**extends** : class kế thừa các class, interface kết thừa interface  
**implements** : class kế thừa các interface.

**Trừu tương**: quan tâm đến các đối tượng chó, mèo, hay con vật nào khác là các con vật, có tên và phát ra tiếng kêu

**đóng gói:** đóng gói tên và phương thức kêu, khi muốn gọi thì phải hỏi (getter settter)

**thừa kế:** chó, mèo thừa kế animal

**đa hình**: tùy theo chó, mèo sẽ có tiếng kêu khác nhau

public abstract class animal{

private String name;

public animal(String name){

this.name=name;

}

public abstract void say();

}

public class dog extends animal{

public cat(String name){

super(name);

}

@override

public void say(){gaugau};

}

animal a = new dog(‘nh’);

1. say;

animal b = new cat(‘nh’);

1. say;

**Tính kế thừa Inheritance**: class con được hưởng các phương thức và thuộc tính của lớp cha, ko phải ghi lại

VD: Class cha là hình, có đỉnh và các cạnh, class con là hcn, hv,...

**Câu hỏi: Tại sao đa kế thừa không được support trong java?**

Để giảm thiểu sự phức tạp và đơn giản hóa ngôn ngữ, đa kế thừa không được support trong java.

**Tính đóng gói (encapsulation)** . Mục đích chính của đóng gói trong java là giảm thiểu mức độ phức tạp phát triển phần mềm.

## Lợi ích của đóng gói trong java

* Trừu tượng là ẩn đi cài đặt chi tiết còn đóng gói là gói code và data vào một khối duy nhất.
* Bạn có thể tạo lớp **read-only** hoặc **write-only** bằng việc cài đặt phương thức setter hoặc getter.
* Nếu một đối tượng khác muốn điều gì từ đối tượng này thì nó phải 'hỏi'. Theo thuật ngữ lập trình hướng đối tượng, phải gửi một thông điệp đến một đối tượng khác để hỏi về tuổi. Theo thuật ngữ Java, phải gọi một phương thức của đối tượng khác để nó trả lại kết quả là tuổi.

**Đa hình trong java (Polymorphism)** trong từng trường hợp, hoàn cảnh khác nhau thì đối tượng có hình thái khác nhau tùy thuộc vào từng ngữ cảnh

* Đó chính là sự đa hình: các đối tượng làm việc theo cách riêng của chúng.
* Có hai kiểu của đa hình trong java, đó là đa hình lúc biên dịch (compile) và đa hình lúc thực thi (runtime). Chúng ta có thể thực hiện đa hình trong java bằng cách nạp chồng phương thức và ghi đè phương thức.

**Đa hình lúc runtime**là quá trình gọi phương thức đã được ghi đè trong thời gian thực thi chương trình. Trong quá trình này, một phương thức được ghi đè được gọi thông qua biến tham chiếu của một lớp cha.

**Một ví dụ về đa hình trong thực tế**. Ta có 3 con vật: chó, mèo, lợn. Cả 3 con vật này đều là động vật (lớp cha). 3 con vật kế thừa (extends) lớp động vật, override phương thức kê. Nhưng khi ta bảo cả 3 động vật kêu thì con chó sẽ kêu gâu gâu, con mèo sẽ kêu meo meo và con heo sẽ kêu ẹt ẹt.

## Tính trừu tượng trong java

Tính trừu tượng là một tiến trình ẩn các cài đặt chi tiết và chỉ hiển thị tính năng tới người dùng. Tính trừu tượng cho phép bạn loại bỏ tính chất phức tạp của đối tượng bằng cách chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình.

