



ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

**IT3100**

# **LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

## **Bài 05. Kết tập và Kế thừa**

# Mục tiêu bài học

- ❖ Giải thích về khái niệm tái sử dụng mã nguồn
- ❖ 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

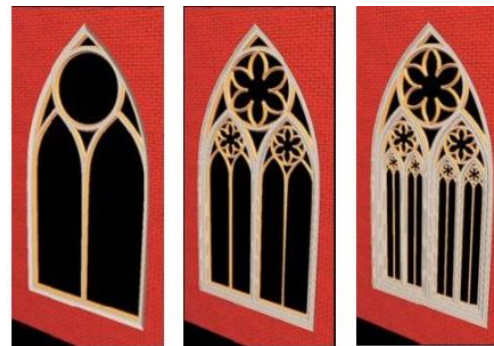
# Nội dung

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

# 1/ TÁI SỬ DỤNG MÃ NGUỒN

# 1. Tái sử dụng mã nguồn

- ❖ 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)

# 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)

## 2/ KẾT TẬP



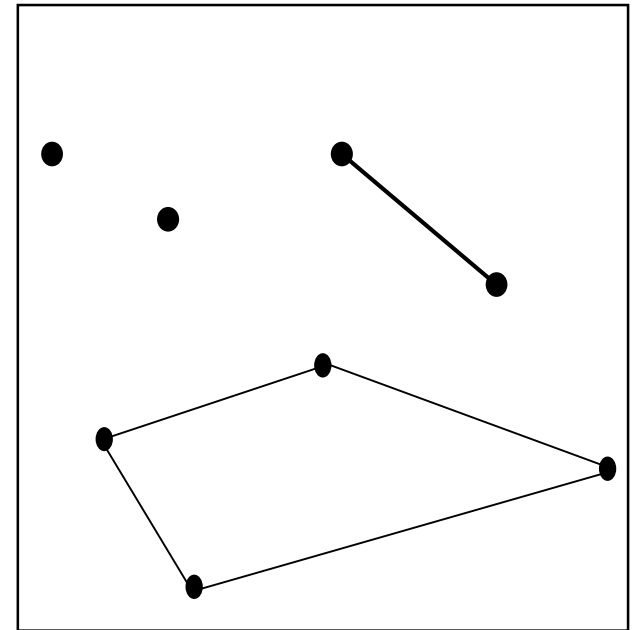
## 2.1 Kết tập

### ❖ Kết tập:

- Quan hệ chứa/có ("has-a") hoặc là một phần ("is-a-part-of")

### ❖ Ví dụ:

- Tứ giác gồm 4 điểm → Kết tập



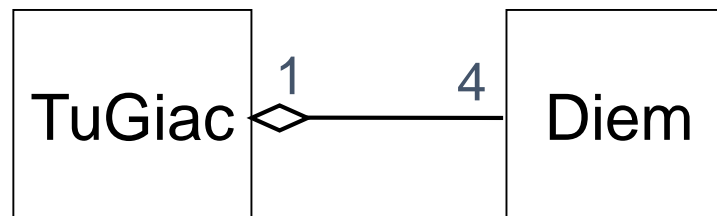
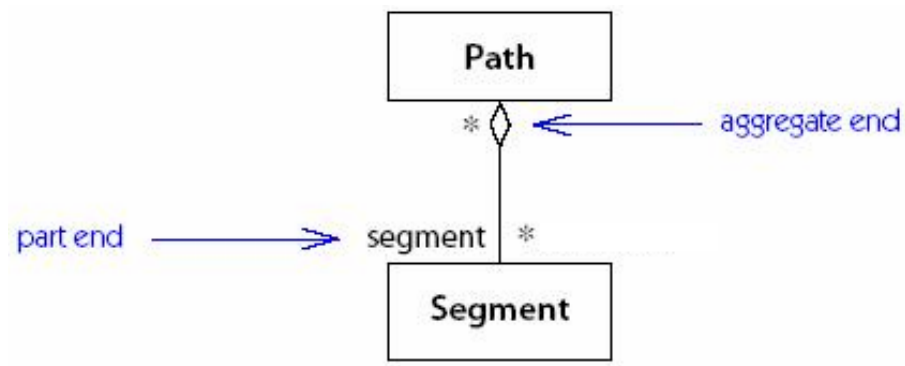
## 2.2. Bản chất của kết tập

