Chương 3

Cấu trúc OOP trong Java

Trường Đại học Công nghệ Sài gòn
Khoa Công nghệ Thông tin

Nội Dung

- Các khái niệm cơ bản về lớp, đối tượng.
- Lớp và đối tượng trong java (class và object)
- Tính bao đóng (**encapsulation)**
- Tính đóng gói (package)
- Tính kế thừa (Inheritance)
- Tính đa hình (Polymorphism)
- Tính trừu tượng Abstract và Interface.

Các khái niệm cơ bản

- Đối tượng (object): Trong thế giới thực, khái niệm đối tượng có thể xem như một thực thể: người, vật, bảng dữ liệu,…
  - Đối tượng giúp ta hiểu rõ thế giới thực
  - Cơ sở cho việc cài đặt trên máy tính
  - Mỗi đối tượng có định danh, thuộc tính, hành vi
- Ví dụ:
  - đối tượng sinh viên MSSV: “TH0701001”;Tên sinh viên: “Nguyễn Văn A”
- Hệ thống các đối tượng: Là 1 tập hợp các đối tượng
  - Mỗi đối tượng đảm trách 1 công việc
  - Các đối tượng có thể trao đổi thông tin với nhau
  - Các đối tượng có thể xử lý song song, hay phân tán

Các khái niệm cơ bản

- Lớp (class): Là khuôn mẫu (template) để sinh ra đối tượng (Là một kiểu dữ liệu)
  - Ví dụ: class các đối tượng Sinhvien
    - Sinh viên "Nguyễn Văn A", mã số TH0701001 → 1 đối tượng thuộc lớp Sinhvien.
    - Sinh viên "Nguyễn Văn B", mã số TH0701002 → là 1 đối tượng thuộc lớp Sinhvien.
- Đối tượng (object) của lớp: Một đối tượng cụ thể thuộc 1 lớp, 1 thể hiện cụ thể của 1 lớp đó.

Lớp và Đối tượng trong java

- Khai báo class:

```
<modifier> class <name> {
    <khai báo thuộc tính>*
    <Khai báo khởi dựng>*
    <Khai báo phương thức>*
}
```

- Giải thích:
  - <*modifier*> phạm vi truy xuất.
  - Lưu ý: khai báo <modifier> cho class ở mức đỉnh chỉ có thể là : public, default, protected

Lớp và Đối tượng trong java

- Thuộc tính: Các đặc điểm mang giá trị của đối tượng, là vùng dữ liệu được khai báo bên trong lớp

class <*ClassName*>{

<*Tiền tố*> <*kiểu dữ liệu*> <*tên thuộc tính*>;

}

- Kiểm soát truy cập đối với thuộc tính
  - public
  - protected
  - private

Lớp và Đối tượng trong java

- Phương thức: Chức năng xử lý, hành vi của các đối tượng.

```
class <ClassName>{
        …
        <Tiền tố> <kiểu trả về> <tên phương thức>(<các đối số>)
        {
                …
        }
}
```

Lớp và Đối tượng trong java

- Phạm vi truy xuất
  - public
  - protected
  - *default*
  - private
- final: Không được khai báo chồng ở các lớp dẫn xuất (không được ghi đè ở lớp con)
- abstract: Không có phần source code, sẽ được cài đặt trong các lớp dẫn xuất.
- static: Phương thức lớp dùng chung cho tất cả các thể hiện của lớp, có thể được thực hiện cả khi không có đối tượng của lớp.
- native: đ ây là từ khoá báo cho java biết phương thức này được viết bằng một ngôn ngữ lập trình nào đó không phải là java (thường được viết bằng C/C++)
- synchoronized: Dùng để ngăn những tác động của các đối tượng khác lên đối tượng đang xét trong khi đang đồng bộ hóa. Dùng trong lập trình multithreads.

Phương thức khởi dựng (contructor)

- Là một phương thức đặc biệt của class , dùng để khởi dựng một đối tượng (tạo mới-object type)
- Cung cấp các giá trị cho các **thuộc tính** của đối tượng
- Cùng tên với tên class
- Không có giá trị trả về
- Có thể có hoặc không tham số

```
class Student {
//Không có định nghĩa khởi dựng nào
}
------
Student stu1=new  Student();
```

```
class Student {
//Không định nghĩa khởi dựng mặc định
        Student(<đối số>();
}
------
Student stu1=new  Student();
```

Khối vô danh

- Không static
  - Được java thực thi khi đối tượng được tạo ra (trước cả hàm khởi dựng)
  - Được đặt trong cặp { ---- }
- Static
  - **Chạy một lần duy nhất khi class load vào bộ nhớ, không phụ thuộc vào việc tạo đối tượng.**
  - Trước khối lệnh đặt từ khóa static: **static** { ---- }
  - Được dùng để khởi tạo các thành phần static

Nạp chồng phương thức (overloading)

- Khai báo các phương thức trùng tên, nhưng có đối số khác nhau trong cùng một lớp

```
class Student {
//các khởi dựng
        Student() {  }
        Student(..) {  }
//các phương thức
    public void getData() {....}
    public void getData(String id) {.....}
}
```

Tham chiếu this

- Một biến ẩn, tồn tại trong tất cả các class, tham khảo đến bản thân của class chứa nó (run time))
- this được dùng trong các tình huống sau:
  - Phân biệt tuộc tính cục bộ từ một biến cục bộ
  - Tham khảo thành phần từ một phương thức không static

```
class ThisDemo1 {
    int data;
    void method(int data) {
        this.data = data;
    }
```

```
class ThisDemo2 {
    int data;
    void method() {
        System.out.println(data); }
    void method2() {
        this.method(); } }
```

Từ Khóa This (tt)

- Dùng cho các khởi dựng khác (là lệnh đầu tiên)

```
class ThisDemo3 {
    int data;
    ThisDemo3() {
            this(100);
    }
    ThisDemo3(int data) {
       this.data = data;
    }
}
```

Tính đóng gói

- Nhóm những gì có liên quan với nhau vào thành một, và có thể sử dụng một tên để gọi
- Ví dụ:
  - Các phương thức đóng gói các câu lệnh
  - Đối tượng đóng gói dữ liệu và các phương thức
- Đóng gói để che dấu một phần hoặc tất cả thông tin, chi tiết cài đặt bên trong với bên ngoài

