

HANDOUT BUỔI 8-9: TÍNH KẾ THỪA (INHERITANCE)

Mục tiêu:

- Hiểu được khái niệm kế thừa trong OOP.
- Viết được lớp kế thừa trong một tình huống cụ thể.
- Hiểu và áp dụng các quy tắc kế thừa trong Java.
- Biết cách sử dụng super để gọi constructor hoặc phương thức của lớp cha.
- Thực hành lập trình với kế thừa và ghi đè phương thức (overriding).

PHẦN 1: LÝ THUYẾT VỀ TÍNH KẾ THỪA

✔ Giới thiệu và đặt vấn đề

Mục tiêu: Giúp sinh viên hiểu vì sao kế thừa quan trọng trong lập trình hướng đối tượng (OOP).

I. Dẫn dắt vấn đề thông qua tình huống thực tế

Trong lập trình hướng đối tượng, **kế thừa** (inheritance) là một cơ chế quan trọng giúp tái sử dụng mã nguồn, giảm thiểu sự dư thừa và nâng cao hiệu quả bảo trì phần mềm. Để hiểu rõ hơn về vai trò của kế thừa, chúng ta sẽ phân tích hai tình huống thực tế sau đây.

1. Tình huống 1: Quản lý nhân sự trong doanh nghiệp

Xét một hệ thống quản lý nhân sự trong doanh nghiệp, nơi có nhiều loại nhân viên với vai trò khác nhau. Các loại nhân viên có một số đặc điểm chung, nhưng cũng có những thuộc tính riêng biệt:

- **Nhân viên chung (Employee):** Mỗi nhân viên đều có các thông tin cơ bản như họ tên, mã nhân viên, mức lương cơ bản.
- **Quản lý (Manager):** Bên cạnh các thông tin chung, quản lý còn có khoản thưởng (bonus).
- **Lập trình viên (Developer):** Ngoài thông tin cơ bản, lập trình viên có thêm số giờ làm thêm (overtimeHours) để tính lương bổ sung.

Phân tích vấn đề:

Nếu không sử dụng kế thừa, lập trình viên sẽ phải định nghĩa lại các thuộc tính và phương thức chung trong từng lớp riêng biệt (Manager và Developer). Điều này dẫn đến việc lặp lại mã nguồn, gây khó khăn trong quá trình bảo trì và mở rộng hệ thống.

Nếu cần sửa đổi một thuộc tính chung (ví dụ: thay đổi cách tính lương cơ bản), chúng ta phải cập nhật nhiều đoạn mã khác nhau, làm tăng nguy cơ sai sót.

Cách giải quyết với kế thừa:

Sử dụng kế thừa, chúng ta có thể định nghĩa một lớp cha (Employee) chứa các thuộc tính và phương thức chung. Các lớp con (Manager, Developer) sẽ kế thừa từ lớp cha và mở rộng thêm các thuộc tính riêng của từng loại nhân viên. Nhờ đó, mã nguồn trở nên gọn gàng hơn, dễ bảo trì và có tính mở rộng cao hơn.

2. Tình huống 2: Hệ thống quản lý phương tiện giao thông

Một hệ thống quản lý phương tiện giao thông cần xử lý nhiều loại phương tiện khác nhau, nhưng chúng có một số đặc điểm chung:

- **Lớp phương tiện chung (Vehicle):** Chứa các thuộc tính chung như hãng sản xuất (brand), tốc độ tối đa (speed), loại nhiên liệu (fuelType).
- **Lớp ô tô (Car):** Kế thừa từ Vehicle và bổ sung thuộc tính số cửa (numberOfDoors).
- **Lớp xe đạp (Bike):** Kế thừa từ Vehicle và bổ sung thuộc tính có bàn đạp hay không (hasPedals).

Phân tích vấn đề:

Nếu không áp dụng kế thừa, mỗi loại phương tiện (Car, Bike) sẽ cần có một đoạn mã riêng để xử lý các thuộc tính chung như brand, speed, fuelType. Điều này dẫn đến sự dư thừa và trùng lặp mã nguồn, gây khó khăn khi cần cập nhật thông tin chung cho tất cả các phương tiện (ví dụ: thay đổi cách hiển thị thông tin phương tiện).

Cách giải quyết với kế thừa:

Bằng cách sử dụng kế thừa, chúng ta có thể tạo một lớp cha Vehicle chứa các thuộc tính và phương thức chung, sau đó cho Car và Bike kế thừa từ Vehicle. Khi đó, cả hai loại phương tiện đều có thể sử dụng lại mã nguồn từ lớp cha mà không cần định nghĩa lại từ đầu. Điều này giúp mã nguồn dễ đọc, dễ bảo trì và tiết kiệm thời gian phát triển.

Hai tình huống trên cho thấy rằng việc không sử dụng kế thừa sẽ dẫn đến mã nguồn dư thừa, trùng lặp và khó bảo trì. Khi có sự thay đổi về thuộc tính hoặc phương thức chung, lập trình viên phải cập nhật nhiều phần mã khác nhau, làm tăng nguy cơ sai sót và mất nhiều thời gian.

