

Quan Hệ Kế Thừa (Inheritance)



Nội dung

- ❑ Từ khóa super (T.T)
- ❑ Kỹ thuật Overriding
- ❑ Lớp ảo & phương thức trừu tượng
- ❑ Tính đa hình



Từ khóa **super**

- ❑ Từ khóa **super** dùng để tham chiếu đến lớp Cha
- ❑ Cho phép truy xuất đến các phương thức khởi tạo (**constructor**) và phương thức (**method**) của lớp Cha thông qua từ khóa này

```
public class Parent{  
    public String name;  
    public void method(){  
    }  
}
```

```
public class Child extends Parent{  
    public String name;  
    public void method(){  
        this.name = super.name;  
        super.method()  
    }  
}
```



Từ khóa **super**

- ❑ Câu lệnh **super()** hoặc **super(parameters)** được dùng để gọi phương thức khởi tạo của lớp Cha
- ❑ Luôn được đặt trong **dòng đầu tiên** của phương thức khởi tạo của lớp Con
- ❑ Nếu không được khai báo tường minh thì **super()** của lớp Cha sẽ được trình biên dịch tự động thêm vào
- ❑ Các phương thức khác của lớp Cha được gọi theo cú pháp

super.method([parameters])



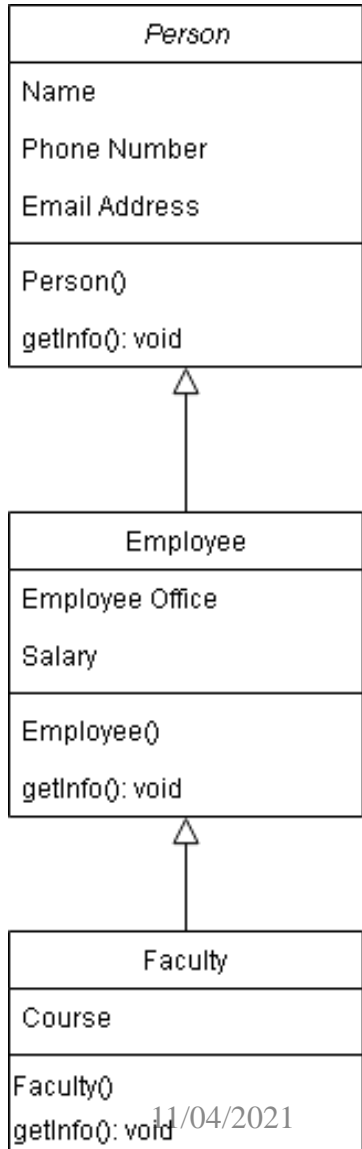
Từ khóa **super**: Ví dụ 1

```
package poly.ho;
public class NhanVien{
    public NhanVien(String hoTen, double luong){...}
    public void xuất(){...}
}
```

```
package poly.hcm;
public class TruongPhong extends NhanVien{
    public double trachNhiem;
    public TruongPhong (String hoTen, double luong, double trachNhiem){
        super(hoTen, luong);
        this.trachNhiem = trachNhiem
    }
    public void xuất(){
        super.xuất()
        System.out.println(trachNhiem)
    }
}
```



Từ khóa **super**: Ví dụ 2



```
class Employee extends Person {
    public Employee() {
        this("This is the second constructor of class Employee");
        System.out.println("This is the first constructor of class Employee");
    }
    public Employee(String s) {
        System.out.println(s);
    }
    public void getInfo() {
        System.out.println("This is a/an employee");
    }
}
```

```
class Person {
    public Person() {
        System.out.println("Person's constructor");
    }
    public void getInfo() {
        System.out.println("This is a/an person");
    }
}
```

```
public class Faculty extends Employee {
    public Faculty() {
        System.out.println("Faculty's constructor");
    }
    public void getInfo() {
        System.out.println("This is a/an faculty");
    }
    public static void main(String[] args) {
        //new Faculty();
        Faculty faculty = new Faculty();
        faculty.getInfo();
    }
}
```