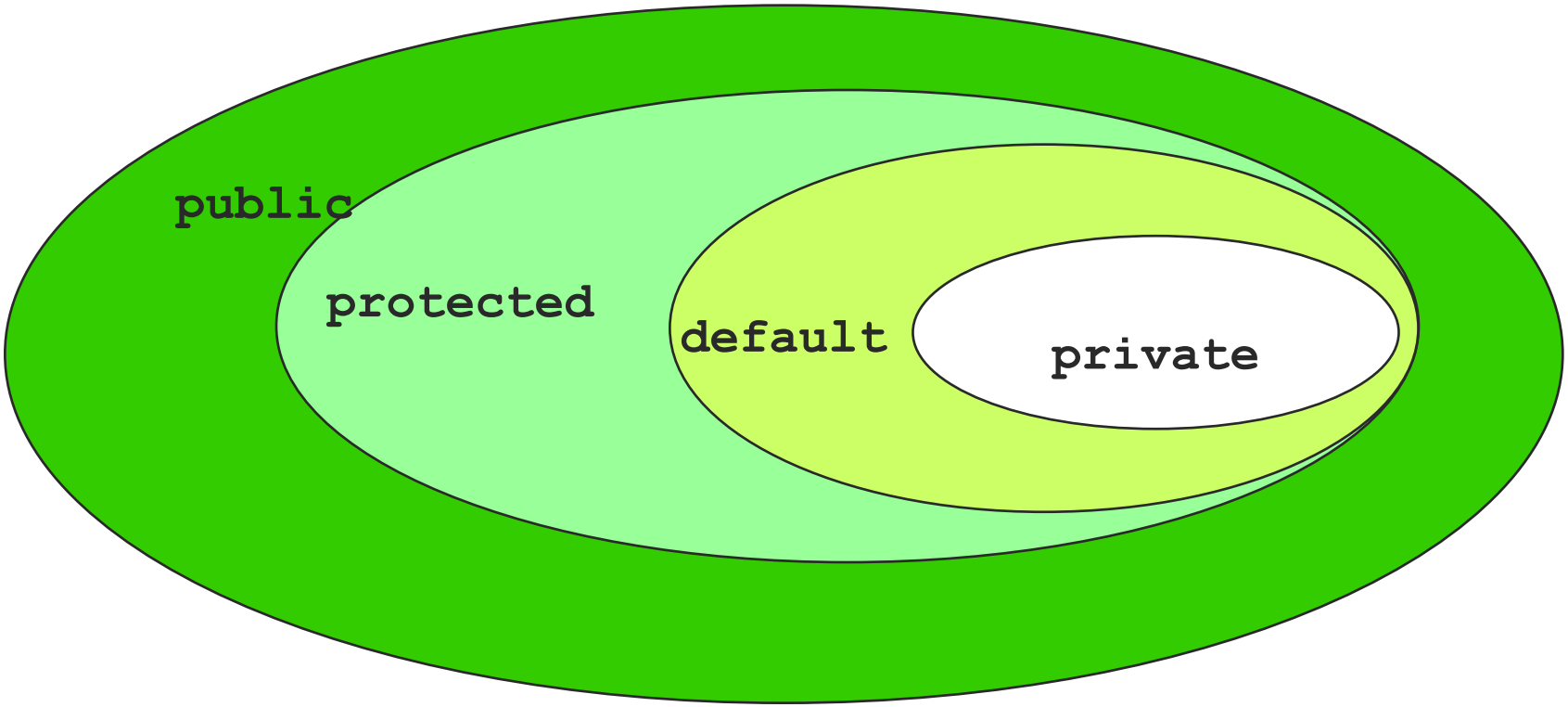
Tính đóng gói

- Nhóm các Class lại nhằm tăng tính bao đóng
- Cú pháp:
  - Khai báo gói chứa class hiện tại:
    - package <packageName>;
  - Khai báo Gói cần sử dụng trong chương trình
    - import <packageName.elementAccessed>;

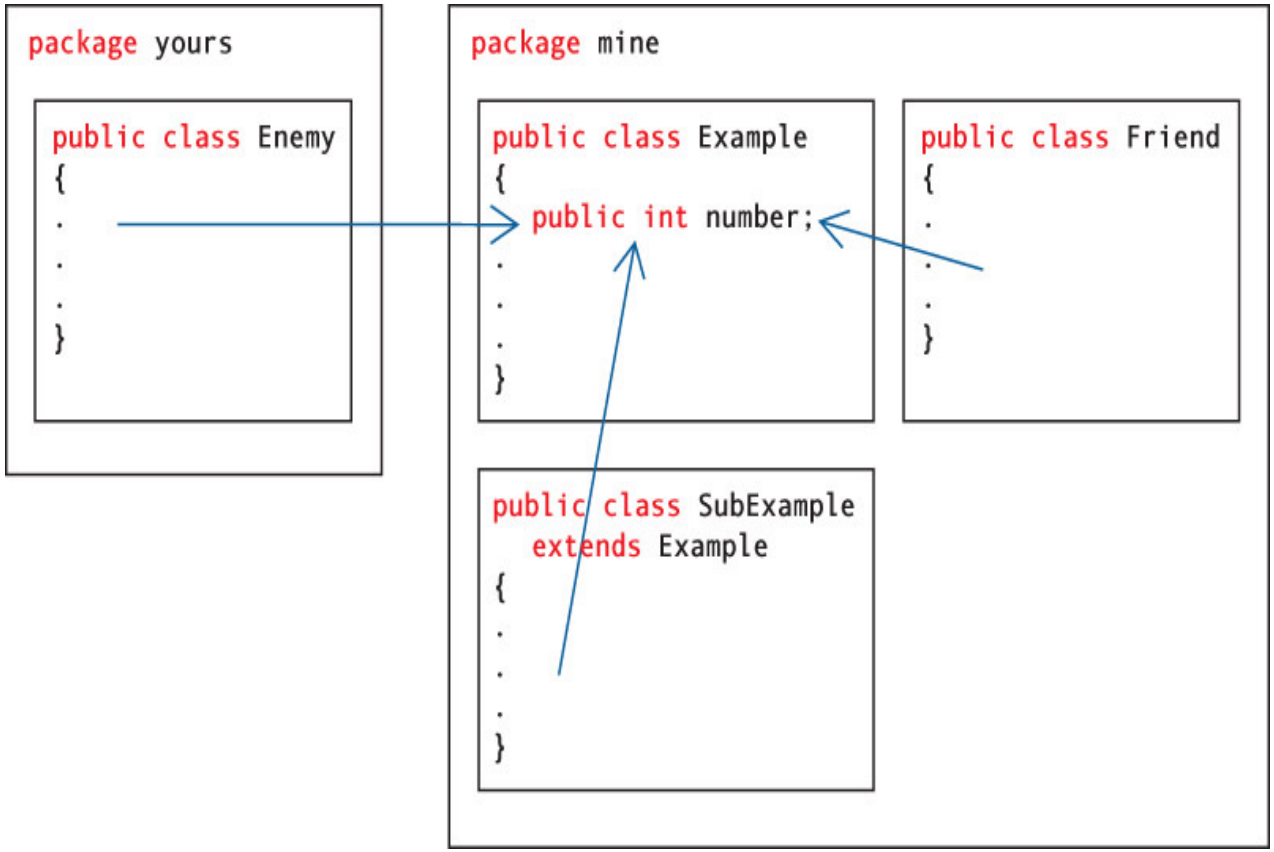
```
package registration.reports;//Tên gói chứa class MyClass
import registration.processing.*; //khai báo thư viện
import java.util.List;
import java.lang.*;
class MyClass {
   /* */
}
```

