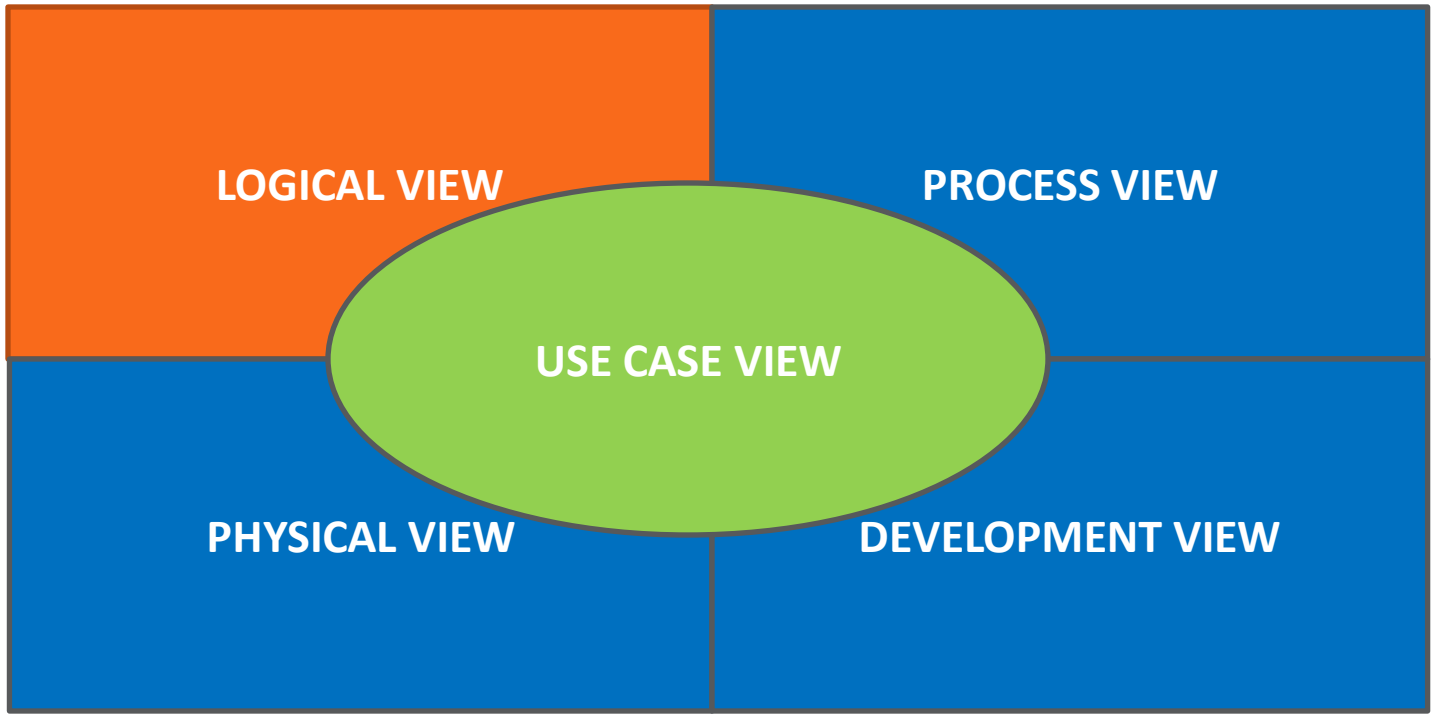
MÔ HÌNH HÓA CẤU TRÚC (STRUCTURE MODELING)

4 View + 1


LOGICAL VIEW
PROCESS VIEW
USE CASE VIEW
PHYSICAL VIEW
DEVELOPMENT VIEW

Analysis Class Diagram

- Class Diagram là mô hình chính để phân tích yêu cầu
  - Mô hình cấu trúc tĩnh của hệ thống:
    - bao gồm các đối tượng và quan hệ giữa chúng
  - Việc phân tích và xây dựng một lược đồ lớp đầy đủ cần kết hợp và bổ sung của các lược đồ khác.

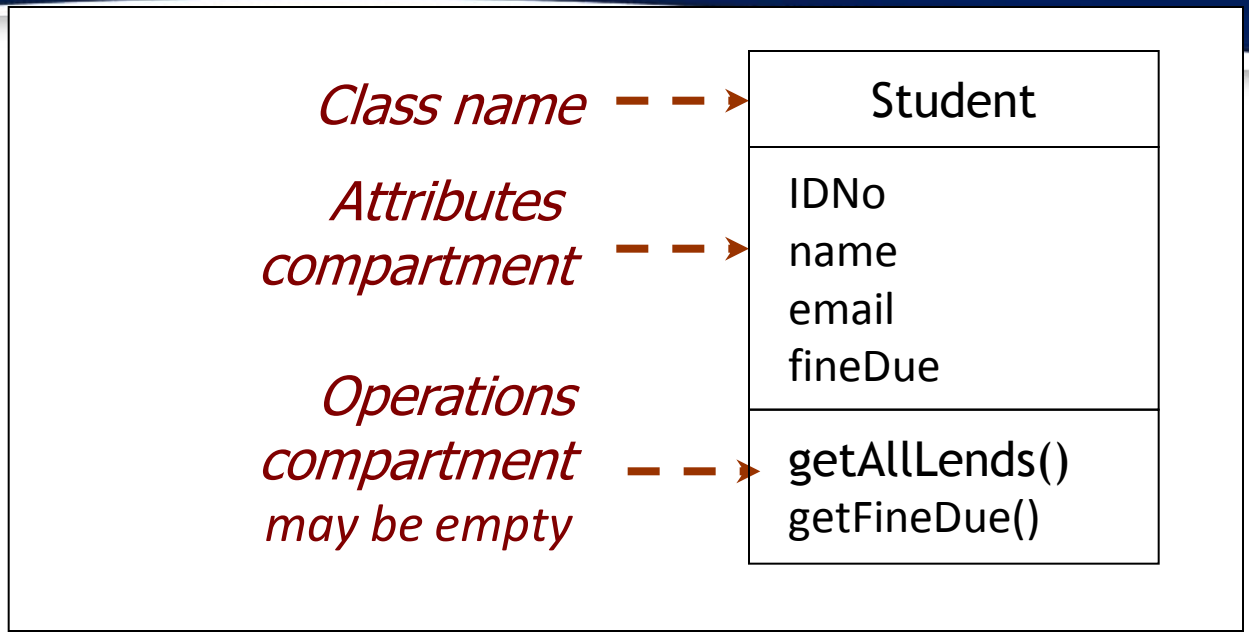
- Các mô hình hỗ trợ khác – mô hình hóa hành vi:
  - **Mô hình tương tác:**
    - Lược đồ cộng tác **(Collaboration Diagram)**
    - Llược đồ tuần tự **(Sequence Diagram)**
  - **Mô hình điều khiển:**
    - Lược đồ chuyển trạng thái **(Statechart Diagram)**
    - Lược đồ hoạt động **(Activity Diagram)**

From Use case to Class Diagram

1. Start with the use case details
   - Numbered list or
   - Activity diagram
2. Identify classes and associations
   - Conceptual class diagram
3. Convert use case details to a sequence diagram
   - Objects and methods
4. Convert methods on sequence diagrams to methods on class diagrams
   - Interface class diagram
5. Convert associations to directed dependencies

