



**FPT POLYTECHNIC**



# **LẬP TRÌNH JAVA 2**

**BÀI 1: REVIEW OOP**

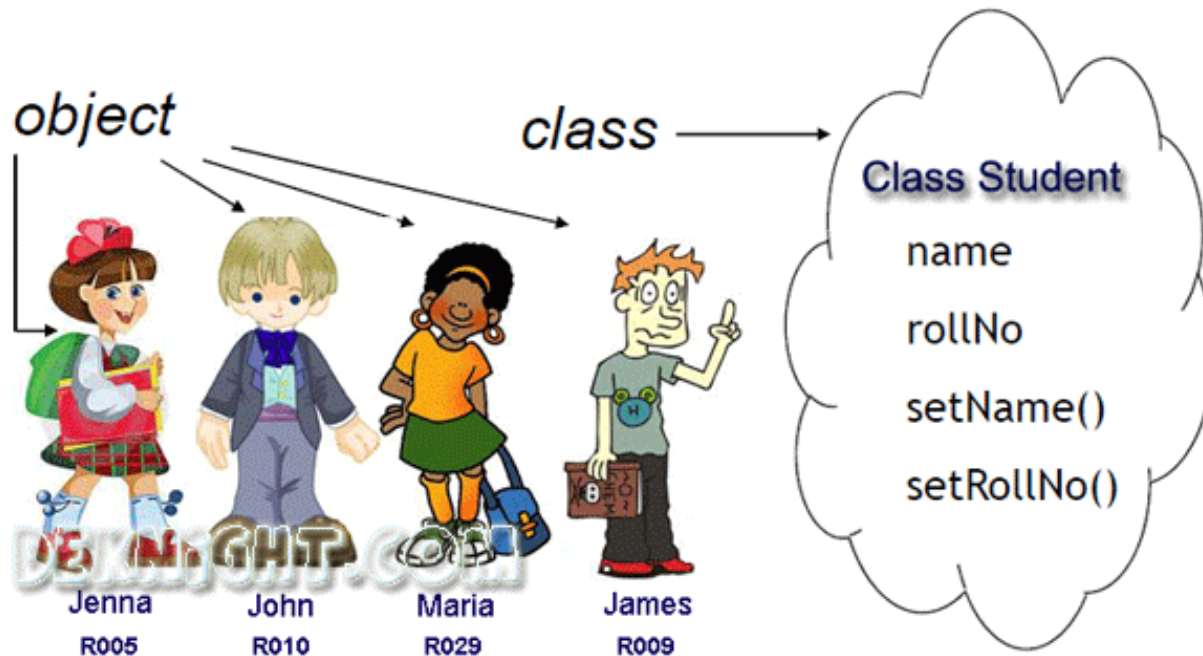
**PHẦN 1: LỚP VÀ ĐỐI TƯỢNG**

[www.poly.edu.vn](http://www.poly.edu.vn)

❑ Kết thúc bài học này bạn sẽ được củng cố kiến thức sau

- ❖ Lớp và đối tượng (Class & Object)
- ❖ Trường dữ liệu (Field) & Phương thức (Method)
- ❖ Hàm tạo (Constructor)
- ❖ Từ khóa this
- ❖ Nạp chồng (Overloading)
- ❖ Gói (Package)
- ❖ Đặc tả truy xuất (Accessing Modifier)
- ❖ Che dấu (Encapsulation)

- ❑ Lớp là một khuôn mẫu được sử dụng để mô tả các đối tượng cùng loại.
- ❑ Lớp bao gồm các thuộc tính (trường dữ liệu) và các phương thức (hàm thành viên)



## ❑ Thuộc tính (field)

- ❖ Hãng sản xuất
- ❖ Model
- ❖ Năm
- ❖ Màu

Danh từ



## ❑ Phương thức (method)

- ❖ Khởi động()
- ❖ Dừng()
- ❖ Phanh()
- ❖ Bật cần gạt nước()