DEMO

Phạm vi truy xuất (Modifier)

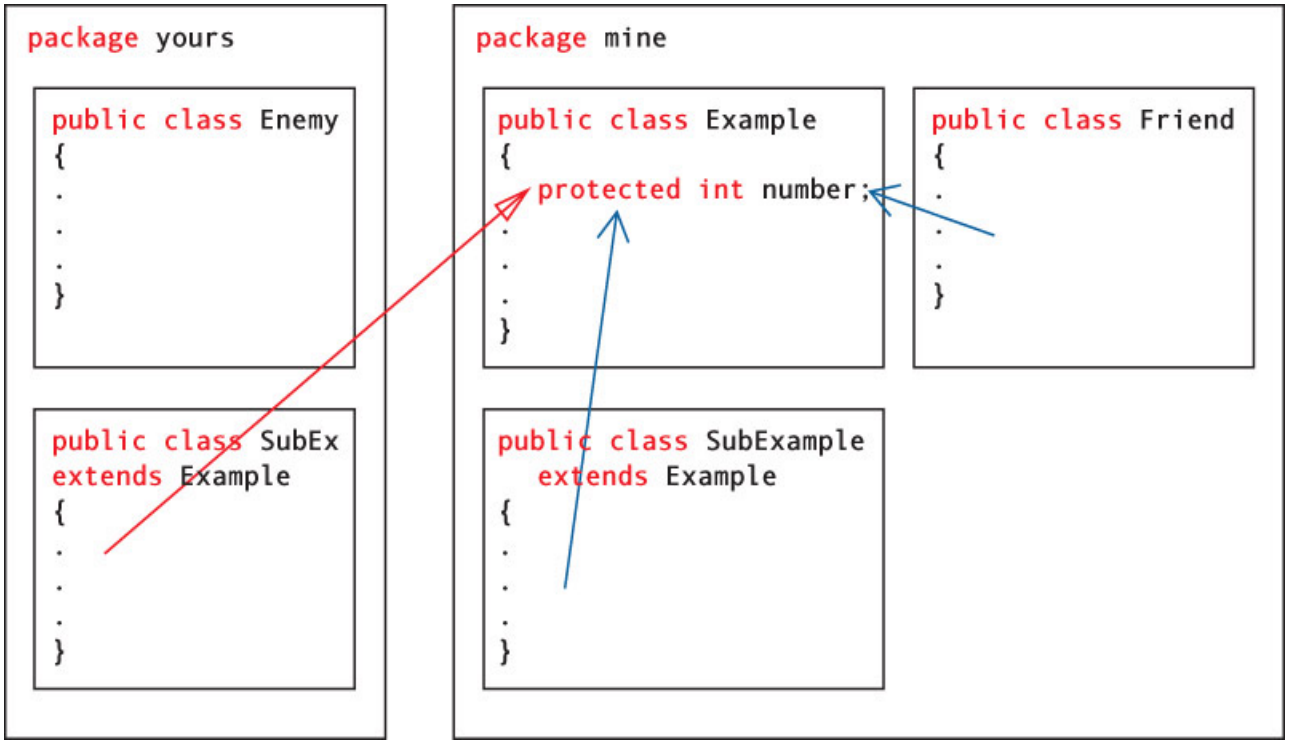


Phạm vi truy xuất (Modifier)