Kế thừa không chỉ giúp tổ chức mã nguồn một cách hợp lý hơn mà còn hỗ trợ việc mở rộng hệ thống một cách dễ dàng. Khi cần bổ sung một loại đối tượng mới (ví dụ: thêm

lớp Intern trong quản lý nhân sự hoặc lớp Truck trong hệ thống phương tiện giao thông), chúng ta chỉ cần tạo một lớp mới kế thừa từ lớp cha, thay vì viết lại toàn bộ mã từ đầu.

II. Khái niệm và cú pháp kế thừa trong lập trình hướng đối tượng

Kế thừa (*inheritance*) là một trong những đặc trưng quan trọng của lập trình hướng đối tượng (OOP), cho phép một lớp (*class*) tái sử dụng thuộc tính và phương thức của một lớp khác. Đây là cơ chế giúp tăng tính tổ chức của mã nguồn, giảm thiểu trùng lặp và nâng cao khả năng mở rộng của chương trình.

1. Định nghĩa kế thừa

Kế thừa là cơ chế trong lập trình hướng đối tượng cho phép một lớp con (*child class*) có thể sử dụng lại các thuộc tính và phương thức của một lớp cha (*parent class*), đồng thời có thể mở rộng hoặc thay đổi các phương thức đó theo nhu cầu riêng.

Ví dụ: Một lớp Vehicle chứa các thuộc tính chung của phương tiện giao thông như hãng sản xuất, tốc độ tối đa, loại nhiên liệu. Lớp Car và Bike có thể kế thừa từ Vehicle để sử dụng lại các thuộc tính này, đồng thời bổ sung những đặc điểm riêng biệt của từng loại phương tiện.

2. Lợi ích của kế thừa

Kế thừa mang lại nhiều lợi ích quan trọng trong thiết kế và triển khai phần mềm:

- **Giảm trùng lặp mã nguồn:** Các thuộc tính và phương thức chung chỉ cần khai báo một lần trong lớp cha, giúp tiết kiệm công sức viết mã và tránh dư thừa.
- **Tăng tính tái sử dụng mã:** Lớp con có thể sử dụng lại các thành phần của lớp cha mà không cần định nghĩa lại, giúp mã nguồn trở nên gọn gàng và hiệu quả hơn.
- **Dễ bảo trì và mở rộng chương trình:** Khi có thay đổi hoặc bổ sung trong lớp cha, tất cả các lớp con kế thừa đều có thể tự động cập nhật theo, giảm thiểu công sức chỉnh sửa.
- **Giúp tổ chức các lớp có quan hệ logic với nhau:** Kế thừa giúp mô hình hóa hệ thống theo cách tự nhiên hơn, phản ánh mối quan hệ giữa các thực thể trong thực tế.

3. Cú pháp kế thừa trong Java

Trong Java, để khai báo một lớp kế thừa từ một lớp khác, ta sử dụng từ khóa `extends`.

Cú pháp chung:

```
class Parent {  
    // Thuộc tính và phương thức của lớp cha  
}  
  
class Child extends Parent {  
    // Mở rộng lớp cha bằng các thuộc tính và phương thức mới  
}
```

Trong đó:

- Parent là lớp cha, chứa các thuộc tính và phương thức có thể được sử dụng lại.
- Child là lớp con, kế thừa các thành phần từ Parent và có thể bổ sung hoặc ghi đè phương thức của lớp cha.

Ví dụ minh họa:

```
class Parent {  
    String name;  
  
    void display() {  
        System.out.println("This is Parent class");  
    }  
}  
  
class Child extends Parent {  
    int age;  
  
    void showAge() {  
        System.out.println("Age: " + age);  
    }  
}
```

Trong đoạn mã trên:

- Lớp Parent có thuộc tính name và phương thức display().
- Lớp Child kế thừa Parent, ngoài các thuộc tính và phương thức của lớp cha, nó bổ sung thêm thuộc tính age và phương thức showAge().

Như vậy, khi một đối tượng của Child được tạo ra, nó có thể sử dụng cả phương thức display() từ Parent và phương thức showAge() riêng của nó.

4. Giải thích chi tiết về quyền truy cập trong kế thừa

Khi một lớp con kế thừa từ lớp cha, nó có thể truy cập các thành phần của lớp cha theo các quy tắc sau:

1. **Thuộc tính và phương thức có phạm vi truy cập public hoặc protected của lớp cha** đều có thể được sử dụng trong lớp con.
2. **Thuộc tính và phương thức có phạm vi truy cập private** trong lớp cha sẽ **không thể** truy cập trực tiếp từ lớp con. Nếu cần truy cập, chúng ta phải sử dụng phương thức getter hoặc setter.
3. **Lớp con có thể ghi đè (override) phương thức của lớp cha** để thay đổi hành vi của nó theo nhu cầu cụ thể.