KHÁI NIỆM CLASS

Ký hiệu lớp
Class Symbol



Object là một thể hiện (Instance) của lớp

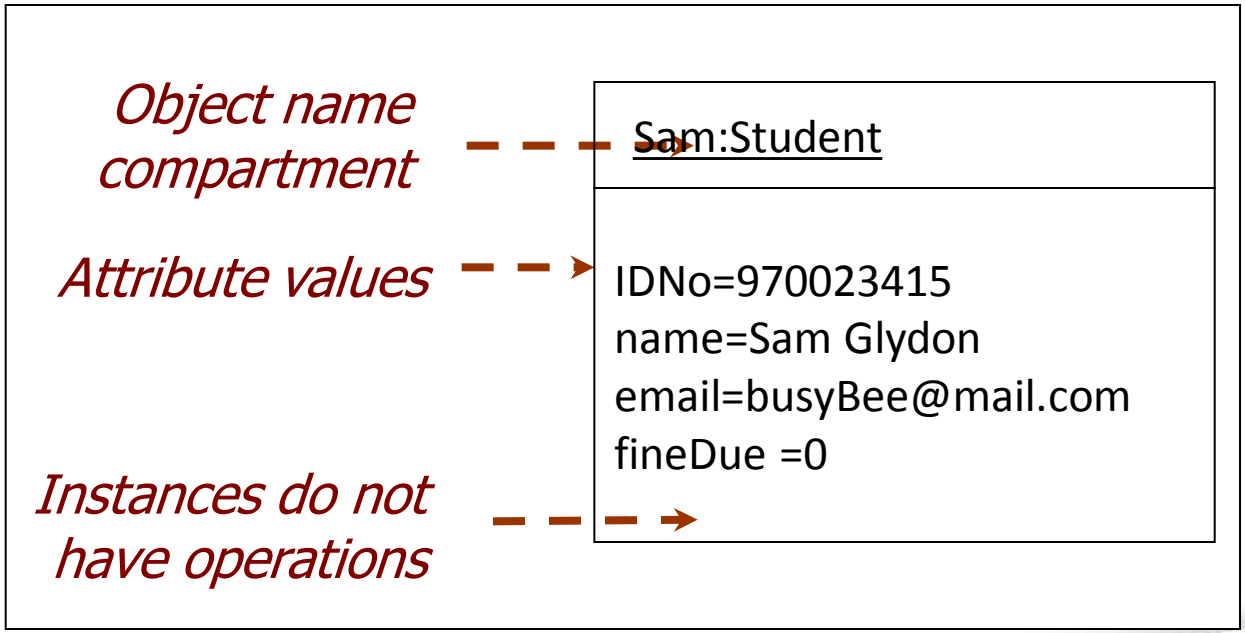


Figure 4-4. Four different ways of showing a class using UML notation

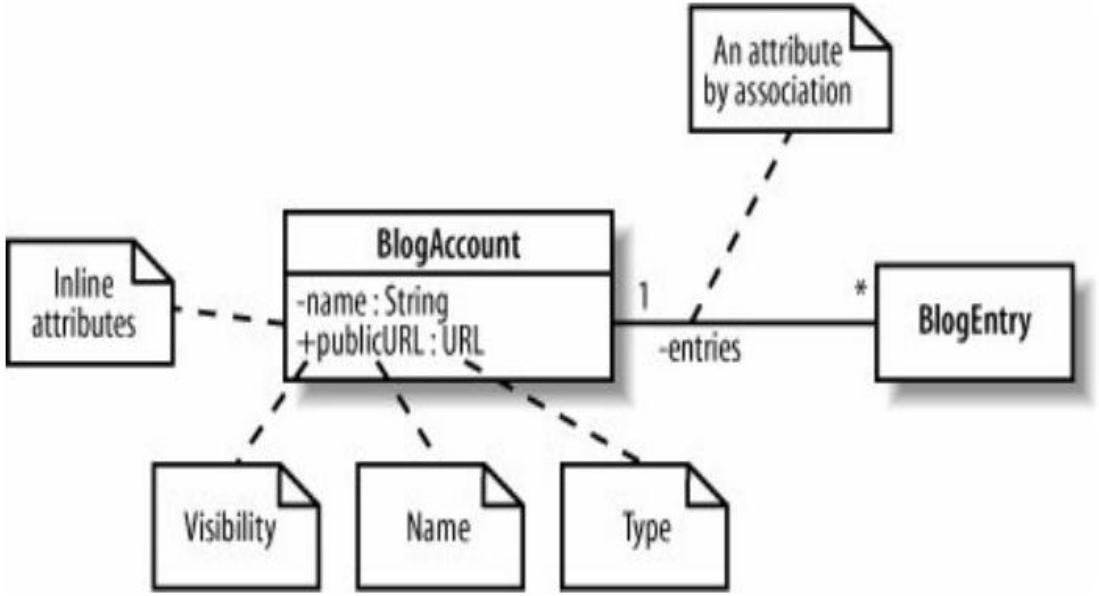
Class name

- Start with a capital letter
- Be centered in the top compartment
- Be written in a boldface font
- Be written in italics if the class is abstract

THUỘC TÍNH (ATTRIBUTE)

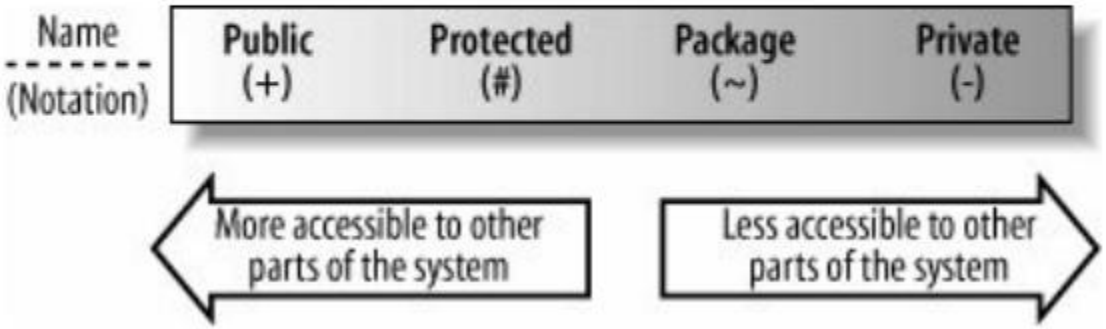
Attribute

Figure 4-11. The BlogAccount class contains two inlined attributes, name and publicURL, as well as an attribute that is introduced by the association between the BlogAccount and BlogEntry classes



Visibility

Figure 4-6. UML's four different visibility classifications



Read Only property

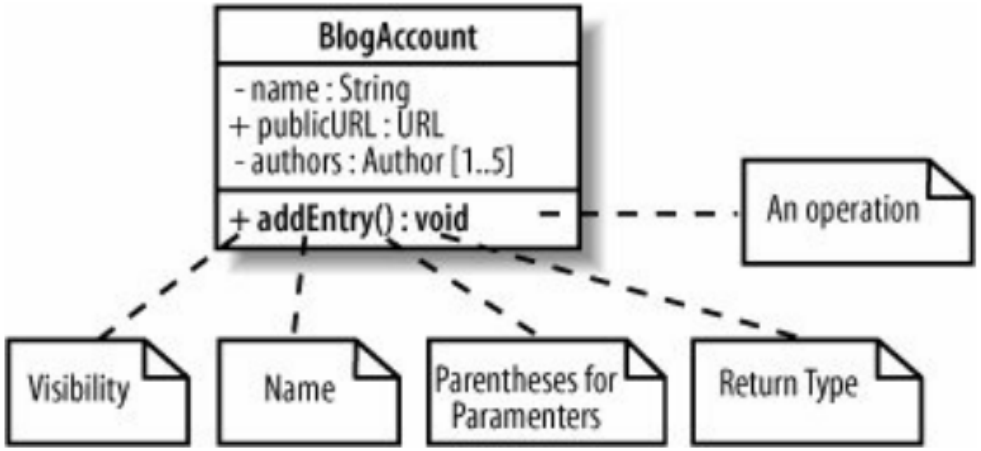
Figure 4-13. The createdBy attribute in the ContentManagementSystem class is given a default initial value and a property of readOnly so that the attribute cannot be changed throughout the lifetime of the system


```
public class ContentManagementSystem
{
    private final String createdBy = "Adam Cook Software Corp.";
}
```

PHƯƠNG THỨC (OPERATION)

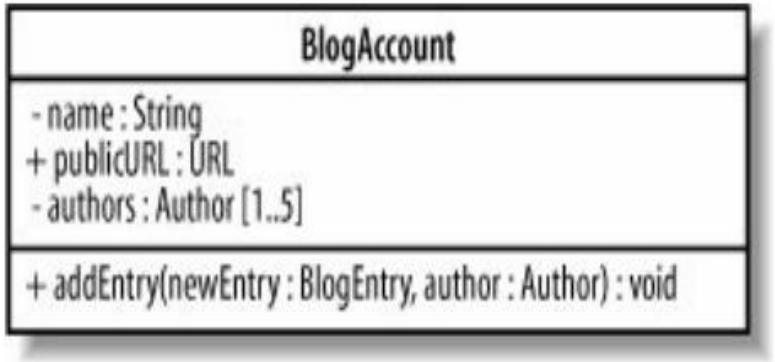
Operation

Figure 4-16. Adding a new operation to a class allows other classes to add a BlogEntry to a BlogAccount



Operation with parameter and return type

Figure 4-21. The BlogAccount class is made up of three regular attributes and one regular operation



XÁC ĐỊNH CLASS

How do we design class ?

- Class identification from project spec / requirements
  - nouns are potential classes, objects, fields
  - verbs are potential methods or responsibilities of a class
- CRC card exercises
  - write down classes' names on index cards
  - next to each class, list the following:
    - responsibilities: problems to be solved; short verb phrases
    - collaborators: other classes that are sent messages by this class (asymmetric)

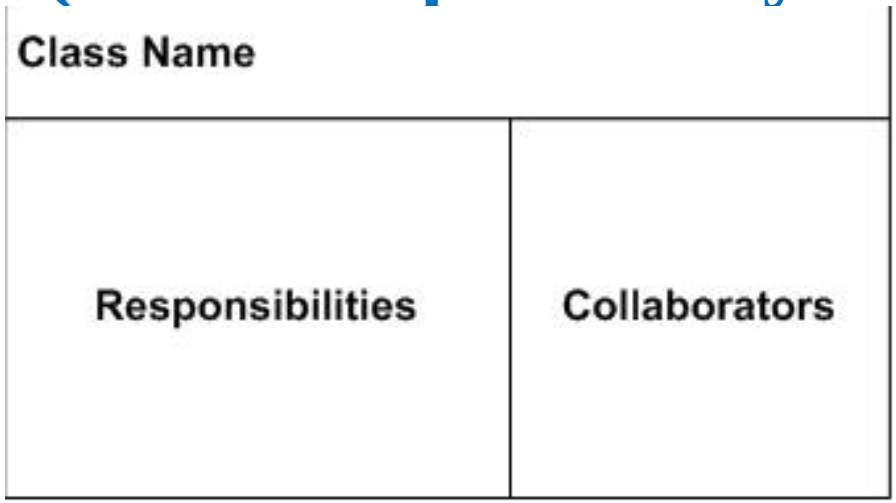
Class identification from project spec / requirements (Phát hiện lớp)