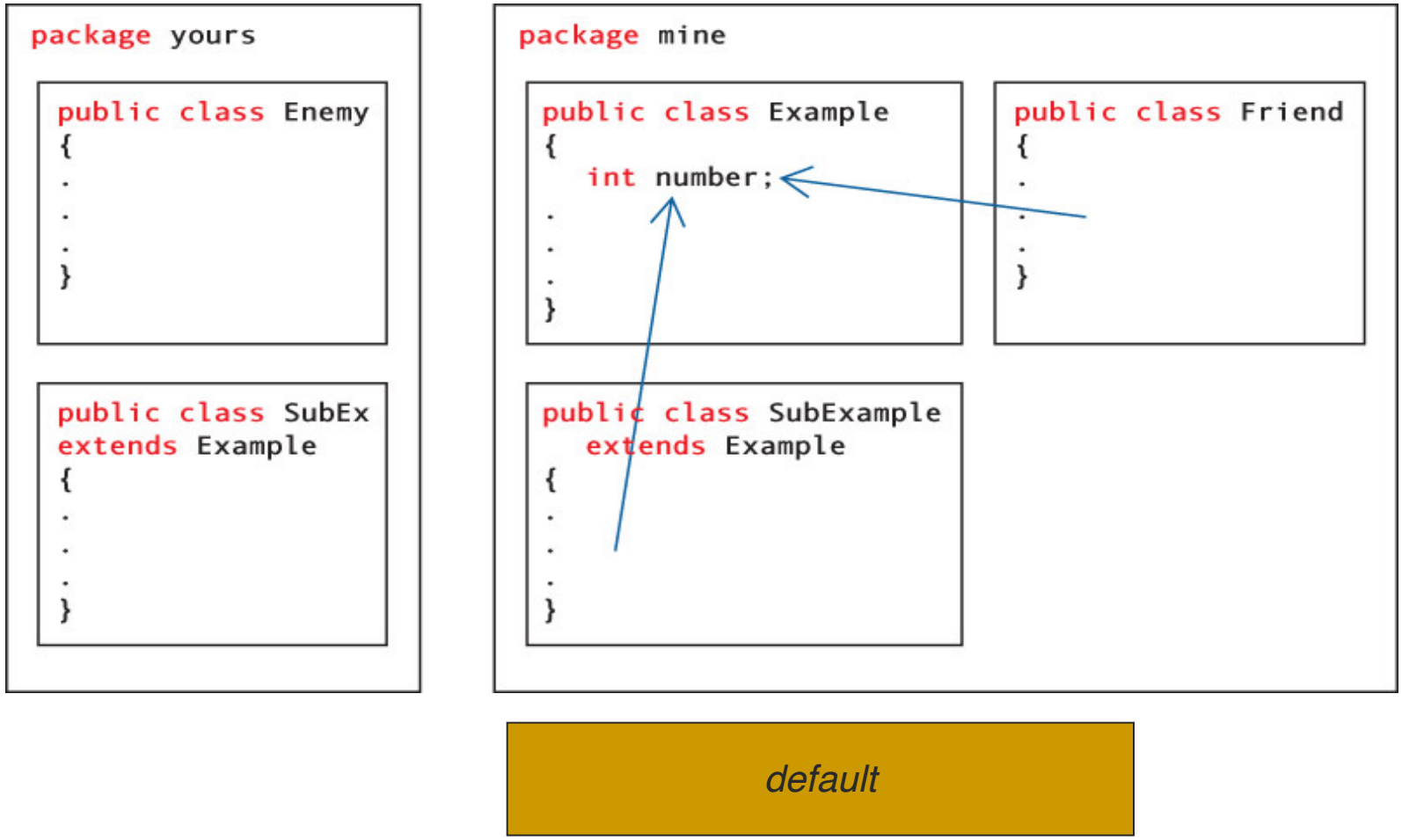
Public access

Phạm vi truy xuất (Modifier)

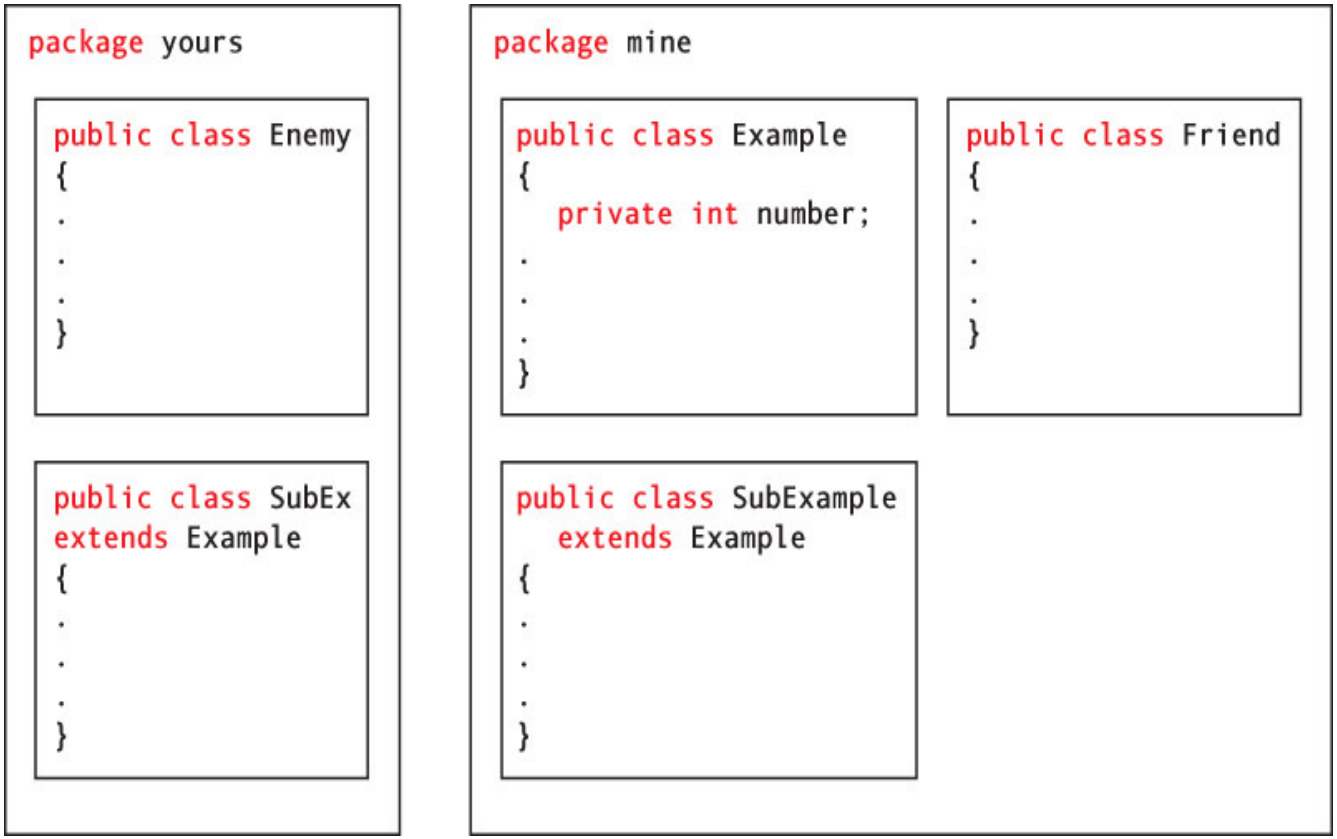


Protected assess

Phạm vi truy xuất (Modifier)



