



**Bộ môn Công nghệ phần mềm**  
**VIỆN CÔNG NGHỆ THÔNG TIN TRUYỀN THÔNG**  
**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**



# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

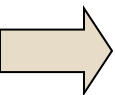
## Bài 05. Kết tập và kế thừa

Cao Tuấn Dũng  
[dungct@soict.hut.edu.vn](mailto:dungct@soict.hut.edu.vn)

# Mục tiêu bài học

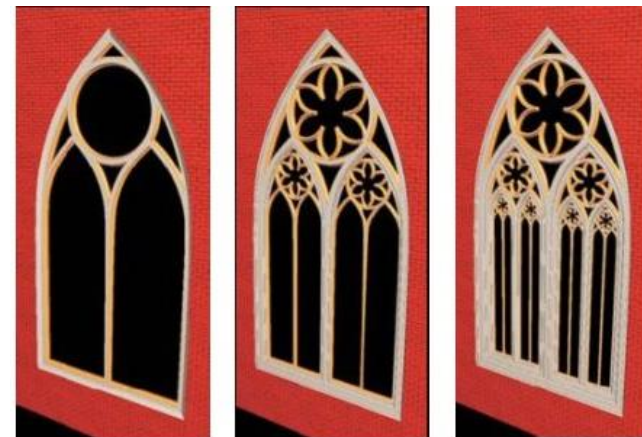
- Giải thích về khái niệm tái sử dụng mã nguồn
- Chỉ ra được bản chất, mô tả các khái niệm liên quan đến đến kết tập và kế thừa
- So sánh kết tập và kế thừa
- Biểu diễn được kết tập và kế thừa trên UML
- Giải thích nguyên lý kế thừa và thứ tự khởi tạo, hủy bỏ đối tượng trong kế thừa
- Áp dụng các kỹ thuật, nguyên lý về kết tập và kế thừa trên ngôn ngữ lập trình Java

# Nội dung

- 
1. Tái sử dụng mã nguồn
  2. Kết tập (Aggregation)
  3. Kế thừa (Inheritance)

# 1. Tái sử dụng mã nguồn (Re-usability)

- Tái sử dụng mã nguồn: Sử dụng lại các mã nguồn đã viết
  - Lập trình cấu trúc: Tái sử dụng hàm/chương trình con
  - OOP: Khi mô hình thế giới thực, tồn tại nhiều loại đối tượng có các thuộc tính và hành vi tương tự hoặc liên quan đến nhau  
→ *Làm thế nào để tái sử dụng lớp đã viết?*



# 1. Tái sử dụng mã nguồn (2)

- Các cách sử dụng lại lớp đã có:
  - *Sao chép* lớp cũ thành 1 lớp khác → Dư thừa và khó quản lý khi có thay đổi
  - Tạo ra lớp mới là sự *tập hợp* hoặc *sử dụng các đối tượng* của lớp cũ đã có → Kết tập (Aggregation)
  - Tạo ra lớp mới trên cơ sở *phát triển* từ lớp cũ đã có → Kế thừa (Inheritance)

# 1. Tái sử dụng mã nguồn (2)

- Ưu điểm
  - Giảm thiểu công sức, chi phí.
  - Nâng cao chất lượng phần mềm
  - Nâng cao khả năng mô hình hóa thế giới thực
  - Nâng cao khả năng bảo trì (maintainability)

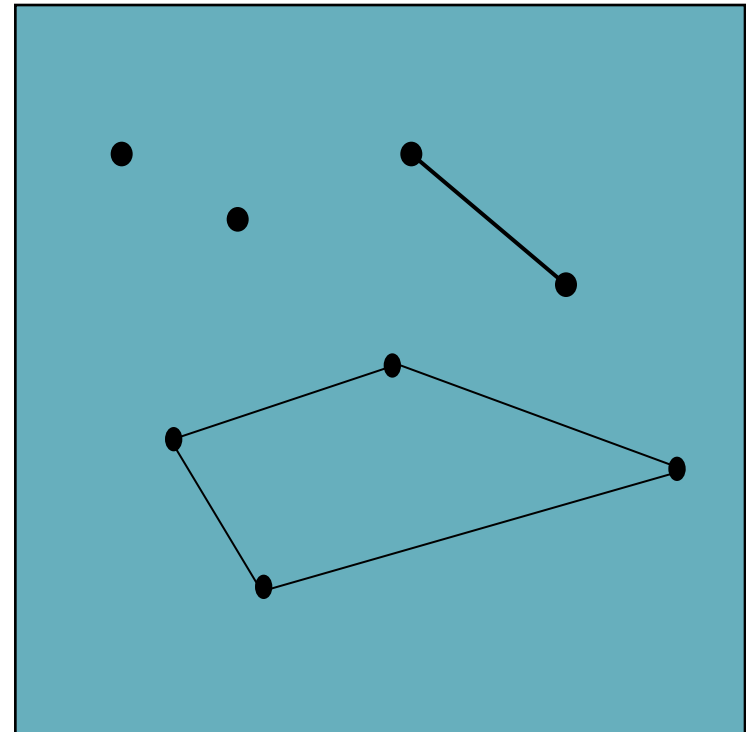


# Nội dung

1. Tái sử dụng mã nguồn
2. Kết tập (Aggregation)
3. Kế thừa (Inheritance)

## 2. Kết tập

- Ví dụ:
  - Điểm
    - Tứ giác gồm 4 điểm  
→ Kết tập
- Kết tập
  - Quan hệ chứa/có (“has-a”) hoặc là một phần (is-a-part-of)





## 2.1. Bản chất của kết tập

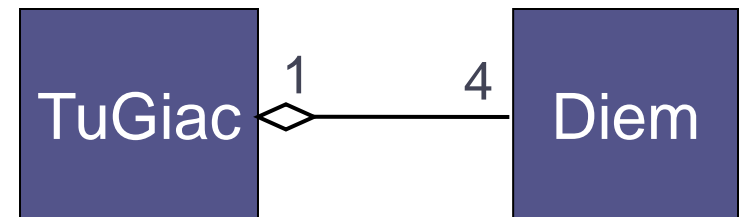
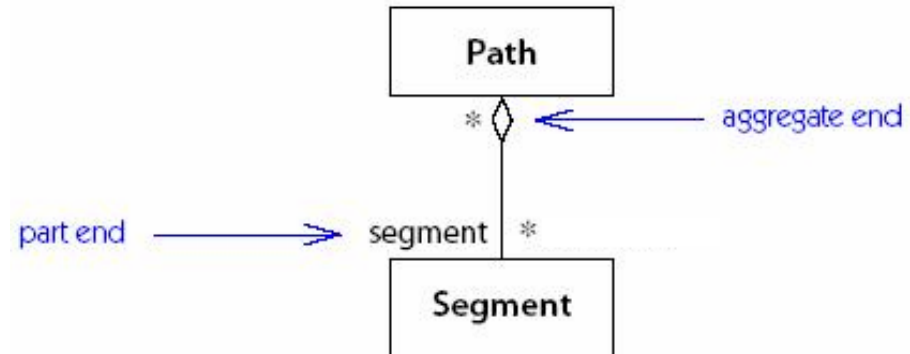
- Kết tập (aggregate)
  - Các thành phần của lớp mới là các đối tượng của các lớp có sẵn.
  - Kết tập tái sử dụng thông qua *đối tượng*
- Lớp mới
  - Lớp toàn thể (Aggregate/Whole),
- Lớp cũ
  - Lớp thành phần (Part).

## 2.1. Bản chất của kết tập (2)

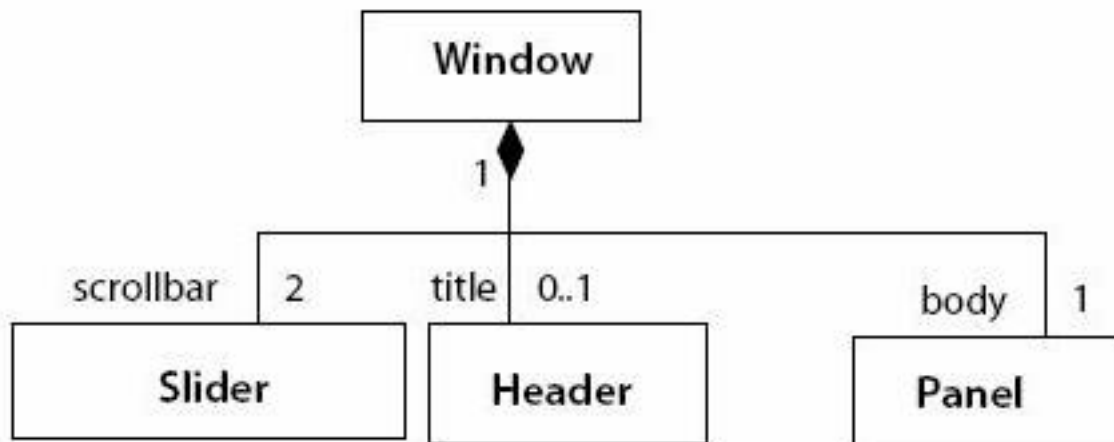
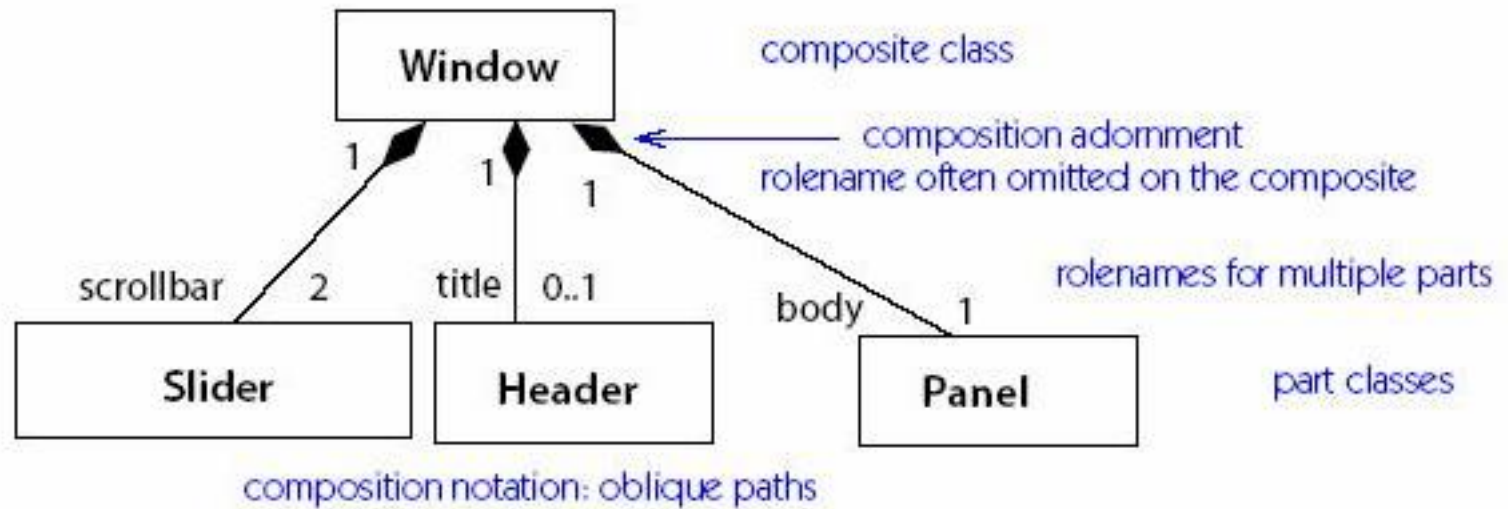
- Lớp toàn thể chứa đối tượng của lớp thành phần
  - Là một phần (is-a-part of) của lớp toàn thể
  - Tái sử dụng các thành phần dữ liệu và các hành vi của lớp thành phần thông qua đối tượng thành phần