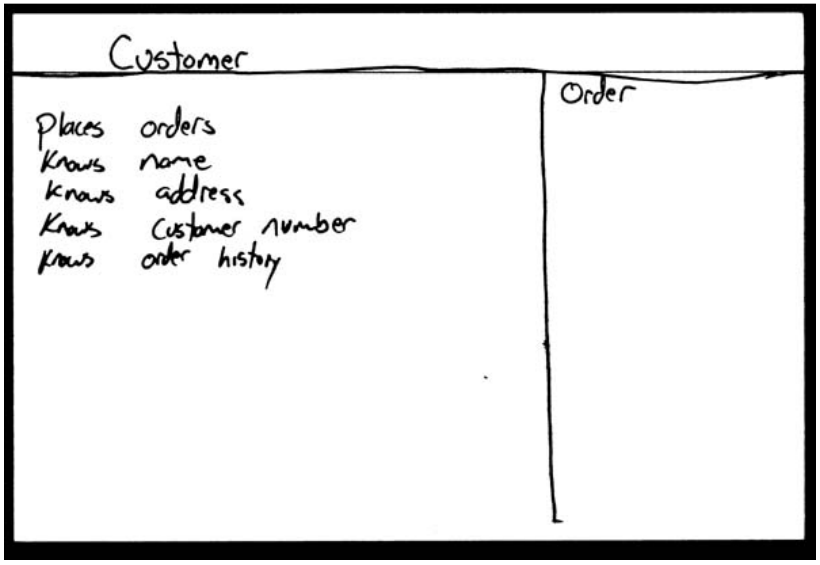
- Từ các danh từ trong phát biểu bài toán
  - Tài liệu yêu cầu phải đầy đủ và đúng.
- Là một phát biểu có mục đích
- Miêu tả cho một tập các đối tượng (nhiều hơn 1)
  - Không xét các lớp chỉ có một thể hiện (Singleton)
- Sở hữu một tập các thuộc tính
  - Thuộc tính định danh: chỉ xem xét nếu có ý nghĩa thực tế.
- Xem xét là lớp hay thuộc tính?
  - Ví dụ: *color* (of a car) có thể 3 thành phần Red, Blue, Green
- Sở hữu một tập các phép toán
  - Phép toán có thể nhận diện sau

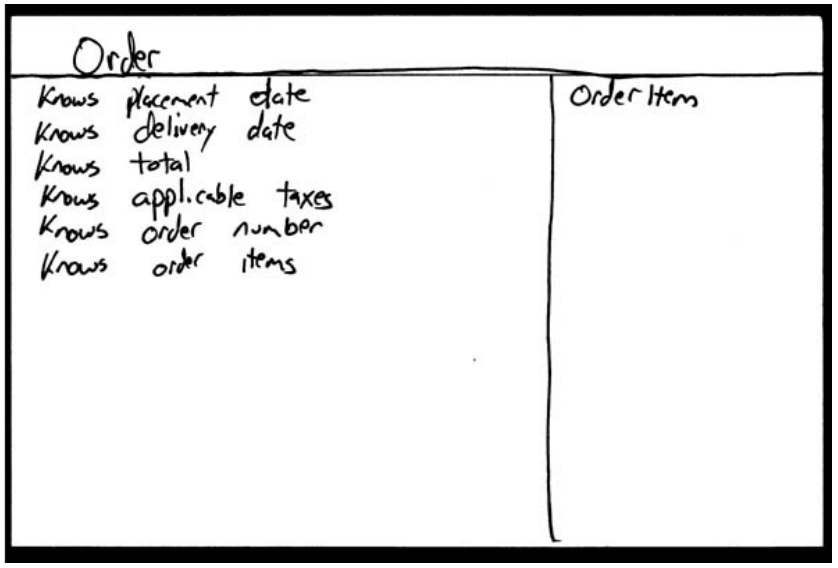
Kiểm tra tính hợp lý của các lớp ứng viên

- Một số test giúp kiểm tra một lớp ứng viên có thể chấp nhận được hay không?
  - Nó có nằm ngoài phạm vi của hệ thống không?
  - Nó có ám chỉ tới toàn bộ hệ thống không?
  - Nó có lập lại một lớp khác không?
  - Nó có quá mơ hồ không?
  - Nó có buộc quá chặt với inputs và outputs vật lý không?
  - Nó có là một thuộc tính hay không?
  - Nó có là một phép toán hay không?
  - Nó có là một mối kết hợp hay không?
- Nếu câu trả lời là "Yes",
  - Có thể xem xét việc mô hình các lớp theo một cách khác hoặc loại bỏ lớp đó

CRC Card (Class Responsibility Collaborator)







Example

- Students have names, addresses, and phone numbers.
- Students also enroll in seminars, drop seminars, and request transcripts.
- The things a class knows and does constitute its responsibilities. Important: A class is able to change the values of the things it knows, but it is unable to change the values of what other classes know.

Professor	
Name Address Phone number Email address Salary Provide information Seminars instructing	Seminar

Transcript	
See the prototype Determine average mark	Student Seminar Professor Enrollment

Seminar	
Name Seminar number Fees Waiting list Enrolled students Instructor Add student Drop student	Student Professor

Enrollment	
Mark(s) received Average to date Final grade Student Seminar	Seminar

Student Schedule	
See the prototype	Seminar Professor Student Enrollment Room

Student	
Name Address Phone number Email address Student number Average mark received Validate identifying info Provide list of seminars taken	Enrollment

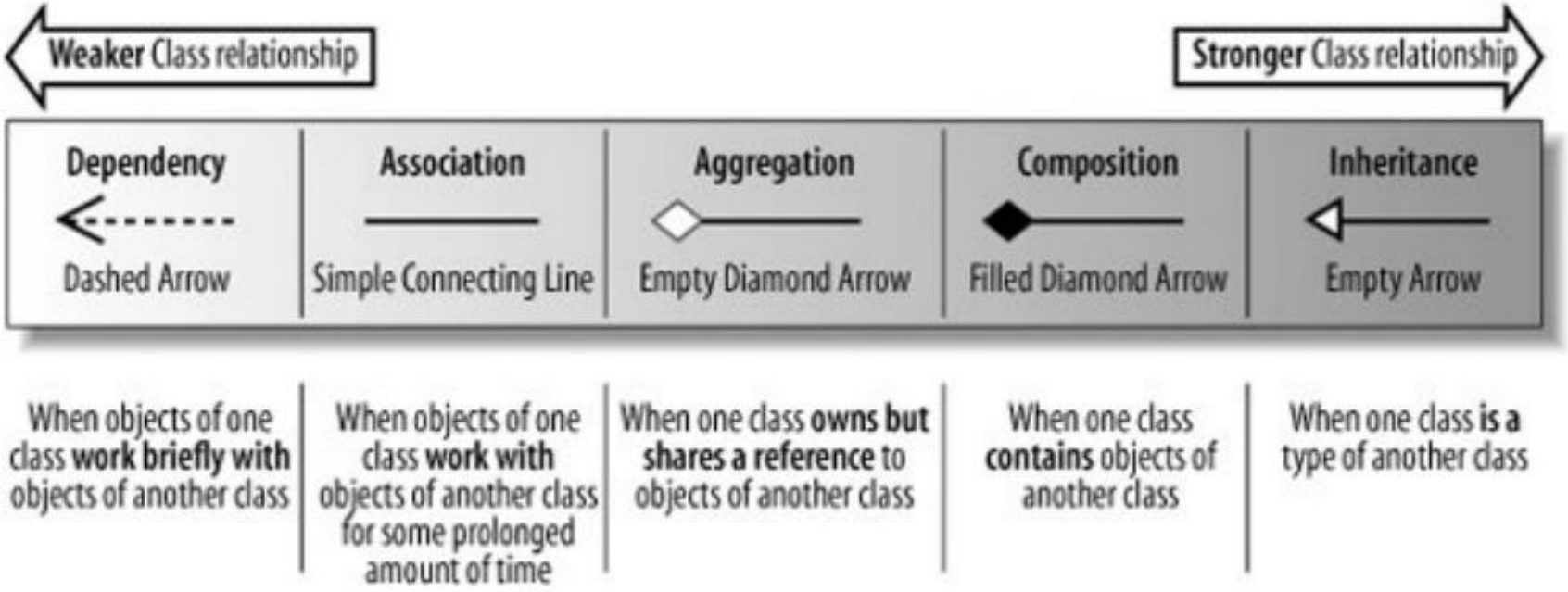
Room	
Building Room number Type (Lab, class, ...) Number of Seats Get building name Provide available time slots	Building

Building	
Building Name Rooms Provide name Provide list of available rooms for a given time period	Room

XÁC ĐỊNH QUAN HỆ GIỮA CÁC LỚP (RELATIONSHIP)

RELATIONSHIP

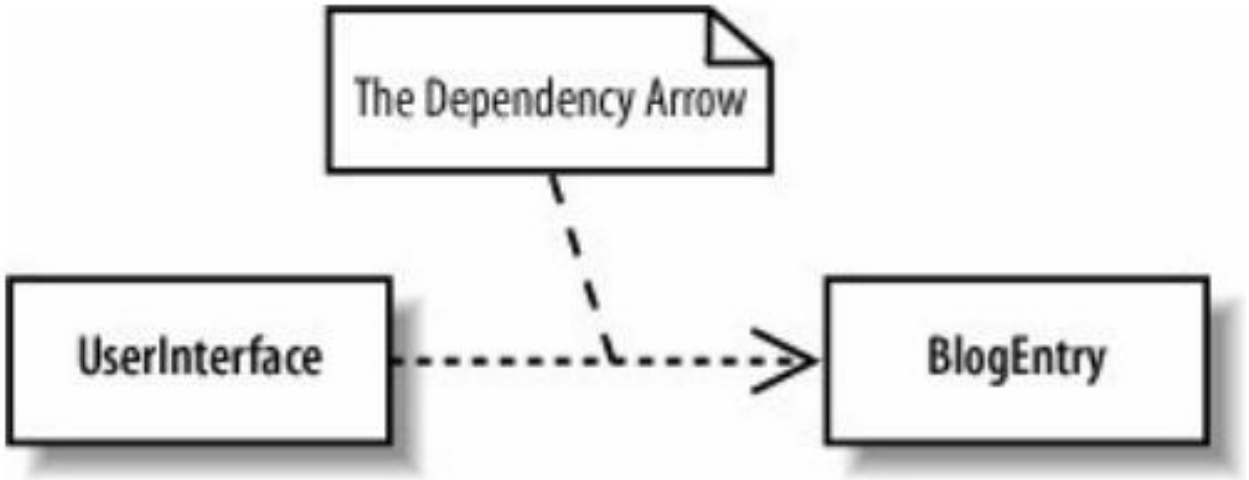
Figure 5-1. UML offers five different types of class relationship