Phạm vi truy xuất (Modifier)



Private access

Phạm vi truy xuất (Modifier)

	private	default/package	***protected***	***public***
Same class	Yes	Yes	Yes	Yes
Same package		Yes	Yes	Yes
Different package (subclass)			Yes	Yes
Different package (non-subclass)				Yes

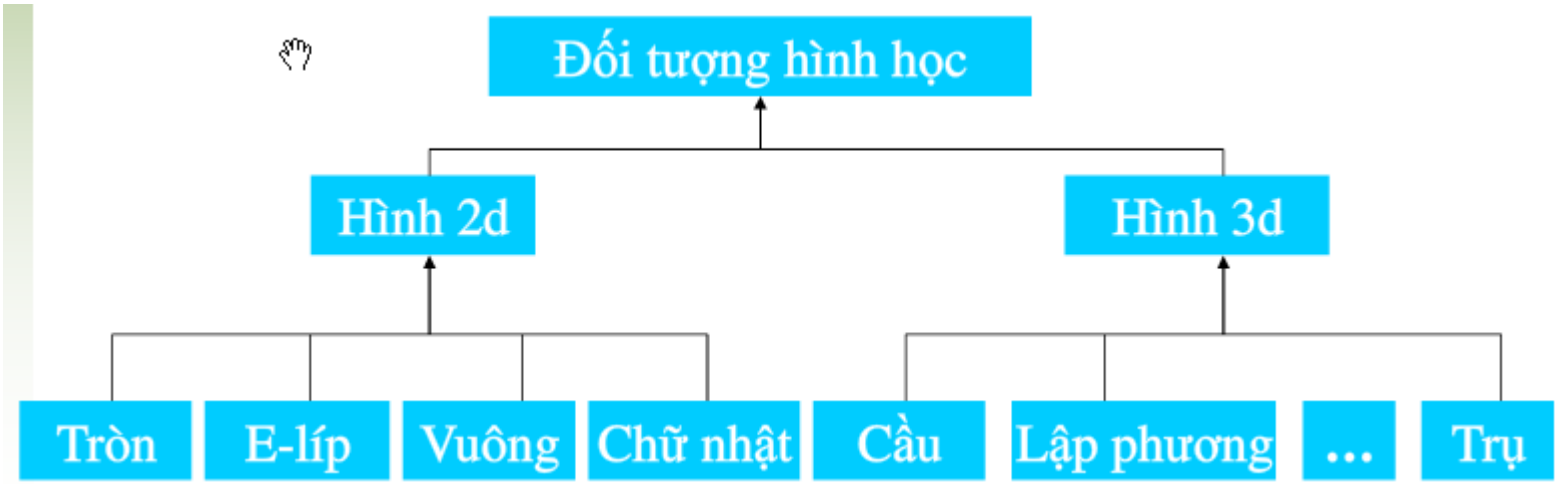
BAO ĐÓNG

- Khai báo private cho các thành phần cần che dấu
- Khai báo protected cho các thành phần (thừa kế và khác gói)

```
class Encapsulation {
    private int secret; //che dấu field
    public boolean setSecret(int secret) {//phương thức truy xuất gián tiếp
        if (secret < 1 || secret > 100) {
            return false;
        }
        this.secret = secret;
        return true;
    }
    public getSecret() { //phương thức truy xuất gián tiếp
        return secret;
    }
}
```

Thừa kế

- Thừa hưởng các thuộc tính và phương thức sẵn có
- Bổ xung, chi tiết hóa cho phù hợp với mục đích sử dụng mới
- Thuôc tính: thêm mới
- Phương thức:
  - Thêm mới
  - Hiệu chỉnh (Overiding)



Thừa kế

- Subclass: lớp dẫn xuất/lớp con
- SuperClass: lớp cơ sở/lớp cha
- Lớp con có thể kế thừa một hay tất cả các thành phần dữ liệu (thuộc tính), phương thức của lớp cha (public,default,protected)
- Dùng từ khóa extends

```
import java.awt.*;
 class Point {
            ----
}
class ColoredPoint extends Point {
            -------
}
```

Phương thức Overriding

- Là một phương thức ở class con định nghĩa lại từ phương thức có sẵn ở class cha.
- Có tên , kiểu trả về và các đối số giống với phương thức của lớp cha
- Nạp chồng là một kỷ thuật đa hình
- class không được nạp chồng các phương thức khai báo final
- Có phạm vi truy cập lớn hơn lớp cha

Từ khóa super

- Tham chiếu đến đối tượng là class cha của class hiện tại.
- super được dùng truy cập đến các thành viên của class cha đã bị che bởi class con và phương thức khởi dựng cùa class cha
- Sử dụng từ khóa *super* khi quan hệ tới thừa kế.
- Dùng viện dẫn các phương thức khởi dựng của cha từ khởi dựng của con.

```
class Person {
    String fName;
    String lName;
    Person(String fname, String lname)
      this.fName = fname;
      this.lName = lname;
    }
}
```

```
class Student extends Person {
    String studNum;
    Student(String fname, String lname, String sNum) {
      super(fname, lname);
      studNum = sNum;
    } }
```

Tính đa hình

```
class A{
    public void method(){
        System.out.println("method of A"); }
}

class B extends A {
     public void method(){
        System.out.println("method of B"); }
}

class C extends A {
    public void method(){
        System.out.println("method of C"); }
}
```

```
// Câu lệnh trong main
A a = new A();
a.method();

a = new B();
a.method();

C c = new C();
a = c;
a.method();

// Kết quả màn hình
method of A
method of B
method of C
```

Từ Khóa static

- Áp dụng tới các thành phần của class .
- Có thể xem như một khai báo toàn cục-> có thể được truy xuất bởi tất cả các thể hiện của class.
- Các phương thức static được sử dụng mà không cần tạo đối tượng của class
- Không tham khảo tới “this” hoặc “super”.
- Có thể áp dụng khai báo khối vô danh { } , và khối chỉ được thực thi một lần khi khối được nạp (khối thường dùng để khởi tạo các biến class).
- Thuộc tính và phương thức static được gọi là class variable, class method.
- ***Lưu ý: Trong thân phương thức static, không được phép truy xuất các thành phần không static***.