## 2.2. Biểu diễn kết tập bằng UML

- Sử dụng “hình thoi” tại đầu của lớp toàn thể
- Sử dụng bội số quan hệ (multiplicity) tại 2 đầu
  - 1 số nguyên dương: 1, 2,...
  - Dải số (0..1, 2..4)
  - \*: Bất kỳ số nào
  - Không có: Mặc định là 1



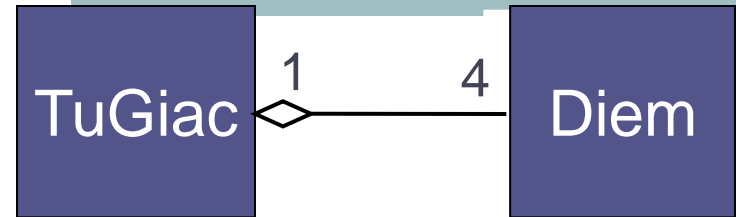
# Ví dụ



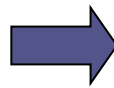
## 2.3. Minh họa trên Java

```
class Diem {  
    private int x, y;  
    public Diem() {}  
    public Diem(int x, int y) {  
        this.x = x; this.y = y;  
    }  
    public void setX(int x) { this.x = x; }  
    public int getX() { return x; }  
    public void hienThiDiem() {  
        System.out.print("(" + x + ", "  
                           + y + ")");  
    }  
}
```

```
class TuGiac {  
    private Diem d1, d2;  
    private Diem d3, d4;  
    public TuGiac(Diem p1, Diem p2,  
                  Diem p3, Diem p4) {  
        d1 = p1; d2 = p2; d3 = p3; d4 = p4;  
    }  
    public TuGiac() {  
        d1 = new Diem();      d2 = new Diem(0,1);  
        d3 = new Diem (1,1);  d4 = new Diem (1,0);  
    }  
    public void printTuGiac() {  
        d1.printDiem();  d2.printDiem();  
        d3.printDiem();  d4.printDiem();  
        System.out.println();  
    }  
}
```



```
public class Test {  
    public static void main(String arg[])  
    {  
        Diem d1 = new Diem(2,3);  
        Diem d2 = new Diem(4,1);  
        Diem d3 = new Diem (5,1);  
        Diem d4 = new Diem (8,4);  
  
        TuGiac tg1 = new TuGiac(d1, d2, d3, d4);  
        TuGiac tg2 = new TuGiac();  
        tg1.printTuGiac();  
        tg2.printTuGiac();  
    }  
}
```

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The window has standard Windows window controls (minimize, maximize, close) on the right. The command prompt shows the output of the Java program: two lines of coordinates separated by angle brackets, followed by a prompt to press any key to continue. The first line is "<2, 3><4, 1><5, 1><8, 4>" and the second line is "<0, 0><0, 1><1, 1><1, 0>". The prompt "Press any key to continue . . ." is on the third line. The background of the command prompt is black, and the text is white. There is a scroll bar on the right side of the window.

```
C:\WINDOWS\system32\cmd.exe  
<2, 3><4, 1><5, 1><8, 4>  
<0, 0><0, 1><1, 1><1, 0>  
Press any key to continue . . .
```

## Ví dụ 2

```
class Person {  
    private String name;  
    private Date birthday;  
    public String getName() { return name; }  
    ...  
}
```

```
class Employee {  
    private Person me;  
    private double salary;  
    public String getName() {  
        return me.getName();  
    }  
    ...  
}
```

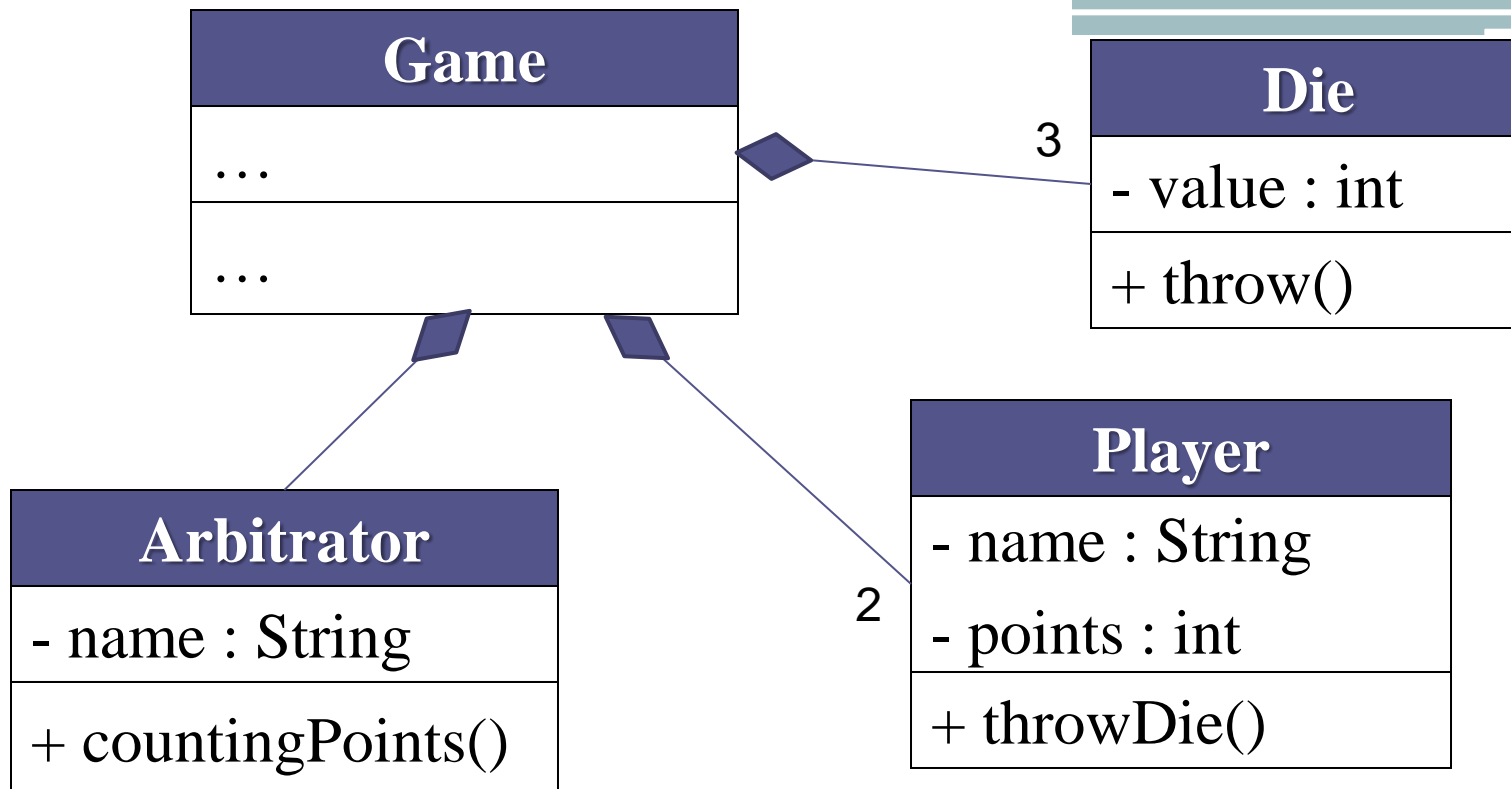


## Ví dụ 2

```
class Manager {  
    private Employee me;  
    private Employee assistant;  
    public setAssistant(Employee e) {...}  
    ...  
}  
...  
Manager junior = new Manager();  
Manager senior = new Manager();  
senior.setAssistant(junior); //error
```

# Một ví dụ về Kết tập

- Một trò chơi gồm 2 đối thủ, 3 quân súc sắc và 1 trọng tài.
  - Cần 4 lớp:
    - Người chơi (Player)
    - Súc sắc (Die)
    - Trọng tài (Arbitrator)
    - Trò chơi (Game)
- Lớp Trò chơi là lớp kết tập của 3 lớp còn lại



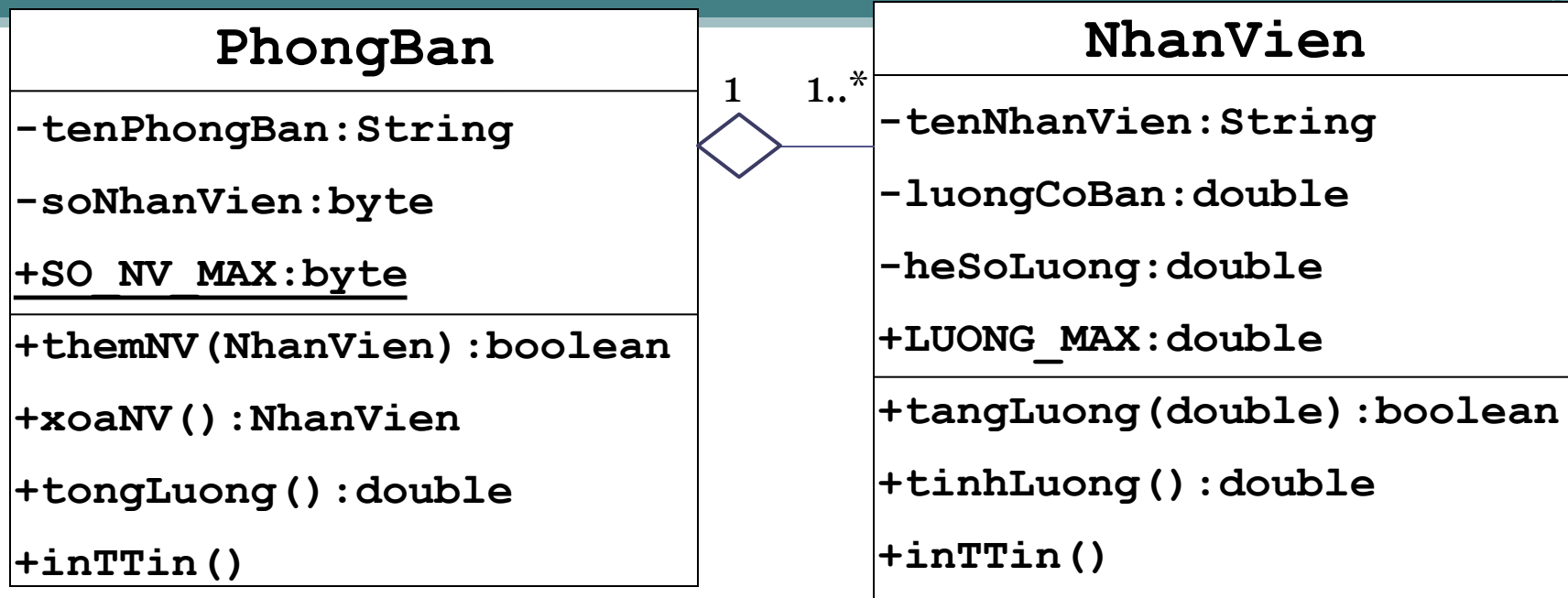
```

class Game
{
    Die die1, die2, die3;
    Player player1, player2;
    Arbitrator arbitrator1;
    ...
}
  