Ví dụ về phạm vi truy cập trong kế thừa:

```
class Parent {  
    public String publicAttribute = "Public Attribute";  
    protected String protectedAttribute = "Protected Attribute";  
    private String privateAttribute = "Private Attribute";  
  
    public void publicMethod() {  
        System.out.println("Public Method");  
    }  
  
    protected void protectedMethod() {  
        System.out.println("Protected Method");  
    }  
}
```

```
private void privateMethod() {  
    System.out.println("Private Method");  
}  
}  
  
class Child extends Parent {  
    void showAttributes() {  
        System.out.println(publicAttribute); // Truy cập được  
        System.out.println(protectedAttribute); // Truy cập được  
        // System.out.println(privateAttribute); // Lỗi: Không thể truy cập  
    }  
  
    void callMethods() {  
        publicMethod(); // Truy cập được  
        protectedMethod(); // Truy cập được  
        // privateMethod(); // Lỗi: Không thể truy cập  
    }  
}
```

Trong đoạn mã trên:

- Lớp Child có thể truy cập publicAttribute và protectedAttribute, nhưng không thể truy cập privateAttribute của lớp cha Parent.
- Các phương thức publicMethod() và protectedMethod() có thể được gọi từ lớp con, nhưng privateMethod() thì không.

Kế thừa là một cơ chế quan trọng giúp tổ chức mã nguồn một cách hiệu quả hơn, giảm thiểu trùng lặp và tăng khả năng mở rộng của chương trình.

- Bằng cách sử dụng kế thừa, các thuộc tính và phương thức chung có thể được khai báo một lần trong lớp cha và tái sử dụng trong nhiều lớp con.

- Lớp con có thể bổ sung các đặc điểm riêng hoặc ghi đè các phương thức của lớp cha để phù hợp với nhu cầu cụ thể.
- Khi thiết kế hệ thống, cần xem xét phạm vi truy cập (public, protected, private) để đảm bảo an toàn dữ liệu và tính đúng đắn của mã nguồn.

III. Các kiểu kế thừa trong java

Kế thừa là một cơ chế quan trọng trong lập trình hướng đối tượng, nhưng không phải tất cả các hệ thống kế thừa đều giống nhau. Tùy thuộc vào cách tổ chức lớp và mối quan hệ giữa các lớp, kế thừa có thể được chia thành nhiều loại khác nhau. Hiểu rõ các loại kế thừa giúp lập trình viên thiết kế hệ thống một cách hiệu quả hơn, đảm bảo mã nguồn dễ bảo trì và mở rộng. Java hỗ trợ ba mô hình kế thừa chính:

1.1. Kế thừa đơn (Single Inheritance)

Kế thừa đơn là mô hình phổ biến nhất, trong đó một lớp con (*child class*) kế thừa từ một lớp cha (*parent class*). Điều này có nghĩa là lớp con có thể sử dụng tất cả các thuộc tính và phương thức của lớp cha, đồng thời có thể mở rộng hoặc ghi đè các phương thức đó.

Ví dụ minh họa:

Một lớp Car kế thừa từ lớp Vehicle.

```
class Vehicle {  
    String brand;  
  
    void start() {  
        System.out.println("Vehicle is starting...");  
    }  
}  
  
class Car extends Vehicle {  
    int numberOfDoors;  
}
```

Phân tích:

- Vehicle là lớp cha, chứa thuộc tính brand và phương thức start().

- Car kế thừa Vehicle, có thể sử dụng brand và start(), đồng thời bổ sung thuộc tính riêng numberOfDoors.
- Khi một đối tượng Car được tạo ra, nó có thể truy cập phương thức start() từ lớp cha.

Ưu điểm của kế thừa đơn:

- ✓ Dễ hiểu, dễ triển khai.
- ✓ Giúp tái sử dụng mã nguồn của lớp cha mà không cần viết lại.

Hạn chế của kế thừa đơn:

- ✗ Nếu hệ thống phát triển lớn hơn, một lớp cha duy nhất có thể không đủ linh hoạt để đáp ứng tất cả các yêu cầu của các lớp con.

1.2. Kế thừa đa cấp (Multilevel Inheritance)

Trong kế thừa đa cấp, một lớp kế thừa từ một lớp con khác, tạo thành một chuỗi kế thừa (*chain of inheritance*).

Ví dụ minh họa:

Lớp ElectricCar kế thừa từ Car, và Car kế thừa từ Vehicle.

```
class Vehicle {
    String brand;
}

class Car extends Vehicle {
    int numberOfDoors;
}

class ElectricCar extends Car {
    int batteryLife;
}
```

Phân tích:

- Vehicle là lớp gốc chứa thuộc tính brand.

- Car kế thừa từ Vehicle và mở rộng với thuộc tính numberOfDoors.
- ElectricCar kế thừa từ Car và bổ sung thuộc tính batteryLife.

Ưu điểm của kế thừa đa cấp:

- ✓ Cho phép mở rộng lớp mà không làm ảnh hưởng đến lớp gốc.
- ✓ Giúp mô hình hóa các hệ thống phân cấp phức tạp, như sinh vật học (Animal → Mammal → Dog).