Động từ



```
class <<ClassName>>
```

```
{
```

```
<<type>> <<field1>>;
```

Khai báo các trường

```
...
```

```
<<type>> <<fieldN>>;
```

Khai báo các phương thức

```
<<type>> <<method1>>([parameters]) {
```

```
    // body of method
```

```
}
```

```
...
```

```
<<type>> <<methodN>>([parameters]) {
```

```
    // body of method
```

```
}
```

```
}
```

```
public class Employee{
    public String fullname;
    public double salary;

    public void input(){
        Scanner scanner = new Scanner(System.in);
        System.out.print(" >> Full Name: ");
        this.fullname = scanner.nextLine();

        System.out.print(" >> Salary: ");
        this.salary = scanner.nextDouble();
    }

    public void output(){
        System.out.println(this.fullname);
        System.out.println(this.salary);
    }
}
```

Trường



Phương thức

Lớp Employee có 2 thuộc tính là fullname và salary và 2 phương thức là input() và output()

- ❑ Đoạn mã sau sử dụng lớp Employee để tạo một nhân viên sau đó gọi các phương thức của lớp.

```
public static void main(String[] args) {  
  
    Employee emp = new Employee();  
    emp.input();  
    emp.output();  
}
```



- ❑ Chú ý:

- ❖ Toán tử **new** được sử dụng để tạo đối tượng
- ❖ Biến emp chứa tham chiếu tới đối tượng
- ❖ Sử dụng dấu chấm (.) để truy xuất các thành viên của lớp (trường và phương thức).

- ❑ Phương thức là một mô-đun mã thực hiện một công việc cụ thể nào đó
  - ❖ Trong lớp Employee có 2 phương thức là input() và output()
- ❑ Phương thức có thể có một hoặc nhiều tham số
- ❑ Phương thức có thể có kiểu trả về hoặc void (không trả về gì cả)
- ❑ Cú pháp

```
<<kiểu trả về>> <<tên phương thức>> ([danh sách tham số])  
{  
    // thân phương thức  
}
```



- ❑ Hàm tạo là một phương thức đặc biệt được sử dụng để tạo đối tượng.
- ❑ Đặc điểm của hàm tạo
  - ❖ Tên trùng với tên lớp
  - ❖ Không trả lại giá trị
- ❑ Ví dụ

|  | Lớp |
|--|-----|
| <pre>public class ChuNhat{<br/>    double dai, rong;<br/>    <b>public ChuNhat</b>(double dai, double rong){<br/>        this.dai = dai;<br/>        this.rong = rong;<br/>    }<br/>}</pre> |     |

| Đối tượng  |
|--|
| <pre>ChuNhat cn1 = new ChuNhat(20, 15);<br/>ChuNhat cn2 = new ChuNhat(50, 25);</pre> |

- ❑ Trong một lớp có thể định nghĩa nhiều hàm tạo khác tham số, mỗi hàm tạo cung cấp 1 cách tạo đối tượng.
- ❑ Nếu không khai báo hàm tạo thì Java tự động cung cấp hàm tạo mặc định (không tham số)

```
public class ChuNhat{  
    double dai, rong;  
    public ChuNhat(double dai, double rong){  
        this.dai = dai;  
        this.rong = rong;  
    }  
    public ChuNhat(double canh){  
        this.dai = canh;  
        this.rong = canh;  
    }  
}
```

```
ChuNhat cn = new ChuNhat(20, 15);  
ChuNhat vu= new ChuNhat(30);
```

- ❑ **this** được sử dụng để đại diện cho đối tượng hiện tại, nó được sử dụng trong lớp để tham chiếu tới các thành viên của lớp (field và method)
- ❑ Sử dụng **this.field** để phân biệt field với các biến cục bộ hoặc tham số của phương thức

```
public class MyClass{  
    int field;  
    void method(int field){  
        this.field = field;  
    }  
}
```

The diagram illustrates the use of **this.field** in the provided code. Two callout boxes are present: one labeled "Trường" (Field) pointing to the **this** part of **this.field**, and another labeled "Tham số" (Parameter) pointing to the **field** part of **this.field**. This highlights how **this** is used to refer to the class's field when a local variable or parameter has the same name.

