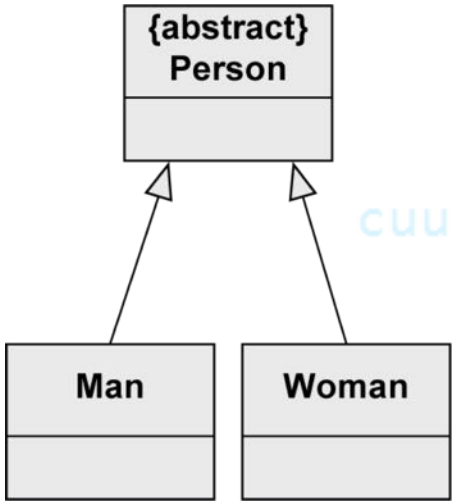
LỚP ABSTRACT



- Được dùng để làm nổi bật các đặc điểm chung của các lớp con của lớp đó.
- Được dùng để đảm bảo rằng không có instance trực tiếp nào của lớp cha.
- Chỉ các lớp con không là abstract mới có thể sinh ra instance.
- Hữu ích trong ngữ cảnh các mối quan hệ tổng quát hoá.
- Ký hiệu: từ khoá **`{abstract}`** hoặc tên lớp ở font in nghiêng.





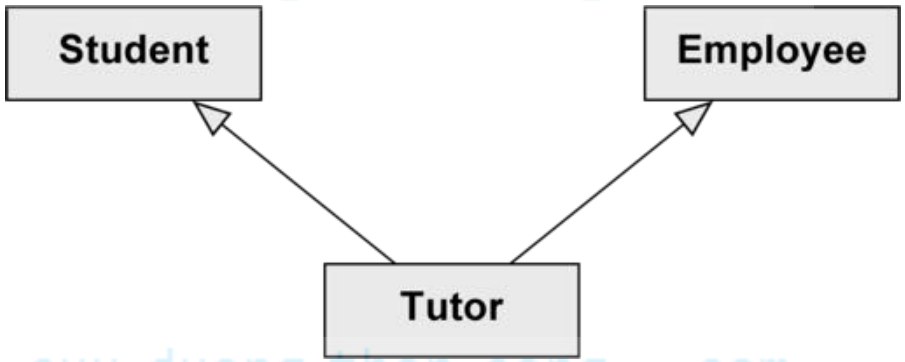


No Person-object possible

Two types of Person: Man and Woman

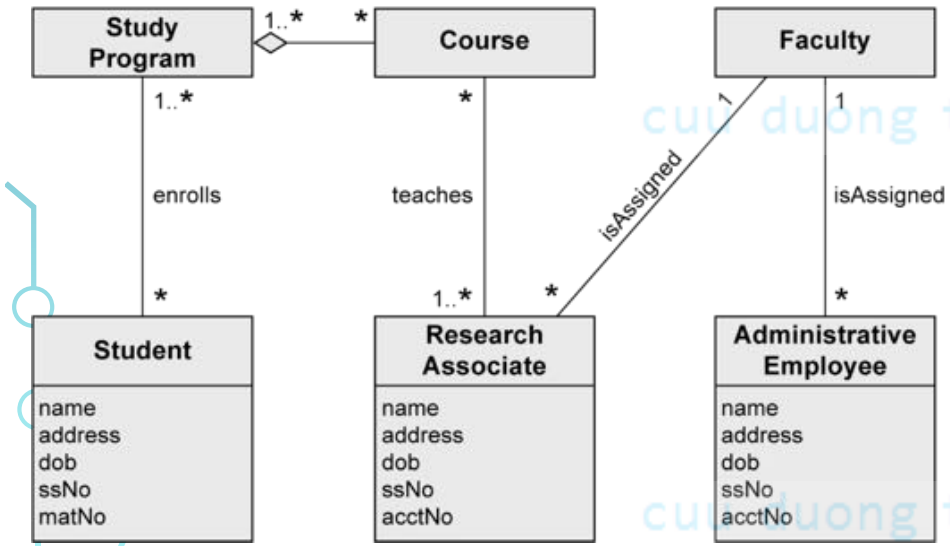
ĐA KẾ THỪA

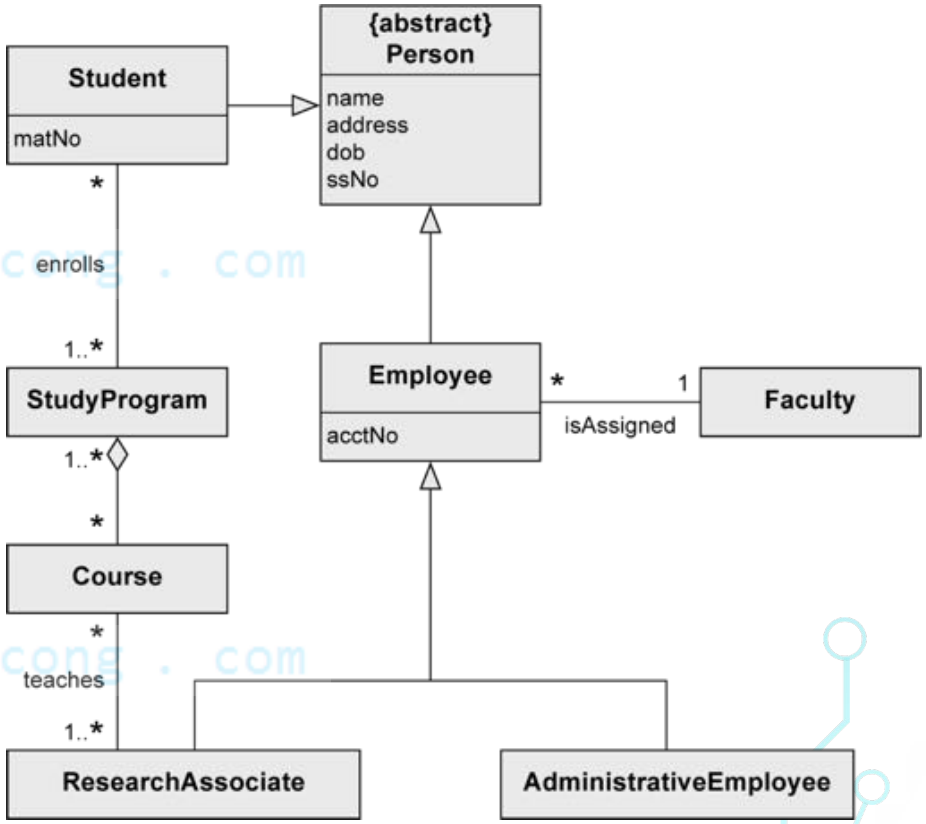
- UML cho phép đa kế thừa.
- Một lớp có thể có nhiều lớp cha.
- Ví dụ:



A Tutor is both an Employee and a Student

CÓ VÀ KHÔNG CÓ TỔNG QUÁT HOÁ



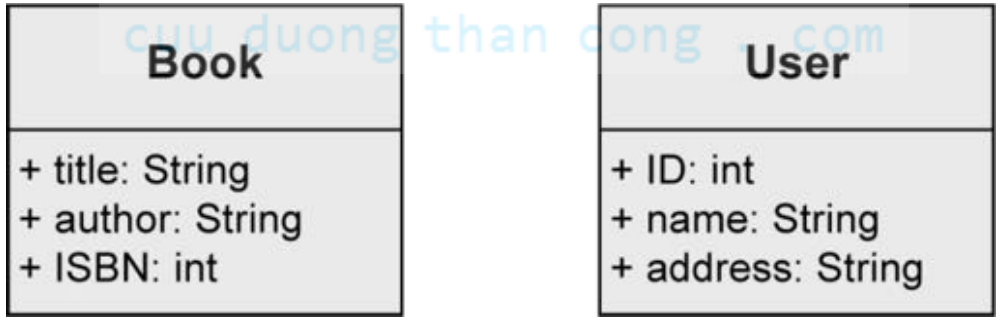


NỘI DUNG

1. Đối tượng
2. Lớp
3. Các thành phần cơ bản
4. **Tạo một biểu đồ lớp**
5. Phát sinh mã nguồn

TẠO MỘT BIỂU ĐỒ LỚP

- Không thể trích xuất các lớp, thuộc tính, liên kết từ văn bản text viết bằng ngôn ngữ tự nhiên một cách tự động.
- Chỉ dẫn
  - Các danh từ thường chỉ các lớp
  - Các tính từ thường chỉ các giá trị thuộc tính
  - Các động từ thường chỉ các hoạt động
- Ví dụ: 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.



CÁC BƯỚC TẠO BIỂU ĐỒ LỚP

1. Nhận diện các lớp
2. Nhận diện các thuộc tính
3. Nhận diện quan hệ giữa các lớp

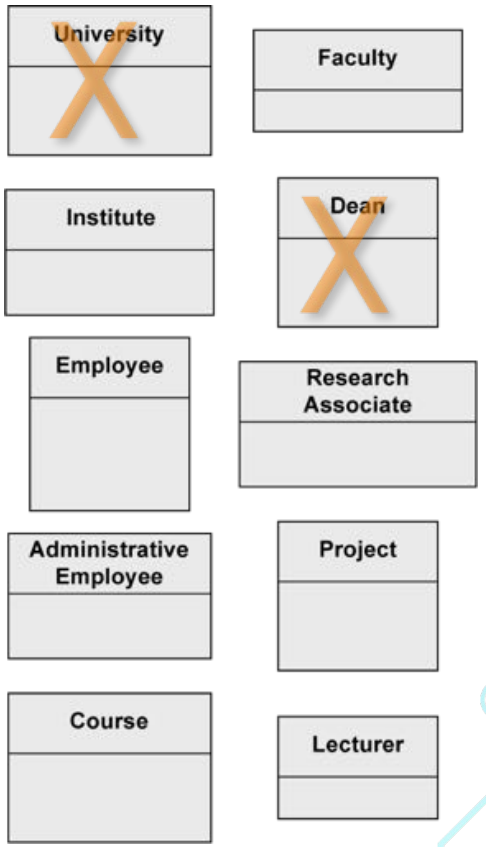
VÍ DỤ: UNIVERSITY INFORMATION SYSTEM

- 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.

BƯỚC 1: NHẬN DIỆN LỚP

- 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.

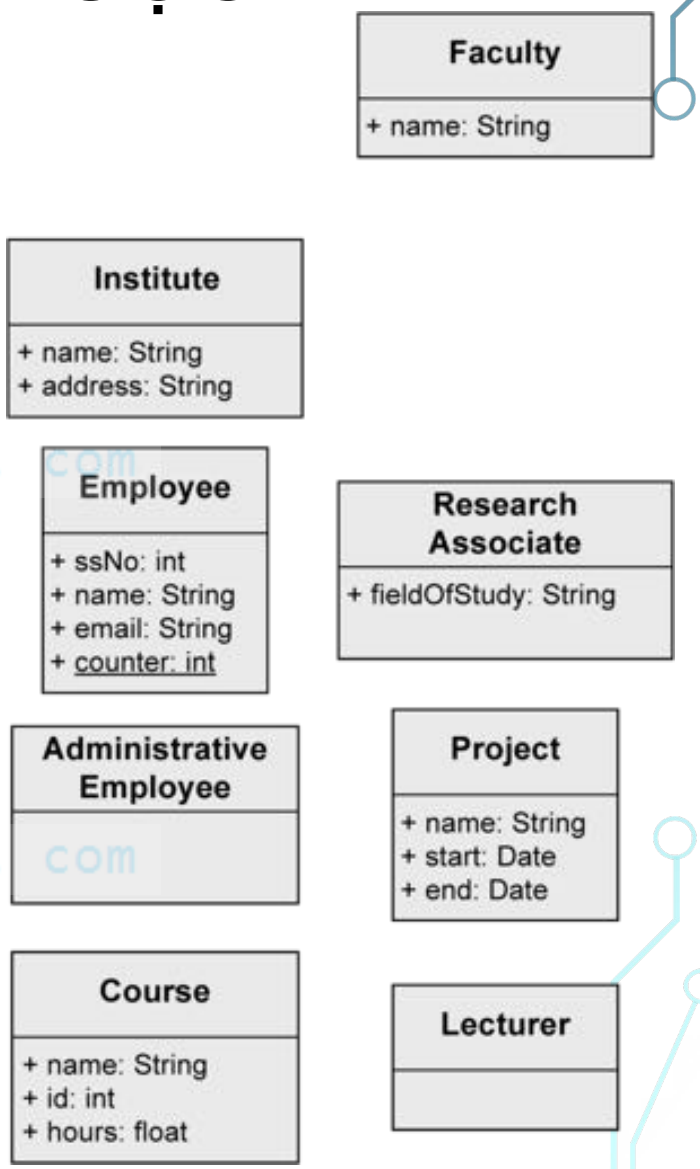
We model the system "University"