Hạn chế của kế thừa đa cấp:

- ✗ Làm tăng độ phức tạp của hệ thống.
- ✗ Khi có thay đổi trong lớp cha, tất cả các lớp con trong chuỗi kế thừa đều bị ảnh hưởng.

1.3. Kế thừa thứ bậc (Hierarchical Inheritance)

Trong mô hình kế thừa thứ bậc, một lớp cha có nhiều lớp con kế thừa từ nó.

Ví dụ minh họa:

Lớp Vehicle có hai lớp con là Car và Bike.

```
class Vehicle {  
    String brand;  
}  
  
class Car extends Vehicle {  
    int numberOfDoors;  
}  
  
class Bike extends Vehicle {  
    boolean hasPedals;  
}
```

Phân tích:

- Vehicle là lớp cha chứa thông tin chung.
- Car kế thừa Vehicle, bổ sung thuộc tính numberOfDoors.

- Bike cũng kế thừa Vehicle, bổ sung thuộc tính hasPedals.

Ưu điểm của kế thừa thứ bậc:

- ✓ Giúp tổ chức mã nguồn theo nhóm đối tượng có chung đặc điểm.
- ✓ Dễ dàng mở rộng hệ thống bằng cách thêm các lớp con mới mà không ảnh hưởng đến các lớp hiện có.

Hạn chế của kế thừa thứ bậc:

- ✗ Khi lớp cha thay đổi, tất cả các lớp con phải thích nghi với sự thay đổi này.
- ✗ Nếu có quá nhiều lớp con, mã nguồn có thể trở nên khó quản lý.

2. Câu hỏi kiểm tra và giải thích

2.1. Java có hỗ trợ kế thừa đa lớp (Multiple Inheritance) không?

Không, Java **không hỗ trợ kế thừa đa lớp trực tiếp**. Điều này có nghĩa là một lớp con không thể kế thừa từ hai lớp cha cùng lúc.

Lý do:

- Nếu một lớp con kế thừa từ nhiều lớp cha và các lớp cha có các phương thức trùng tên, trình biên dịch sẽ không biết phương thức nào nên được sử dụng. Điều này gây ra **vấn đề mơ hồ (ambiguity problem)**.
- Để giải quyết vấn đề này, Java hỗ trợ **đa kế thừa thông qua interface**, cho phép một lớp có thể thực hiện nhiều interface cùng lúc.

Ví dụ về vấn đề mơ hồ khi sử dụng kế thừa đa lớp (trong một ngôn ngữ hỗ trợ đa kế thừa):

```
class A {  
    void show() {  
        System.out.println("Class A");  
    }  
}  
  
class B {  
    void show() {  
        System.out.println("Class B");  
    }  
}
```

```
    }  
}  
  
// Lỗi! Java không cho phép kế thừa từ hai lớp cha  
class C extends A, B {  
    // Lỗi: Không biết nên gọi show() từ A hay B  
}
```

Giải pháp: Sử dụng interface

```
interface A {  
    void show();  
}  
  
interface B {  
    void display();  
}  
  
class C implements A, B {  
    public void show() {  
        System.out.println("Class C implements A");  
    }  
  
    public void display() {  
        System.out.println("Class C implements B");  
    }  
}
```

Trong Java, interface giúp tránh xung đột phương thức và vẫn đảm bảo tính linh hoạt trong thiết kế.

2.2. Khi nào nên dùng kế thừa đa cấp thay vì kế thừa đơn?

- **Sử dụng kế thừa đơn** khi hệ thống đơn giản và không yêu cầu nhiều lớp con kế thừa từ nhau. Ví dụ, nếu chỉ có một mối quan hệ cha-con trực tiếp, kế thừa đơn là lựa chọn phù hợp.
- **Sử dụng kế thừa đa cấp** khi có một chuỗi quan hệ kế thừa logic. Ví dụ, trong mô hình sinh vật học: Animal → Mammal → Dog. Trong hệ thống xe cộ: Vehicle → Car → ElectricCar.

✅ Kế thừa đa cấp thích hợp khi:

- ✓ Có một chuỗi các đối tượng có quan hệ kế thừa rõ ràng.
- ✓ Cần mô hình hóa các hệ thống phân cấp phức tạp.
- ✓ Muốn mở rộng một lớp con mà không cần thay đổi lớp cha ban đầu.

❌ Kế thừa đa cấp không nên dùng khi:

- ✗ Cấu trúc phân cấp không rõ ràng, dễ gây nhầm lẫn.
- ✗ Việc thay đổi một lớp cha có thể ảnh hưởng đến toàn bộ hệ thống.

Kế thừa là một công cụ mạnh mẽ trong lập trình hướng đối tượng, nhưng cần được sử dụng một cách hợp lý. Việc lựa chọn loại kế thừa phù hợp sẽ giúp mã nguồn trở nên dễ đọc, dễ mở rộng và dễ bảo trì hơn. Trong phần tiếp theo, chúng ta sẽ tìm hiểu về các kỹ thuật nâng cao trong kế thừa, bao gồm ghi đè phương thức (method overriding) và sử dụng từ khóa super.

