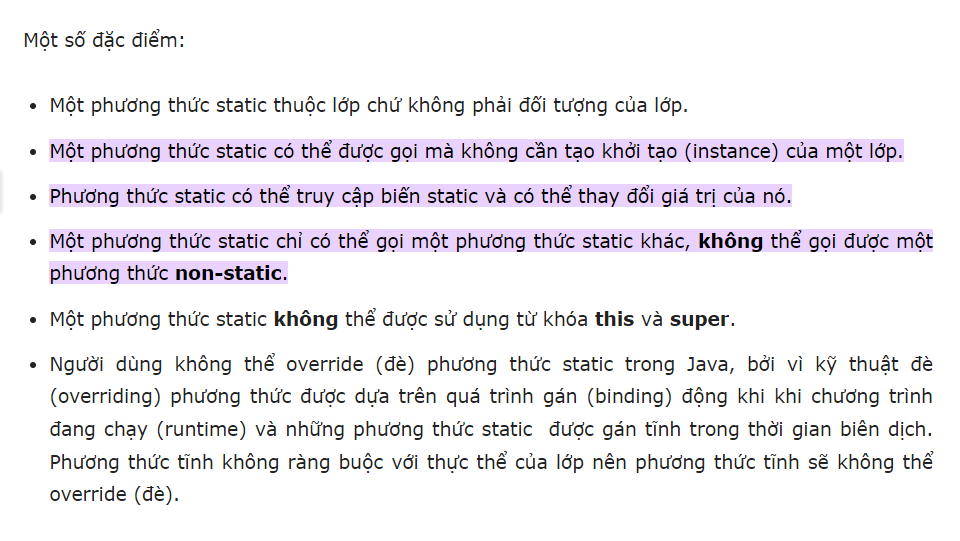
**Constructor:**

* Khi khởi tạo đối tượng thì lớp Constructor tự động được gọi
* Constructor lớp con luôn gọi tới Constructor lớp cha
* Nếu không có Constructor được tạo thì Constructor mặc định được tự động tạo ra, ngược lại nếu đã có sẵn Constructor thì mặc định sẽ không được sinh ra.
* Có thể có nhiều Constructor với nhiều kiểu tham số
* C kế thừa B kế thừa A 🡪 gọi constructor A 🡪 B 🡪 (Bởi vì từ khóa super() luôn được tự tạo ở đầu của các class con)

**Static:**

- biến static là biến chung của các đối tượng, biến static thay đổi thì đổi ở mọi nơi, --> Math.pow(int x,int y), static ko được coi là thuộc tính vì ko thuộc đối tượng nào



**Inheritance:**

- kế thừa Students **extends** Person :...

- super.

- this.

- Tất cả các lớp trong Java đều được kế thừa từ lớp Object

+ Object có các phương thức hashCode(),toString(),getClass(),equals(); khi gọi tới các PT này, ta không cần xài super;

+ toString() : biểu diễn đối tượng dưới dạng xâu;

Lưu ý: Lúc định nghĩa lại toString() nhớ @Override

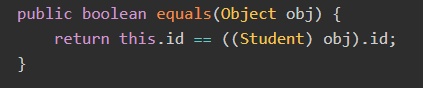
Lệnh System.out.println() đã tự động gọi đến PT toString nên không cần phải ghi ra s.toString mà chỉ cần System.out.println(s) thì sẽ tự in ra thông tin của **ĐỐI TƯỢNG**

* So sánh 2 đối tượng trong cùng 1 lớp:

+ Không thể dung == vì nó so sánh địa chỉ => false;

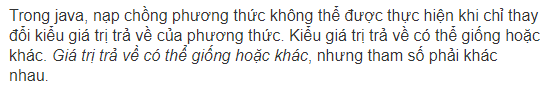
+ Dùng PT equals để so sánh

Ex: So sánh ID của 2 student:



\*Object obj vì đây là PT lấy từ class Object, khi override phải giữ nguyên tham số, (Student) obj ép obj thành class Student.