Dean has no further attributes than any other employee

BƯỚC 2: NHẬN DIỆN THUỘC TÍNH

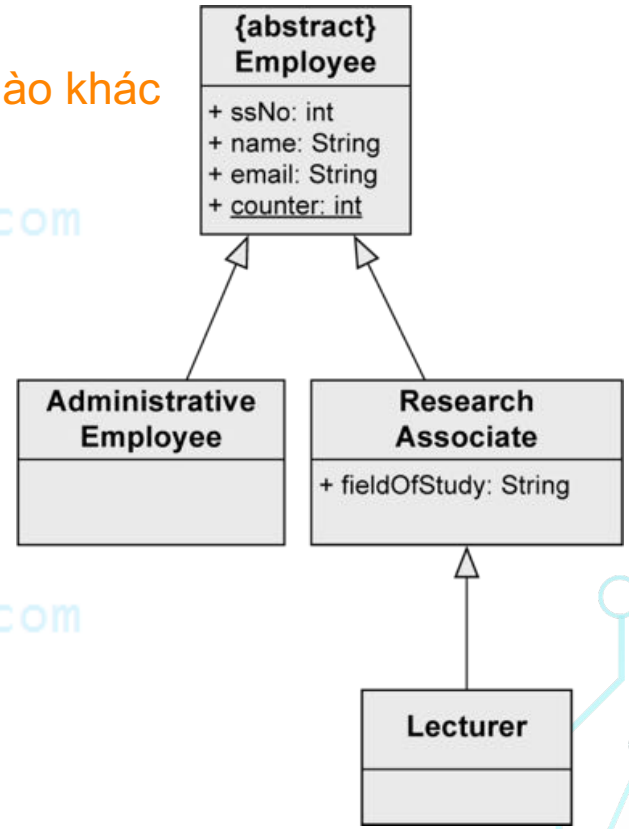
- A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.
- Each faculty is led by a dean, who is an employee of the university.
- The total number of employees is known. Employees have a 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.



BƯỚC 3: NHẬN DIỆN QUAN HỆ [1]

- Có 3 loại quan hệ :
  - Association
  - Generalization
  - Aggregation

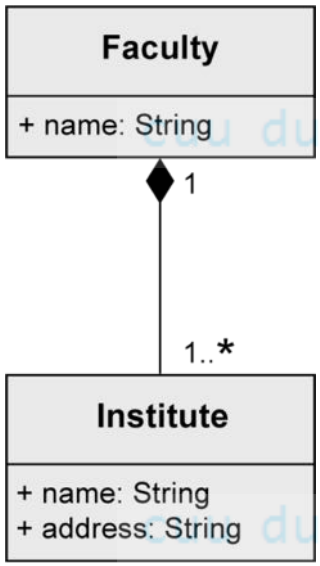
Abstract vì Employee chỉ có
2 loại duy nhất, không có loại nào khác



- Nhận diện quan hệ Generalization
- *"There is a distinction between research and administrative personnel."*
- *"Some research associates hold courses. Then they are called lecturers."*

BƯỚC 3: NHẬN DIỆN QUAN HỆ [2]

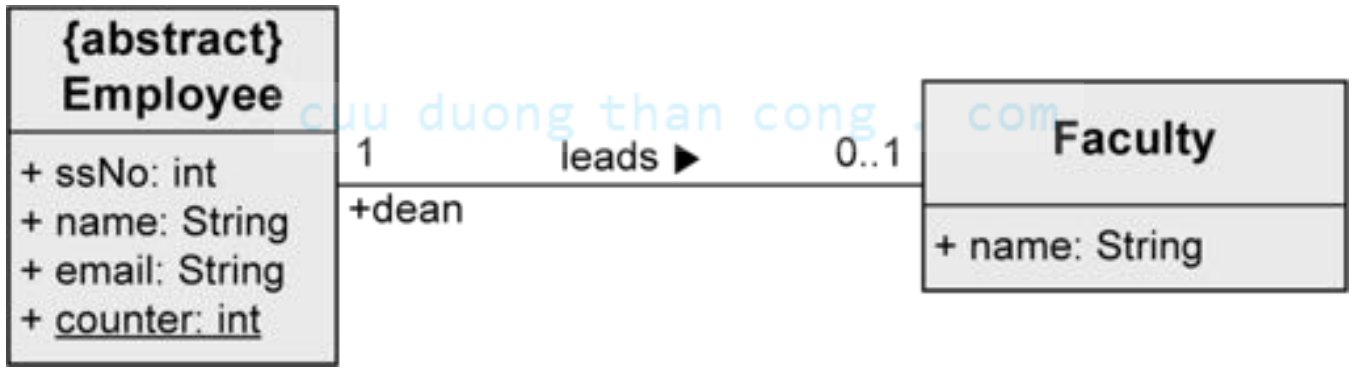
- *"A university consists of multiple faculties which are composed of various institutes."*



Composition to show existence dependency

BƯỚC 3: NHẬN DIỆN QUAN HỆ [3]