BÀI TẬP VẬN DỤNG

Bài tập 1: Quản lý viện phí bệnh nhân

Mục tiêu:

- Hiểu cách kế thừa giúp tái sử dụng mã nguồn.
- Biết cách ghi đè phương thức (override).
- Áp dụng từ khóa super để gọi constructor của lớp cha.

🔗 Mô tả bài toán

Một bệnh viện cần quản lý viện phí của bệnh nhân bao gồm bệnh nhân nội trú và bệnh nhân ngoại trú. Tất cả bệnh nhân có các thông tin chung như mã bệnh nhân, họ tên và tiền thuốc, nhưng mỗi loại bệnh nhân có cách tính viện phí riêng:

- BenhNhan (Lớp cha) có các thuộc tính: ma, hoTen, tienThuoc.
- BNNgoaiTru (Lớp con) có thêm phiKham, phiXetNghiem.
- BNNoiTru (Lớp con) có thêm phiNgay, soNgayNamVien, và quy tắc tính phụ phí chăm sóc.

Phân tích bài toán

1. Tạo lớp cha BenhNhan:

- Chứa thuộc tính chung (ma, hoTen, tienThuoc).
- Có phương thức vienPhi() trả về tiền thuốc.
- Có phương thức displayInfo() để in thông tin bệnh nhân.

2. Tạo lớp con BNNgoaiTru kế thừa BenhNhan:

- Thêm thuộc tính phiKham, phiXetNghiem.
- Ghi đè (override) phương thức vienPhi() để tính viện phí = tienThuoc + phiKham + phiXetNghiem.

3. Tạo lớp con BNNoiTru kế thừa BenhNhan:

- Thêm thuộc tính phiNgay, soNgayNamVien.
- Ghi đè phương thức vienPhi() để tính viện phí theo công thức:

$$\text{vienPhi} = \text{tienThuoc} * \text{soNgayNamVien} + \text{phiNgay} * \text{soNgayNamVien} + \text{phuPhi}$$

- Trong đó phuPhi là 50 nếu soNgayNamVien < 10, ngược lại là 100.

4. Trong main(), tạo danh sách bệnh nhân, hiển thị thông tin.

Code chương trình

```
// Lớp cha: BenhNhan
class BenhNhan {
    protected int ma;
    protected String hoTen;
    protected double tienThuoc;

    public BenhNhan(int ma, String hoTen, double tienThuoc) {
```

```
this.ma = ma;

this.hoTen = hoTen;

this.tienThuoc = tienThuoc;
}

public double vienPhi() {
    return tienThuoc;
}

public void displayInfo() {
    System.out.println("Ma: " + ma + ", Ho ten: " + hoTen + ", Vien phi: " + vienPhi());
}
}

// Lớp con: BNNGoaiTru kế thừa BenhNhan
class BNNGoaiTru extends BenhNhan {
    private double phiKham;
    private double phiXetNghiem;

    public BNNGoaiTru(int ma, String hoTen, double tienThuoc, double phiKham,
double phiXetNghiem) {
        super(ma, hoTen, tienThuoc);
        this.phiKham = phiKham;
        this.phiXetNghiem = phiXetNghiem;
    }

    @Override
    public double vienPhi() {
```

```

        return tienThuoc + phiKham + phiXetNghiem;
    }
}

// Lớp con: BNNoiTru kế thừa BenhNhan
class BNNoiTru extends BenhNhan {
    private double phiNgay;
    private int soNgayNamVien;

    public BNNoiTru(int ma, String hoTen, double tienThuoc, double phiNgay, int
soNgayNamVien) {
        super(ma, hoTen, tienThuoc);
        this.phiNgay = phiNgay;
        this.soNgayNamVien = soNgayNamVien;
    }

    @Override
    public double vienPhi() {
        double phuPhi = (soNgayNamVien < 10) ? 50 : 100;
        return tienThuoc * soNgayNamVien + phiNgay * soNgayNamVien + phuPhi;
    }
}

// Lớp Main để kiểm tra chương trình
public class Main {
    public static void main(String[] args) {
        BenhNhan bn1 = new BNNgoaiTru(101, "Nguyen Van A", 500, 200, 300);
        BenhNhan bn2 = new BNNoiTru(102, "Tran Thi B", 700, 100, 12);
    }
}

```


```
System.out.println("Thông tin bệnh nhân ngoại trú:");
bn1.displayInfo();

System.out.println("\nThông tin bệnh nhân nội trú:");
bn2.displayInfo();
}
}
```

Kết quả mong đợi khi chạy chương trình

Thông tin bệnh nhân ngoại trú:
Ma: 101, Ho ten: Nguyen Van A, Vien phi: 1000.0

Thông tin bệnh nhân nội trú:
Ma: 102, Ho ten: Tran Thi B, Vien phi: 9300.0

Bài tập này giúp sinh viên hiểu rõ cách tổ chức kế thừa trong lập trình hướng đối tượng, áp dụng vào bài toán thực tế về quản lý viện phí. 