```

# Thứ tự khởi tạo trong kết tập

- Khi một đối tượng được tạo mới, các thuộc tính của đối tượng đó đều phải được khởi tạo và gán những giá trị tương ứng.
- Các đối tượng thành phần được khởi tạo trước  
→ Các phương thức khởi tạo của các lớp của các đối tượng thành phần được thực hiện trước

# Bài tập



- Viết mã nguồn cho lớp **PhongBan** với các thuộc tính và phương thức như biểu đồ trên cùng phương thức khởi tạo với số lượng tham số cần thiết, biết rằng:
  - Giá trị của dữ liệu hằng tĩnh **SO\_NV\_MAX = 100**
  - Việc thêm/xóa nhân viên được thực hiện theo cơ chế của stack
  - tongLuong()** trả về tổng lương của các nhân viên trong phòng.
  - inTTin()** hiển thị thông tin của phòng và thông tin của các nhân viên trong phòng.

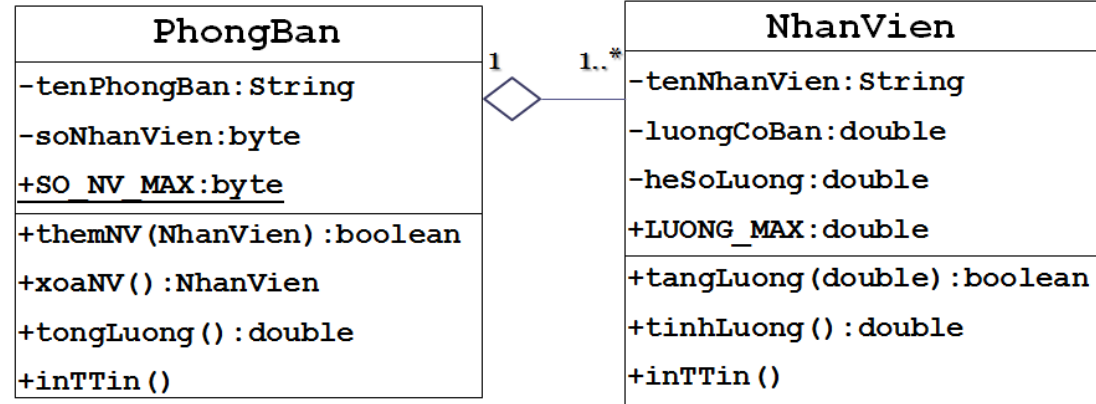
```
public class PhongBan {
    private String tenPhongBan; private byte soNhanVien;
    public static final SO_NV_MAX = 100;
    private NhanVien[] dsnv;
    public boolean themNhanVien(NhanVien nv) {
        if (soNhanVien < SO_NV_MAX) {
            dsnv[soNhanVien] = nv; soNhanVien++;
            return true;
        } else return false;
    }
    public boolean xoaNhanVien(NhanVien nv) {
        if (soNhanVien > 0) {
            NhanVien tmp = dsnv[soNhanVien-1];
            dsnv[soNhanVien-1] = null; soNhanVien--;
            return dsnv[soNhanVien];
        } else return null;
    }
    // (cont) ...
}
```

```
public PhongBan(String tenPB){
    dsnv = new NhanVien[SO_NV_MAX];
    tenPhongBan = tenPB; soNhanVien = 0;
}
public double tongLuong(){
    double tong = 0.0;
    for (int i=0;i<soNhanVien;i++)
        tong += dsnv[i].tinhLuong();
    return tong;
}
public void inTTin(){
    System.out.println("Ten phong: "+tenPhong);
    System.out.println("So NV: "+soNhanVien);
    System.out.println("Thong tin cac NV");
    for (int i=0;i<soNhanVien;i++)
        dsnv[i].inTTin();
}
}
```

# Thảo luận

## Trong ví dụ trên

- Lớp cũ? Lớp mới?
  - Lớp cũ: NhanVien
  - Lớp mới: PhongBan
- Lớp mới tái sử dụng lớp cũ thông qua?
  - Mảng đối tượng của lớp NhanVien: dsnv
- Lớp mới tái sử dụng được những gì của lớp cũ?
  - tinhLuong() trong phương thức tongLuong()
  - inTTin() trong phương thức inTTin()



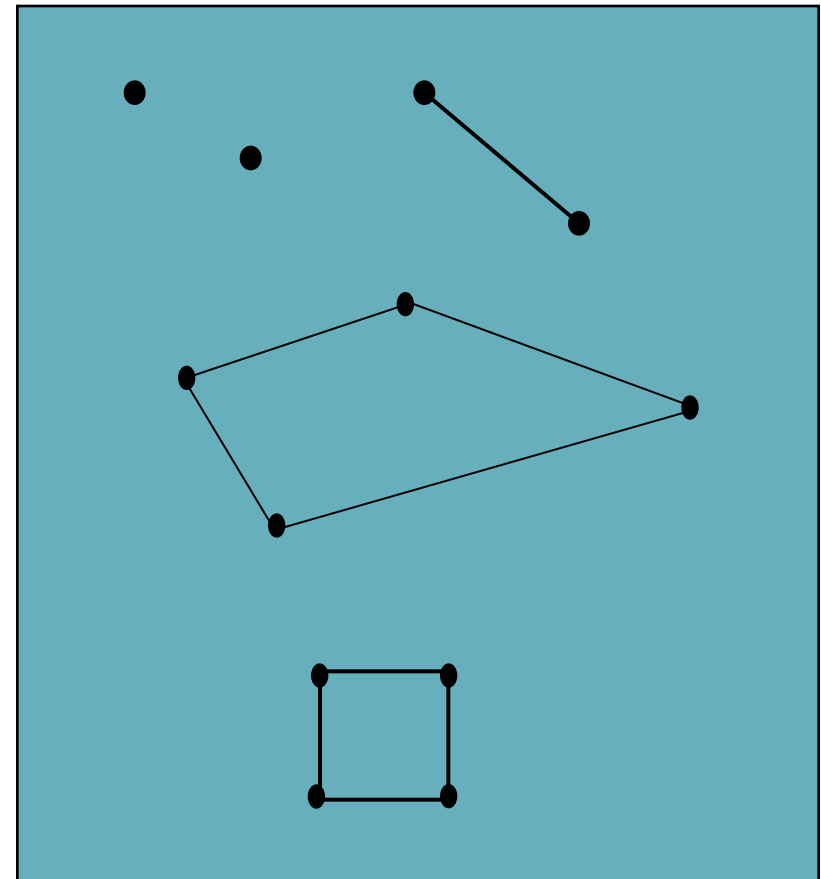


# Nội dung

1. Tái sử dụng mã nguồn
2. Kết tập (Aggregation)
3. Kế thừa (Inheritance)

### 3. Tổng quan về kế thừa

- Ví dụ:
  - Điểm
    - Tứ giác gồm 4 điểm  
→ Kết tập
  - Tứ giác
    - Hình vuông  
→ Kế thừa



## 3.1.1. Bản chất kế thừa

- Kế thừa (Inherit, Derive)
  - Tạo lớp mới bằng cách phát triển lớp đã có.
  - Lớp mới kế thừa những gì đã có trong lớp cũ và phát triển những tính năng mới.
- Lớp cũ:
  - Lớp cha (parent, superclass), lớp cơ sở (base class)
- Lớp mới:
  - Lớp con (child, subclass), lớp dẫn xuất (derived class)

# Kế thừa

- Nguyên lý mô tả một lớp trên cơ sở mở rộng/cụ thể hơn một lớp đã tồn tại, hay nhiều lớp (trong trường hợp đa thừa kế)
- Trên cách nhìn mô đun hóa: Nếu B thừa kế A, mọi dịch vụ của A sẽ sẵn có trong B (theo các cách thực hiện khác nhau)
- Trên cách nhìn xuất phát từ kiểu: Nếu B thừa kế A, bất cứ khi nào một thể hiện của A được yêu cầu, thể hiện của B có thể là một đáp ứng.

# Kế thừa

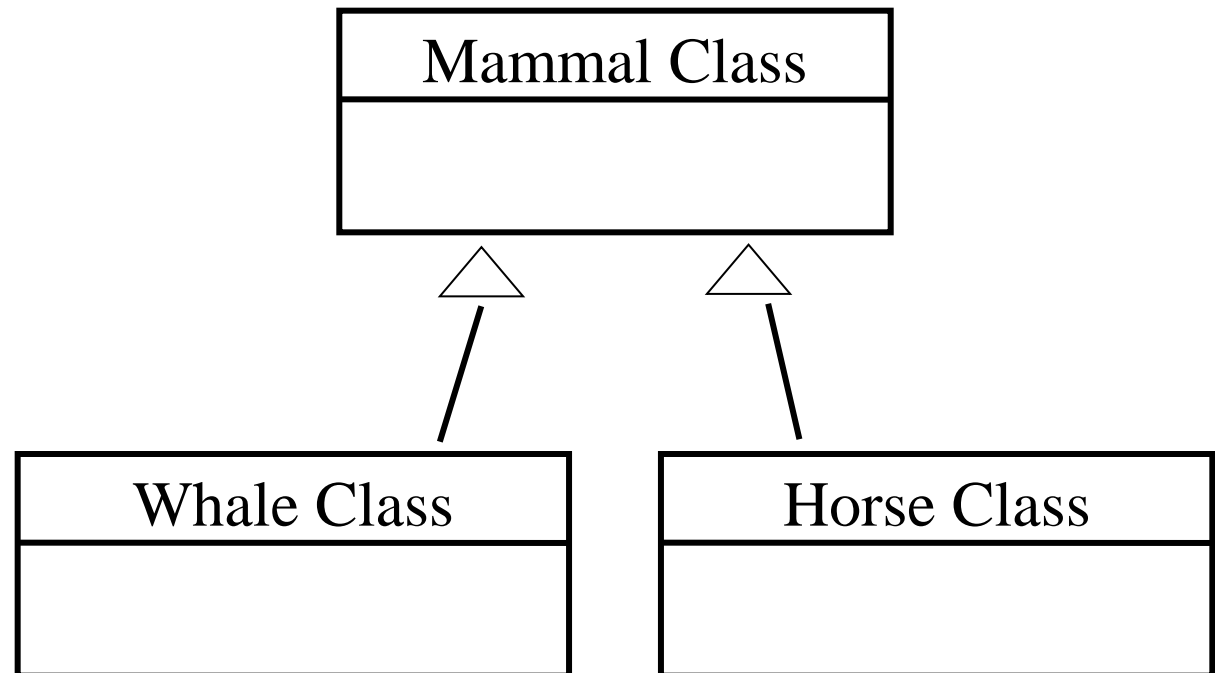
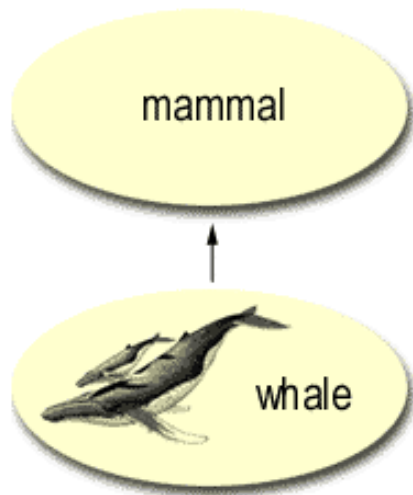
- *Inheritance* xác định 1 quan hệ (relationship ) giữa các lớp khi 1 lớp chia sẻ cấu trúc và/hoặc hành vi của 1 hay nhiều lớp khác
- 1 cây phả hệ bởi các lớp được tạo ra trong đó 1 lớp con - *subclass* kế thừa từ 1 hay nhiều lớp cha - *superclasses*
- Kế thừa còn được gọi là quan hệ là : *is-a* .