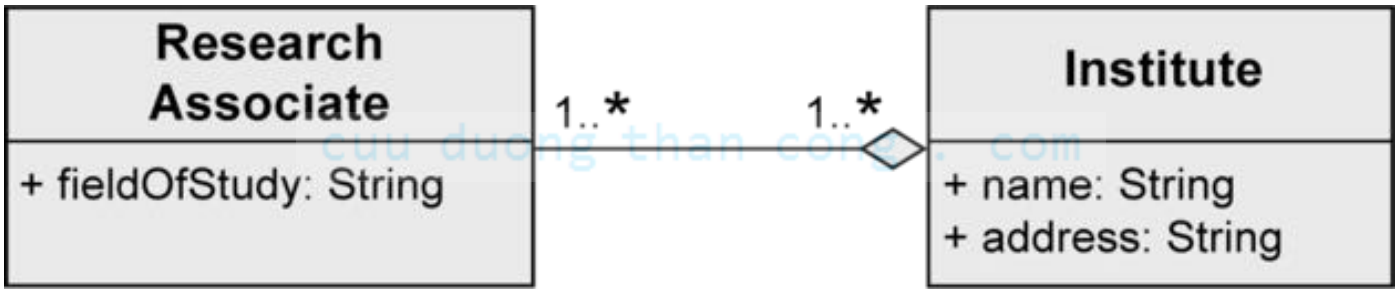
- *"Each faculty is led by a dean, who is an employee of the university"*



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

BƯỚC 3: NHẬN DIỆN QUAN HỆ [4]

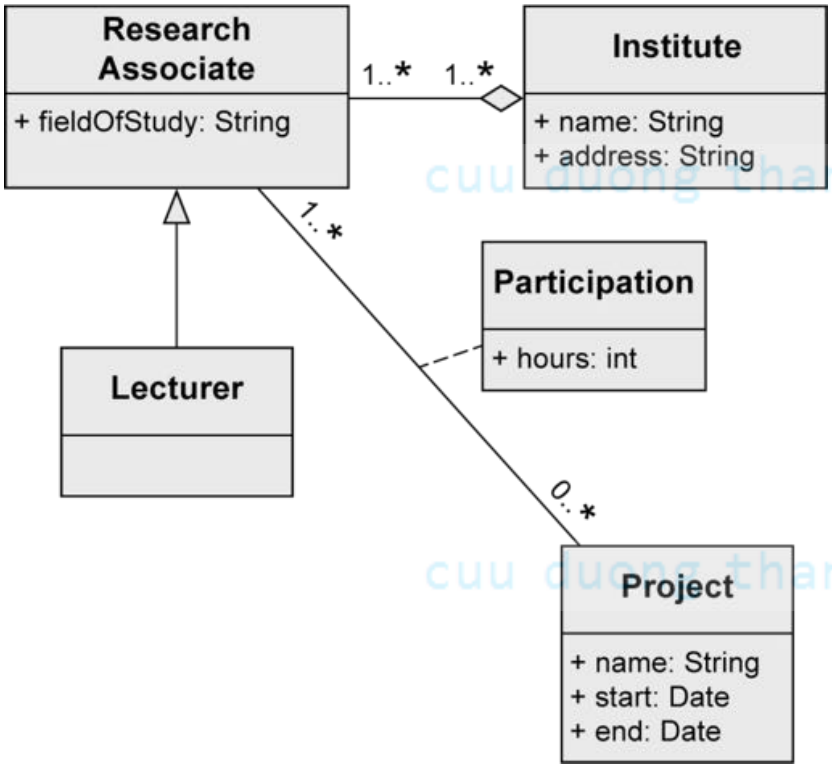
- "Research associates are assigned to at least one institute."



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

BƯỚC 3: NHẬN DIỆN QUAN HỆ [5]

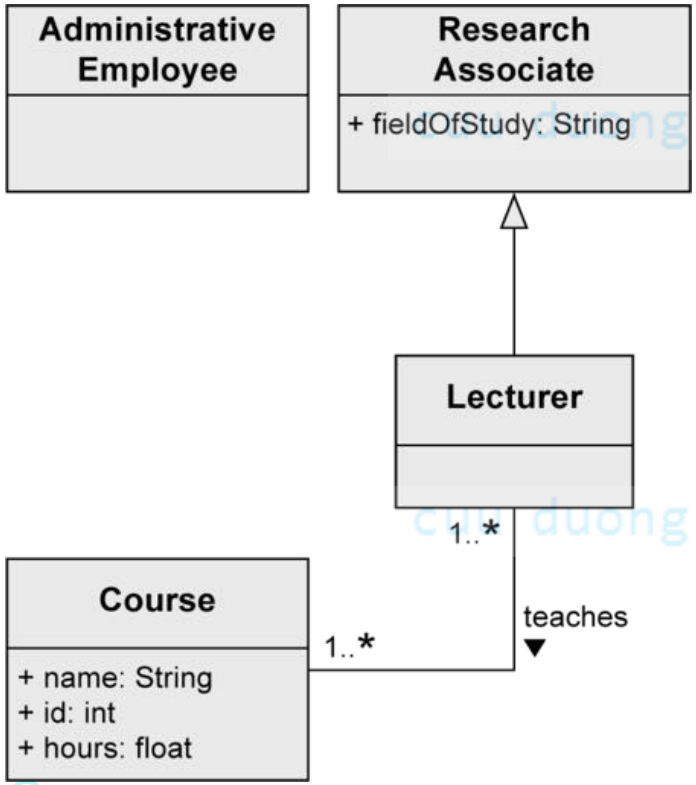
- *"Furthermore, research associates can be involved in projects for a certain number of hours."*