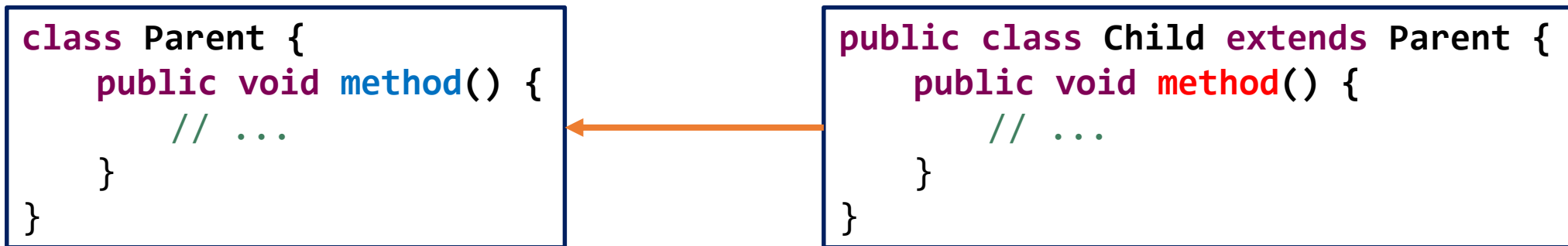


Kĩ thuật **Overriding**

❑ **Overriding** (ghi đè)

❑ Là kĩ thuật cho phép định nghĩa lại một phương thức đã có ở lớp Cha để sử dụng cho phù hợp với lớp Con (Điều lại phương thức cho phù hợp với mục đích sử dụng của lớp Con)

❑ **Overriding** xảy ra khi lớp Con và lớp Cha có phương thức cùng cú pháp






Kỹ thuật **Overriding** :: Minh họa

- ❑ Lớp Parent và Child đều có method() cùng cú pháp thì method() trong Parent sẽ bị ghi đè trong Child


```
class Parent {  
    public void method() {  
        // ...  
    }  
}
```

```
public class Child extends Parent {  
    public void method() {  
        // ...  
    }  
}
```



```
public static void main(String[] args) {  
    Parent p = new Child();  
    p.method();  
}
```

Mặc dù p có kiểu là Parent nhưng khi gọi p.method() thì method() của lớp Child sẽ được thực thi do cơ chế ghi đè (overriding)





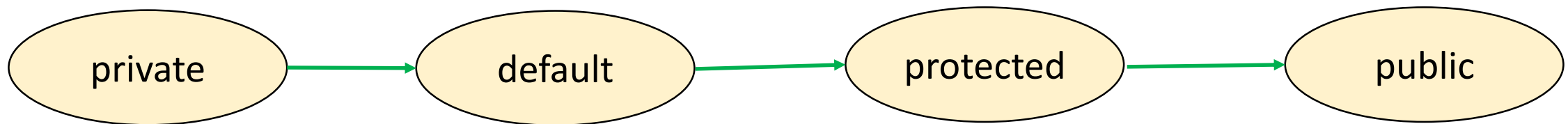
Kĩ thuật **Overriding** :: Đặc điểm

- ❑ Lớp Con ghi đè phương thức của lớp Cha thì sẽ che dấu phương thức của lớp Cha
- ❑ Mục đích của ghi đè là để sửa lại phương thức của lớp Cha trong lớp Con
- ❑ Dùng từ khóa **super** để truy cập đến phương thức nguyên thủy của lớp Cha



Phương thức ghi đè :: **Overriding** method

- ❑ **Overriding** method được định nghĩa trong lớp Con
- ❑ Thường có dòng chữ @Override ở bên trên để chú thích
- ❑ Có tên, kiểu trả về và các tham số giống với phương thức của lớp Cha
- ❑ Có kiểu, phạm vi truy cập tương đương hoặc rộng hơn so với phương thức gốc trong lớp Cha





Phương thức ghi đè :: **Overriding** method

- ❑ Nếu một phương thức được khai báo **private** ở lớp Cha và một phương thức cùng tên được khai báo **private** ở lớp Con thì hai phương thức này sẽ không liên quan gì đến nhau

```
class Parent {  
    private void method() {  
        // ...  
    }  
}
```



```
public class Child extends Parent {  
    private void method() {  
        // ...  
    }  
}
```

