**Kế thừa trong java** là sự liên quan giữa hai class với nhau, trong đó có class cha (superclass) và class con (subclass). Khi kế thừa class con được hưởng tất cả các phương thức và thuộc tính của class cha. Tuy nhiên, nó chỉ được truy cập các thành viên public và protected của class cha. Nó không được phép truy cập đến thành viên private của class cha.

Ưu điểm của kế thừa

* Tính kế thừa giúp tăng khả năng tái sử dụng. Khi một lớp kế thừa một lớp khác, thì nó có thể truy cập tất cả các chức năng của lớp mà nó kế thừa.
* Khả năng tái sử dụng nâng cao độ tin cậy. Chúng ta chỉ cần kiểm tra và gỡ lỗi với mã lớp cha, chứ không cần kiểm tra từng lớp con.
* Khi mã được tái sử dụng, sẽ giúp giảm chi phí phát triển và bảo trì.
* Các lớp con sẽ tuân thủ theo một giao diện (interface) chuẩn.
* Tính kế thừa giúp hạn chế sự dư thừa mã và hỗ trợ khả năng mở rộng mã.

Nhược điểm của kế thừa

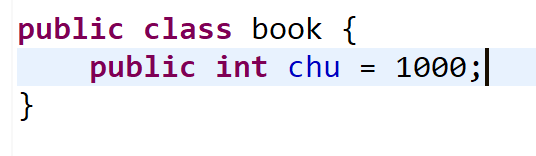
* Các chức năng được kế thừa hoạt động chậm hơn chức năng bình thường, vì nó được thực hiện gián tiếp (lấy từ lớp cha) chứ không phải trực tiếp.
* Thông thường, các dữ liệu thành viên trong lớp cha không được sử dụng. Điều này có thể dẫn đến lãng phí bộ nhớ.
* Tính kế thừa làm tăng sự kết nối giữa lớp cơ sở và lớp kế thừa. Một thay đổi trong lớp cha sẽ ảnh hưởng đến tất cả các lớp con.
* Nếu sử dụng Tính kế thừa không đúng cách có thể dẫn đến các cách giải quyết sai lầm sau này.

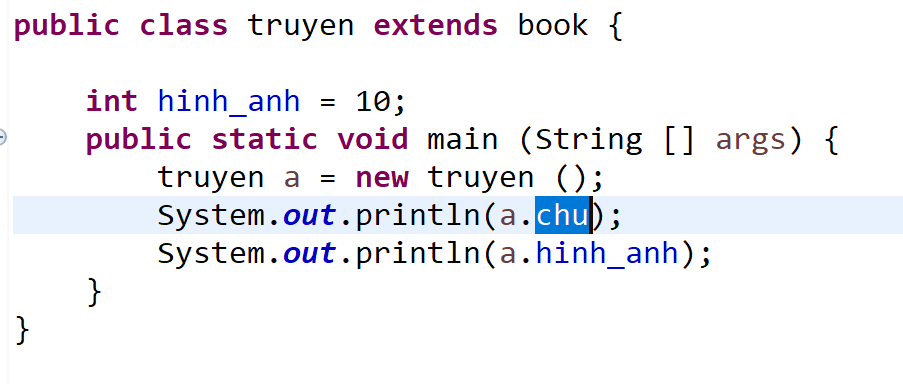
Cú pháp của kế thừa : sử dụng từ khoá extends để kế thừa