- ❖ Kết tập (aggregation): Tạo ra thành viên của lớp mới từ các đối tượng của các lớp có sẵn.
- ❖ Lớp mới
  - Lớp toàn thể (Aggregate/Whole): Lớp toàn thể chứa đối tượng của lớp thành phần
- ❖ Lớp cũ
  - Lớp thành phần (Part): Là một phần (is-a-part of) của lớp toàn thể
  - Kết tập tái sử dụng các thành phần dữ liệu và các hành vi của lớp thành phần thông qua đối tượng thành phần

## 2.3. Biểu diễn kết tập bằng UML

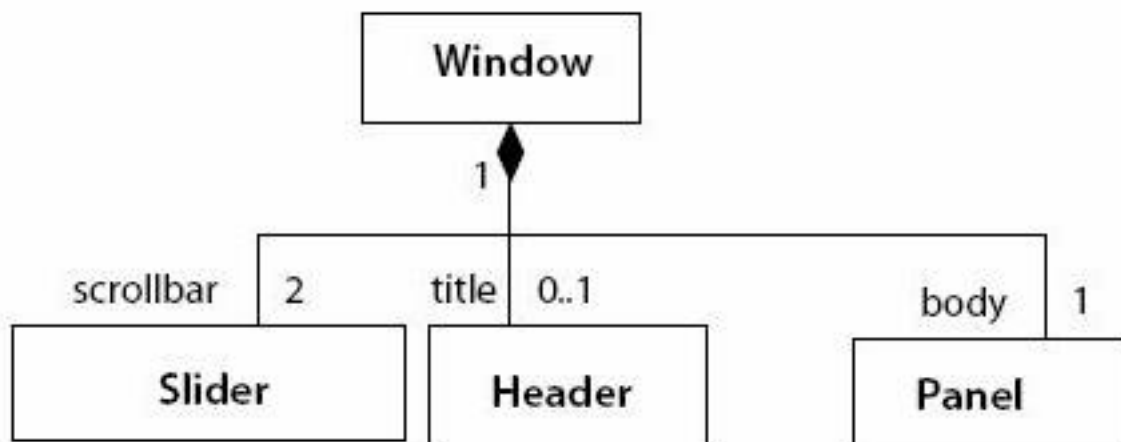
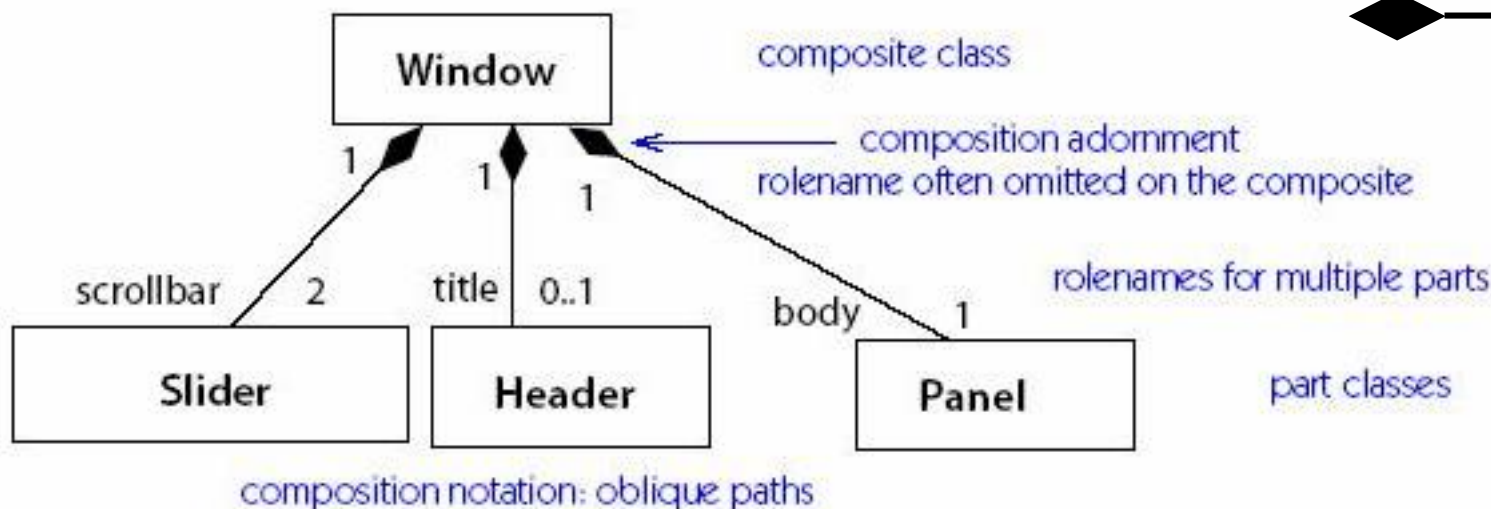
- ❖ Sử dụng "hình thoi" tại đầu của lớp toàn thể
- ❖ Sử dụng bội số quan hệ (multiplicity) tại 2 đầu
  - 1 số nguyên dương: 1, 2,...
  - Dải số (0..1, 2..4)
  - \*: Bất kỳ số nào
  - Không có: Mặc định là 1
- ❖ Tên vai trò (rolename)
  - Nếu không có thì mặc định là tên của lớp (bỏ viết hoa chữ cái đầu)