- ❑ Trong một lớp có thể có nhiều phương thức cùng tên nhưng khác nhau về tham số (kiểu, số lượng và thứ tự)

```
public class MyClass{  
    void method()  
    void method(int x)  
    void method(float x)  
    void method(int x, double y)  
}
```

- ❑ Trong lớp MyClass có 4 phương thức cùng tên là method nhưng khác nhau về tham số

- ❑ Xét trường hợp overload sau

```
class MayTinh{  
    int tong(int a, int b){return a + b;}  
    int tong(int a, int b, int c){return a + b + c;}  
}
```

- ❑ Với lớp trên, bạn có thể sử dụng để tính tổng 2 hoặc 3 số nguyên.

```
MayTinh mt = new MayTinh();  
int t1 = mt.tong(5, 7);  
int t2 = mt.tong(5, 7, 9);
```

- ❑ Package được sử dụng để chia các class và interface thành từng gói khác nhau.
  - ❖ Việc làm này tương tự quản lý file trên ổ đĩa trong đó class (file) và package (folder)
- ❑ Ví dụ sau tạo lớp MyClass thuộc gói com.poly

```
package com.poly;  
public class MyClass{...}
```
- ❑ Trong Java có rất nhiều gói được phân theo chức năng
  - ❖ java.util: chứa các lớp tiện ích
  - ❖ java.io: chứa các lớp vào/ra dữ liệu
  - ❖ java.lang: chứa các lớp thường dùng...

- ❑ Lệnh import được sử dụng để chỉ ra lớp đã được định nghĩa trong một package
- ❑ Các lớp trong gói java.lang và các lớp cùng định nghĩa trong cùng một gói với lớp sử dụng sẽ được import ngầm định

```
package com.polyhcm;  
import com.poly.MyClass;  
import java.util.Scanner;  
public class HelloWorld{  
    public static void main(String[] args){  
        MyClass obj = new MyClass();  
        Scanner scanner = new Scanner(System.in);  
    }  
}
```

❑ Đặc tả truy xuất được sử dụng để định nghĩa khả năng cho phép truy xuất đến các thành viên của lớp. Trong java có 4 đặc tả khác nhau:

❖ **private**: chỉ được phép sử dụng nội bộ trong class

❖ **public**: công khai hoàn toàn

❖ **{default}**:

➤ Là public đối với các lớp truy xuất cùng gói

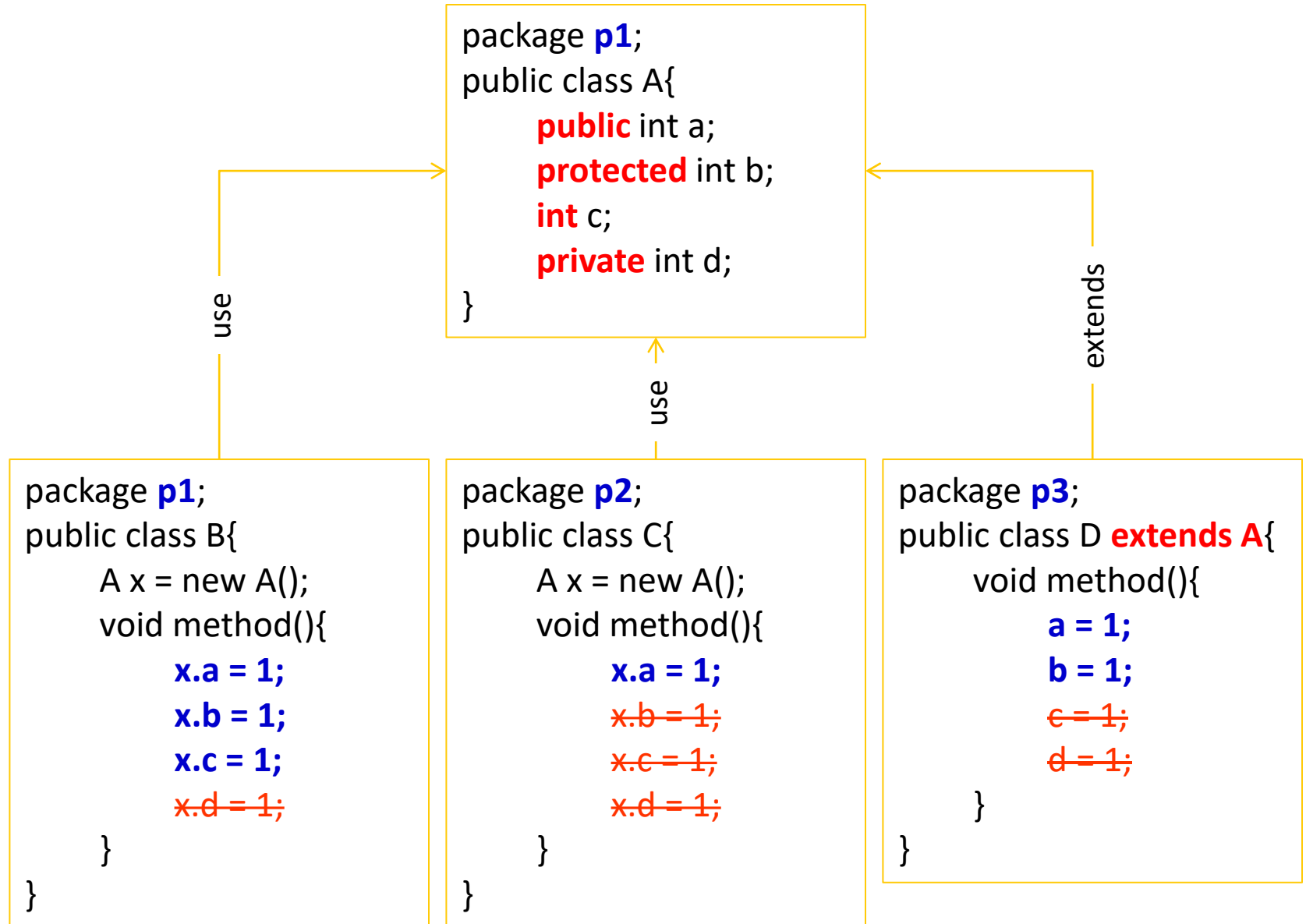
➤ Là private với các lớp truy xuất khác gói.

❖ **protected**: tương tự {default} nhưng cho phép kế thừa dù lớp con và cha khác gói.

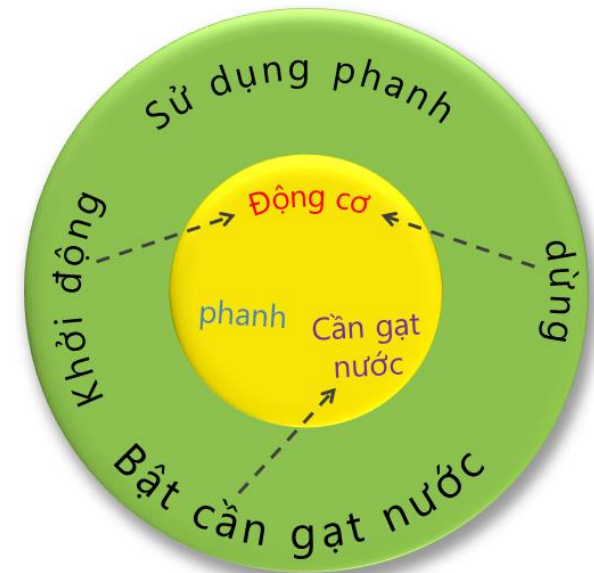
❑ Mức độ che dấu tăng dần theo chiều mũi tên

**public** —→ **protected** —→ **{default}** —→ **private**





- ❑ Encapsulation là tính che dấu trong hướng đối tượng.
  - ❖ Nên che dấu các trường dữ liệu
  - ❖ Sử dụng phương thức để truy xuất các trường dữ liệu
- ❑ Mục đích của che dấu
  - ❖ Bảo vệ dữ liệu
  - ❖ Tăng cường khả năng mở rộng



- ❑ Để che dấu thông tin, sử dụng private cho các trường dữ liệu.