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

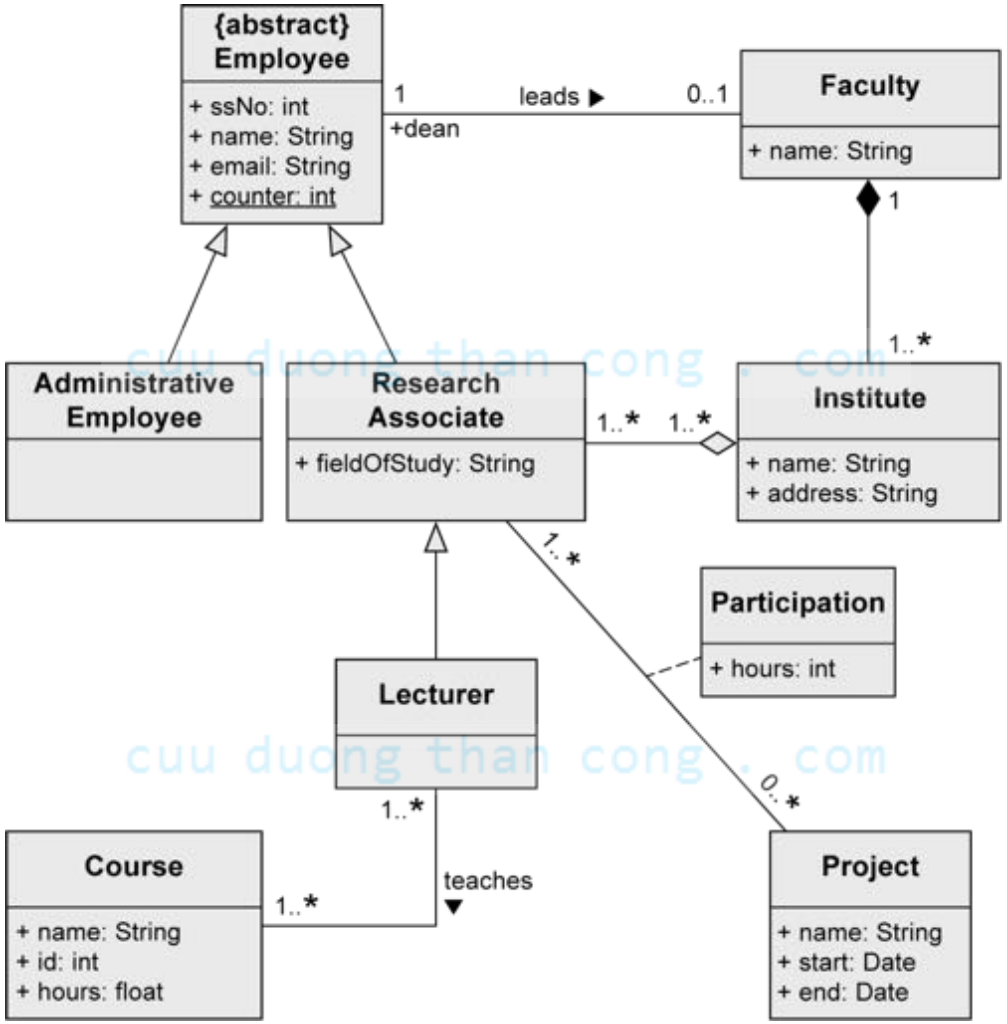
BƯỚC 3: NHẬN DIỆN QUAN HỆ [6]

- *"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`.

BIỂU ĐỒ LỚP HOÀN THIỆN



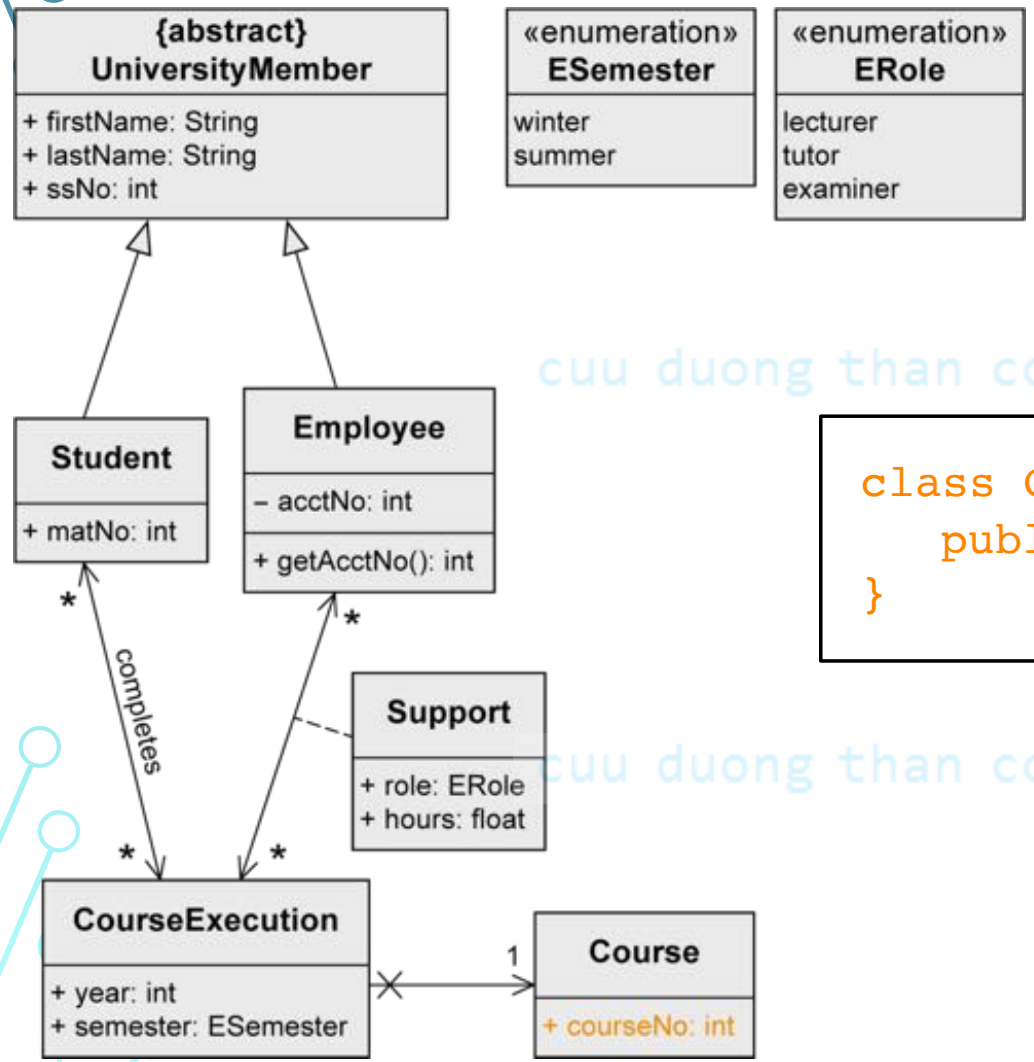
NỘI DUNG

1. Đối tượng
2. Lớp
3. Các thành phần cơ bản
4. Tạo một biểu đồ lớp
5. **Phát sinh mã nguồn**