Lớp nội (inner class)

- Là lớp được khai báo trong một class khác

```
public class A {
        // ...
        int <field_1>
        class B {
                // ...
                int <field_2>
                public B(int par_1){
                        field_2 = par_1 + field_1;
                }
        }
}
```

Từ khóa final

- Được áp dụng khi khai báo biến, phương thức và class.
- Khai báo hằng số:
  - final int data = 10;
- Phương thức khai báo final không thể nạp chồng ở class con.
  - final void myMethod() {//trong class cha }
- Một class final không được thừa kế
  - final public class MyClass { }
- Phương thức không thay đổi giá trị đối số

```
public final class A
{
        …
}
```

Lớp trừu tượng (abstract)

- Lớp trừu tượng chỉ được dùng làm lớp cha cho các lớp khác, nó không có thể hiện
- Định nghĩa các thuộc tính chung cho các lớp con
- Có ít nhất một phương thức trừu tượng (không có phần thân): public abstract void draw();
- Khai báo class trừu tượng: public abstract class name{}
- Các class con phải cài đặt tất cả phương thức trừu tượng
- Không thể tạo đối tượng của class trừu tượng. Nhưng có thể khai báo biến thuộc class trừu tượng để tham chiếu đến class con của chúng.

Interface

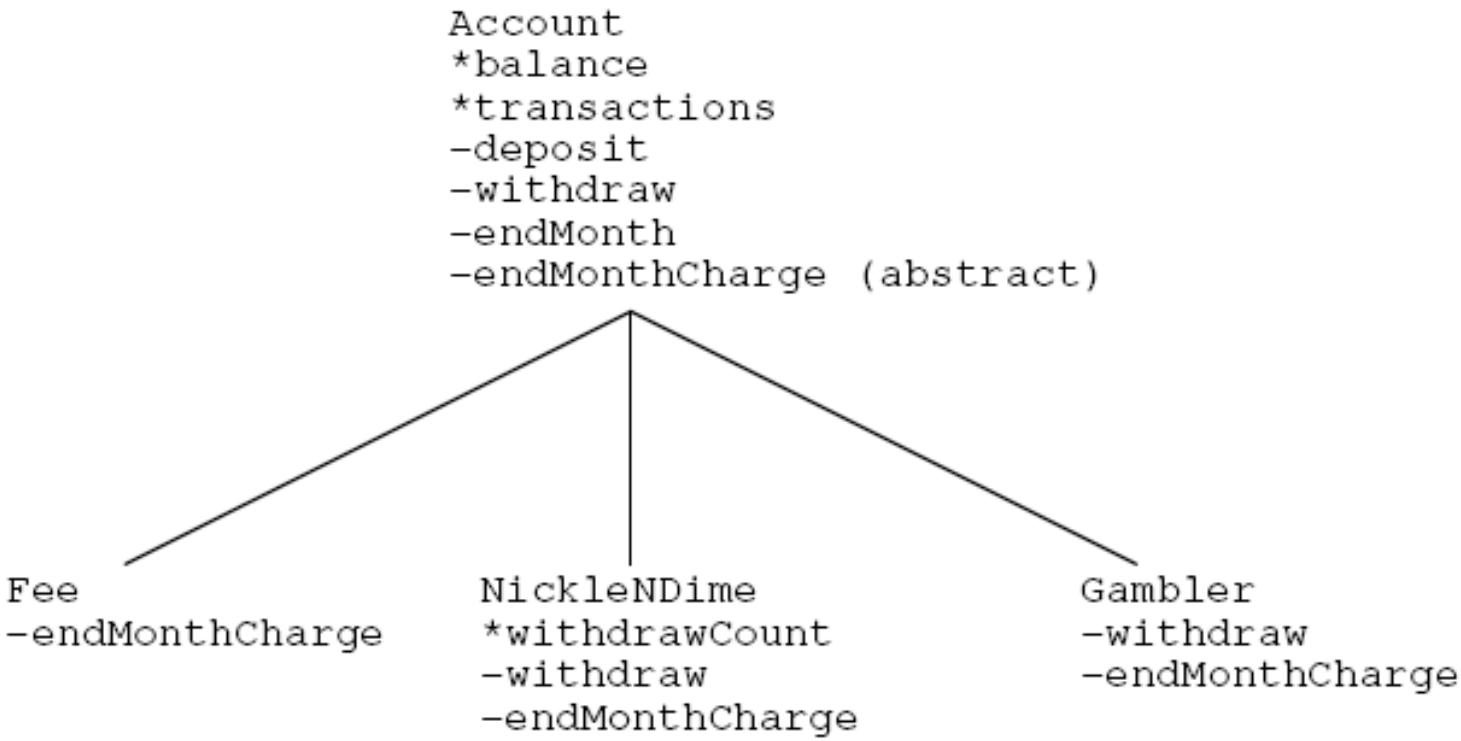
- Interface có 2 mục đích
  - Để tạo ra class cơ sở thuần ảo
  - Thực hiện hành vi kế tửa bội
- Dùng từ khóa interface để tạo ra interface
- Để triển khai interface dùng từ khóa implements
- Một class có thể triển khai nhiều interface
- Lưu ý:
  - Các thuộc tính trong interface là static và final
  - Mặc định các thành phần là public
  - Interface có thể thừa kế interface khác
  - Một class có thể cài đặt một hay nhiều interface nhưng chỉ có thể thừa kế 1
  - Class triển khai phải khai báo tất cả phương thức trong interface, nếu không phải khai báo là abstract
  - Interface không có thể hiện

Bài tập

- Xây dựng cadidate gồm các thuộc tính : mã, ngày sinh, điểm toán , văn, anh và các phương thức cần thiết
  - Xây dựng chương trình test :
    - Nhập n thí sinh (n do người dùng nhập) q
    - In thông tin thí sinh có tổng điểm > 15
- Xét phần mềm quản lý nhân sự. Công ty có 2 lọai nhân viên : văn phòng và sản xuất. Viết chương trình tính lương cho từng nhân viên
  - Thông tin nhân viên: họ tên, năm sinh, lương
  - Nhân viên sản xuất: lương=LCB + số sản phẩm*10000
  - Nhân viên văn phòng: lương=số ngày làm việc*10000