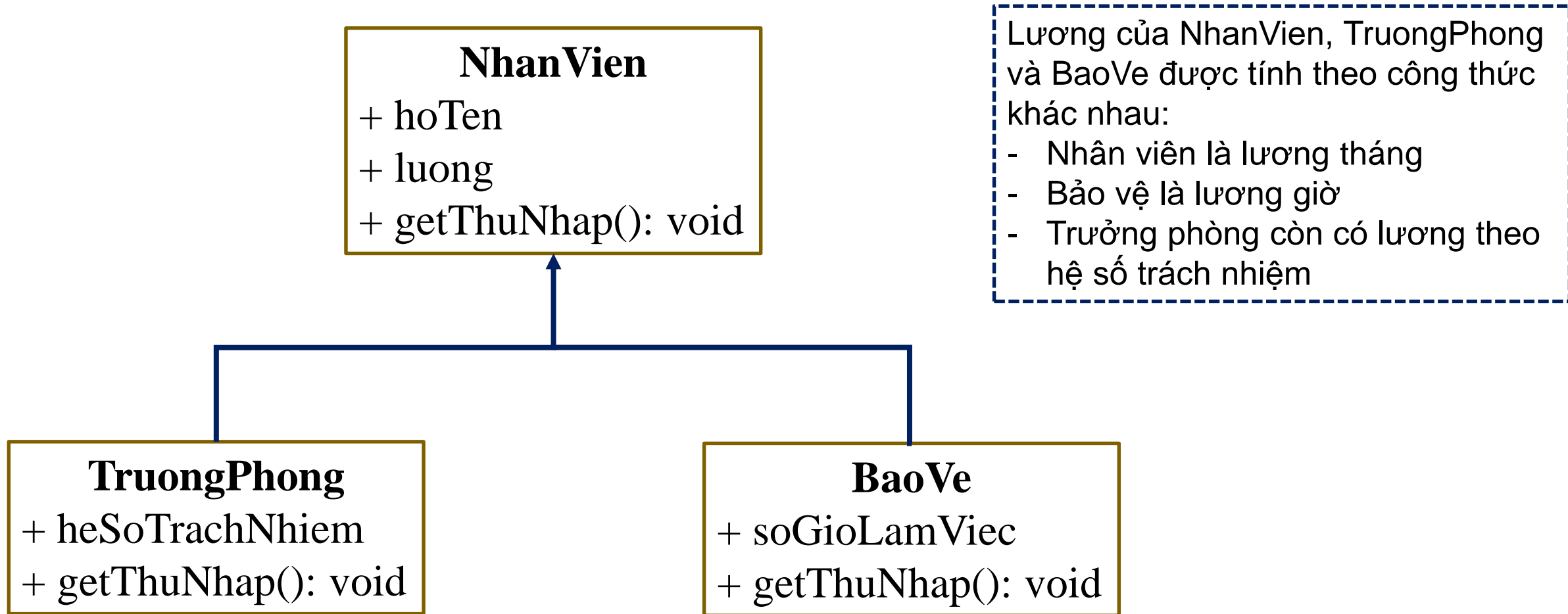


Phương thức ghi đè :: **Overriding** method

- ❑ Phương thức **static** không được phép ghi đè
- ❑ Nếu phương thức **static** được định nghĩa lại ở lớp Con thì `super.staticMethod()` sẽ bị thay bằng `SuperClassName.staticMethod()`