PHÁT SINH MÃ NGUỒN

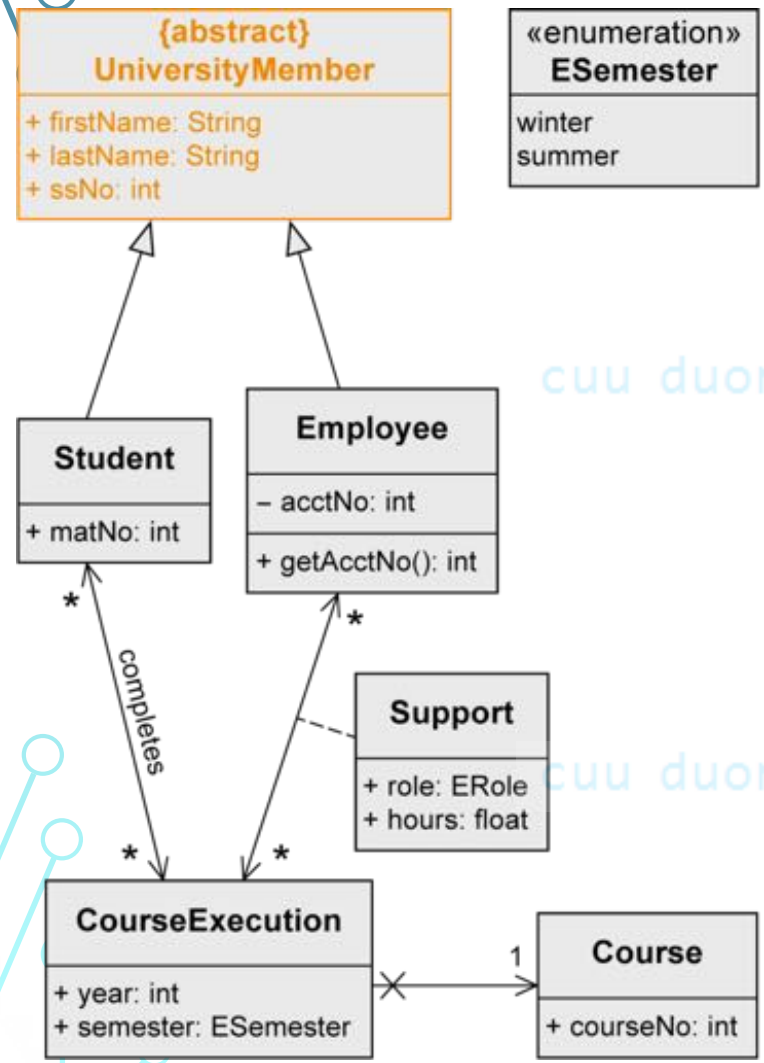
- Các biểu đồ lớp thường được tạo ra với ý định cài đặt các thành phần đã mô hình hoá trong một ngôn ngữ lập trình hướng đối tượng.
- Thông thường, việc dịch là bán tự động và chỉ yêu cầu can thiệp thủ công ở mức tối thiểu.

VÍ DỤ [1]



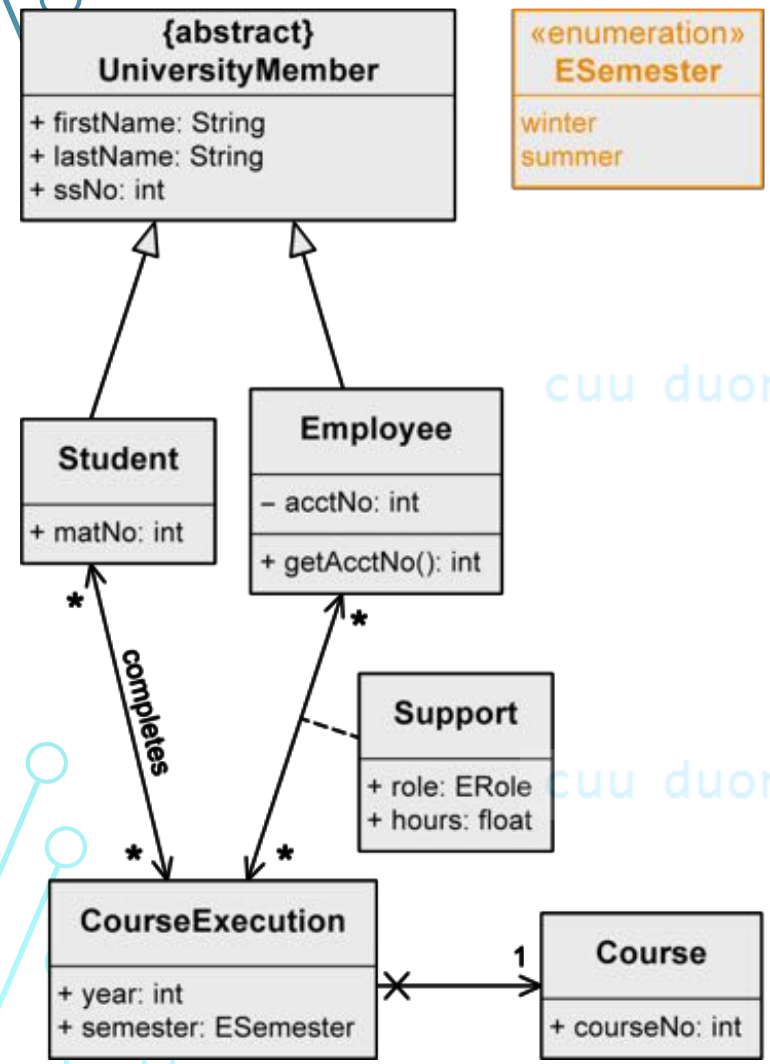

```
class Course {
   public int courseNo;
}
```

VÍ DỤ [2]




```
abstract class UniversityMember {
    public String firstName;
    public String lastName;
    public int ssNo;
}
```