## 3.1.1. Bản chất kế thừa (2)

- Lớp con
  - Là một loại (is-a-kind-of) của lớp cha
  - Tái sử dụng bằng cách kế thừa các thành phần dữ liệu và các hành vi của lớp cha
  - Chi tiết hóa cho phù hợp với mục đích sử dụng mới
    - Extension: Thêm các thuộc tính/hành vi mới
    - Redefinition (Method Overriding): Chỉnh sửa lại các hành vi kế thừa từ lớp cha

# Tính tương đồng

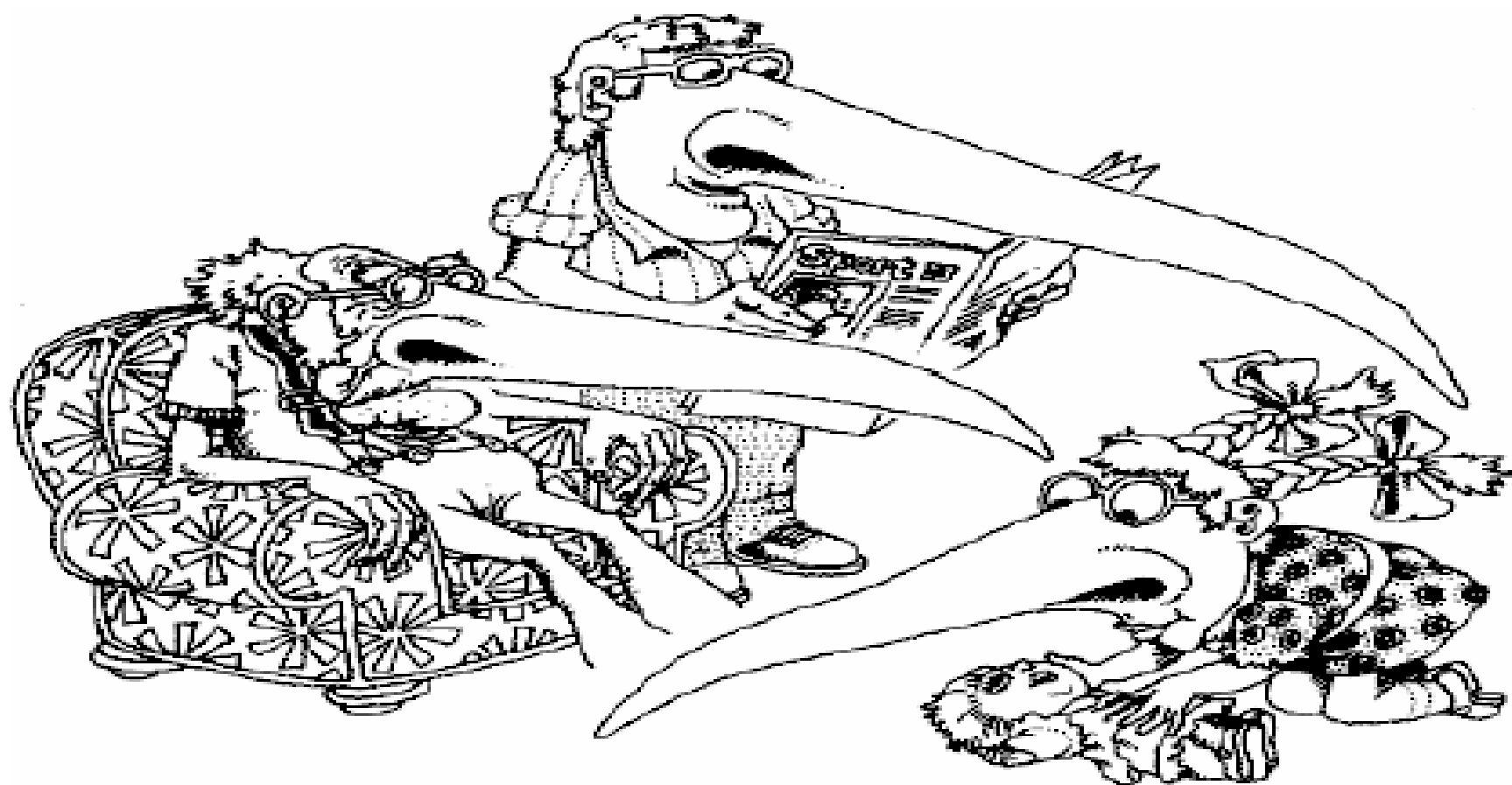
- Lớp cá voi kế thừa từ lớp động vật có vú.
- 1 con cá voi là 1 đv có vú ( *is-a* mammal )
- Lớp cá voi là *subclass*, lớp DVCV là *superclass*



# Kế thừa

- Cả Whale và Horse có quan hệ *is-a* với mammal class
- Cả Whale và Horse có 1 số hành vi thông thường của Mammal
- Inheritance là chìa khóa để tái sử dụng code – Nếu 1 lớp cha đã được tạo, thì lớp con có thể được tạo và thêm vào một số thông tin





## 3.1.2. Kết tập và kế thừa

- So sánh kết tập và kế thừa?
  - Giống nhau
    - Điều là kỹ thuật trong OOP để tái sử dụng mã nguồn
  - Khác nhau?

# Phân biệt kế thừa và kết tập

## Kế thừa

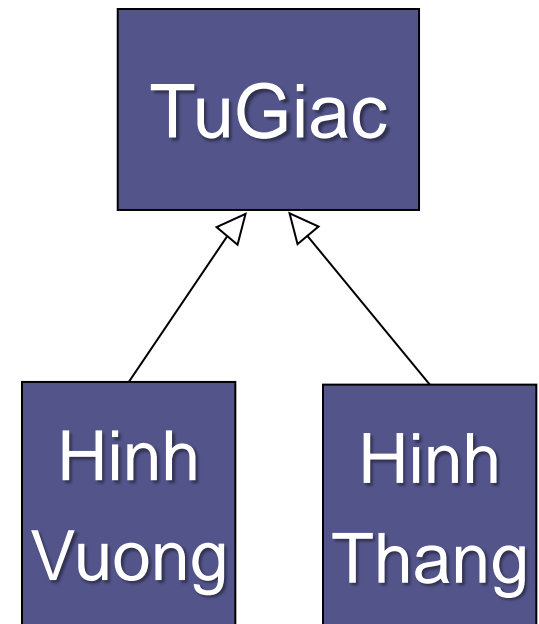
- Kế thừa **tái sử dụng** thông qua **lớp**.
  - Tạo lớp mới bằng cách phát triển lớp đã có sẵn
- Quan hệ “**là một loại**” (“is a kind of”)
- Ví dụ: Ô tô là một loại phương tiện vận tải

## Kết tập

- Kết tập **tái sử dụng** thông qua **đối tượng**.
  - Tạo ra tham chiếu đến các đối tượng của các lớp có sẵn trong lớp mới
- Quan hệ “**là một phần**” (“is a part of”)
- Ví dụ: Ô tô có 4 bánh xe