DEPENDENCY

DEPENDENCY

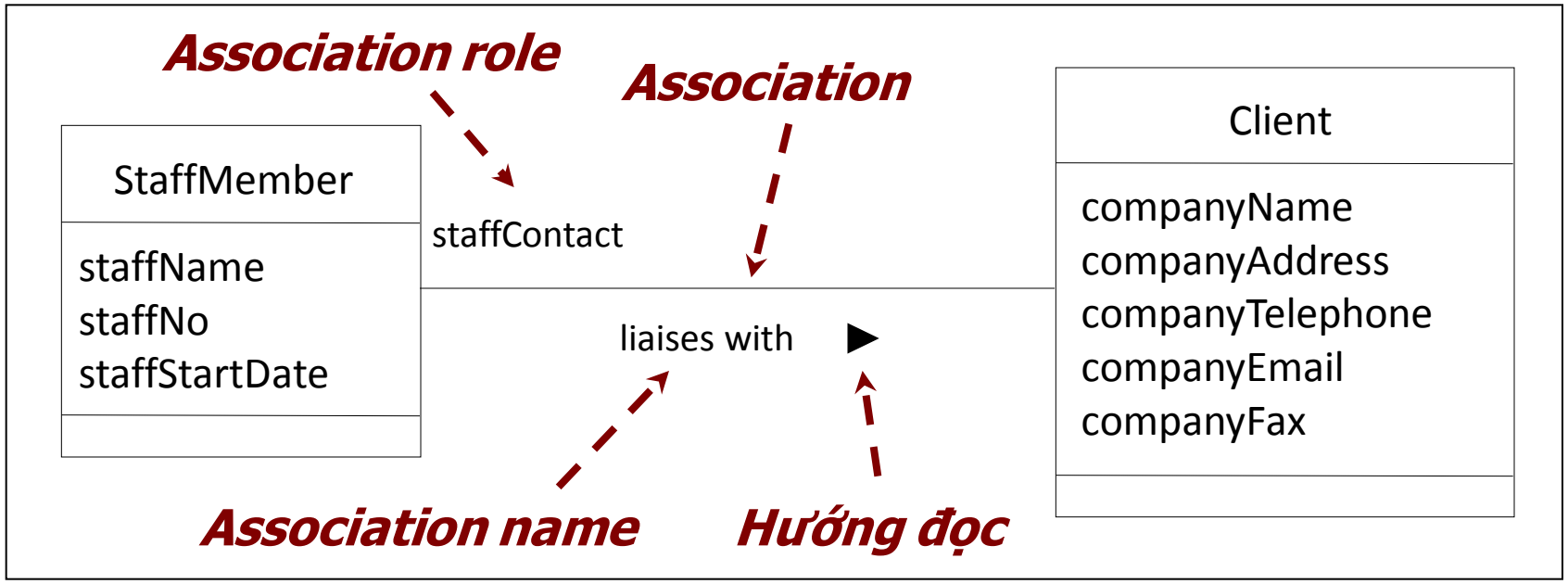
Figure 5-2. The UserInterface is dependent on the BlogEntry class because it will need to read the contents of a blog's entries to display them to the user



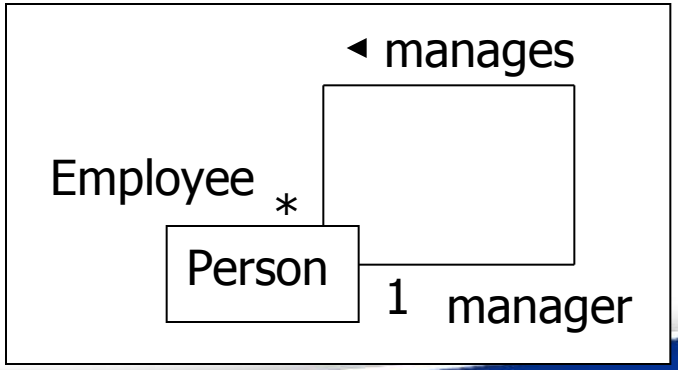
ASSOCIATION

Association – Mối kết hợp

- Khả năng xảy ra liên kết logic hay ngữ nghĩa giữa các đối tượng của lớp này và các đối tượng của lớp khác

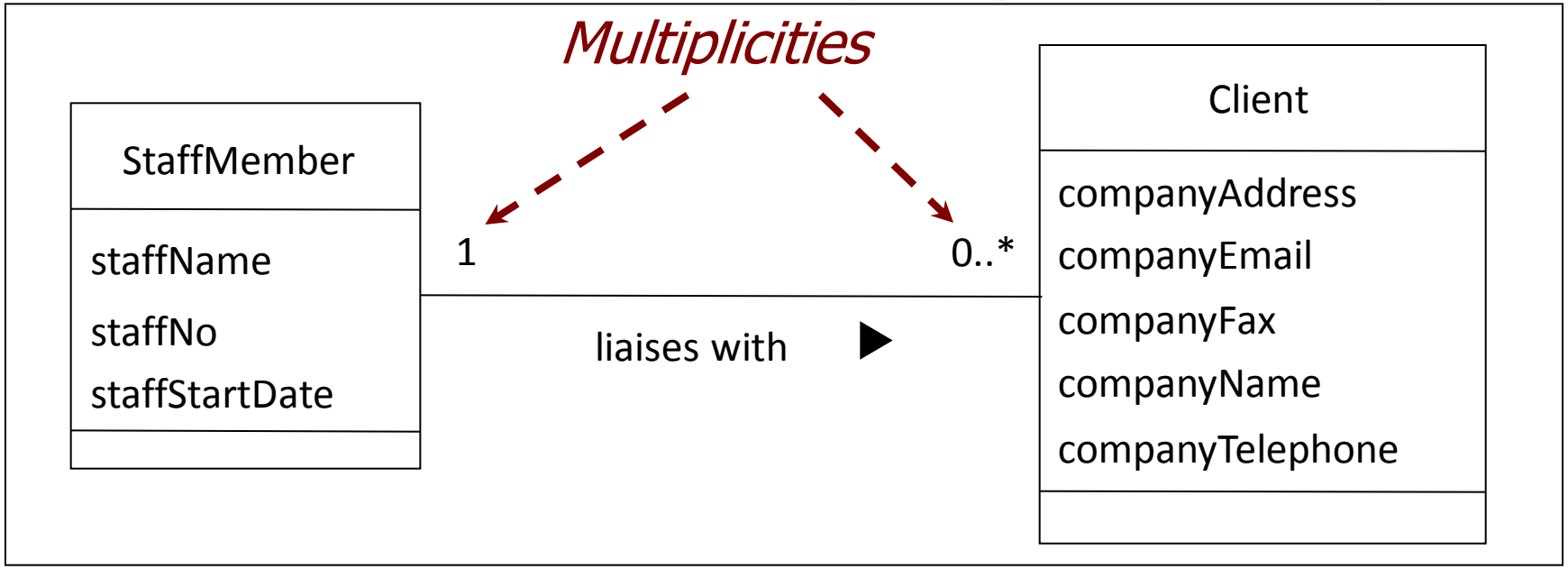


- **Role xác định vai trò của đối tượng tham gia vào quan hệ.**
  - Bắt buộc đối với quan hệ phản xạ



Multiplicity - Bản số của quan hệ

- Là phạm vi số lượng cho phép của các thể hiện của từng lớp tham gia vào quan hệ.
  - Thể hiện các qui định (hay ràng buộc) thương mại.