Bài tập: Account

- Cần lưu trữ thông tin cho tài khoản ngân hàng
- Giả sử chỉ cần lưu trữ số dư và tổng số lần giao dịch của tài khoản
- Tránh lập lại mã giữa 3 dạng tài khoản
- Một tài khoản cần đáp ứng các thông điệp sau
  - constructor(initialBalance)
  - deposit(amount)
  - withdraw(amount)
  - endMonth()
- Các dạng tài khoản
  - Normal: Cố định 50000 cuối mỗi tháng
  - Dime: 5000 cho mỗi lần withdraw
  - Gamber:
    - 4900 cuối mội tháng nếu không withdraw
    - 5100 cho hai lần withdraw


Account
*balance
*transactions
-deposit
-withdraw
-endMonth
-endMonthCharge (abstract)
Fee
-endMonthCharge
NickleNDime
*withdrawCount
-withdraw
-endMonthCharge
Gambler
-withdraw
-endMonthCharge

Bài tập

- Case Study 1: Giả sử chúng ta có một class human. Hãy cho biết sự liên quan của class human với các khái niệm bên dưới:
  - Properties
  - Behaviors
  - Encapsulation
  - Inheritance
  - Polymorphism
  - API
- Hãy nghĩ ra một đối tượng bất kì. Tìm ra sự liên quan của 6 khái niêm tới đối tượng đó.

Bài tập

- Case Study 2: Tạo một class Shapes có 2 thuộc tính chuỗi: name và size và các phương thức.
  - void printShapeInfo(): Xuất giá trị của name và size
  - void printShapeName() : Xuất name
  - void printShapeSize(): Xuất size
- Sử dụng thừa kế , tạo một class Square cùng thuộc tính và phương thức với class Shapes, ngoài ra Square có thêm 2 thuộc tính số nguyên: length và width và 2 phương thức
  - void printShapeLength(): xuất thuộc tính length
  - void printShapeWidth(): xuất thuộc tính width.
  - Nạp chồng printShapeInfo() để xuất thêm 2 thuộc tính length và width.

Bài tập

- Case study 3: Tạo một interface Animal có 2 phương thức : eat() và move(). Các phương thức không tham số và trị trả về. Tạo class Fish và Bear hiện thực Animal.
  - phương thức eat(): xuất ra cách ăn của ……
  - phương thức move(): xuất ra cách di chuyển của…..

Câu hỏi ôn

- Các đối tượng trong thế giới thực chứa ___ và ___.
- Một trạng thái của đối tượng được lưu trữ trong ___.
- Một hành vi của đối tượng được phơi bày thông qua ___.
- Các dữ liệu nội tại được che đậy từ thế giới bên ngòai và chỉ truy xuất qua phương thức được biết như dữ liệu ___.

Câu hỏi ôn

- Bản thiết kế cho một đối tượng phần mềm được gọi là ___.
- Hành vi chung, phổ biến có thể được định nghĩa trong ___ và được thừa kế vào ___
- Một tập phương thức với không hiện thực được gọi là ___.

Câu hỏi ôn

- Sự khác biệt chính của OOP so với lập trình cấu trúc?
- Một đối tượng trong thế giới thực bao gồm:
- Một đối tượng phần mềm bao gồm:
- Thuộc tính của đối tượng được gọi là:
- Phương thức đặc trưng của đối tượng được gọi là:

Câu hỏi ôn

- Trong lập trình biến class được xem là:
- Điều gì xảy ra khi không có đa hình trong thừa kế:
- Bạn nghĩ sao về phát biểu: Bao đóng hạn chế việc dùng lại và mở rộng hiện thực class
- API đem lại những thuận lợi gì?

Quan hệ giữa các đối tượng

- Có 3 quan hệ phổ biến nhất giữa các lớp là:
  - Dependence(“uses–a”)
  - Aggregation(“has–a”)
  - Inheritance(“is–a”)

UML

Table 4–1 UML Notation for Class Relationships

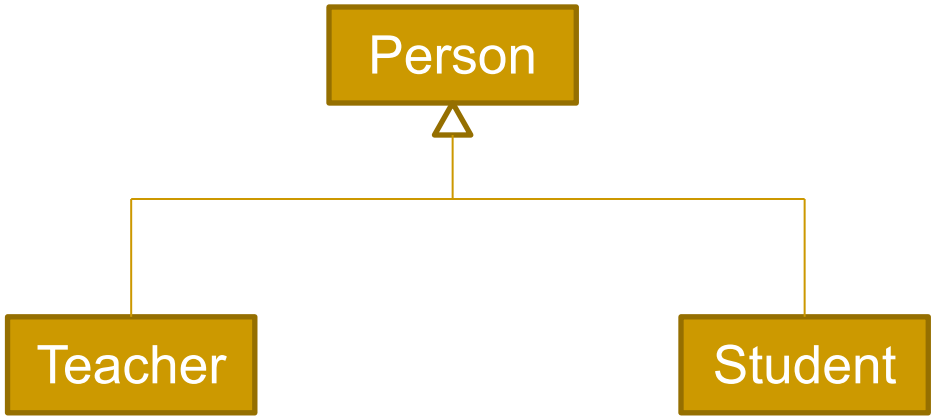
Relationship	UML Connector
Inheritance	
Interface inheritance	
Dependency	
Aggregation	
Association	
Directed association	

Dependence (“use-a”)

- Bên trong lớp Course có thuộc tính tham chiếu đến lớp Student.
- Lớp Course có sử dụng phương thức, thuộc tính tĩnh của lớp Student.
- Phương thức của lớp Course có tham số truyền vào là Student.
- Lớp Course sử dụng lớp Student như biến toàn cục.

→ Khi Course sử dụng Student thì ta nói Courser phụ thuộc vào Student.