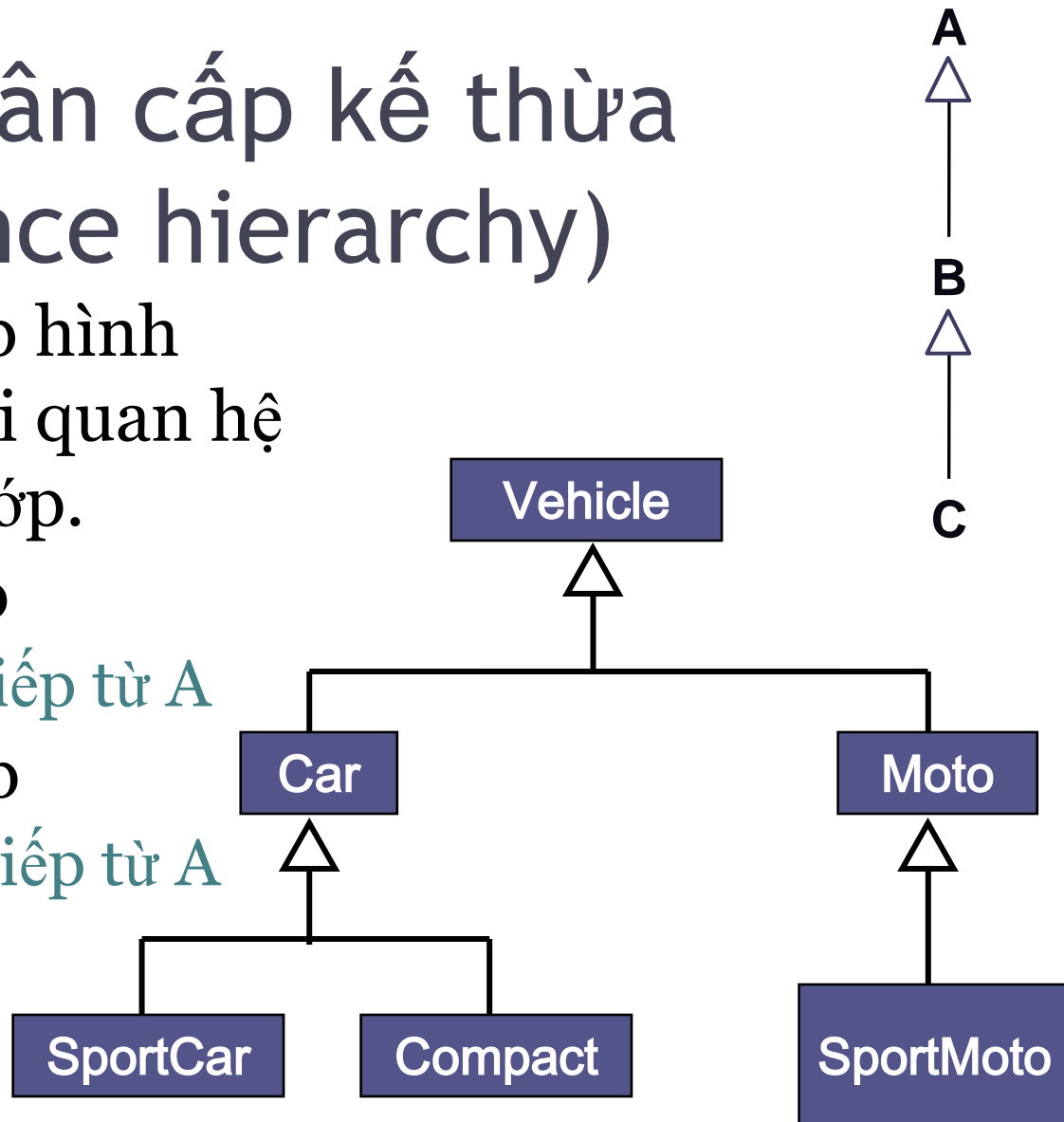
### 3.1.3. Biểu diễn kế thừa trong UML

- Sử dụng “tam giác rỗng” tại đầu Lóp cha



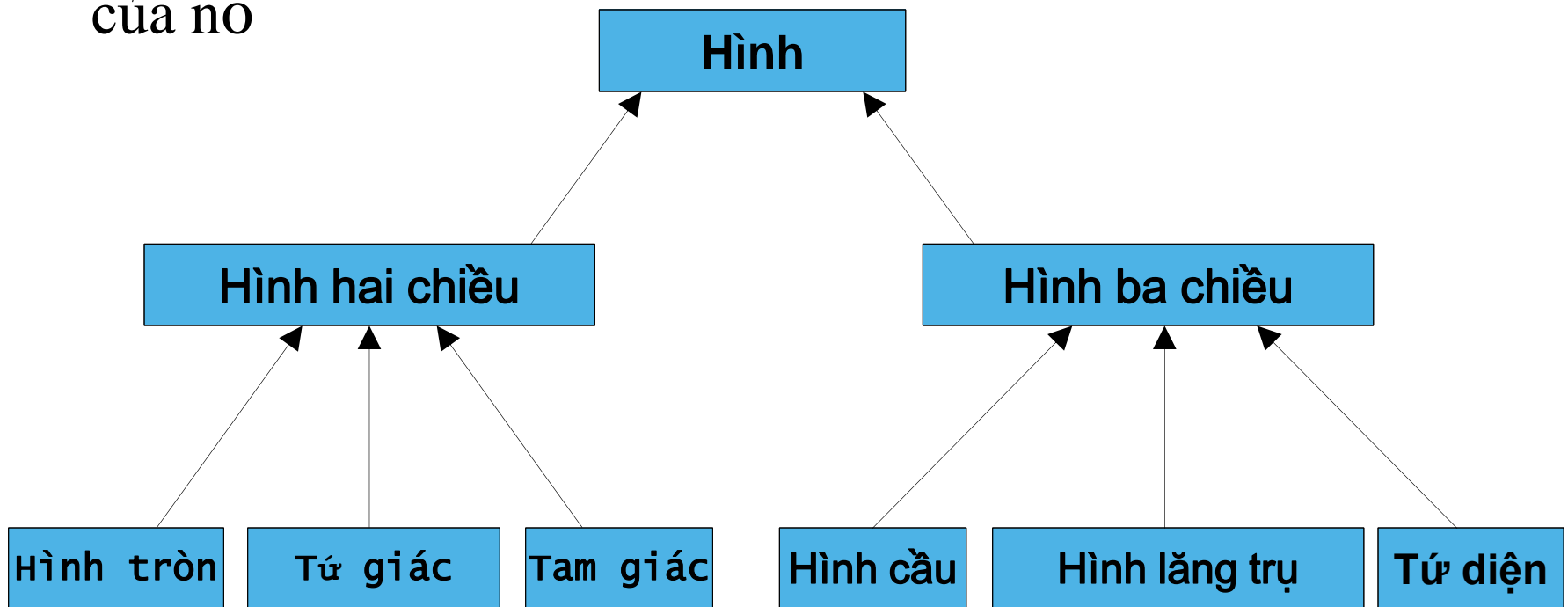
### 3.1.4. Cây phân cấp kế thừa (Inheritance hierarchy)

- Cấu trúc phân cấp hình cây, biểu diễn mối quan hệ kế thừa giữa các lớp.
- Dẫn xuất trực tiếp
  - B dẫn xuất trực tiếp từ A
- Dẫn xuất gián tiếp
  - C dẫn xuất gián tiếp từ A



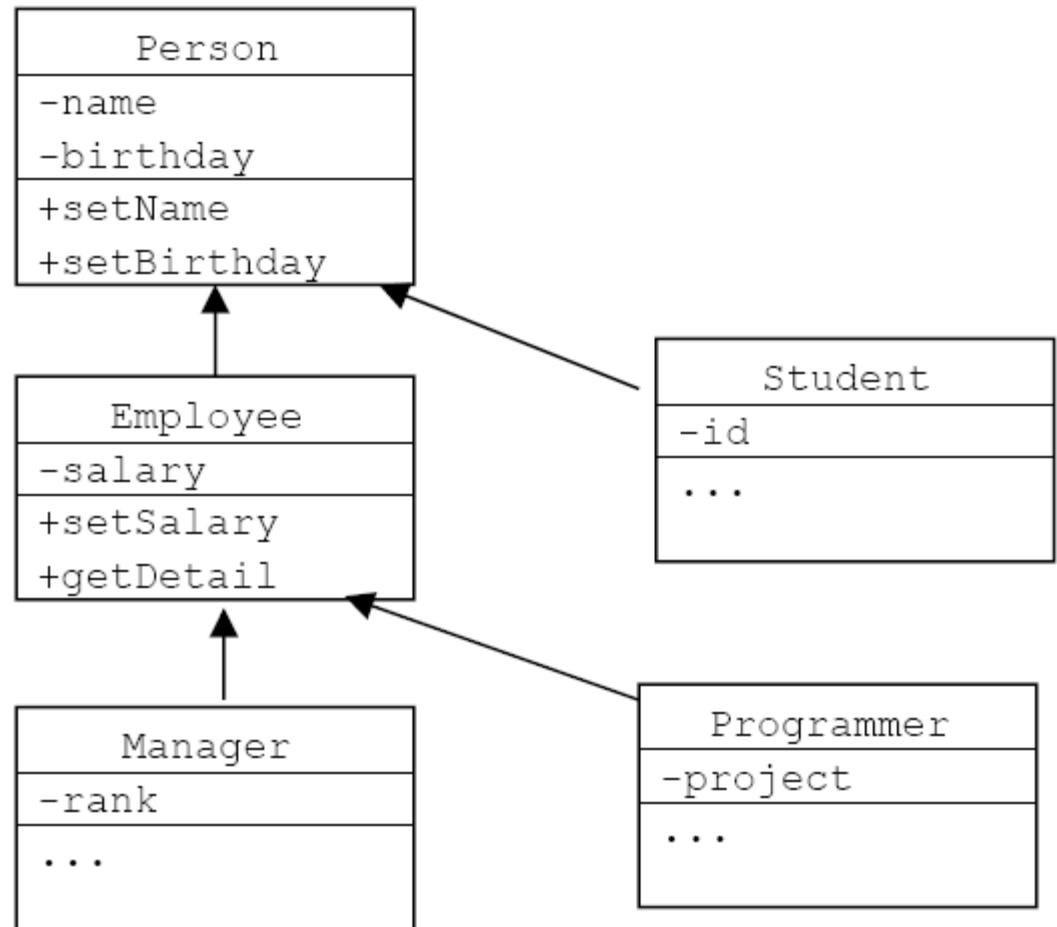
## 3.1.4. Cây phân cấp kế thừa (2)

- Lớp con có cùng lớp cha gọi là siblings (anh chị em)
- Thành viên được kế thừa sẽ được kế thừa xuống dưới trong cây phân cấp → Lớp con kế tất cả các lớp tổ tiên của nó



## 3.1.4. Cây phân cấp kế thừa (2)

Mọi đối tượng  
đều kế thừa từ  
lớp gốc Object



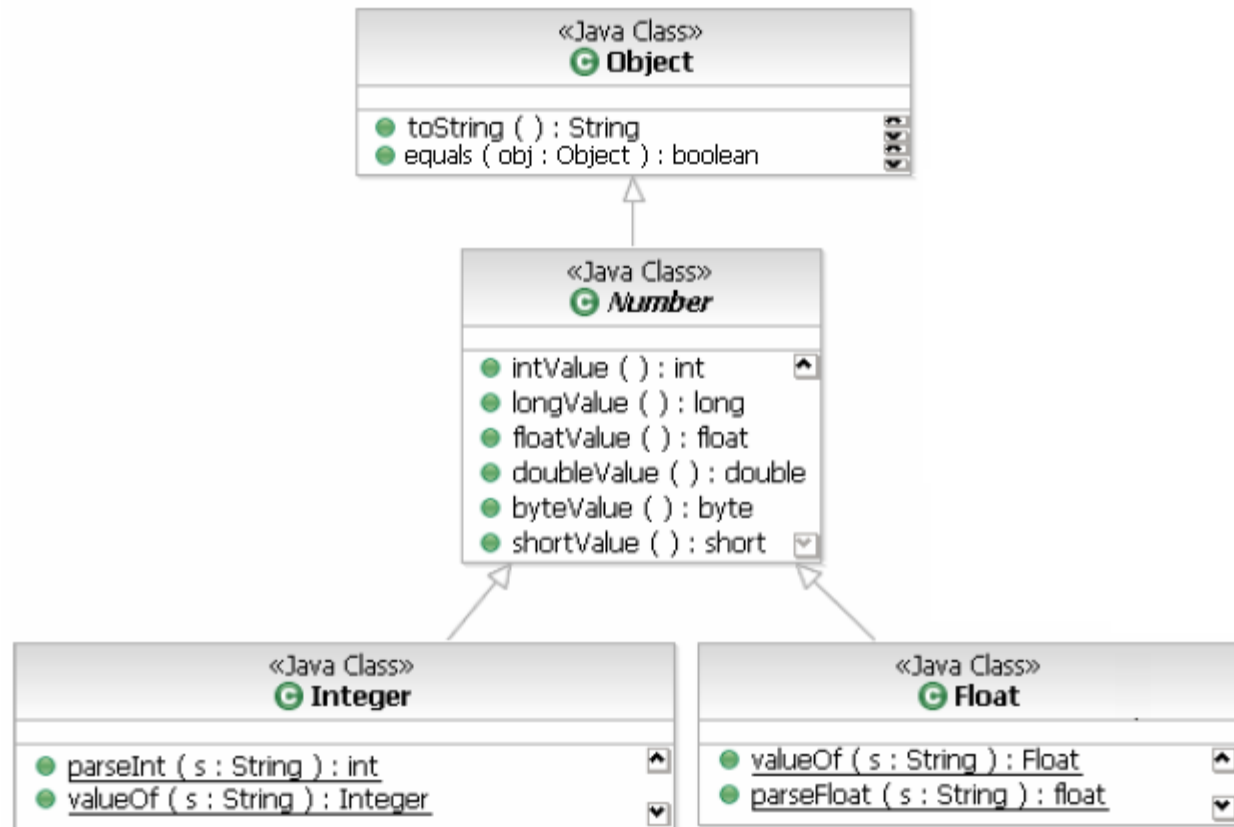
# Lớp Object

- Lớp có tên `Object` định nghĩa trong package chuẩn `java.lang`
- Nếu một lớp không được định nghĩa là lớp con của một lớp khác thì mặc định nó là lớp con trực tiếp của lớp `Object`.
  - Lớp `Object` là lớp gốc trên cùng của tất cả các cây phân cấp kế thừa



# Lớp Object (2)

- Chứa một số phương thức hữu ích kế thừa lại cho tất cả các lớp, ví dụ: `toString()`, `equals()`...