Có chính xác một nhân viên liên lạc với từng khách hàng

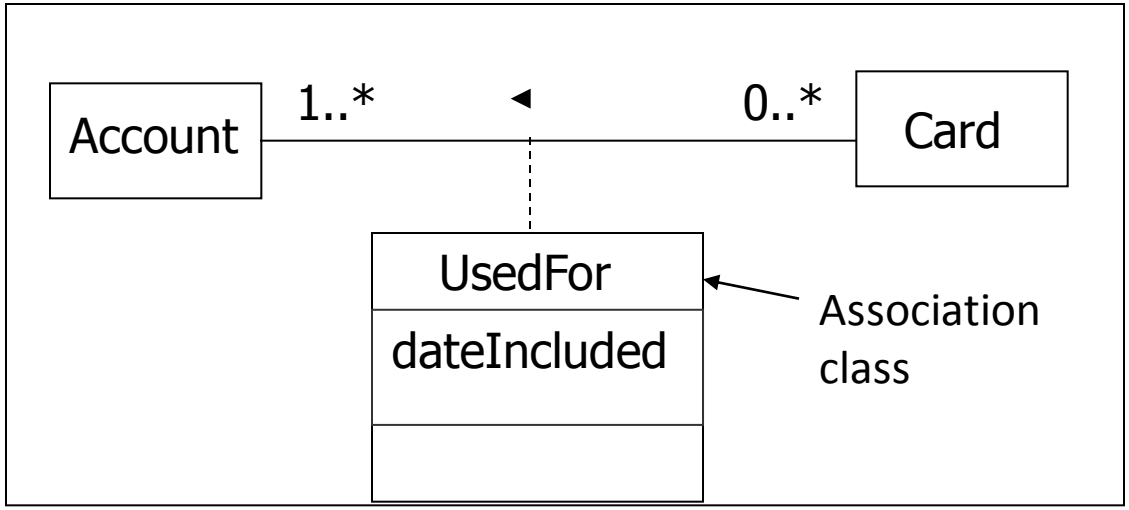
Một nhân viên có thể liên lạc với không, một hay nhiều khách hàng

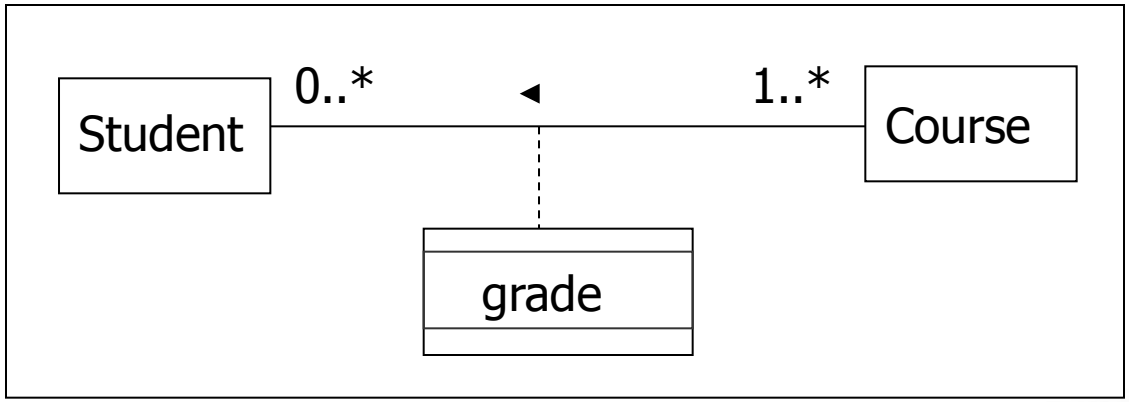
Association class - Lớp kết hợp

- Dùng cho các quan hệ có bản số m*n

Dùng lớp kết hợp khi:

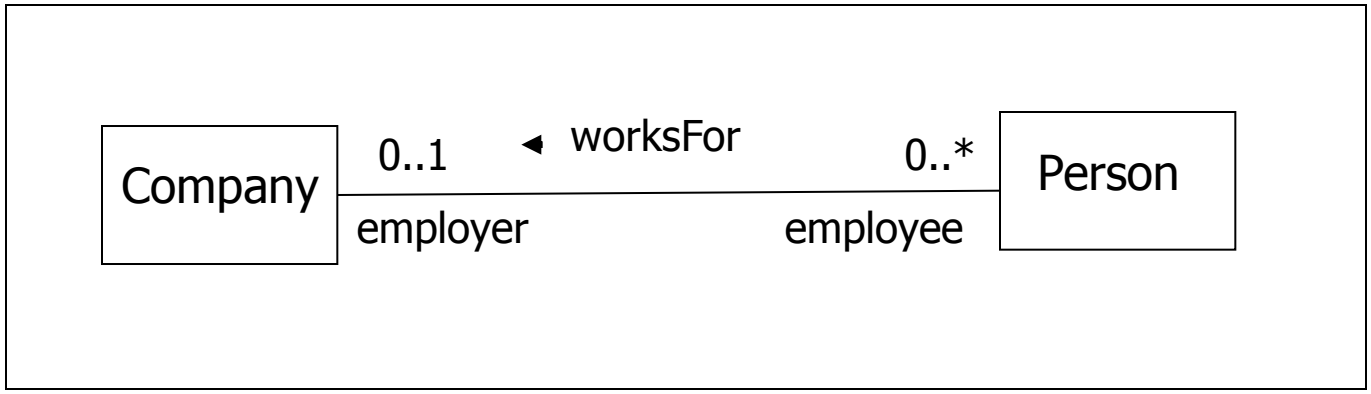
- Để giữ các thuộc tính của quan hệ
- Cần bao gồm một phép toán nào đó
- Cần có quan hệ với các đối tượng khác





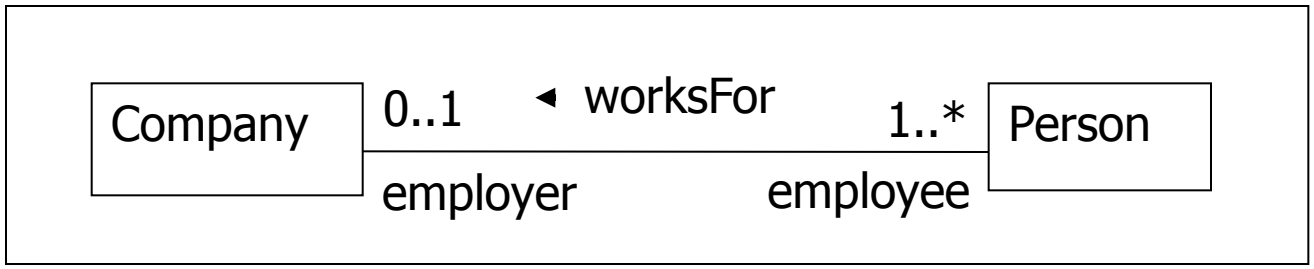
Nhận diện quan hệ Association

- Biểu diễn các quy định thương mại (business rules)
- Quan hệ yếu giữa các lớp: Sự tồn tại của các đối tượng độc lập với nhau.
- Hướng liên kết có thể bao gồm sau trong bước thiết kế lớp.

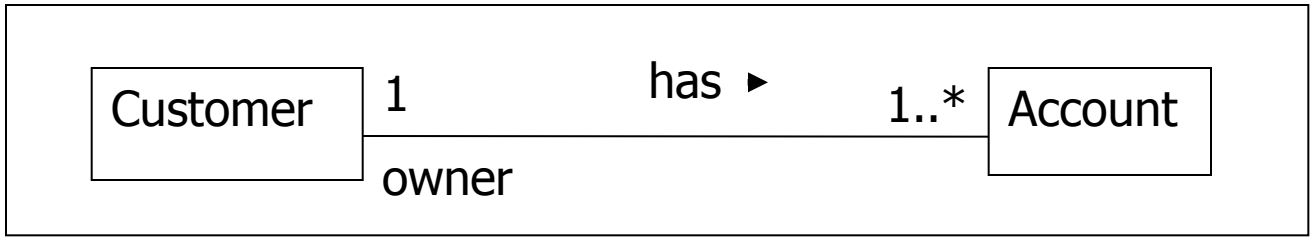


Ví dụ

- Một công ty phải có ít nhất 1 hoặc nhiều nhân viên.
- Một người làm việc cho không hay chỉ một công ty

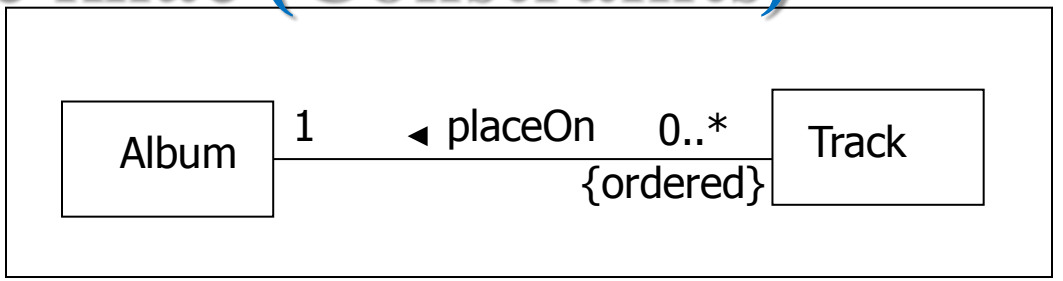


- Mỗi khách hàng của ngân hàng (bank customer) có thể có 1 hay nhiều tài khoản (accounts)
- Mỗi tài khoản thuộc duy nhất 1 khách hàng

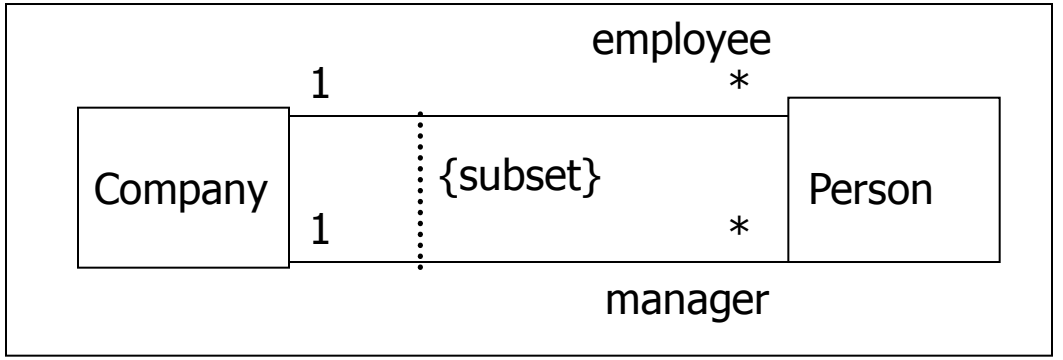


Các ràng buộc khác (Constraints)

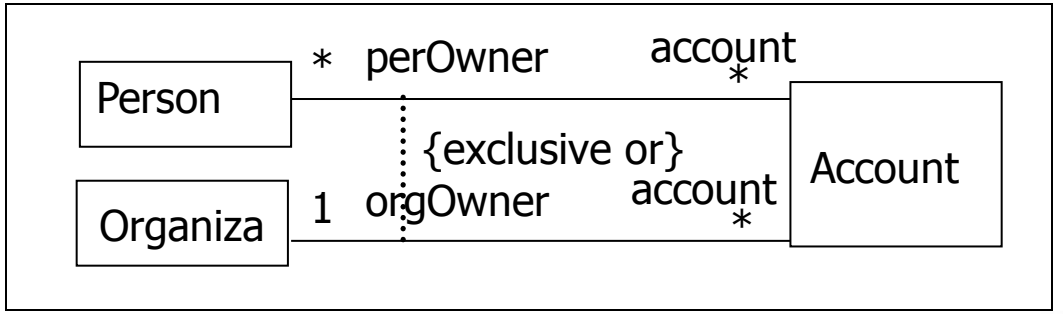
- Ordered: có thứ tự
  - Trong một Album, các Track phải được ghi theo một thứ tự nào đó.