Bài tập 2: Quản lý nhân viên trong công ty

Mục tiêu:

- Hiểu cách kế thừa giúp mở rộng tính năng hệ thống.
- Biết cách sử dụng `super()` để gọi constructor lớp cha.
- Thực hành ghi đè (override) phương thức.

Mô tả bài toán

Công ty có nhiều loại nhân viên, mỗi loại có cách tính lương khác nhau:

- Employee (Lớp cha) có các thuộc tính chung: name, id, baseSalary.
- Manager (Lớp con) có thêm bonus (thưởng).

- Developer (Lớp con) có thêm overtimeHours (số giờ làm thêm).

Yêu cầu:

- Viết chương trình quản lý nhân viên bằng kế thừa.
- Viết phương thức calculateSalary() trong mỗi lớp để tính lương:
 - **Nhân viên thông thường:** lương = baseSalary.
 - **Quản lý:** lương = baseSalary + bonus.
 - **Lập trình viên:** lương = baseSalary + overtimeHours * 200000.
- Hiển thị thông tin nhân viên.

✚ Phân tích bài toán

1. Tạo lớp cha Employee:

- Chứa thuộc tính chung name, id, baseSalary.
- Có phương thức calculateSalary().

2. Tạo lớp con Manager kế thừa Employee:

- Thêm thuộc tính bonus.
- Ghi đè phương thức calculateSalary().

3. Tạo lớp con Developer kế thừa Employee:

- Thêm thuộc tính overtimeHours.
- Ghi đè phương thức calculateSalary().

4. Trong main(), tạo danh sách nhân viên, hiển thị thông tin.

✚ Code chương trình

```
// Lớp cha: Employee
class Employee {
    protected String name;
    protected int id;
    protected double baseSalary;

    public Employee(String name, int id, double baseSalary) {
```

```
this.name = name;

this.id = id;

this.baseSalary = baseSalary;
}

public double calculateSalary() {
    return baseSalary;
}

public void displayInfo() {
    System.out.println("ID: " + id + ", Name: " + name + ", Salary: " +
calculateSalary());
}
}

// Lớp con: Manager kế thừa Employee
class Manager extends Employee {
    private double bonus;

    public Manager(String name, int id, double baseSalary, double bonus) {
        super(name, id, baseSalary);
        this.bonus = bonus;
    }

    @Override
    public double calculateSalary() {
        return baseSalary + bonus;
    }
}
```

```
}

// Lớp con: Developer kế thừa Employee
class Developer extends Employee {
    private int overtimeHours;

    public Developer(String name, int id, double baseSalary, int overtimeHours) {
        super(name, id, baseSalary);
        this.overtimeHours = overtimeHours;
    }

    @Override
    public double calculateSalary() {
        return baseSalary + (overtimeHours * 200000);
    }
}

// Lớp Main để kiểm tra chương trình
public class Main {
    public static void main(String[] args) {
        Employee emp1 = new Employee("Alice", 1, 5000000);
        Manager emp2 = new Manager("Bob", 2, 7000000, 2000000);
        Developer emp3 = new Developer("Charlie", 3, 6000000, 5);

        System.out.println("Employee Information:");
        emp1.displayInfo();
        emp2.displayInfo();
    }
}
```

```
emp3.displayInfo();  
}  
}
```

Kết quả mong đợi khi chạy chương trình

Employee Information:
ID: 1, Name: Alice, Salary: 5000000.0
ID: 2, Name: Bob, Salary: 9000000.0
ID: 3, Name: Charlie, Salary: 7000000.0

CÂU HỎI TRẮC NGHIỆM

Phần 1: Khái niệm và lợi ích của kế thừa

- Kế thừa trong lập trình hướng đối tượng (OOP) giúp:
 - Giảm trùng lặp mã nguồn
 - Tăng tính tái sử dụng mã
 - Dễ bảo trì và mở rộng chương trình
 - Tất cả các đáp án trên
- Trong lập trình hướng đối tượng, một lớp được gọi là lớp con khi:
 - Nó kế thừa từ một lớp khác
 - Nó có nhiều phương thức hơn lớp cha
 - Nó có từ khóa this trong constructor
 - Nó chỉ chứa các phương thức tĩnh
- Tại sao kế thừa lại giúp việc bảo trì phần mềm dễ dàng hơn?
 - Vì nó giảm số lượng dòng code cần sửa đổi
 - Vì tất cả các lớp con luôn độc lập với lớp cha
 - Vì kế thừa giúp chương trình chạy nhanh hơn
 - Vì kế thừa không làm thay đổi cấu trúc chương trình
- Lợi ích của kế thừa KHÔNG bao gồm điều nào sau đây?
 - Giúp tránh trùng lặp mã nguồn
 - Giảm số lượng đối tượng được tạo ra trong chương trình
 - Dễ dàng thêm các tính năng mới mà không ảnh hưởng đến mã hiện tại
 - Cải thiện khả năng tổ chức mã nguồn