Phương thức ghi đề :: **Minh họa**





Abstract class & Abstract method

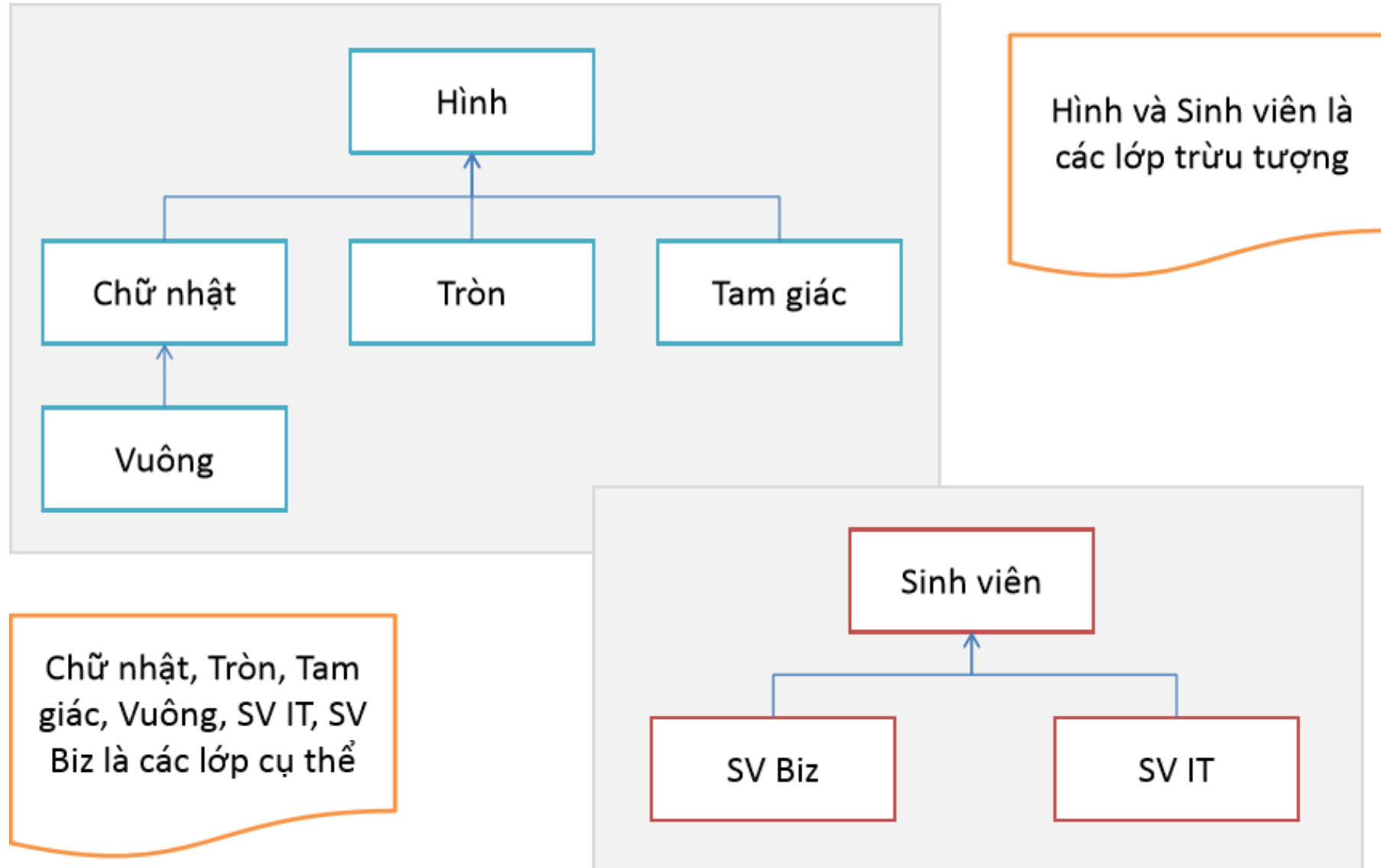


Lớp trừu tượng :: **Abstract class**

- ❑ Lớp **trừu tượng** là lớp có các hành vi chưa được xác định rõ
- ❑ Ví dụ 1: Một đối tượng Hình Học thì chưa xác định rõ được hình dạng là gì nên không thể áp dụng công thức tính diện tích hay chu vi
- ❑ Ví dụ 2: Một đối tượng Sinh Viên thì chưa xác định được cách tính điểm trung bình vì sinh viên mỗi ngành học khác nhau có cách tính khác nhau
- ❑ Vậy lớp Hình Học và lớp Sinh Viên là những lớp trừu tượng vì chúng có các hành vi chưa thể xác định được rõ ràng



Lớp trừu tượng :: **Abstract class**





Abstract class & Abstract method

- ❑ Phương thức trừu tượng là phương thức không có phần thân và có từ khóa **abstract** ở đầu trong cú pháp khai báo
- ❑ Một lớp và phương thức trừu tượng được khai báo bằng cách dùng từ khóa **abstract**
- ❑ Lớp có chứa phương thức trừu tượng thì lớp đó phải là lớp trừu tượng
- ❑ Trong lớp trừu tượng vẫn có thể định nghĩa các trường và phương thức cụ thể
- ❑ Lớp trừu tượng không cho phép tạo đối tượng cụ thể



Abstract class & Abstract method

```
abstract public class MyClass{  
    abstract public type MyMethod();  
}
```