# Ví dụ



## Ví dụ (2)

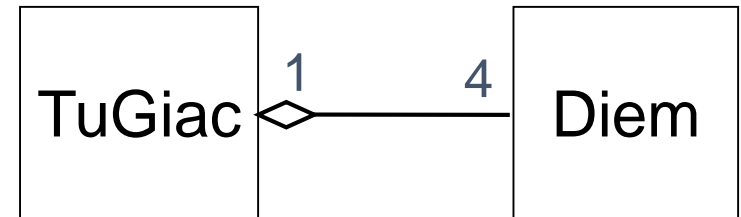
**Cấu thành (composition):**  
là một dạng kết tập, bộ phận không thể tồn tại nếu toàn thể bị hủy bỏ



alternate notation: grouping paths as a tree

## 2.4. 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 printDiem(){  
        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 args[])  
    {  
        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();  
    }  
}
```



# Cách cài đặt khác

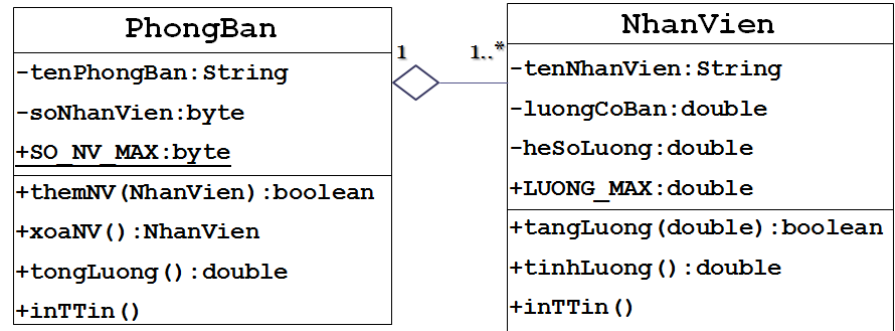
```
class TuGiac {  
    private Diem[] diem = new Diem[4];  
    public TuGiac(Diem p1, Diem p2, Diem p3, Diem p4) {  
        diem[0] = p1; diem[1] = p2;  
        diem[2] = p3; diem[3] = p4;  
    }  
    public void printTuGiac() {  
        diem[0].printDiem(); diem[1].printDiem();  
        diem[2].printDiem(); diem[3].printDiem();  
        System.out.println();  
    }  
}
```

## 2.5. 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 đối tượng thành phần được thực hiện trước

## 2.6 Thảo luận