**class** Subclass-name **extends** Superclass-name {

}

Ví dụ về kế thừa



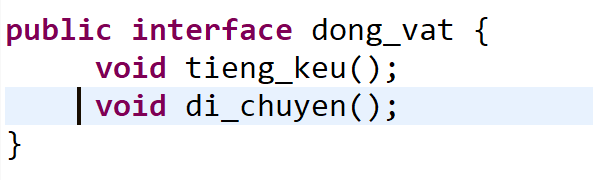


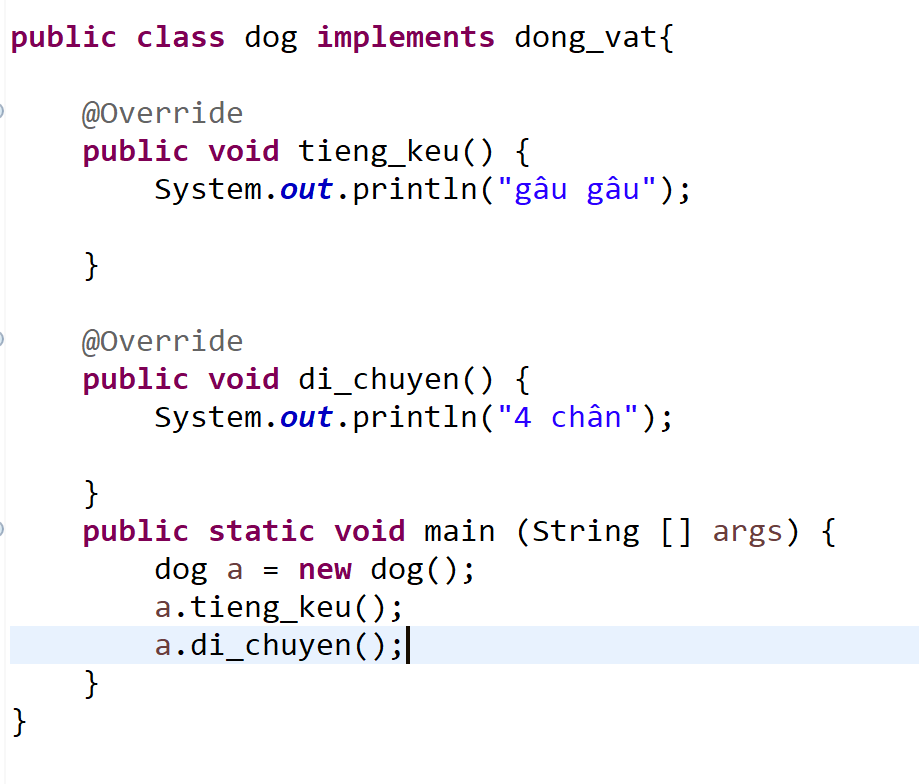
Kết quả

1000 và 10

từ khoá implements

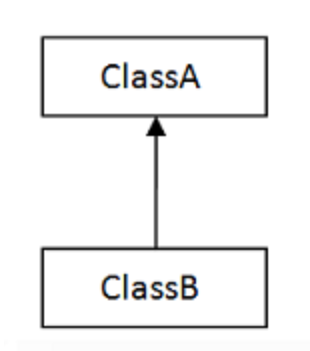
**Từ khóa implements** được sử dụng bởi các **lớp** mà kế thừa **từ** Interface





Các laoij kế thừa trong java : đơn kế thừa , kế thừa nhiều cấp , kế thừa thứ bậc

Đơn kế thừa



**class** Animal {

**void** eat() {

System.***out***.println("eating...");

}

}

**class** Dog **extends** Animal {

**void** bark() {

System.***out***.println("barking...");

}

}

**public** **class** TestInheritance1 {

**public** **static** **void** main(String args[]) {

Dog d = **new** Dog();

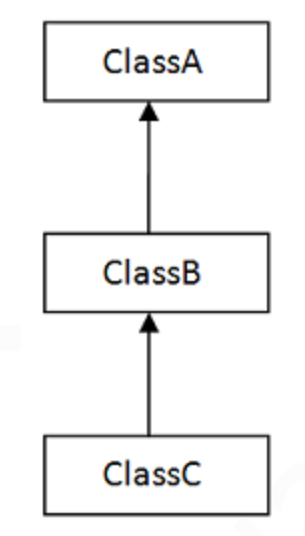
d.bark();

d.eat();

}

}

Kế thừa nhiều cấp



**class** Animal\_1 {

**void** eat() {

System.***out***.println("eating...");

}

}

**class** Dog\_1 **extends** Animal\_1 {

**void** bark() {

System.***out***.println("barking...");

}

}

**class** BabyDog **extends** Dog\_1 {

**void** weep() {

System.***out***.println("weeping...");

}

}

**public** **class** TestInheritance2 {

**public** **static** **void** main(String args[]) {

BabyDog d = **new** BabyDog();

d.weep();

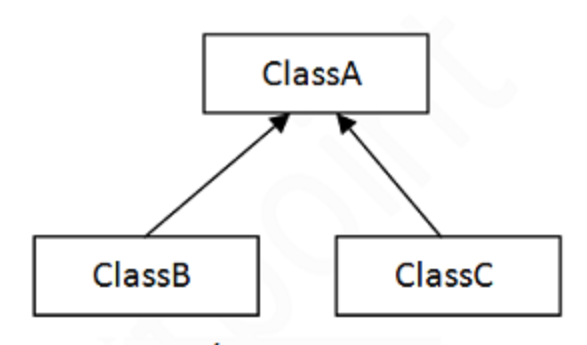
d.bark();

d.eat();

}

}

Kế thừa thứ bậc



**class** Animal\_2 {

**void** eat() {

System.***out***.println("eating...");

}

}

**class** Dog\_2 **extends** Animal\_2 {

**void** bark() {

System.***out***.println("barking...");

}

}

**class** Cat **extends** Animal\_2 {

**void** meow() {

System.***out***.println("meowing...");

}

}

**public** **class** TestInheritance3 {

**public** **static** **void** main(String args[]) {

Cat c = **new** Cat();

c.meow();

c.eat();

// c.bark(); // compile error

}

**Từ khóa this trong java** là một biến tham chiếu được sử dụng để tham chiếu tới đối tượng của lớp hiện tại.

**Từ khóa super trong java** là một biến tham chiếu được sử dụng để tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất.