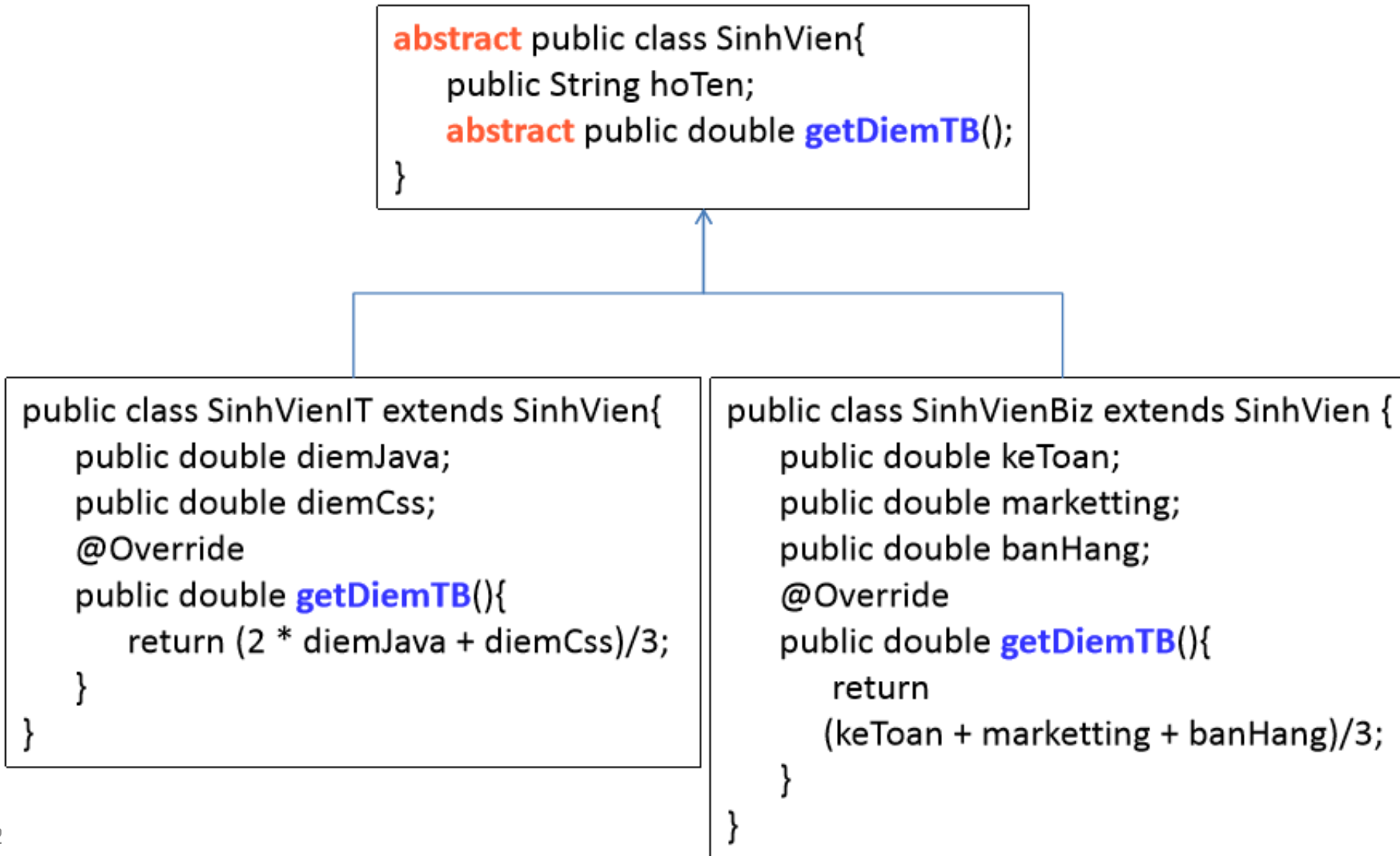
Sử dụng từ khóa
abstract để định
nghĩa lớp và
phương thức trừu
tượng

```
abstract public class SinhVien{  
    abstract public double getDiemTB();  
}
```

```
abstract public class Hinh{  
    abstract public double getChuVi();  
    abstract public double getDienTich();  
}
```