**Overload (Nạp chồng phương thức) (Compile time)**

****

**@Override (Ghi đè) (Runtime)**

- Phương thức override và được override phải có chung kiểu trả về,tên và danh sách tham số , vdu trên là chung kiểu void, tên display và không có tham số

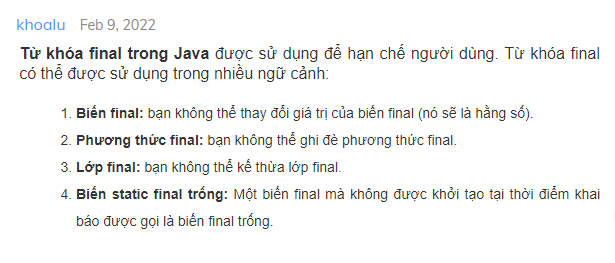
- Override chỉ xảy ra giữa các lớp có quan hệ kế thừa

- PT gốc / PT ba không bị thay đổi, chỉ thay đổi ở PT con --> gọi PT ba = super , PT con = this

- Các phương thức static không thể bị override

-------------------------------------------------------------------------------------------------------------

**TỪ KHÓA “final”:**



-------------------------------------------------------------------------------------------------------------

- private-->default-->protected-->public

- Muốn **OVERRIDE PHƯƠNG THỨC (Thuộc tính không ảnh hưởng)** thì phạm vi truy cập cái cái con >= cái cha aka **không được làm giảm phạm vi của nó (nhớ là chỉ có phương thức, còn thuộc tính thì không)**

\*ex:

protected void display()

@Override

default void display()/void display() : Không hợp lệ , default<protected

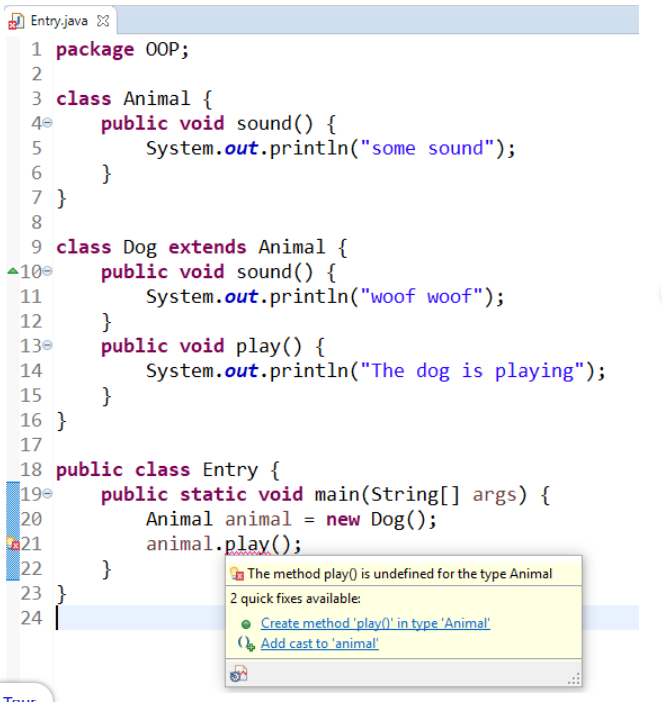
public void display(): Hợp lệ

* **Up-casting và down-casting:**
* **Up-casting:**

Đối tượng lớp con bị ép thành lớp cha (lớp cha tham chiếu tới đối tượng lớp con)

* Khi đó chỉ có thể gọi tới thuộc tính và phương thức của lớp cha
* Nếu lớp con ghi đè lên phương thức lớp cha thì lúc gọi sẽ gọi phương thức ở lớp con (Override)

Ex:



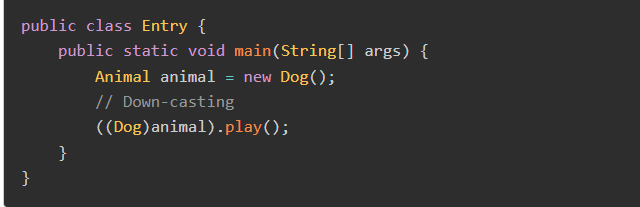
Đối tượng Dog bị ép về Animal 🡪 Phuong thức play() chỉ có ở dog chứ không có animal (Không gọi được) 🡪 Phương thức sound bị ghi đè ở lớp Dog

**Ứng dụng :** Thừa hưởng đời cha và được sửa lại theo tính chất đời con

* **Down-Casting:**

**Note**: Sử dụng downcasting khi chúng ta cần sử dụng các tính chất riêng của subclass mà ở supperclass không có.

* Ngược lại với up-casting, lớp cha bị ép về lớp con
* Sử dụng được các phương thức của lớp con



* Dùng instanceof để kiểm tra đối tượng này có phải thuộc (class,subclass,interface) của cái kia không;

Class Parent()

Class Child extends Parent()