- **Subset: tập con**
  - Manager phải là một employee

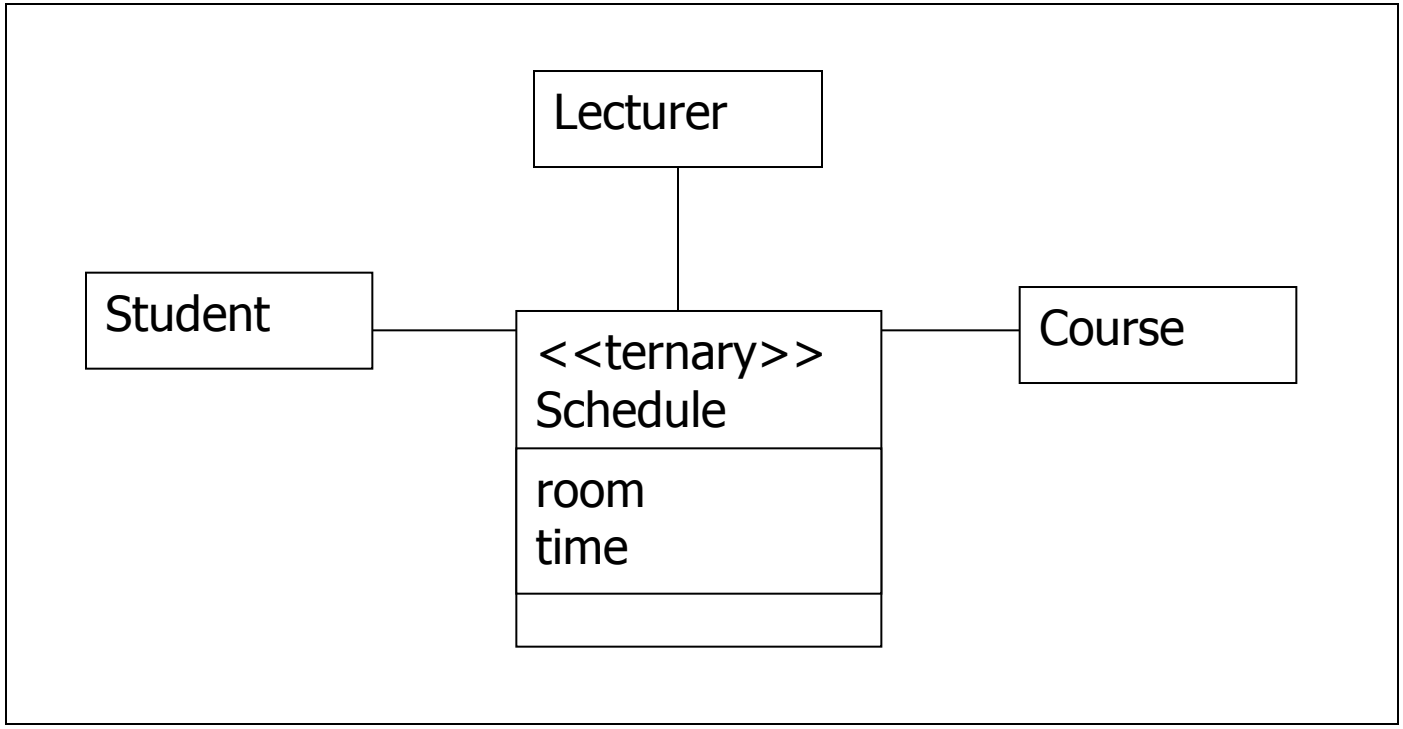


- **XOR:**
  - Loại trừ lẫn nhau



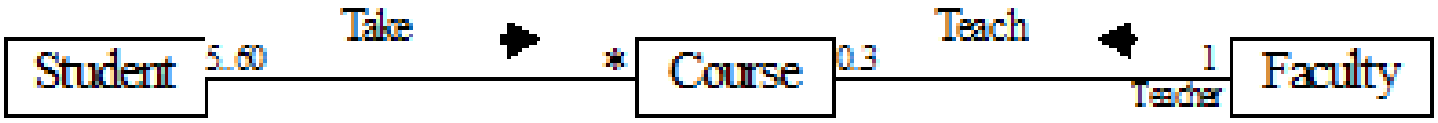
Tertiary Relations - Quan hệ bậc 3

- Thuộc tính 'room' đặt ở đâu?
  - Lecture "L1" dạy Course "C1" cho Student "S1" trong Room "R1"
  - Lecture "L1" dạy Course "C1" cho Student "S2" trong Room "R2"



Association

Association represents a general binary relationship that describes an activity between two classes.



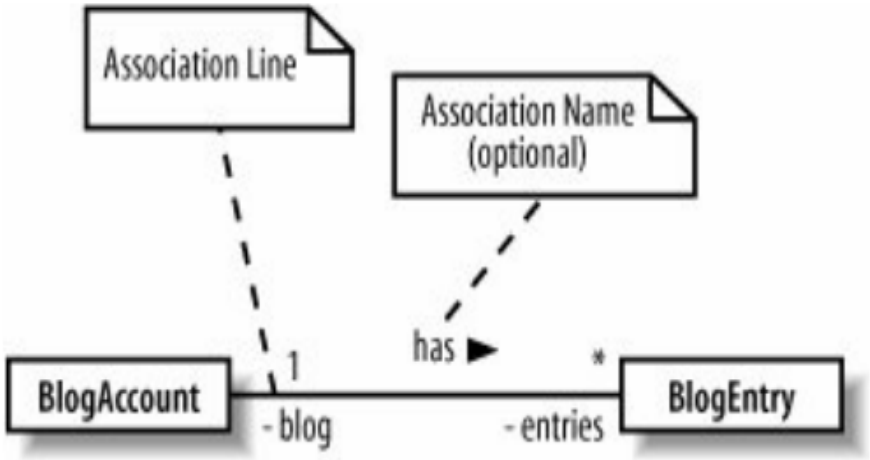

```
public class Student {
  /** Data fields */
  private Course[]
    courseList;

  /** Constructors */
  /** Methods */
}
```

```
public class Course {
  /** Data fields */
  private Student[]
    classList;
  private Faculty faculty

  /** Constructors */
  /** Methods */
}
```

```
public class Faculty {
  /** Data fields */
  private Course[]
    courseList;

  /** Constructors */
  /** Methods */
}
```

An association is usually represented as a data field in the class.

Figure 5-3. The BlogAccount class is optionally associated with zero or more objects of the BlogEntry class; the BlogEntry is also associated with one and only one BlogAccount



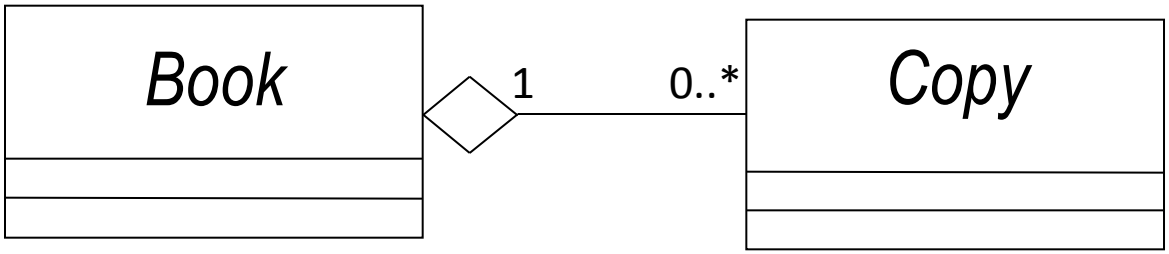
Example 5-1. The BlogAccount and BlogEntry classes without navigability applied to their association relationship

```
public class BlogAccount {

   // Attribute introduced thanks to the association with the BlogEntry class
   private BlogEntry[] entries;

   // ... Other Attributes and Methods declared here ...
}

public class BlogEntry {

   // Attribute introduced thanks to the association with the Blog class
   private BlogAccount blog;

   // ... Other Attributes and Methods declared here ...
}
```

AGGREGATION & COMPOSITION

Cấu trúc Aggregation

- Một kiểu quan hệ association đặc biệt
  - Là quan hệ whole–part hay is-a-part-of
- Có sự phụ thuộc mạnh giữa các lớp
  - Một lớp đóng vai trò quan trọng hơn
- Có tính chất bổ sung (Transitive) và bất đối xứng (Asymmetry)