Abstract class & Abstract method





Tính đa hình :: **Polymorphism**

- ❑ Tính đa hình là hiện tượng các đối tượng thuộc các lớp khác nhau có thể hiểu cùng một thông điệp theo các cách khác nhau
- ❑ Cho phép mỗi đối tượng của lớp Cha có thể tham chiếu vào một đối tượng thuộc lớp Con, nhưng điều ngược lại thì không được



Tính đa hình :: **Polymorphism**

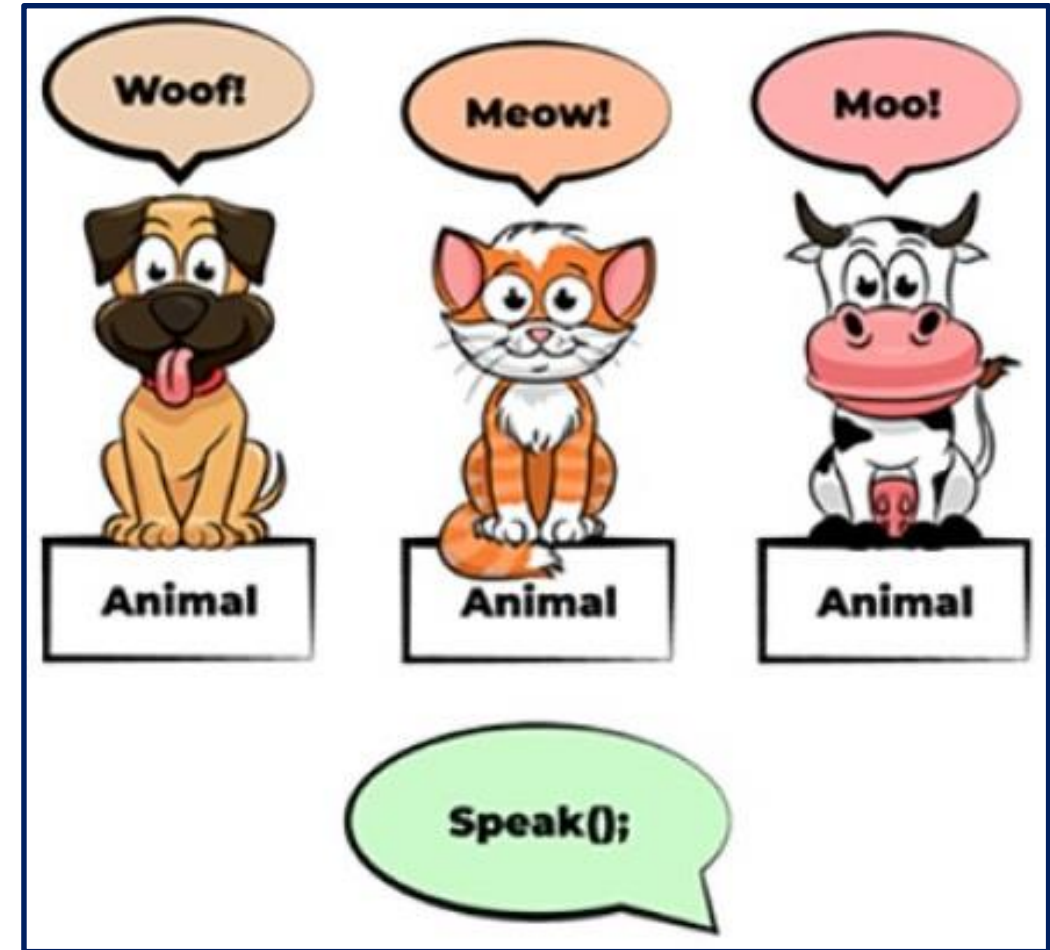
□ Để hiện thực được tính đa hình:

- Các lớp phải có quan hệ thừa kế với cùng 1 lớp Cha nào đó
- Phương thức đa hình phải được ghi đè (override) ở các lớp con