**private** double diem;

- ❑ Bổ sung các phương thức getter và setter để đọc ghi các trường đã che dấu

```
public void setDiem(double diem){  
    this.diem = diem;  
}  
  
public String getDiem() {  
    return this.diem;  
}
```

```
public class SinhVien{
    private String hoTen;
    private double diem;
    public void setHoTen(String hoTen){
        this.hoTen = hoTen;
    }
    public String getHoTen() {
        return this.hoTen;
    }
    public void setDiem(double diem){
        if(diem < 0 || > 10){
            System.out.println("Điểm không hợp lệ");
        }
        else{
            this.diem = diem;
        }
    }
    public String getDiem() {
        return this.diem;
    }
}
```

❑ Chỉ cần thêm mã vào phương thức **setDiem()** để có những xử lý khi dữ liệu không hợp lệ

```
public class MyClass{
    public static void main(String[] args){
        SinhVien sv = new SinhVien();
        sv.setHoTen("Nguyễn Văn Tèo");
        sv.setDiem(20);
    }
}
```



**FPT POLYTECHNIC**



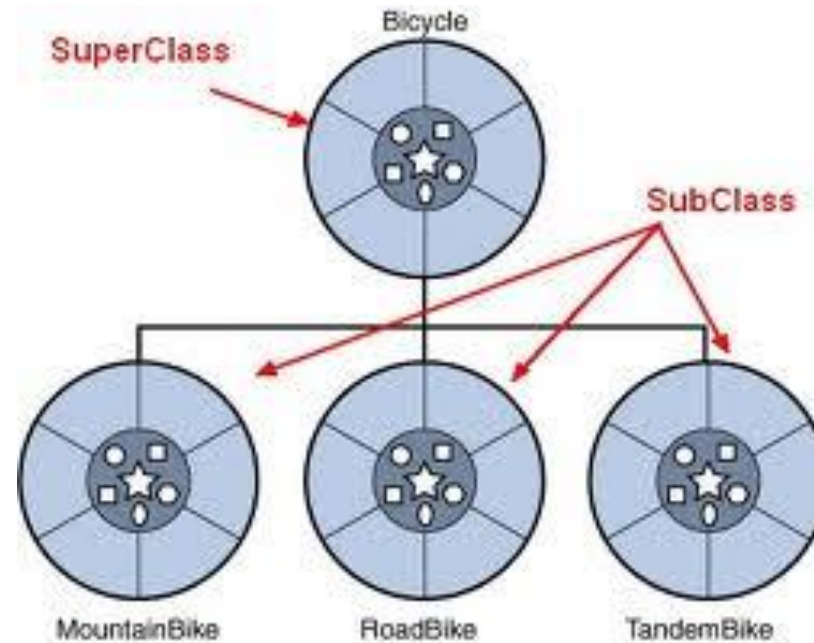
# **LẬP TRÌNH JAVA 2**

**BÀI 1: REVIEW OOP**

**PHẦN 2: KẾ THỪA VÀ ĐA HÌNH**

[www.poly.edu.vn](http://www.poly.edu.vn)

- ❑ Kết thúc bài học này bạn sẽ được củng cố kiến thức sau:
  - ❖ Thừa kế (Inheritance)
  - ❖ Ghi đè và đa hình (Overriding & Polymorphism)
  - ❖ Lớp trừu tượng (Abstract Class)
  - ❖ Giao tiếp lập trình (Interface)
  - ❖ Lớp nội và lớp ẩn danh (Inner & Anonymous class)
  - ❖ Từ khóa static và final (static and final keywords)



```
class Bicycle{...}  
class MountainBike extends Bicycle{...}  
class RoadBike extends Bicycle{...}  
class TandemBike extends Bicycle{...}
```