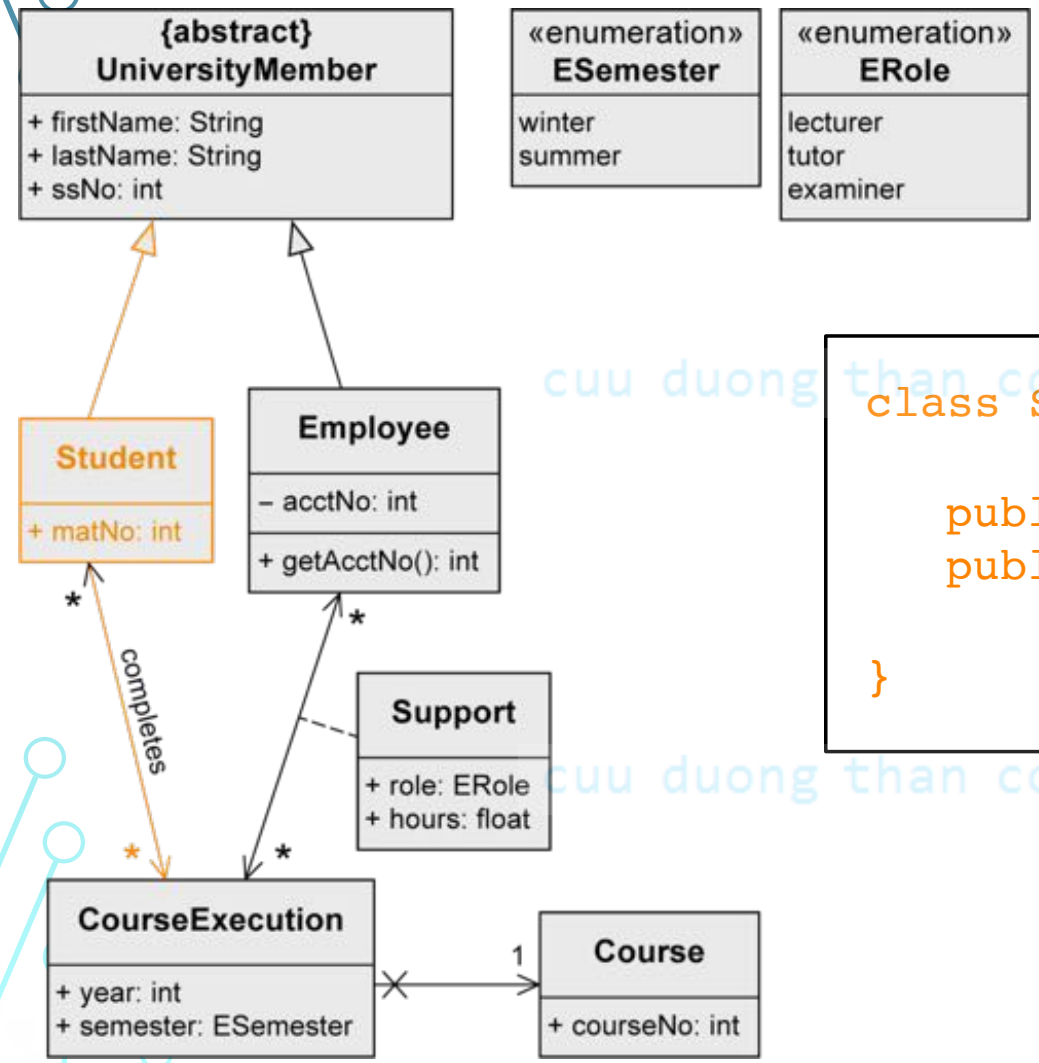
VÍ DỤ [3]




```
Enumeration ESemester {
    winter,
    summer
}
```

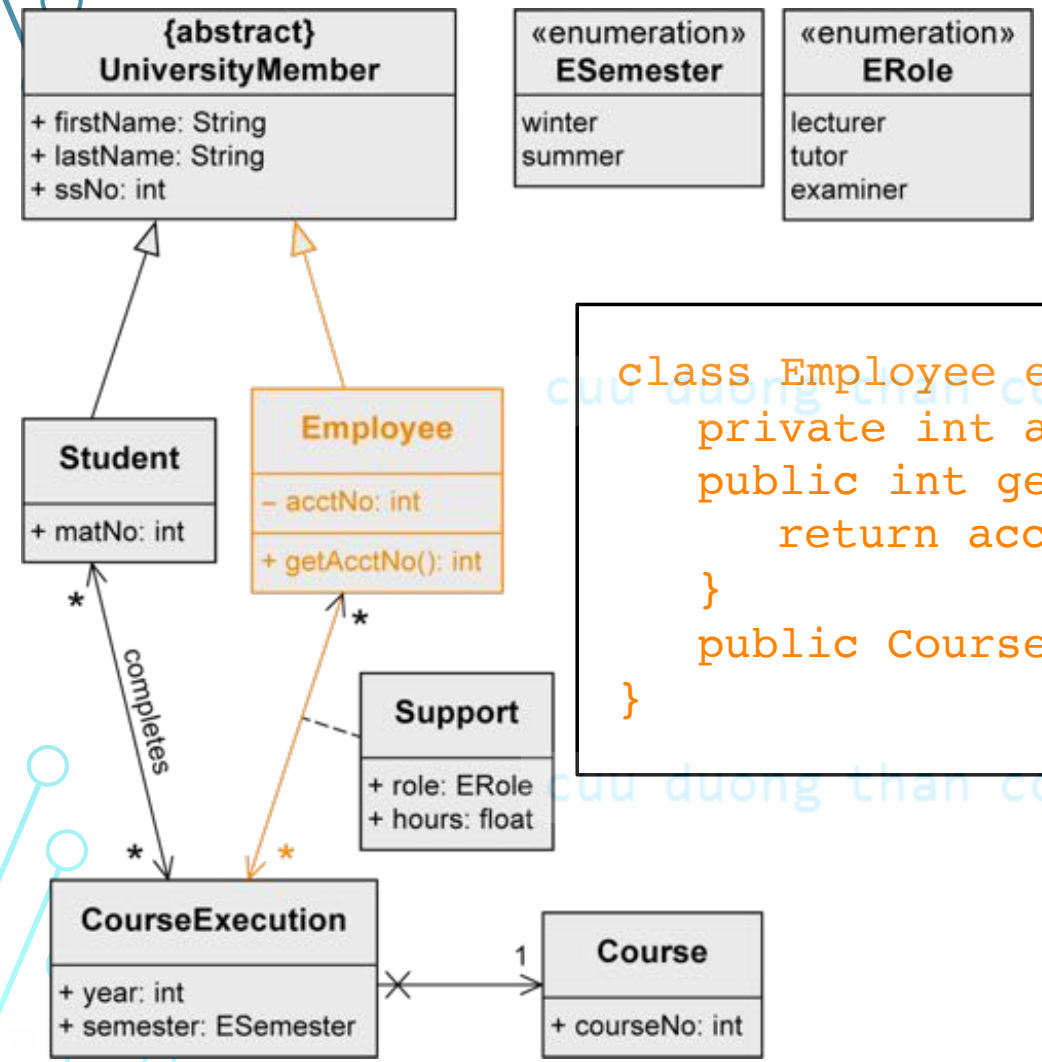
```
Enumeration ERole {
    lecturer,
    tutor,
    examiner
}
```