### **6. Sự khác nhau giữa nạp chồng và ghi đè là gì?**

|  |  |  |
| --- | --- | --- |
| No. | Nạp chồng phương thức (overloading) | Ghi đè phương thức (overriding) |
| 1) | Nạp chồng phương thức được sử dụng để giúp code của chương trình **dễ đọc hơn**. | Ghi đè phương thức được sử dụng để  cung cấp cài đặt cụ thể  cho phương thức được khai báo ở lớp cha. |
| 2) | Nạp chồng được thực hiện bên trong một class. | Ghi đè phương thức xảy ra trong 2 class  có quan hệ kế thừa. |
| 3) | Nạp chồng phương thức thì tham số phải khác nhau. | Ghi đè phương thức thì tham số phải giống nhau. |
| 4) | Nạp chồng phương thức là ví dụ về đa hình lúc biên dịch. | Ghi đè phương thức là ví dụ về đa hình lúc runtime. |
| 5) | Trong java, nạp chồng phương thức không thể được thực hiện khi chỉ thay đổi kiểu giá trị trả về của phương thức. Kiểu giá trị trả về có thể giống hoặc khác. Giá trị trả về có thể giống hoặc khác, nhưng tham số phải khác nhau. | Giá trị trả về phải giống nhau |

### **Có thể khai báo một phương thức của interface với từ khóa static không?**

Không. Vì mặc định các phương thức của một interface là abstract, từ khóa static và abtract không thể được sử dụng chung với nhau.

### **Sự khác nhau giữa lớp abstract và interface là gì?**

Abstract: là một class cha trừu tượng cho tất cả các class có cùng bản chất.

using System;

namespace oop

{

abstract class Animal

{

public abstract void Speak();

public virtual void Eat(string something)

{

Console.WriteLine("Eat " + something);

}

}

}

using System;

namespace oop

{

class Dog : Animal

{

public override void Speak()

{

Console.WriteLine("Dog speaks go go");

}

}

}

using System;

namespace oop

{

class Cat : Animal

{

public override void Speak()

{

Console.WriteLine("Cat speaks meo meo");

}

}

}

* **Abstract class**: Giống như demo trên bạn có thể hiểu khi định nghĩa một đối tượng có những chức năng A,B,C trong đó tính năng A,B chắc chắn sẽ thực thi theo cách nào đó, còn tính năng C phải tùy thuộc vào đối tượng cụ thể là gì, như đối tượng Dog, Cat tuy chúng đều có thể phát ra âm thanh nhưng âm thanh là khác nhau. Vì vậy method Speak() là abstract method để **chỉ ra rằng tính năng này còn dang dở chưa rõ thực thi,** các lớp extend phải hoàn thành nốt tính năng này, còn những tính năng đã hoàn thành vẫn sử dụng như bình thường đây là những tính năng chung.

**Interfacce:** là một **chức năng** mà bạn có thể thêm và bất kì class nào; khuông mẫu, để các lớp con implement. Sử dụng khi thừa kế từ nhiều interface

* **Interface** : **Khi bạn muốn tạo dựng một bộ khung chuẩn gồm các chức năng mà những module hay project cần phải có. Giống như sau khi nhận requirement của khách hàng về team ngồi với nhau và phân tích các đầu mục các tính năng của từng module, sau đó triển khai vào code viết các interface như đã phân tích,để các bạn dev có thể nhìn vào đó để thực hiện đủ các tính năng (khi đã implement rồi thì không sót một tính năng nào ^^).**

|  |  |
| --- | --- |
| Lớp Abstract | Interface |
| 1)Abstract/ non-abstract method -> định nghĩa được tại chỗ | abstract method -> chỉ có thể định nghĩa trong các lớp con |
| 3)Có constructor, static method, main | Ko có |
| Biến final, static, non-final, non-static | Final,static |
| 5) Một lớp chỉ có thể extends một lớp abstract. | Một lớp thể implement nhiều interface. |
| 6/ Access modifier: public, private, protected | Luôn là **public abstract** |

Nếu chúng ta thêm một phương thức mới vào interface khi đó chúng ta phải lùng sục hết tất cả các lớp nào đang thực thi interface và định nghĩa thực thi cho phương thức mới.