- ❑ Mục đích của thừa kế là tái sử dụng.
- ❑ Lớp con được phép sở hữu các tài sản (trường và phương thức) của lớp cha
  - ❖ Lớp con được phép sở hữu các tài sản public hoặc protected của lớp cha
  - ❖ Lớp con cũng được phép sở hữu các tài sản mặc định {default} của lớp cha nếu lớp con và lớp cha được định nghĩa cùng gói
  - ❖ Lớp con không thể truy cập thành viên private của lớp cha



```
package poly.ho;
public class NhanVien{
    public String hoTen;
    protected double luong;
    public NhanVien(String hoTen, double luong){...}
    void xuat(){...}
    private double thueThuNhap(){...}
}
```


- A. super.hoTen
- B. super.luong
- C. super.xuat()
- D. super.thueThuNhap()

```
package poly.hcm;
public class TruongPhong extends NhanVien{
    public double trachNhiem;
    public TruongPhong_(String hoTen, double luong, double trachNhiem){...}
    public void xuat(){
        // Mã ở đây có thể sử dụng những tài sản nào của lớp cha
    }
}
```

- ❑ Truy cập đến các thành viên của lớp cha bằng cách sử dụng từ khóa `super`
- ❑ Có thể sử dụng `super` để gọi hàm tạo của lớp cha

```
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()  
    }  
}
```



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

```
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 xuat(){
        super.xuat()
        System.out.println(trachNhiem)
    }
}
```

- ❑ Overriding là trường hợp lớp con và lớp cha có phương thức cùng cú pháp.



- ❑ Lớp Parent và Child đều có phương thức `method()` cùng cú pháp nên `method()` trong Child sẽ đè lên `method()` trong Parent

```
Parent o = new Child();  
o.method();
```

Mặc dù o có kiểu là Parent nhưng o.method() thì method() của lớp Child sẽ chạy do bị đè

- ❑ Lớp trừu tượng là lớp có các hành vi chưa được xác định rõ
  - ❖ Ví dụ 1: Đã là hình thì chắc chắn là có diện tích và chu vi nhưng chưa xác định được cách tính mà phải là một hình cụ thể như chữ nhật, tròn, tam giác... mới có thể xác định cách tính
  - ❖ Ví dụ 2: Sinh viên thì chắc chắn có điểm trung bình nhưng chưa xác định được cách tính như thế nào mà phải là sinh viên của ngành nào mới biết được môn học và công thức tính điểm cụ thể.
- ❑ Vậy lớp hình và lớp sinh viên là các lớp trừu tượng vì phương thức tính chu vi, diện tích và tính điểm chưa thực hiện được.

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

```
public class SinhVienIT extends SinhVien{  
    public double diemJava;  
    public double diemCss;  
    @Override  
    public double getDiemTB(){  
        return (2 * diemJava + diemCss)/3;  
    }  
}
```

```
public class SinhVienBiz extends SinhVien {  
    public double keToan;  
    public double marketing;  
    public double banHang;  
    @Override  
    public double getDiemTB(){  
        return  
            (keToan + marketing + banHang)/3;  
    }  
}
```

- ❑ Overriding thực hiện tính đa hình trong lập trình hướng đối tượng (một hành vi được thể hiện với các hình thái khác nhau)
- ❑ Gọi phương thức bị ghi đè được quyết định lúc chạy chương trình (runtime) chứ không phải lúc biên dịch chương trình (compile time)





# TÍNH ĐA HÌNH (POLYMORPHISM)

```
abstract public class DongVat{  
    abstract public void speak();  
}
```

```
public class Cho extends DongVat{  
    public void speak() {  
        System.out.println("Woof");  
    }  
}
```

```
public class Meo extends DongVat{  
    public void speak() {  
        System.out.println("Meo");  
    }  
}
```

```
public class Vit extends DongVat{  
    public void speak() {  
        System.out.println("Quack");  
    }  
}
```

```
DongVat cho = new Cho();  
DongVat meo = new Meo();  
DongVat vit = new Vit();
```

```
cho.speak();  
meo.speak();  
vit.speak();
```