## 3.2. Nguyên lý kế thừa

- Chỉ định truy cập protected
- Thành viên protected trong lớp cha được truy cập trong:
  - Các thành viên lớp cha
  - Các thành viên lớp con
  - Các thành viên các lớp cùng thuộc 1 package với lớp cha
- Lớp con có thể kế thừa được gì?
  - Kế thừa được các thành viên được khai báo là public và protected của lớp cha.
  - Không kế thừa được các thành viên private.

## 3.2. Nguyên lý kế thừa (2)

	<b>public</b>	<b>Không có</b>	<b>protected</b>	<b>private</b>
Cùng lớp				
Lớp con cùng gói				
Lớp con khác gói				
Khác gói, non-inher				

## 3.2. Nguyên lý kế thừa (2)

	<b>public</b>	<b>Không có</b>	<b>protected</b>	<b>private</b>
Cùng lớp	Yes	Yes	Yes	Yes
Lớp con cùng gói	Yes	Yes	Yes	No
Lớp con khác gói	Yes	No	Yes	No
Khác gói, non-inher	Yes	No	No	No

## 3.2. Nguyên lý kế thừa (3)

- Các phương thức không được phép kế thừa:
  - Các phương thức khởi tạo và hủy
    - Làm nhiệm vụ khởi đầu và gỡ bỏ các đối tượng
    - Chúng chỉ biết cách làm việc với từng lớp cụ thể
  - Toán tử gán =
    - Làm nhiệm vụ giống như phương thức khởi tạo

## 3.3. Cú pháp kế thừa trên Java

- Cú pháp kế thừa trên Java:
  - `<Lớp con> extends <Lớp cha>`
- Ví dụ:

```
class HìnhVuong extends TuGiac {  
    ...  
}  
class Bird extends Animal {  
    ...  
}
```

# Ví dụ 1.1

```
public class TuGiac {
    protected Diem d1, d2, d3, d4;
    public void setD1(Diem _d1) {d1=_d1;}
    public Diem getD1(){return d1;}
    public void printTuGiac(){...}
    ...
}
```

Sử dụng các thuộc tính  
protected của lớp cha  
trong lớp con

```
public class HinhVuong extends TuGiac {
    public HinhVuong(){
        d1 = new Diem(0,0); d2 = new Diem(0,1);
        d3 = new Diem(1,0); d4 = new Diem(1,1);
    }
}

public class Test{
    public static void main(String args[]){
        HinhVuong hv = new HinhVuong();
        hv.printTuGiac();
    }
}
```

Gọi phương thức public  
lớp cha của đối tượng lớp con

## Ví dụ 1.2

```
public class TuGiac {
    protected Diem d1, d2, d3, d4;
    public void printTuGiac(){...}
    public TuGiac(){...}
    public TuGiac(Diem d1, Diem d2,
                  Diem d3, Diem d4) { ...}
}

public class HinhVuong extends TuGiac {
    public HinhVuong(){ super(); }
    public HinhVuong(Diem d1, Diem d2,
                    Diem d3, Diem d4){
        super(d1, d2, d3, d4);
    }
}

public class Test{
    public static void main(String args[]){
        HinhVuong hv = new HinhVuong();
        hv.printTuGiac();
    }
}
```

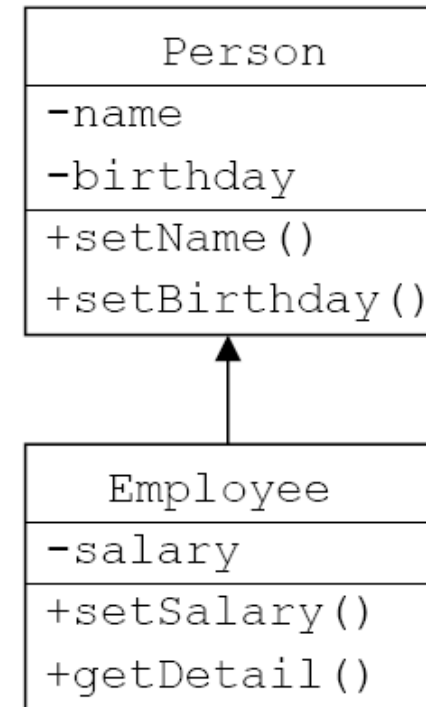


# Ví dụ 2

protected

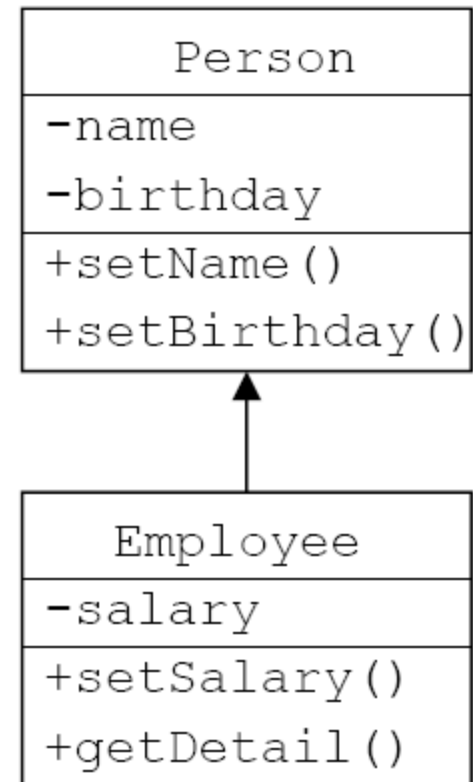
```
class Person {
    private String name;
    private Date birthday;
    public String getName() {return name;}
    ...
}

class Employee extends Person {
    private double salary;
    public boolean setSalary(double sal) {
        salary = sal;
        return true;
    }
    public String getDetail(){
        String s = name+", "+birthday+", "+salary; //Loi
    }
}
```



## Ví dụ 2 (tiếp)

```
public class Test{  
    public static void main(String args[]){  
        Employee e = new Employee();  
        e.setName("John");  
        e.setSalary(3.0);  
    }  
}
```



## Ví dụ 3 - Cùng gói

```
public class Person {  
    Date birthday;  
    String name;  
    ...  
}  
public class Employee extends Person {  
    ...  
    public String getDetail() {  
        String s;  
        String s = name + "," + birthday;  
        s += "," + salary;  
        return s;  
    }  
}
```

## Ví dụ 3 - Khác gói

```
package abc;
public class Person {
    protected Date birthday;
    protected String name;
    ...
}

import abc.Person;
public class Employee extends Person {
    ...
    public String getDetail() {
        String s;
        s = name + "," + birthday + "," + salary;
        return s;
    }
}
```

# Khởi tạo và huỷ bỏ đối tượng trong kế thừa

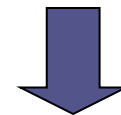
- Khởi tạo đối tượng:
  - Lớp cha được khởi tạo trước lớp con.
  - Các phương thức khởi tạo của lớp con luôn gọi phương thức khởi tạo của lớp cha ở câu lệnh đầu tiên
    - Tự động gọi (không tường minh - implicit): Khi lớp cha CÓ phương thức khởi tạo mặc định
    - Gọi trực tiếp (tường minh - explicit)
- Huỷ bỏ đối tượng:
  - Ngược lại so với khởi tạo đối tượng

### 3.4.1. Tự động gọi constructor của lớp cha

```
public class TuGiac {
    protected Diem d1, d2;
    protected Diem d3, d4;
    public TuGiac() {
        System.out.println
            ("Lop cha TuGiac()");
    }
    //...
}

public class HinhVuong
    extends TuGiac {
    public HinhVuong() {
        //Tu dong goi TuGiac()
        System.out.println
            ("Lop con HinhVuong()");
    }
}
```

```
public class Test {
    public static void
        main(String arg[])
    {
        HinhVuong hv =
            new HinhVuong();
    }
}
```



```
C:\WINDOWS\system32\cmd...
Lop cha TuGiac()
Lop con HinhVuong()
Press any key to continue . . .
```

## 3.4.2. Gọi trực tiếp constructor của lớp cha

- Câu lệnh đầu tiên trong phương thức khởi tạo của lớp con có thể gọi phương thức khởi tạo của lớp cha
  - **super(Danh\_sach\_tham\_so) ;**
  - Điều này là bắt buộc nếu lớp cha không có phương thức khởi tạo mặc định
    - Đã viết phương thức khởi tạo của lớp cha với một số tham số
    - Phương thức khởi tạo của lớp con không bắt buộc phải có tham số.

