**1. Kế thừa là gì? Nó giúp ích gì trong lập trình hướng đối tượng?**

**Kế thừa (Inheritance)** là một trong bốn nguyên tắc chính của lập trình hướng đối tượng (OOP), cho phép một lớp **(lớp con - subclass)** kế thừa các thuộc tính và phương thức của một lớp khác **(lớp cha - superclass)**.

**Lợi ích của kế thừa:**

* **Tái sử dụng mã nguồn**: Tránh việc viết lại mã bằng cách kế thừa từ lớp cha.
* **Mở rộng chức năng**: Lớp con có thể sử dụng hoặc ghi đè các phương thức của lớp cha để phù hợp với nhu cầu.
* **Dễ bảo trì và mở rộng**: Khi cần thay đổi logic chung, chỉ cần cập nhật trong lớp cha thay vì sửa từng lớp riêng lẻ.

**2. Sự khác biệt giữa kế thừa và đóng gói (Encapsulation) trong Java**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Đặc điểm | Kế thừa (Inheritance) | |  | | --- | |  |  |  | | --- | | Đóng gói (Encapsulation) | |
| Khái niệm | Cơ chế cho phép lớp con kế thừa các thuộc tính và phương thức của lớp cha. | Cơ chế bảo vệ dữ liệu bằng cách ẩn chi tiết bên trong lớp và chỉ cho phép truy cập thông qua các phương thức cụ thể. |
| Mục đích | Tái sử dụng mã nguồn, giảm trùng lặp. | Bảo vệ dữ liệu, kiểm soát quyền truy cập. |
| Từ khóa chính | extends (kế thừa từ lớp cha). | private, protected, public (kiểm soát truy cập). |
| Ứng dụng | Dùng khi nhiều lớp có điểm chung và cần kế thừa. | Dùng để giới hạn quyền truy cập và bảo vệ dữ liệu bên trong lớp. |

**3. Ví dụ về kế thừa trong Java**

class Animal {

String name;

public void eat() {

System.out.println(name + " đang ăn...");

}

}

class Dog extends Animal {

public void bark() {

System.out.println(name + " sủa: Gâu gâu!");

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.name = "Buddy"; // Kế thừa thuộc tính từ Animal

myDog.eat(); // Kế thừa phương thức từ Animal

myDog.bark(); // Phương thức riêng của Dog

}

}

**4. Lợi ích và hạn chế của kế thừa trong Java**

**Lợi ích của kế thừa:**

**Tái sử dụng mã nguồn**: Tránh viết lại mã khi các lớp có nhiều điểm chung.  
 **Dễ bảo trì và mở rộng**: Thay đổi ở lớp cha sẽ tự động áp dụng cho lớp con.  
 **Dễ dàng mở rộng chức năng**: Lớp con có thể ghi đè (@Override) phương thức để có hành vi riêng.

**Hạn chế của kế thừa:**

**Thiếu linh hoạt**: Lớp con bị ràng buộc với lớp cha, nếu lớp cha thay đổi có thể ảnh hưởng đến nhiều lớp con.  
 **Gây khó hiểu khi kế thừa quá nhiều tầng**: Nếu kế thừa quá sâu (đa cấp), sẽ khó theo dõi logic chương trình.  
 **Không thể kế thừa từ nhiều lớp cha**: Java không hỗ trợ đa kế thừa (một lớp chỉ có thể kế thừa một lớp cha).