* + Từ khóa super được sử dụng để tham chiếu trực tiếp đến biến instance của lớp cha gần nhất.
  + **class** Vehicle {
  + **int** speed = 50;
  + }
  + **public** **class** Bike2 **extends** Vehicle {
  + **int** speed = 100;
  + **void** display() {
  + System.out.println(**super**.speed);//in speed của lớp Vehicle
  + }
  + **public** **static** **void** main(String args[]) {
  + Bike2 b = **new** Bike2();
  + b.display();
  + }
  + }
  + super() được sử dụng để gọi trực tiếp Constructor của lớp cha.
  + **class** Vehicle {
  + Vehicle() {
  + System.out.println("Vehicle is created");
  + }
  + }
  + **class** Bike2 **extends** Vehicle {
  + Bike2() {
  + **super**();//gọi Constructor của lớp cha
  + System.out.println("Bike is created");
  + }
  + **public** **static** **void** main(String args[]) {
  + Bike2 b = **new** Bike2();
  + }
  + }
  + Từ khóa super được sử dụng để gọi trực tiếp phương thức của lớp cha.
  + **class** Person {
  + **void** message() {
  + System.out.println("welcome");
  + }
  + }
  + **public** **class** Student16 **extends** Person {
  + **void** message() {
  + System.out.println("welcome to java");
  + }
  + **void** display() {
  + message();// gọi phương thức message() của lớp hiện tại
  + **super**.message();// gọi phương thức message() của lớp cha
  + }
  + **public** **static** **void** main(String args[]) {
  + Student16 s = **new** Student16();
  + s.display();
  + }
  + }

Override là hiện tượng 1 phương thức thuộc lớp cha được định nghĩa lại ở lớp con.

* Phương thức override và được override phải có chung kiểu trả về, tên phương thức và danh sách tham số.
* Override chỉ xảy ra giữa các lớp có quan hệ kế thừa.
* **class** Shape {
* **void** draw() {
* System.***out***.println("drawing...");
* }
* }
* **class** Rectangle **extends** Shape {
* **void** draw() {
* System.***out***.println("drawing rectangle...");
* }
* }
* **class** Circle **extends** Shape {
* **void** draw() {
* System.***out***.println("drawing circle...");
* }
* }
* **class** Triangle **extends** Shape {
* **void** draw() {
* System.***out***.println("drawing triangle...");
* }
* }
* **class** TestPolymorphism4 {
* **public** **static** **void** main(String args[]) {
* Shape s;
* s = **new** Rectangle();
* s.draw();
* s = **new** Circle();
* s.draw();
* s = **new** Triangle();
* s.draw();
* }
* }

1 phương thức final hoặc static không thể được override,

Constructor được gọi tại thời điểm tạo đối tượng. Nó khởi tạo các giá trị để cung cấp dữ liệu cho các đối tượng, đó là lý do tại sao nó được gọi là constructor.

### Có 2 kiểu constructor

### Constructor không tham số

class ConstructorDemo

{

        ConstructorDemo(){

        System.out.println("Đây là hàm constructor");

    }

    public static void main(String args[]){

        ConstructorDemo a = new ConstructorDemo();

    }

}

### Constructor có tham số

class ParaConst {

    int id;

    String name;

    ParaConst(int i, String n) {

        id = i;

        name = n;

    }

    void display(){

        System.out.println(id + " " + name);

    }

    public static void main(String args[]) {

                ParaConst s1 = new ParaConst(121, "ITSol");

        ParaConst s2 = new ParaConst(232, "KHTN");

        s1.display();

        s2.display();

    }

}

Sủa đổi quyền truy cập của lớp con

Lớp con phải có quyền truy cập lớn hơn lớp cha mới có quyền sửa đổi

**class** A {

**protected** **void** method()

{

System.***out***.println("Hello");

}

}

**public** **class** B **extends** A {

**public** **void** method()

{

System.***out***.println("He");

}

**public** **static** **void** main(String args[])

{

B b = **new** B();

b.method();

}

}

Out : he

**class** A {

**public** **void** method()

{

System.***out***.println("Hello");

}

}

**public** **class** B **extends** A {

**protected** **void** method()

{

System.***out***.println("He");

}

**public** **static** **void** main(String args[])

{

B b = **new** B();

b.method();

}

}

Lỗi