VÍ DỤ [4]



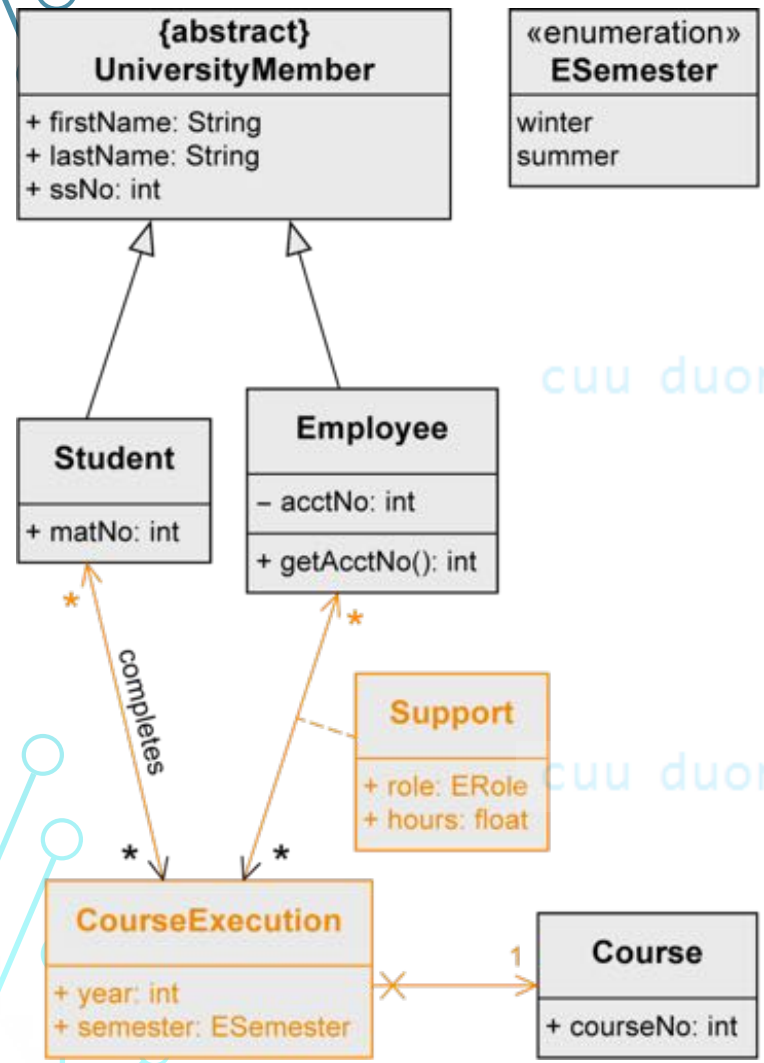

```
class Student extends
                    UniversityMember {
   public int matNo;
   public CourseExecution [ ]
                    completedCourses;
}
```

VÍ DỤ [5]



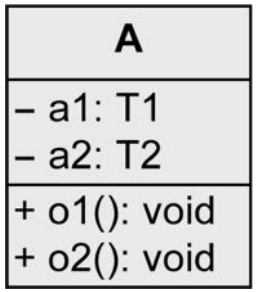
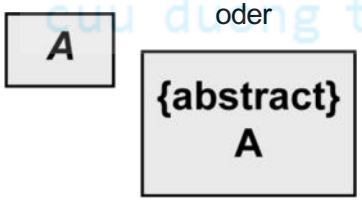
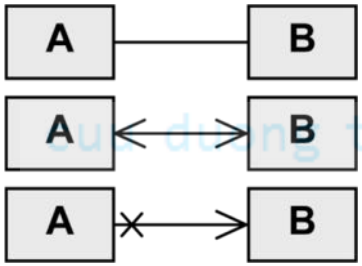

```
class Employee extends UniversityMember {
    private int acctNo;
    public int getAcctNo () {
        return acctNo;
    }
    public CourseExecution [] courseExecutions;
}
```

VÍ DỤ [6]

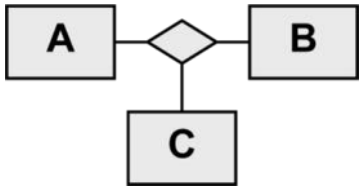
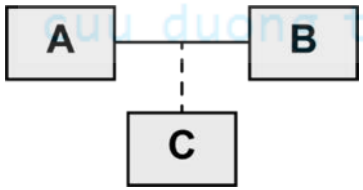
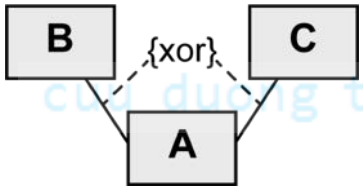



```
class CourseExecution {
    public int year;
    public ESemester semester;
    public Student [] student;
    public Course course;
    public Hashtable support;
        // Key: employee
        // Value: (role, hours)
}
```

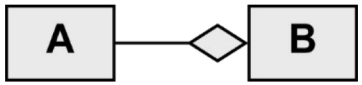
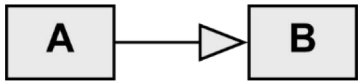
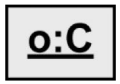
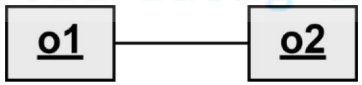
CÁC THÀNH PHẦN KÝ HIỆU [1]

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

CÁC THÀNH PHẦN KÝ HIỆU [2]

Name	Notation	Description
n-ary association		Relationship between n (here 3) classes
Association class		More detailed description of an association
xor relationship		An object of **C** is in a relationship with an object of **A** or with an object of **B** but not with both

CÁC THÀNH PHẦN KÝ HIỆU [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

Câu hỏi?