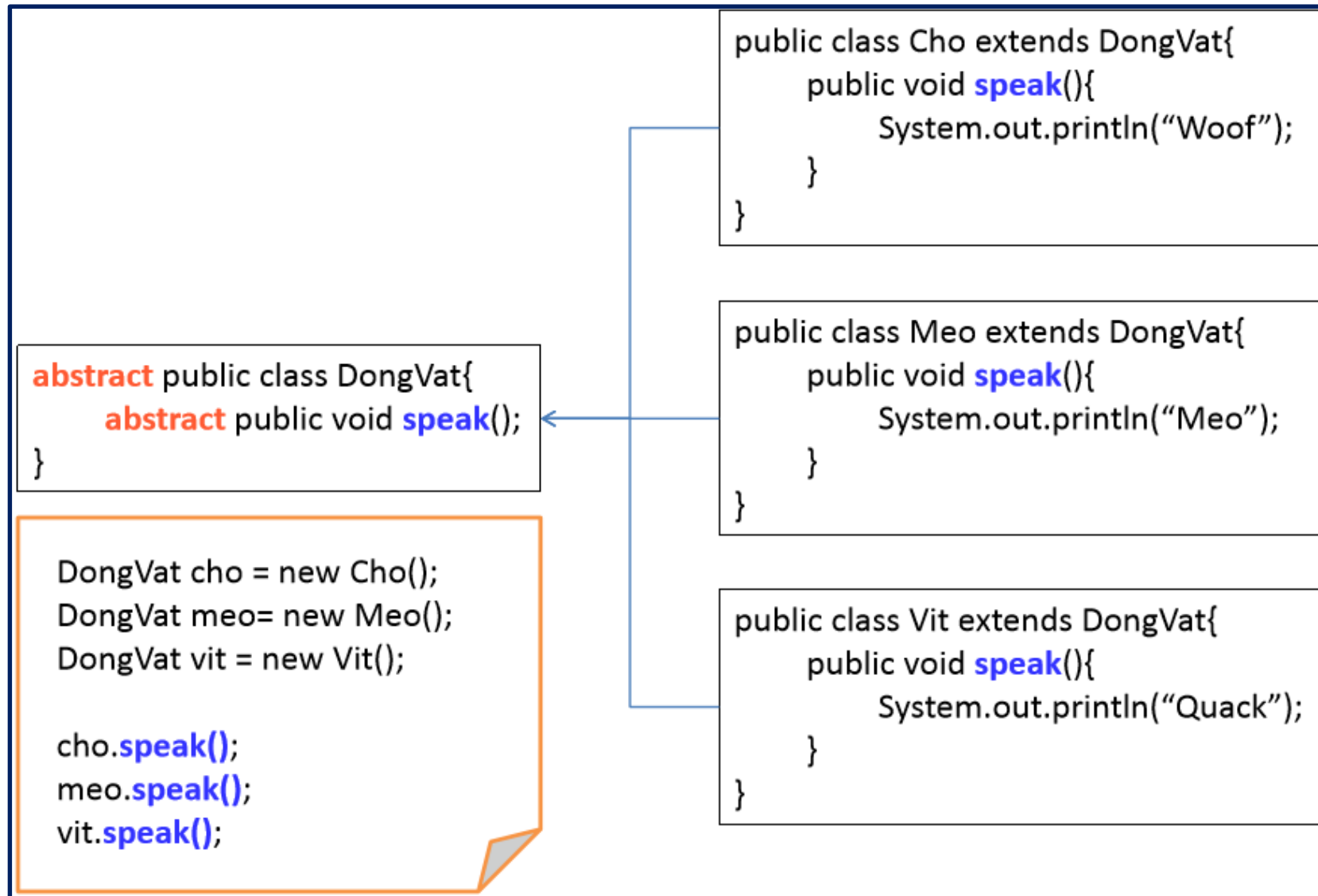
Tính đa hình :: **Polymorphism**

❑ Một ví dụ về đa hình trong thực tế. Ta có 3 con vật: chó, mèo, bò. Cả 3 con vật này đều là động vật. Nhưng khi ta bảo cả 3 động vật kêu thì con chó sẽ kêu gâu gâu, con mèo sẽ kêu meo meo và con bò sẽ kêu um bò





Tính đa hình :: Polymorphism





Tài liệu tham khảo

- Y. Daniel Lang, “**Introduction to Java Programming Comprehension Version**” 10th Edition.
- Jose M. Garrido, “**Object-Oriented Programming: From Problem Solving to Java**”
- Paul Deitel, Harvey Deitel, “**Java : How to program**”, 9th edition, 2012
- Oracle, “**The Java™ Tutorials**”,
<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html>,
6:20PM, 18/01/2018
- Java tutorial, <https://howtodoinjava.com/java/basics/>, 15:00PM,
20/02/2020