- ❑ Trong interface chỉ chứa các method **abstract** và các biến **final**
- ❑ Khi một class **thực thi** một interface, nó phải viết mã (**override**) các method trong interface.
- ❑ Interface là public hoặc default
- ❑ Interface có thể được kế thừa
- ❑ Một interface có thể được thực hiện bởi **nhiều** class, và một class có thể thực thi **nhiều** interface.

```
package mypackage;  
  
interface MyInterface {  
    void myMethod1 ();  
    void myMethod2 ();  
}
```

```
package mypackage;
```

```
public class MyClass implements MyInterface {  
    public void myMethod1 () { //phải là public  
        System.out.println("Override my method 1");  
    }
```

```
    public void myMethod2 () {  
        System.out.println("Override my method 2");  
    }
```

```
// method riêng của class
```

```
void mymethod3 () {  
    System.out.println("My method 3");  
}
```

```
}
```

```
public static void main (String a[]) {  
  
    MyClass myObject1 = new MyClass ();  
    obj1.mymethod1 ();  
    obj1.mymethod2 ();  
    obj1.mymethod3 ();  
  
    MyInterface myObject2 = new MyClass ();  
    obj2.mymethod1 ();  
    obj2.mymethod2 ();  
    obj2.mymethod3 (); // error;  
  
}
```

```
public interface MyInter1{  
    void meth1();  
    void meth2();  
}
```

```
public interface MyInter2{  
    void meth3();  
}
```

```
public class MyClass implements MyInter1, MyInter2 {  
    public void meth1() {  
        System.out.println("Implements method 1");  
    }  
    public void meth2() {  
        System.out.println("Implements method 2");  
    }  
    public void meth3() {  
        System.out.println("Implements method 3");  
    }  
}
```

- ❑ Lớp nội là lớp được khai báo bên trong một lớp khác
- ❑ Có hai loại: lớp nội tĩnh và lớp nội thông thường
- ❑ Lớp bên trong chỉ có thể xác định trong phạm vi lớp ngoài cùng và có thể truy cập các thành viên của lớp bao nó

```
public class MyClass{  
    static public class MyInnerStaticClass{}  
    public class MyInnerClass{}  
}
```

Sử dụng lớp nội

```
MyClass.MyInnerStaticClass x = new MyClass.MyInnerStaticClass();  
MyClass.MyInnerClass y = new MyClass().new MyInnerClass();
```

- ❑ Từ khóa static được sử dụng để định nghĩa cho khối và các thành viên tĩnh (**lớp nội, phương thức, trường**).

```
public class MyClass{  
    static public int X;  
    static{  
        X+=100;  
    }  
    static public void method(){  
        X+=200;  
    }  
    static class MyInnerClass{}  
}
```

MyClass.X = 700;  
MyClass.method()



```
public class MyClass{  
    static public int X = 100;  
    static{  
        X+=100;  
    }  
    static public void method(){  
        X+=200;  
    }  
}
```

→

```
MyClass o = new MyClass();  
o.X += 300;  
MyClass.X += 500;  
MyClass.method()
```

MyClass.X, o.X có  
giá trị là bao nhiêu

## ❑ Trong Java có 3 loại hằng

- ❖ Lớp hằng là lớp không cho phép thừa kế
- ❖ Phương thức hằng là phương thức không cho phép ghi đè
- ❖ Biến hằng là biến không cho phép thay đổi giá trị

## ❑ Sử dụng từ khóa final để định nghĩa hằng

```
final public class MyFinalClass{...}
```

```
public class MyClass{  
    final public double PI = 3.14  
    final public void method(){...}  
}
```

## □ Class

- ❖ Constructor, Filed, Method, This, Package, Modifier, Accessing Overloading

## □ Inheritance

- ❖ Resuse, Overriding, super

## □ Abstract & Interface

- ❖ Framework

## □ Static, final

## □ Inner Class