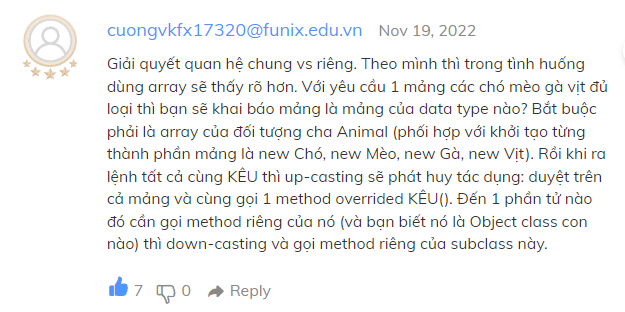
Child cobj = new Child();

* cobj instanceof Child = true ; //cobj là đối tượng của class Child
* cobj instanceof Parent = true; //cobj thuộc Child là subclass của Parent
* cobj instanceof Object = true;//cobj 🡪Child 🡪 Parent 🡪 Object

Fruit fobj = new Orange();

* fobj instanceof Apple = false // cobj là đối tượng của orange
* fobj instanceof Orange = true// Như trên

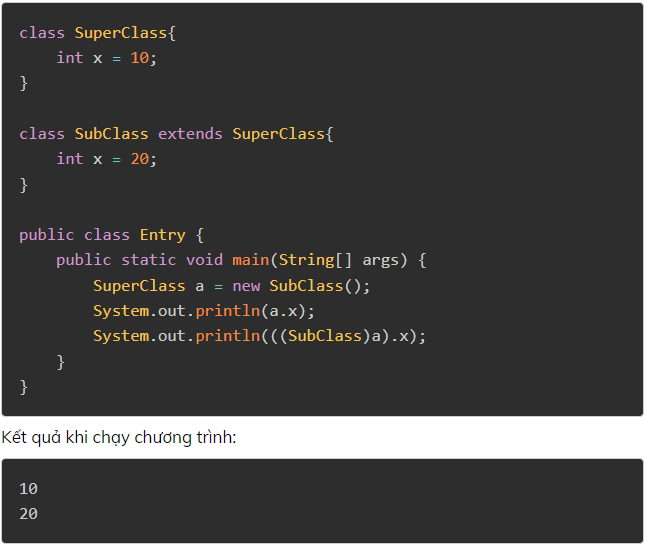
Ứng dụng down-casting:



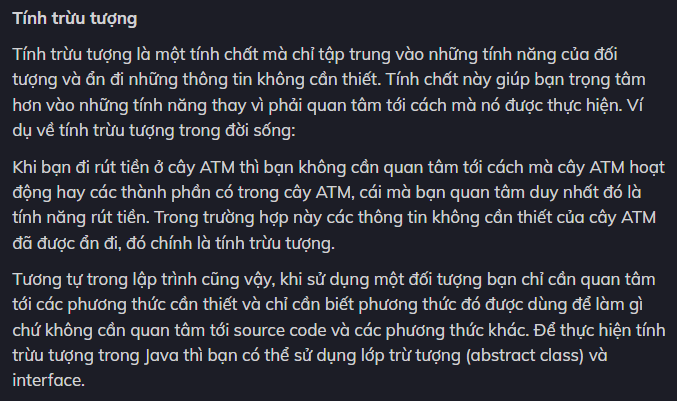
Ứng dụng

**Polymorphism (Đa hình)**

* One thing in many form
* 3 hình thức: nạp chồng phương thức, override, các đối tượng đa hình (Up – down casting);
* **Variable hiding (Giấu biến) :** Xảy ra khi lớp con khai báo thuộc tính có tên và kiểu giống lớp cha, khi đó thuộc tính của lớp con sẽ bị ẩn đi

****

**Abstraction (Trừu tượng)**

****

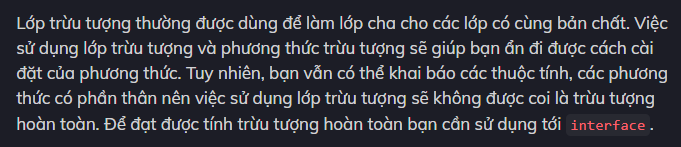
Khởi tạo bằng từ khóa “abstract”: abstract class Animal{}

Đặc điểm của lớp trừu tượng:

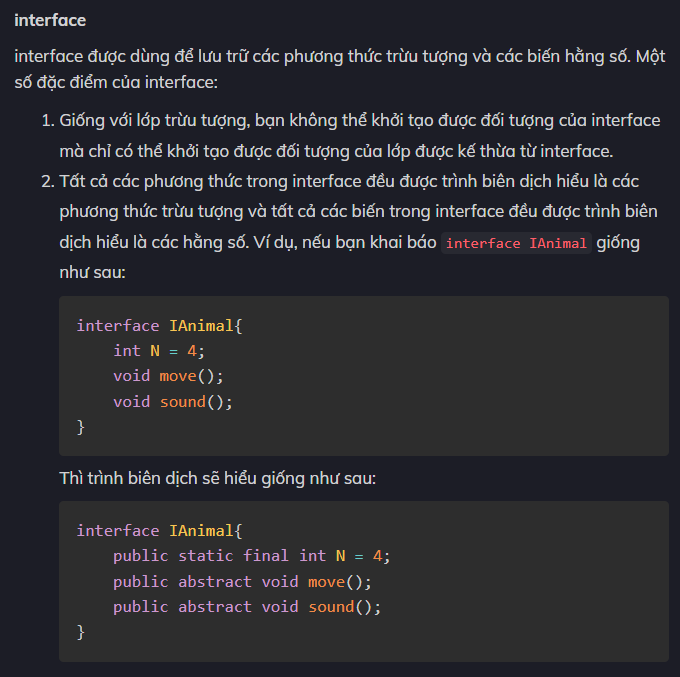
1. Không thể khởi tạo đối tượng thuộc lớp trừu tượng, chỉ khởi tạo được đối tượng từ **LỚP CON** của lớp trừu tượng.
2. Lớp trừu tượng **CÓ THỂ CÓ** các phương thức trừu tượng là pt mà chỉ có phần khai báo, không có phần thân;

Ex: public abstract void makeSound();