```
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 NhanVien xoaNhanVien(){  
        if (soNhanVien > 0) {  
            NhanVien tmp = dsnv[soNhanVien-1];  
            dsnv[soNhanVien-1] = null;  
            soNhanVien--;  
            return tmp;  
        } else return null;  
    }  
    public double tongLuong(){  
        double tong = 0.0;  
        for (int i=0;i<soNhanVien;i++)  
            tong += dsnv[i].tinhLuong();  
        return tong;  
    }  
}
```



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

## 2.6 Thảo luận

Trong ví dụ trên

❖ Lớp cũ? Lớp mới?

- 
- 

❖ Lớp mới tái sử dụng lớp cũ thông qua?

- 

❖ Lớp mới tái sử dụng được những gì của lớp cũ?

- 
-

# 3/ KẾ THỪA

# 3.1. Tổng quan về kế thừa

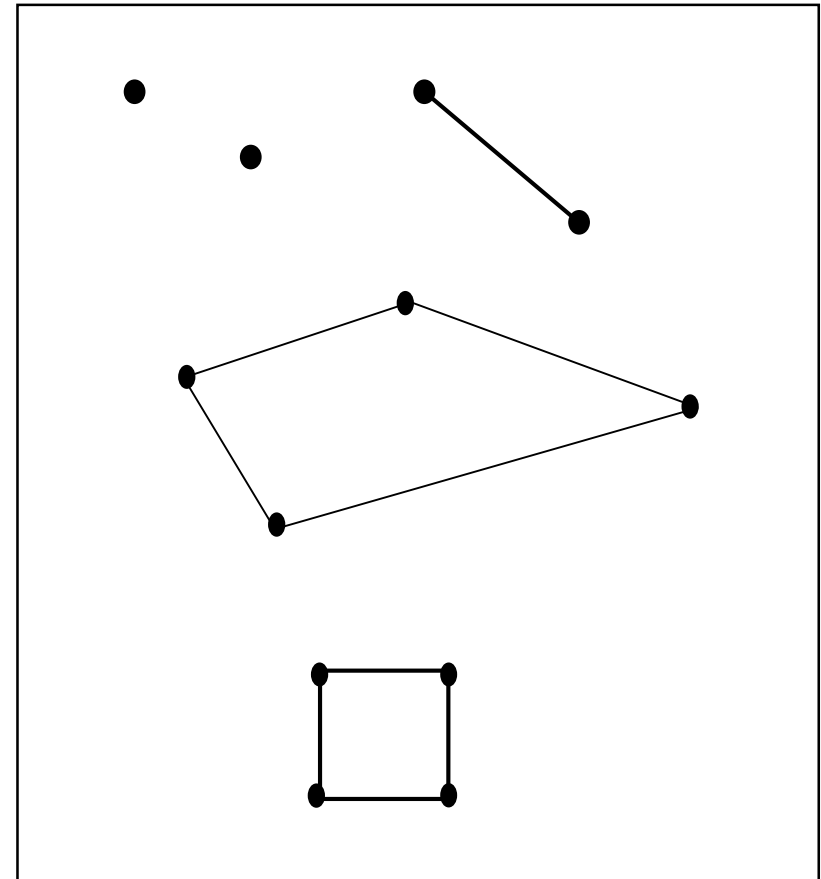
❖ Ví dụ:

- Điểm

- Tứ giác gồm 4 điểm  
→ Kết tập

- Tứ giác

- Hình vuông  
→ Kế thừa



## 3.2. 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)

## 3.2. 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



## 3.3. Nguyên lý kế thừa

### ❖ Lớp con có thể kế thừa được gì?

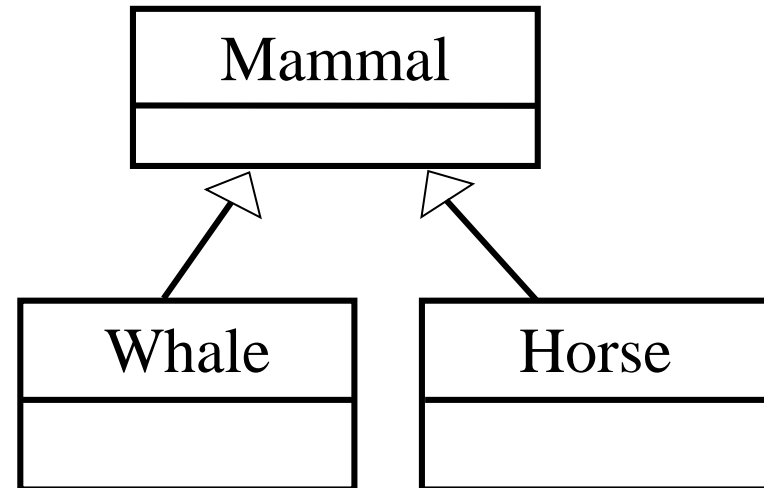
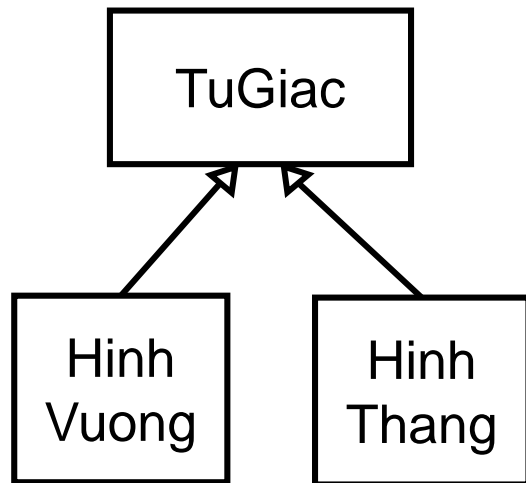
- Kế thừa được các thành viên được khai báo là `public` và `protected` của lớp cha.
- Không kế thừa được các thành viên `private`.
- Các thành viên có chỉ định truy cập mặc định nếu lớp cha cùng gói với lớp con

## 3.3. Nguyên lý kế thừa (2)

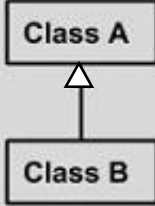
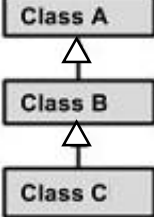
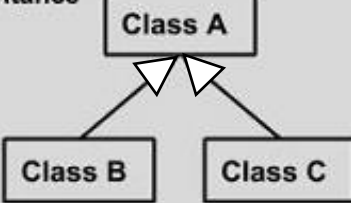
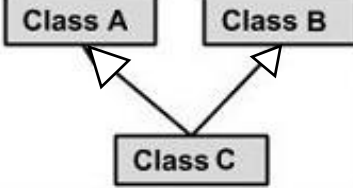
- ❖ Các trường hợp không được phép kế thừa:
  - Các phương thức khởi tạo và hủy
    - Làm nhiệm vụ khởi đầu và gỡ bỏ các đối tượng
    - Chúng chỉ biết cách làm việc với từng lớp cụ thể
  - Toán tử gán =
    - Làm nhiệm vụ giống như phương thức khởi tạo

## 3.4. Biểu diễn kế thừa trong UML

❖ Sử dụng "tam giác rỗng" tại đầu Lớp cha