Inheritance (“is-a”)

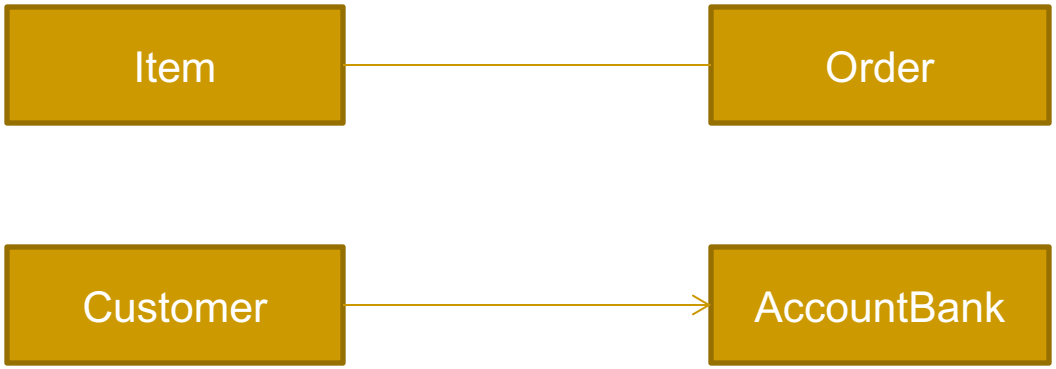



```
class Person { … }

class Teacher extends Person { … }

class Student extends Person { … }
```

Aggregation (has-a)


```
class Teacher { }

class Department {
        List<Teacher> teachers;
}
```

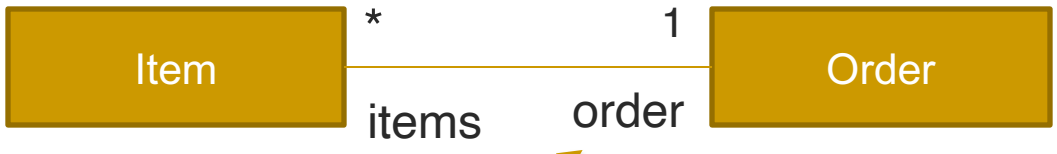
Association



Chỉ số mối kết hợp

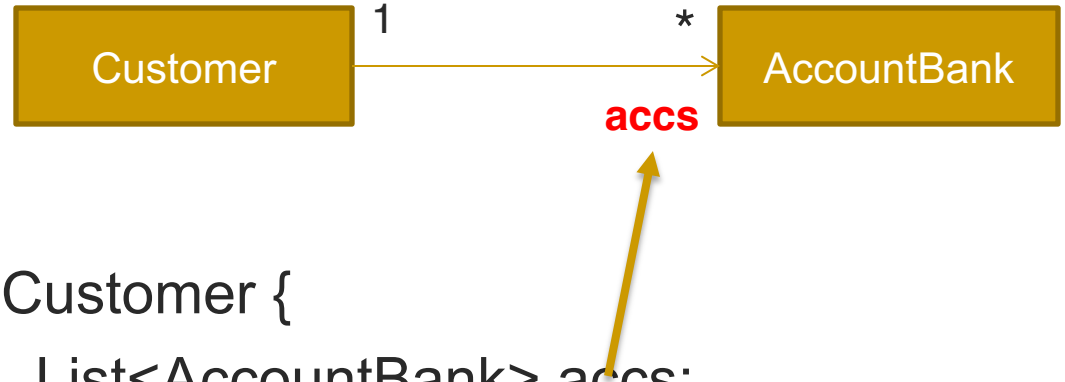
- 1 : chỉ 1
- * : không hoặc nhiều
- 2..4: từ 2 đến 4
- 2..*: từ 2 trở lên

Chỉ số mối kết hợp

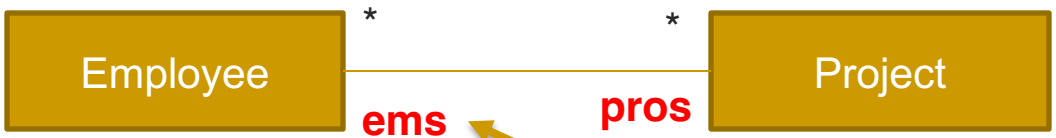



```
class Item {
        Order order;
}
```

```
class Order {
        List<Item> items;
}
```

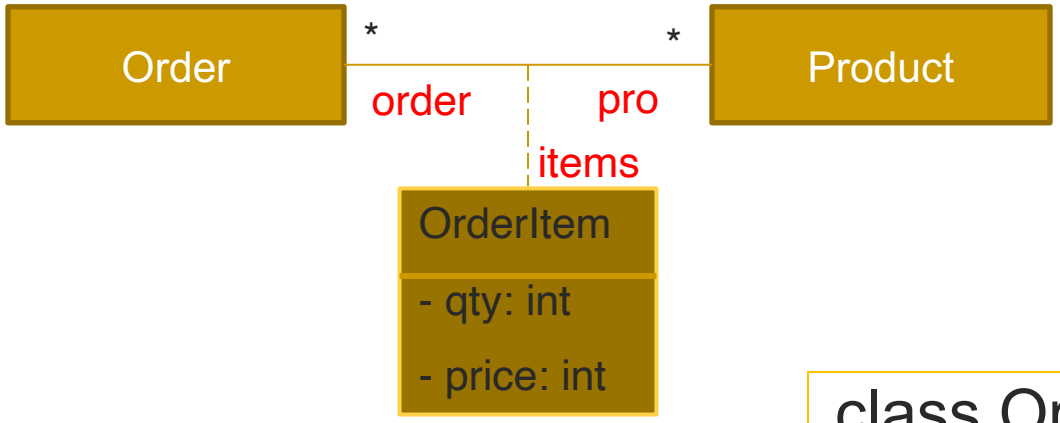



```
class Customer {
        List<AccountBank> accs;
}

class AccountBank {
}
```




```
class Employee {
        List<Project> pros;
}
```

```
class Project {
        List<Employee> ems;
}
```




```
class Order {
        List<OrderItem> items;
}
class Product {
        List<OrderItem> items;
}
```

```
class OrderItem {
        Order order;
        Product pro;
        int qty;
        int price;
}
```

Casestudy: Khai báo mô hình đối tượng