1. Nếu 1 lớp được kế thừa từ lớp trừu tượng thì nó phải ghi đè lên **CHỈ TẤT CẢ** **PHƯƠNG THỨC TRỪU TƯỢNG**.



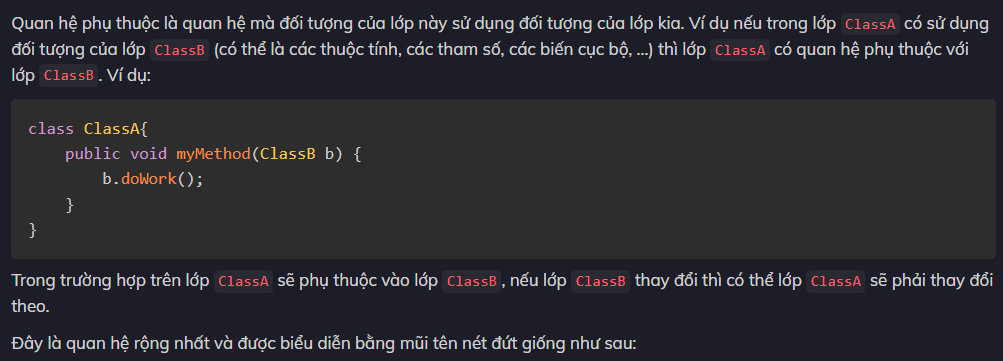
**Interface: Trừu tượng hoàn toàn**



1. Một lớp có thể thừa hưởng nhiều interface bằng từ khóa **“implements”**

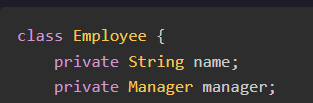
**Relationship**

* **Quan hệ phụ thuộc:**

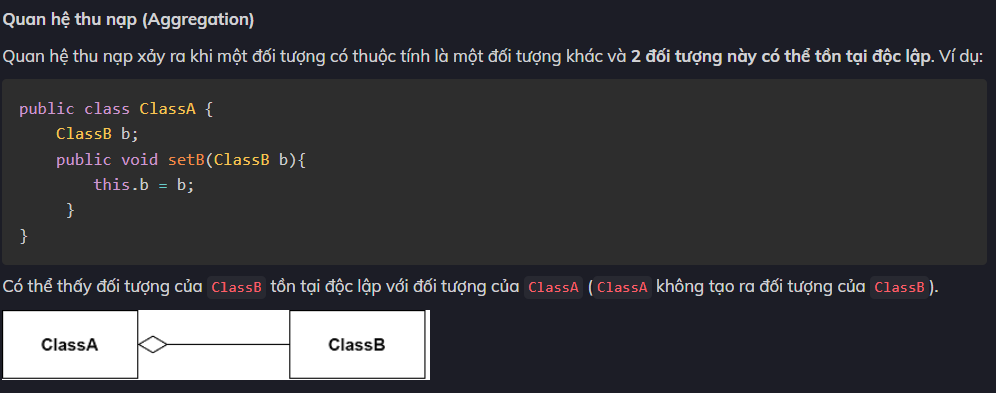


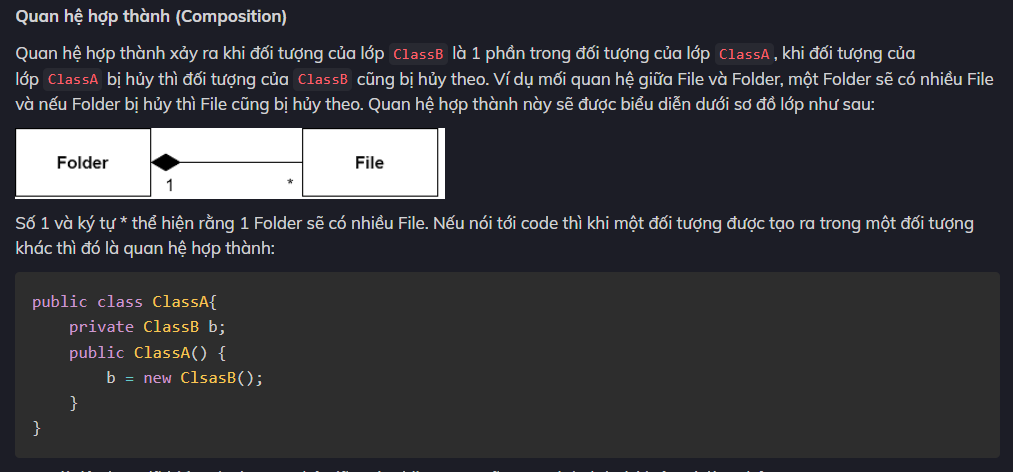
* **Quan hệ kết hợp(Association):**

\*Khi có thuộc tính là đối tượng của một lớp khác



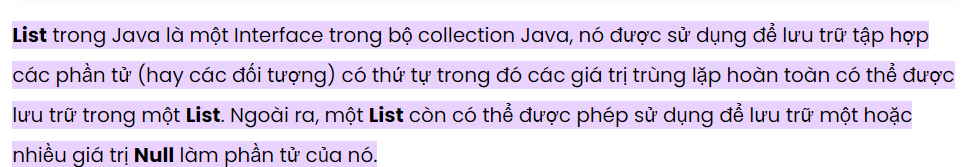
+ Chia làm 2 loại : **Quan hệ thu nạp (Aggregation) && Quan hệ hợp thành (Composition)**

****

****

**\*b=new ClassB(); ảnh sai**

**List trong Java:**

****

**List là interface 🡪 ArrayList là kiểu triển khai interface**

**KIỂU : ARRAYLIST**

**Khai báo:** ArrayList<T> listName = new ArrayList<>() ; trong đó T là kiểu dữ liệu của các phần tử (không được là kiểu nguyên thủy)

**Khắc phục nhược điểm** của việc them và bớt phần tử của Array

**Các phương thức:**

* **.add(Đối Tượng) : thêm vào cuối list**
* **.add(index,đối tượng): them vào vị trí index**
* **.get(index): lấy ra đối tượng ở vị trí index**
* **ToArray(tên mảng): biến list thành mảng và lưu vào mảng được truyền**
* **….v….v**

**LỚP BAO (DỮ LIỆU NGUYÊN THỦY)**

* Mỗi kiểu dữ liệu nguyên thủy đều có 1 lớp bao tương ứng
* 8 kiểu: integer,short,long,float,double,byte, character,boolean
* Khi muốn tạo list các số nguyên với **ArrayList<int> numbers**,ta gặp lỗi và lúc này ta cần tới lớp bao: **ArrayList<Integer> numbers**;
* Khi ta thực hiện**: numbers.add(5)**, chương trình tự động gói giá trị này lại thành 1 đối tượng của lớp bao**: numbers.add(Integer.valueOf(5));**
* **Quá trình này gọi là autoboxing**
* Khi muốn **lấy giá trị**, ta thực hiện **int a=numbers.get(i);** chương trình sẽ tự động thành **int a= numbers.get(i).intValue();**
* **Quá trình này gọi là unboxed;**
* Integer,Float,Short,Double,Byte,Character, Boolean,Long
* **Ứng dụng :** Khai báo biến như 1 đối tượng của lớp bao để sử dụng những phương thức có sẵn do chương trình cung cấp

**Interface Comparable (Dùng để sắp xếp theo tiêu chí)**

**\*import java.util.Collections;**

**Ex:** public class Student implement Comparable<Student>{}

Trong Interface này chứa 1 phương thức duy nhất là compareTo()

Muốn sắp xếp theo tiêu chí, ta phải override lại phương thức này

**@Override**

**public int compareTo(Student student) {**

**return this.getName().compareTo(student.getName());**

**}**

* **Sắp xếp tên theo tang dần.**

Sau đó dùng Collections.sort(Tên danh sách) để sắp xếp;