Phần 2: cú pháp và các kiểu kế thừa

5. Trong Java, để một lớp kế thừa một lớp khác, ta sử dụng từ khóa nào?
 - A. extend
 - B. extends
 - C. inherit
 - D. super
6. Một lớp con có thể kế thừa bao nhiêu lớp cha trong Java?
 - A. 1
 - B. 2
 - C. Không giới hạn
 - D. Không thể kế thừa lớp khác
7. Câu nào sau đây mô tả đúng về **kế thừa đơn**?
 - A. Một lớp kế thừa từ một lớp cha duy nhất
 - B. Một lớp kế thừa từ nhiều lớp cha
 - C. Một lớp có thể kế thừa từ lớp cha và lớp ông nội
 - D. Một lớp có thể có nhiều đối tượng kế thừa từ nó
8. Java có hỗ trợ kế thừa đa lớp không?
 - A. Có, thông qua từ khóa extends
 - B. Không, nhưng có thể sử dụng interface để thay thế
 - C. Không, Java không hỗ trợ kế thừa đa lớp dưới mọi hình thức
 - D. Có, bằng cách sử dụng từ khóa superclass
9. Một lớp ElectricCar kế thừa lớp Car, và Car kế thừa lớp Vehicle. Đây là loại kế thừa nào?
 - A. Kế thừa đơn
 - B. Kế thừa đa cấp
 - C. Kế thừa thứ bậc
 - D. Kế thừa lai
10. Nếu Car và Bike cùng kế thừa Vehicle, đây là loại kế thừa nào?
 - A. Kế thừa đơn
 - B. Kế thừa đa cấp
 - C. Kế thừa thứ bậc
 - D. Kế thừa đa lớp

Phần 3: từ khóa super và ghi đè phương thức (override)

11. Từ khóa super được sử dụng để:
 - A. Truy cập các thuộc tính và phương thức của lớp con

- B. Gọi constructor của lớp cha
- C. Khởi tạo một đối tượng mới của lớp cha
- D. Tất cả các đáp án trên

12. Khi nào cần sử dụng từ khóa super?

- A. Khi muốn truy cập biến private của lớp cha
- B. Khi gọi constructor của lớp cha từ lớp con
- C. Khi muốn ngăn chặn ghi đè phương thức
- D. Khi khai báo một biến static

13. Điều gì xảy ra khi lớp con ghi đè (override) một phương thức của lớp cha?

- A. Phương thức của lớp cha bị xóa hoàn toàn khỏi bộ nhớ
- B. Phương thức của lớp con thay thế hoàn toàn phương thức của lớp cha khi được gọi từ lớp con
- C. Chương trình sẽ báo lỗi biên dịch
- D. Không có gì xảy ra, cả hai phương thức sẽ chạy đồng thời

14. Trong đoạn code sau, phương thức displayInfo() trong lớp con có thể gọi displayInfo() của lớp cha bằng cách nào?

```
class Parent {  
    void displayInfo() {  
        System.out.println("Parent class method");  
    }  
}  
  
class Child extends Parent {  
    void displayInfo() {  
        System.out.println("Child class method");  
    }  
}
```

- A. super.displayInfo();
- B. this.displayInfo();
- C. Parent.displayInfo();
- D. Không thể gọi được

15. Khi nào nên sử dụng ghi đè phương thức (override)?

- A. Khi cần thay đổi hành vi của phương thức kế thừa từ lớp cha
- B. Khi muốn tạo một phương thức mới hoàn toàn khác biệt trong lớp con

- C. Khi muốn sử dụng lại phương thức của lớp cha mà không thay đổi nó
- D. Khi muốn ngăn chặn phương thức được gọi từ lớp con

Phần 4: Áp dụng kế thừa trong thực tế

16. Trong bài toán quản lý phương tiện giao thông, lớp Car kế thừa từ Vehicle. Điều nào sau đây là hợp lý?
 - A. Car có thể có thêm thuộc tính numberOfDoors
 - B. Vehicle có thể có thuộc tính brand, color, maxSpeed
 - C. Car có thể ghi đè phương thức displayInfo() để hiển thị thêm thông tin về numberOfDoors
 - D. Tất cả các đáp án trên
17. Trong bài toán quản lý bệnh viện, phương thức vienPhi() trong BNNoiTru có gì đặc biệt?
 - A. Được ghi đè từ lớp BenhNhan
 - B. Tính toán viện phí dựa trên soNgayNamVien và phiNgay
 - C. Sử dụng thêm phụ phí dựa vào số ngày nằm viện
 - D. Tất cả các đáp án trên
18. Trong bài toán quản lý nhân viên, lớp Manager kế thừa từ Employee và có thuộc tính bonus. Làm thế nào để tính tổng lương?
 - A. $Lương = baseSalary + bonus$
 - B. $Lương = baseSalary * bonus$
 - C. $Lương = baseSalary + (bonus * 2)$
 - D. $Lương = bonus - baseSalary$
19. Trong bài toán quản lý tài khoản ngân hàng, SavingAccount kế thừa từ BankAccount. Thuộc tính nào sau đây nên có trong SavingAccount?
 - A. interestRate (Lãi suất)
 - B. balance (Số dư)
 - C. accountNumber (Số tài khoản)
 - D. Cả A và B
20. Trong hệ thống kế thừa, khi nào nên sử dụng interface thay vì class?
 - A. Khi cần hỗ trợ đa kế thừa
 - B. Khi tất cả các phương thức đều là abstract
 - C. Khi muốn ép buộc tất cả các lớp con phải tuân theo một bộ quy tắc chung
 - D. Tất cả các đáp án trên