Nhận diện Aggregation

Bốn ngữ nghĩa có thể của Aggregation

- Sở hữu độc quyền (Exclusive Owns): vd: Book has Chapter
  - Có sự phụ thuộc tồn tại (không có chapter nếu không có book)
  - Transitivity
  - Asymmetricity
  - Là thuộc tính cố định (một chapter không thể chuyển sang book khác)
- Sở hữu (Owns) ví dụ: Car has Tire
  - Không là thuộc tính cố định (có thể chuyển tire sang một car khác)
- Có (Has) ví dụ: Division has Department
  - Không có sự phụ thuộc tồn tại.
  - Không có thuộc tính cố định.
- Thành viên (Member) ví dụ: Meeting has Chairperson
  - Không có đặc trưng gì đặc biệt ngoại trừ là tư cách thành viên

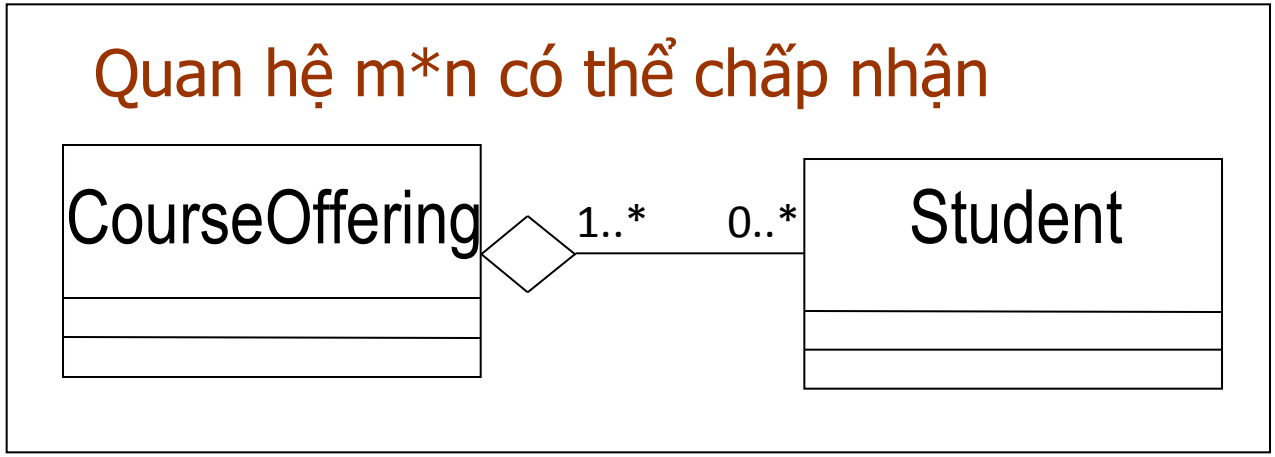
Composition VS Aggregation

- Aggregation về cơ bản là bất kỳ quan hệ whole–part
  - Ngữ nghĩa có thể rất mơ hồ
  - Tương ứng với ngữ nghĩa **"Has"** và **"Member"**.
  - Một đối tượng thành phần (part) có thể thuộc nhiều hơn một đối tượng bao gồm (whole)

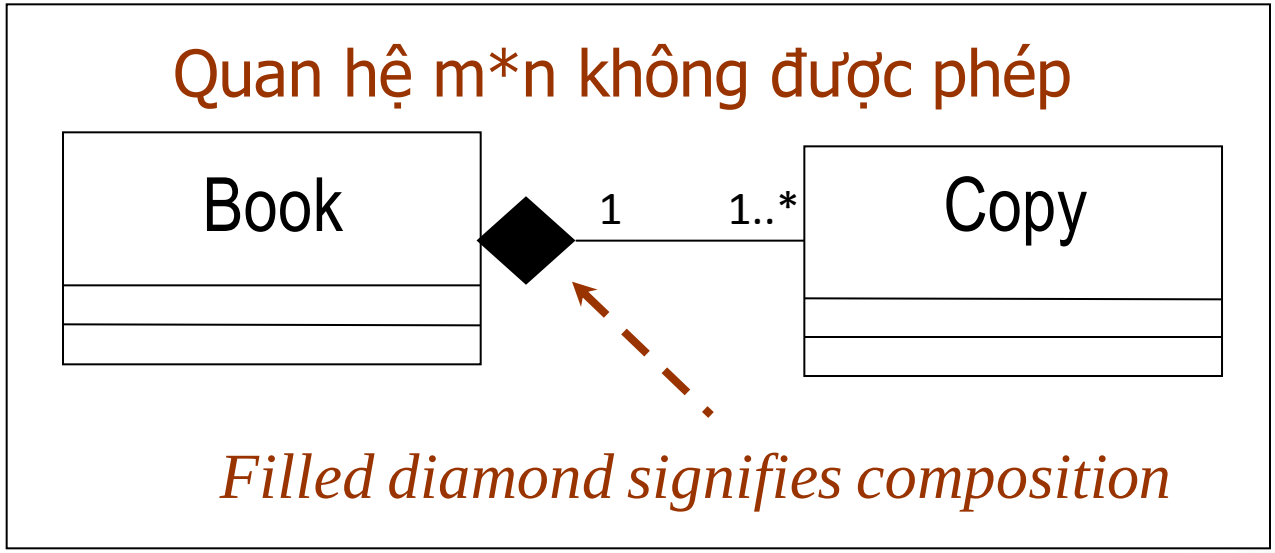
- Composition là Aggregation mạnh hơn
  - Tại một thời điểm, mỗi đối tượng thành phần (part) chỉ có thể thuộc chỉ một đối tượng bao gồm (whole).
  - Có sự phụ thuộc tồn tại từ "part" vào "whole"
    - Khi đối tượng "whole" bị hủy thì các "part" cũng bị hủy
  - Tương ứng với ngữ nghĩa **"ExclusiveOwns"** và **"Owns"**

Aggregation VS Composition

Các student có thể trong nhiều class
Nếu class bị hủy, các student không bị hủy!



Nếu book bị xóa, các bản sao (copies) cũng bị xóa!



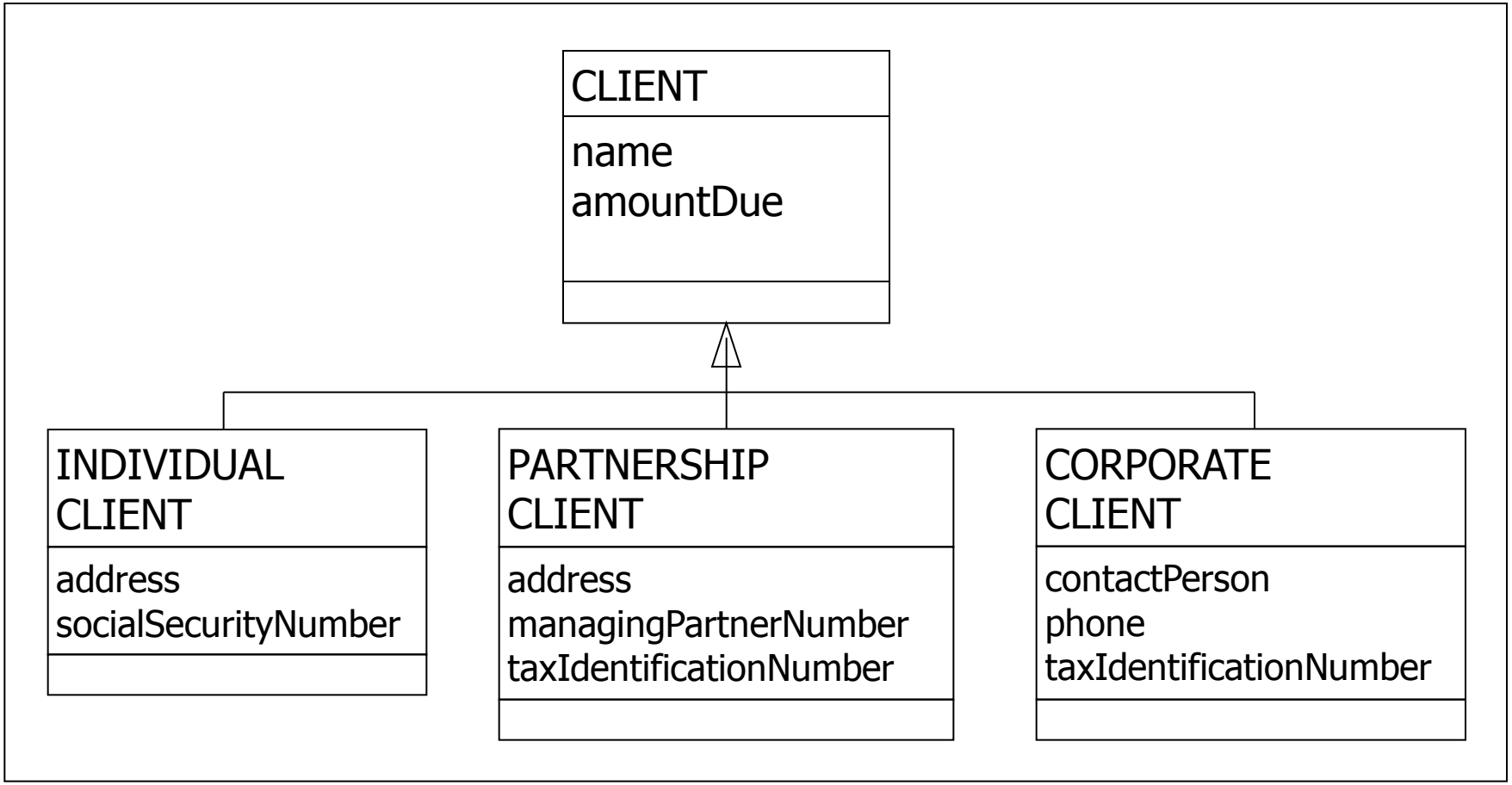
GENERALIZATION-SPECIALIZATION

Nhận diện cấu trúc Generalization-Specialization

- Thêm cấu trúc generalization khi
  - Hai lớp có nhiều chi tiết giống nhau.
  - Có quan hệ Is-A
- Các đặc trưng chung được trừu tượng hóa thành một lớp tổng quát hơn.
- Các lớp con được kế thừa các đặc trưng (thuộc tính, phép toán, quan hệ) của lớp cha
- Các tính chất khác:
  - Substitutability – đối tượng lớp con có thể là giá trị thay thế hợp lệ cho một biến thuộc lớp cha
  - Polymorphism – cùng một phép toán có thể cài đặt khác nhau ở các lớp khác nhau.
  - Abstract operation – cài đặt được cung cấp ở lớp con, còn khai báo ở lớp cha
  - Abstract class – lớp không có thể hiện đối tượng trực tiếp
    - Một lớp với một phép toán abstract là lớp abstract