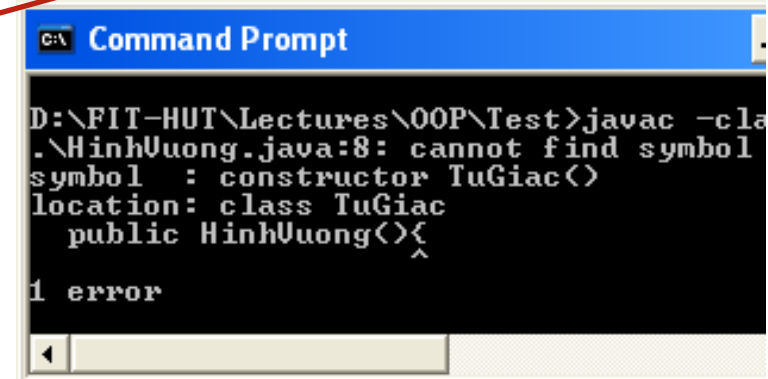
# Ví dụ

```
public class TuGiac {
    protected Diem d1, d2;
    protected Diem d3, d4;
    public TuGiac(Diem d1,
        Diem d2, Diem d3, Diem d4){
        System.out.println("Lop cha
            TuGiac(d1, d2, d3, d4)");
        this.d1 = d1; this.d2 = d2;
        this.d3 = d3; this.d4 = d4;
    }
}

public class HìnhVuong extends
    TuGiac {
    public HìnhVuong() {
        System.out.println
            ("Lop con HìnhVuong()");
    }
}
```

```
public class Test {
    public static
    void main(String
        arg[])
    {
        HìnhVuong hv =
        new
        HìnhVuong();
    }
}
```

**Lỗi** ↓



```
C:\ Command Prompt

D:\FIT-HUT\Lectures\OOP\Test>javac -cla
.\HìnhVuong.java:8: cannot find symbol
symbol : constructor TuGiac()
location: class TuGiac
    public HìnhVuong()<^
1 error
```



# Gọi trực tiếp constructor của lớp cha

## Phương thức khởi tạo lớp con **KHÔNG** tham số

```
public class TuGiac {
    protected Diem d1,d2,d3,d4;
    public TuGiac(Diem d1, Diem d2,
        Diem d3, Diem d4){
        System.out.println("Lop cha
            TuGiac(d1, d2, d3, d4)");
        this.d1 = d1; this.d2 = d2;
        this.d3 = d3; this.d4 = d4;
    }
}
```

```
public class HinhVuong extends TuGiac {
    public HinhVuong() {
        super(new Diem(0,0), new Diem(0,1),
            new Diem(1,1),new Diem(1,0));
        System.out.println("Lop con HinhVuong()");
    }
}
```

. . .

```
HinhVuong hv = new
    HinhVuong();
```



```
C:\WINDOWS\system32\cmd.exe
Lop cha TuGiac(d1, d2, d3, d4)
Lop con HinhVuong()
Press any key to continue . . .
```

# Gọi trực tiếp constructor của lớp cha

## Phương thức khởi tạo lớp con **CÓ** tham số

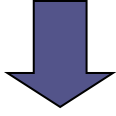
```

public class TuGiac {
    protected Diem d1,d2,d3,d4;
    public TuGiac(Diem d1,
Diem d2, Diem d3, Diem d4) {
        System.out.println
            ("Lop cha TuGiac(d1,d2,d3,d4)");
        this.d1 = d1; this.d2 = d2;
        this.d3 = d3; this.d4 = d4;
    }
}

public class HinhVuong extends TuGiac {
    public HinhVuong(Diem d1, Diem d2,
Diem d3, Diem d4) {
        super(d1, d2, d3, d4);
        System.out.println("Lop con
            HinhVuong(d1,d2,d3,d4)");
    }
}

```

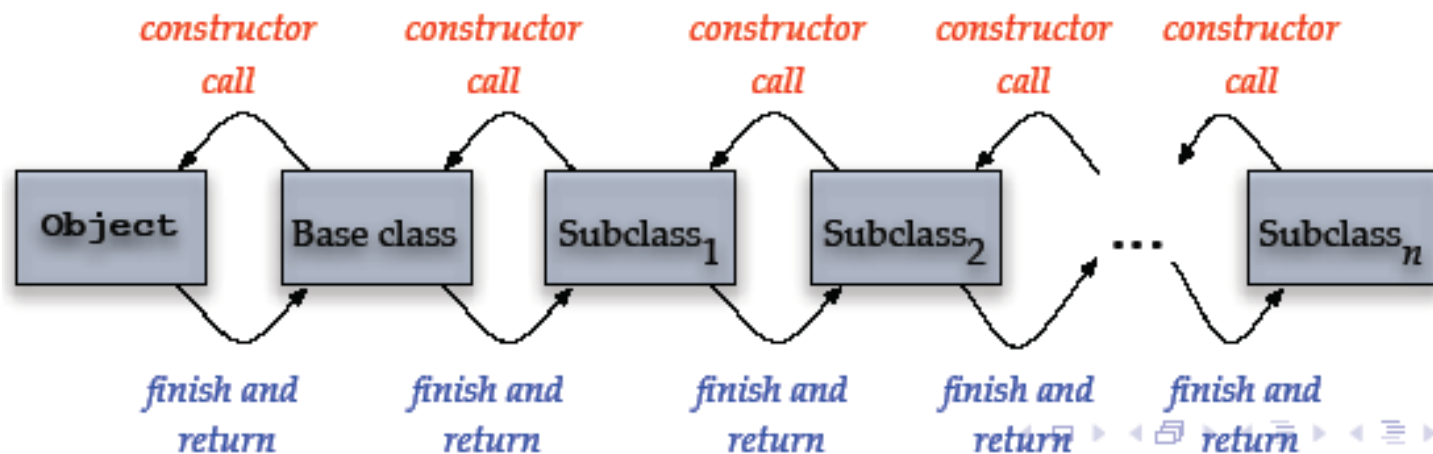
. . .  
HinhVuong hv =  
new HinhVuong(  
new Diem(0,0) ,  
new Diem(0,1) ,  
new Diem(1,1) ,  
new Diem(1,0) );



Lop cha TuGiac(d1, d2, d3, d4)  
 Lop con HinhVuong(d1, d2, d3, d4)

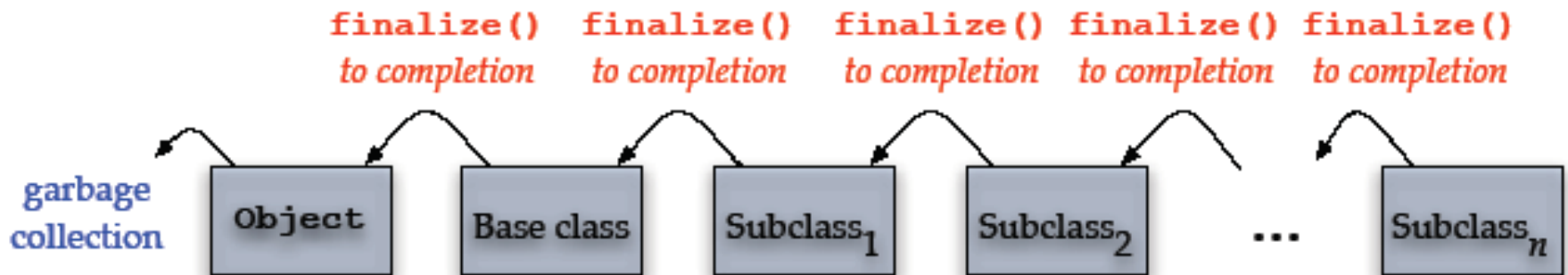
# Lời gọi constructor không tường minh

- Khi khởi tạo một đối tượng, một chuỗi các lời gọi hàm thiết lập sẽ được thực hiện một cách tường minh (qua lời gọi tới phương thức `super()`) hoặc không tường minh một cách tự động
- Lời gọi hàm thiết lập của lớp cơ sở cao nhất trong cây kế thừa sẽ được thực hiện sau cùng, nhưng kết thúc trước. Hàm thiết lập của lớp dẫn xuất sẽ kết thúc cuối cùng.



# Lời gọi finalize() không tương minh

- Khi một đối tượng bị hủy (thu dọn bởi GC) một chuỗi các phương thức finalize() sẽ được tự động thực hiện.
- Trình tự ngược lại với chuỗi lời gọi hàm thiết lập
  - Phương thức finalize() của lớp dẫn xuất được gọi đầu tiên, sau đó đến lớp cha của nó,...



# Đặc điểm về tính kế thừa trong C++

# Khai báo kế thừa

```
class DerivedClass : access-specifier BaseClass  
{  
    // Body of the derived class  
};
```

DerivedClass: lớp dẫn xuất

BaseClass: lớp cơ sở

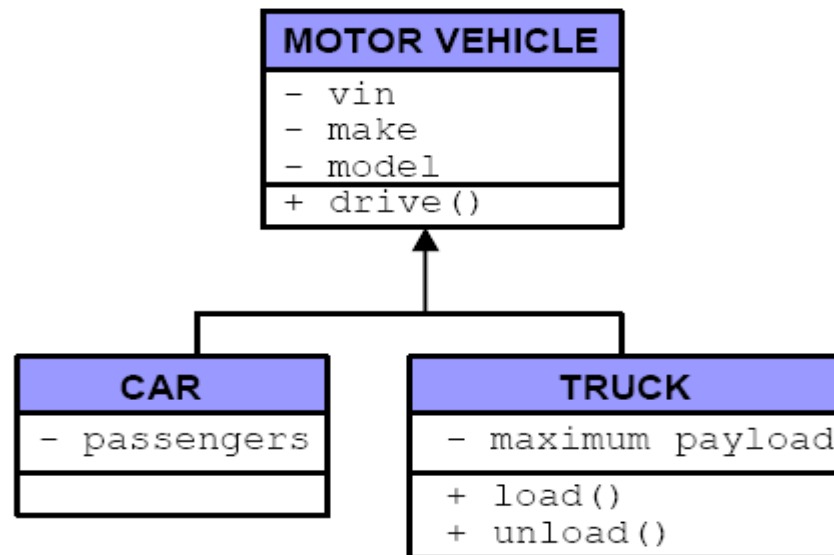
Access-specifier: public, protected, private

# Khai báo kế thừa

**Giả sử đã có sẵn các lớp A và B; để xây dựng lớp C dẫn xuất từ A và B ta có thể viết như sau:**

```
class C : public A, public B
{
    private:
        // khai báo các thuộc tính
    public:
        // các phương thức
};
```

# Ví dụ MotorVehicle





# Ví dụ lớp MotorVehicle

- Bắt đầu bằng định nghĩa lớp cơ sở, MotorVehicle

MOTOR VEHICLE	
*	vin
-	make
-	model
+	drive()

```
class MotorVehicle {  
public:  
    MotorVehicle(int vin, string make, string model);  
    ~MotorVehicle();  
    void drive(int speed, int distance);  
private:  
    int vin;  
    string make;  
    string model;  
};
```

# Ví dụ lớp MotorVehicle

- Tiếp theo, định nghĩa constructor, destructor, và hàm drive() (ở đây, ta chỉ định nghĩa tạm drive())

```
MotorVehicle::MotorVehicle(int vin, string make, string model)
{
    this->vin = vin;
    this->make = make;
    this->model = model;
}
MotorVehicle::~~MotorVehicle() {}
void MotorVehicle::drive(int speed, int distance)
{
    cout << "Dummy drive() of MotorVehicle." << endl;
}
```

# Định nghĩa lớp con

- Mô tả một lớp con cũng giống như biểu diễn nó trong sơ đồ đối tượng quan hệ, ta chỉ tập trung vào những điểm khác với lớp cha
- Ích lợi
  - đơn giản hoá khai báo lớp,
  - hỗ trợ nguyên lý đóng gói của hướng đối tượng
  - hỗ trợ tái sử dụng code (sử dụng lại định nghĩa của các thành viên dữ liệu và phương thức)
  - việc che dấu thông tin cũng có thể có vai trò trong việc tạo cây thừa kế

# Định nghĩa lớp con Car

Chỉ rõ quan hệ giữa lớp con **Car** và lớp cha **MotorVehicle**



```
class Car : public MotorVehicle
{
    public:
        Car (int passengers);
        ~Car();
    private:
        int passengers;
};
```

# Lớp Car

```
class Car : public MotorVehicle
{
public:
    Car (int vin, string make, string model, int passengers);
    ~Car();
private:
    int passengers;
};
```

Quy ước: đặt các tham số cho lớp cha lên đầu danh sách.

```
Car::Car(int vin, string make, string model, int passengers)
{
    this->vin = vin;
    this->make = make;
    this->model = model;
    this->passengers = passengers;
}
Car::~~Car() {}
```

# Định nghĩa lớp con

- Nhược điểm: trực tiếp truy nhập các thành viên dữ liệu của lớp cơ sở
  - thiếu tính đóng gói : phải biết sâu về chi tiết lớp cơ sở và phải can thiệp sâu
  - không tái sử dụng mã khởi tạo của lớp cơ sở
  - không thể khởi tạo các thành viên private của lớp cơ sở do không có quyền truy nhập
- Nguyên tắc: một đối tượng thuộc lớp con bao gồm một đối tượng lớp cha cộng thêm các tính năng bổ sung của lớp con
  - một thể hiện của lớp cơ sở sẽ được tạo trước, sau đó "gắn" thêm các tính năng bổ sung của lớp dẫn xuất
- Vậy, ta sẽ sử dụng constructor của lớp cơ sở.