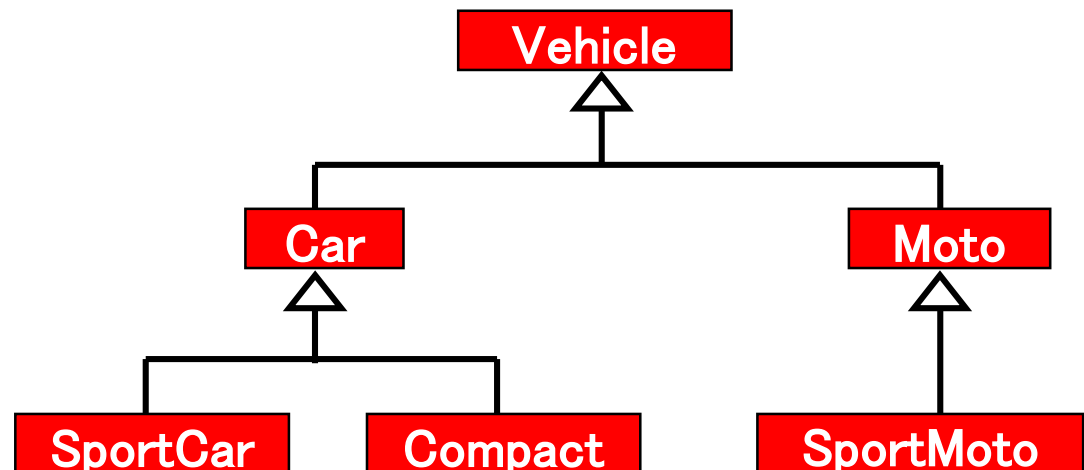
## 3.4. Biểu diễn kế thừa trong UML

<b>Single Inheritance</b> 	<pre>public class A {     ..... } public class B extends A {     ..... }</pre>
<b>Multi Level Inheritance</b> 	<pre>public class A { .....} public class B extends A {.....} public class C extends B {.....}</pre>
<b>Hierarchical Inheritance</b> 	<pre>public class A { .....} public class B extends A {.....} public class C extends A {.....}</pre>
<b>Multiple Inheritance</b> 	<pre>public class A { .....} public class B {.....} public class C extends A,B {     ..... } // Java does not support mutiple Inheritance</pre>

## 3.5 Cây phân cấp kế thừa

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

Mọi lớp  
đều kế thừa từ  
lớp gốc Object



## 3.6. Kết tập và kế thừa

**Giống nhau:** Đều là kỹ thuật trong OOP để tái sử dụng mã nguồn

### Kế thừa

- ❖ Kế thừa tái sử dụng thông qua lớp.
- ❖ Tạo lớp mới bằng cách phát triển lớp đã có
- ❖ Lớp con kế thừa dữ liệu và hành vi của lớp cha
- ❖ 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 lớp mới là tập hợp các đối tượng của các lớp đã có
- ❖ Lớp toàn thể có thể sử dụng dữ liệu và hành vi thông qua các đối tượng thành phần
- ❖ Quan hệ "là một phần" ("is a part of")
- ❖ Ví dụ: Bánh xe là một phần của Ô tô

## 3.7. Cú pháp kế thừa trên Java

- ❖ Cú pháp kế thừa trên Java:

`<Lớp con> extends <Lớp cha>`

- ❖ Ví dụ:


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

- ❖ *Lớp cha nếu được định nghĩa là final thì không thể có lớp dẫn xuất từ nó.*

# Ví dụ 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(){...}  
    ...  
}  
  
public class HìnhVuong extends TuGiac {  
    public HìnhVuong(){  
        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[]){  
        HìnhVuong hv = new HìnhVuong();  
        hv.printTuGiac();  
    }  
}
```