Ví dụ, Generalization-Specialization

Một khách hàng có thể là khách hàng cá nhân, hoặc khách hàng thành viên, hoặc khách hàng công ty. Mỗi loại khách hàng sẽ có các thuộc tính riêng của nó

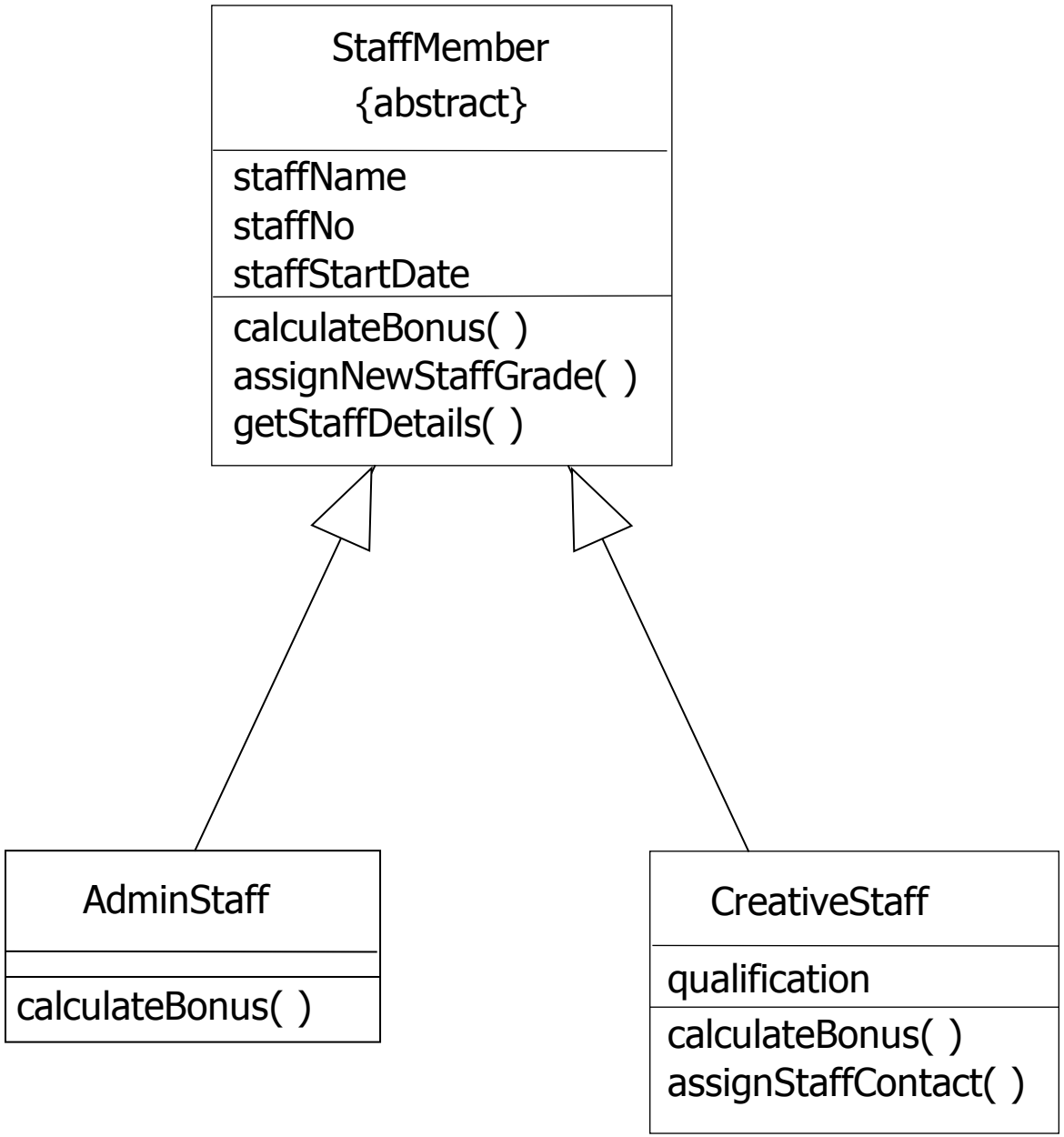


Ví dụ, Generalization-Specialization

- Có 2 kiểu nhân viên (Staff):

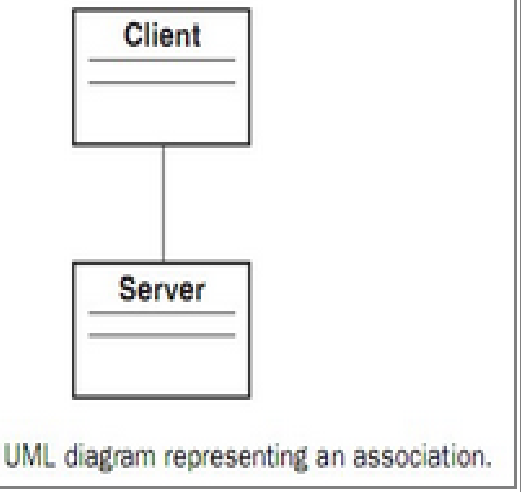
Creative	Cần ghi nhận thuộc tính qualifications Có thể liên hệ với khách hàng Có tiền thưởng tính toán dựa trên lợi nhuận các chiến dịch quảng cáo tham gia
Admin	Không ghi nhận thuộc tính Qualifications Không liên hệ với khách hànng Tiền thưởng tính toán không dựa trên lợi nhuận của các chiến dịch quảng cáo

Adding Generalization-Specialization structure to Class Diagram

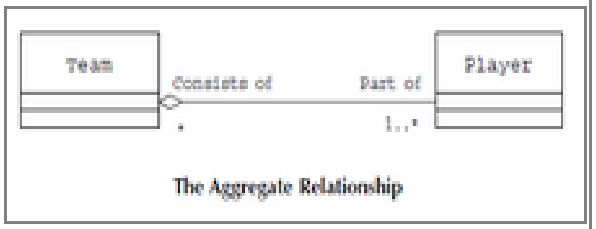


TÓM TẮT

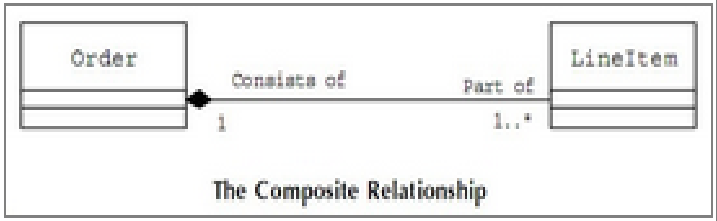
Association	Aggregation	Composition
Class A **uses** Class B.	Class A is **owns** Class B.	Class A **contains** Class B.
Example: • Employee uses BusService for transportation. • Client-Server model. • Computer uses keyboard as input device.	**Example:** • Manager has N Employees for a project. • Team has Players.	**Example:** • Order consists of LineItems. • Body consists of Arm, Head, Legs. • BankAccount consists of Balance and TransactionHistory.
An association is **used when** one object wants another object to perform a service for it. Eg. Computer uses keyboard as input device.	An aggregation is **used when** life of object is independent of container object But still container object owns the aggregated object. Eg. Team has players, If team dissolve, Player will still exists.	A composition is **used where** each part may belong to only one whole at a time. Eg. A line item is part of an order so A line item cannot exist without an order.
Association UML Notation: Associations are represented by just the line (no diamond).	**Aggregation UML Notation:** Aggregations are represented by the line with diamond.	**Composition UML Notation:** Compositions are represented by the line with filled diamond.



UML diagram representing an association.



The Aggregate Relationship



The Composite Relationship

Life or existance of the associated objects are independent of each other, They just provide some kind of service to each other.

Life or existance of the aggregated objects are independent of each other, But one object is playing the role of Owner of the other object.

Life or existance of the composite object is dependent on the existance of container object, Existence of composite object is not meaningful without its container object.

References

- http://agilemodeling.com/artifacts/crcModel.htm
- Study UML2 (Kim Hamilton, Russell Miles)
- Based on lectures by R Poet 2003, modified by P Gray and R Welland.
- Learning UML 2.0 By Kim Hamilton, Russell Miles