# Định nghĩa lớp con

- Để sử dụng constructor của lớp cơ sở, ta dùng danh sách khởi tạo của constructor (tương tự như khi khởi tạo các hằng thành viên)
  - cách duy nhất để tạo phần thuộc về thể hiện của lớp cha tạo

```
Car::Car(int vin, string make, string model, int passengers)
: MotorVehicle(vin, make, model)
{
    this->passengers = passengers;
}
```

Gọi constructor của **MotorVehicle** với các tham số **vin, make, model**

Ta không cần khởi tạo các thành viên **vin, make, model** từ bên trong constructor của **Car** nữa

# Định nghĩa lớp con

- Để đảm bảo rằng một thể hiện của lớp cơ sở luôn được tạo trước, nếu ta bỏ qua lời gọi constructor lớp cơ sở tại danh sách khởi tạo của lớp dẫn xuất, trình biên dịch sẽ tự động chèn thêm lời gọi constructor mặc định của lớp cơ sở
- Tuy ta cần gọi constructor của lớp cơ sở một cách tường minh, tại destructor của lớp dẫn xuất, lời gọi tương tự cho destructor của lớp cơ sở là không cần thiết
  - việc này được thực hiện tự động



# Định nghĩa lớp con

```
class Truck : public MotorVehicle {
public:
    Truck (int vin, string make, string model,
            int maxPayload);
    ~Truck();
    void Load();
    void Unload();
private:
    int maxPayload;
};
```

k như

```
Truck::Truck(int vin, string make, string model,
             int maxPayload)
    : MotorVehicle(vin, make, model)
{
    this->maxPayload = maxPayload;
}
Truck::~~Truck() {}
void Truck::Load() {...}
void Truck::Unload() {...}
```

# Truy cập tới thành viên kế thừa

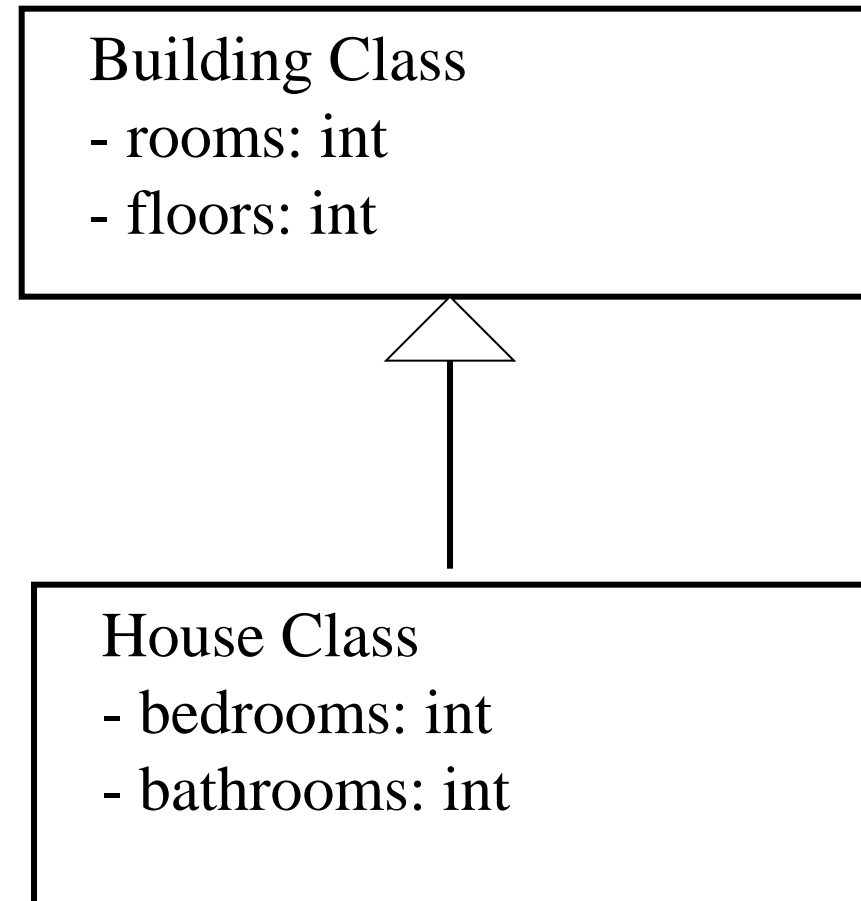
- Truy cập tới các thành viên được kế thừa phụ thuộc không chỉ vào việc chúng được khai báo với từ khóa public, protected hay private trong base class.
- Access còn phụ thuộc vào kiểu kế thừa - public, protected hay private – Kiểu kế thừa được xác định trong **định nghĩa của lớp dẫn xuất**.

# Kế thừa public

- Public inheritance có nghĩa:
  - public members của base class trở thành public members của derived class;
  - protected members của base class trở thành protected members của derived class;
  - private members của base class không thể truy cập được trong derived class.

## Ví dụ

- Giả sử có hai lớp kế thừa sau



**class Building**

```
{  
public:  
    void setRooms(int numRooms);  
    int getRooms() const;  
    void setFloors(int numFloors);  
    int getFloors() const;  
private:    // ??  
    int rooms; // Number of rooms  
    int floors; // Number of floors  
};  
class House : public Building  
{  
public:  
    void setBedrooms(int numBedrooms);  
    int getBedrooms() const;  
    void setBathrooms(int numBathrooms);  
    int getBathrooms() const;  
private:  
    int bedrooms; // Number of bedrooms  
    int bathrooms; // Number of bathrooms  
};
```

Không truy cập được từ lớp  
Dẫn xuất => chuyển thành :  
protected

# Lớp MotorVehicle và Truck

- Quay lại cây thừa kế với MotorVehicle là lớp cơ sở
  - Mọi thành viên dữ liệu đều được khai báo **private**, do đó chúng *chỉ có thể* được truy nhập từ các thể hiện của MotorVehicle
  - tất cả các lớp dẫn xuất không có quyền truy nhập các thành viên private của MotorVehicle
- Vậy, đoạn mã sau sẽ có lỗi

```
class MotorVehicle {  
    ...  
    private:  
        int vin;  
        string make;  
        string model;  
};
```

```
void Truck::Load() {  
    if (this->make == "Ford") {  
        ...  
    }  
}
```

# Lớp MotorVehicle và Truck

- Giả sử ta muốn các lớp con của MotorVehicle có thể truy nhập dữ liệu của nó
- Thay từ khoá **private** bằng **protected**,
- Đoạn mã sau sẽ không còn lỗi

```
class MotorVehicle {  
    ...  
    protected:  
        int vin;  
        string make;  
        string model;  
};
```

```
void Truck::Load() {  
    if (this->make == "Ford") {  
        ...  
    }  
}
```

# Kế thừa protected

- public members của base class trở thành **protected** members của derived class;
- protected members của base class trở thành protected members của derived class;
- private members của base class không thể truy cập được trong derived class.
- quan hệ thừa kế sẽ được nhìn thấy từ
  - mọi phương thức bên trong Car,
  - mọi phương thức thuộc các lớp con của Ca
- Tuy nhiên, mọi đối tượng khác của C++ không nhìn thấy quan hệ này



# Private inheritance

- public members của base class trở thành **private** members của derived class;
- protected members của base class trở thành **private** của derived class;
- private members của base class không thể truy cập được trong derived class.
- chỉ có chính thể hiện đó biết nó được thừa kế từ lớp cha
- các đối tượng bên ngoài không thể tương tác với lớp con như với lớp cha, vì mọi thành viên được thừa kế đều trở thành **private** trong lớp con
- Có thể dùng thừa kế **private** để tạo lớp con có mọi chức năng của lớp cha nhưng lại không cho bên ngoài biết về các chức năng đó.



# Chú ý về Bài tập tuần

- Bài làm giống nhau – 0 điểm
- CNPM: Ngô Thị Hà = Hoàng Hải Nam=Nguyễn Hồng Nhung
- CNPM: Nguyễn Hữu Mạnh = Hoàng Hải Nam (bài complex, bài mydate chép của ai đó)
- CNPM: Ngô Văn Phú = Hồ Văn Hiền
- CNPM: Nguyễn Văn Tự (Tin Pháp) = Đặng Minh Sang = Nguyễn Minh Tuyên (chep doc, ppt)
- CNPM: Nông Việt Dũng = Ngô Thị Bích Thuận
- HTTT: Trần Đức Việt=Trần Khắc Giao=Vũ Văn Toán
- KTMT: Hoàng Văn Hùng = Nguyễn Tiệp Anh
- KTMT: Đậu Hoàng Nhật = Nguyễn Đình Thiện:
- HTTT: Nguyễn Tiến Dũng = Đậu Thanh bình HTTT = Nguyễn Thế cường:
- Hoàng Thanh Tùng HTTT =Trần Bình KTMT:
- Đỗ Đức Đông HTTT=Lê quang Vịnh:

# Chú ý về Bài tập tuần

- Nhóm có bài doc, ppt giống nhau: 5đ
- Nguyễn Thành Luân
- Nguyễn Văn Thịnh
- Nguyễn Văn Tú
- Nguyễn Xuân Thịnh
- Trần Ngọc Mạnh
- Vũ Tuấn Anh (Trung tâm mạng)
- Trương Đức Nhật
- Nguyễn Quang Hải (KTMT)
- 
- Vũ Tuấn Thanh có doc giống Nguyễn Văn Tiến 2007885:
- 
- Trương Minh Khôi KTMT, Hoàng Thị Kim Nhung TTM giống doc, ppt của KTMT: Hoàng Văn Hùng = Nguyễn Tiệp Anh
- Vũ Việt Dũng(HTTT) = Đỗ Đức Đông (HTTT) về doc