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



Gọi phương thức *public* trong lớp cha từ đối tượng lớp con





## Ví dụ 2

**protected** String name;  
**protected** Date bithday;

**Nếu cùng gói:**

String name;  
Date bithday;

```
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;  
        ...  
    }  
}
```

## 3.8. Khởi tạo và hủy bỏ ĐT

### ❖ Khởi tạo đối tượng:

- Các phương thức khởi tạo của lớp con **luôn** gọi phương thức khởi tạo của lớp cha ở câu lệnh đầu tiên:
  - Nếu không có lời gọi tường minh, JVM sẽ tự động chèn lời gọi PT khởi tạo không tham số của lớp cha thành câu lệnh đầu tiên
  - Nếu gọi tường minh: sử dụng cú pháp `super()`, `super(tham số)` để gọi PT khởi tạo của lớp cha.

### ❖ Hủy bỏ đối tượng:

- Ngược lại so với khởi tạo đối tượng

## 3.8. Khởi tạo và hủy bỏ ĐT (2)

```
class TuGiac {  
  
}  
  
class HìnhVuong extends TuGiac {  
  
}  
  
public class Demo {  
    public static void main(String[] args) {  
        HìnhVuong hv = new HìnhVuong();  
    }  
}
```

## 3.8. Khởi tạo và hủy bỏ ĐT (2)

```
class TuGiac {  
    public TuGiac(int sodiem){  
        System.out.println("Tứ giác có "+ sodiem + " đỉnh");  
    }  
  
}  
  
class HìnhVuong extends TuGiac {  
    public HìnhVuong(){  
        super(4);  
        System.out.println("Hình vuông");  
    }  
}  
  
public class Demo {  
    public static void main(String[] args) {  
        HìnhVuong hv = new HìnhVuong();  
    }  
}
```

# Bài tập

## ❖ **Bài 1:** Xây dựng lớp TruongPhong với các yêu cầu sau:

- Kế thừa lớp NhanVien và bổ sung thêm các thuộc tính phụ cấp và số năm đương chức như biểu đồ bên.
- Viết 2 phương thức khởi tạo cho lớp TruongPhong, một phương thức khởi tạo mặc định (không có tham số) và một phương thức khởi tạo với các tham số: tên nhân viên, lương cơ bản, hệ số lương, phụ cấp và số năm đương chức