BÀI TẬP TỰ LUYỆN

Bài 1: Kế thừa đơn giản - Động vật kêu

Đề bài:

- Tạo lớp Animal có phương thức makeSound() in ra "Some sound...".
- Tạo hai lớp con Dog và Cat kế thừa Animal.
- Ghi đè phương thức makeSound() trong từng lớp để hiển thị:
 - Dog: "Woof! Woof!"
 - Cat: "Meow! Meow!"
- Viết chương trình trong main() để tạo đối tượng Dog và Cat, gọi makeSound().

Hướng dẫn:

1. Tạo lớp Animal

- Khai báo lớp Animal với một phương thức makeSound().
- Phương thức này sẽ in ra "Some sound..." khi được gọi.

2. Tạo lớp Dog và Cat kế thừa Animal

- Sử dụng từ khóa extends để khai báo kế thừa.
- Ghi đè (override) phương thức makeSound() trong từng lớp để thay đổi nội dung in ra.

3. Viết main() để kiểm tra chương trình

- Tạo đối tượng Dog và Cat.
- Gọi phương thức makeSound() cho từng đối tượng.

Bài 2: Kế thừa đơn giản - Hình học

Đề bài:

- Tạo lớp Shape có phương thức draw() in ra "Drawing a shape".
- Tạo hai lớp con Circle và Rectangle kế thừa Shape.
- Ghi đè phương thức draw() để hiển thị:
 - Circle: "Drawing a circle"
 - Rectangle: "Drawing a rectangle"

- Trong main(), tạo danh sách các đối tượng Shape, gọi draw() từng đối tượng.

Hướng dẫn:

1. Tạo lớp Shape

- Khai báo lớp Shape với phương thức draw().
- Phương thức này mặc định in ra "Drawing a shape".

2. Tạo lớp Circle và Rectangle kế thừa Shape

- Sử dụng từ khóa extends để kế thừa.
- Ghi đè phương thức draw() để thay đổi nội dung in ra.

3. Viết main() để kiểm tra chương trình

- Tạo danh sách các đối tượng Shape, Circle và Rectangle.
- Gọi draw() trên từng đối tượng để kiểm tra kết quả.

Bài 3: Kế thừa đơn giản - Nhân viên

Đề bài:

- Tạo lớp Employee có các thuộc tính: name, id và phương thức displayInfo().
- Tạo hai lớp con Manager và Developer kế thừa Employee.
- Ghi đè phương thức displayInfo() để hiển thị:
 - Manager: "Manager: <name>, ID: <id>"
 - Developer: "Developer: <name>, ID: <id>"
- Trong main(), tạo danh sách nhân viên, gọi displayInfo().

Hướng dẫn:

1. Tạo lớp Employee

- Khai báo các thuộc tính name, id.
- Viết phương thức displayInfo() để in thông tin nhân viên.

2. Tạo lớp Manager và Developer kế thừa Employee

- Sử dụng extends để kế thừa.
- Ghi đè phương thức displayInfo() để hiển thị thông tin phù hợp.

3. Viết main() để kiểm tra chương trình

- Tạo danh sách các đối tượng Employee, Manager và Developer.
- Gọi displayInfo() trên từng đối tượng.

Bài 4: Kế thừa với super - Phương tiện giao thông

Đề bài:

- Tạo lớp Vehicle có các thuộc tính brand, speed và constructor.
- Tạo hai lớp con Car và Bike kế thừa Vehicle.
- Ghi đè phương thức displayInfo() để hiển thị thông tin xe.
- Trong main(), tạo danh sách xe và gọi displayInfo().

Hướng dẫn:

1. Tạo lớp Vehicle

- Khai báo các thuộc tính brand, speed.
- Viết constructor để khởi tạo giá trị.

2. Tạo lớp Car và Bike kế thừa Vehicle

- Sử dụng từ khóa extends để kế thừa.
- Thêm thuộc tính riêng cho từng lớp.
- Dùng từ khóa super để gọi constructor của lớp cha.
- Ghi đè phương thức displayInfo() để hiển thị thông tin đầy đủ.

3. Viết main() để kiểm tra chương trình

- Tạo danh sách các phương tiện Car và Bike.
- Gọi displayInfo() để hiển thị thông tin.

Bài 5: Kế thừa với super - Hệ thống tài khoản ngân hàng

Đề bài:

- Tạo lớp BankAccount có accountNumber, balance và phương thức displayInfo().
- Tạo hai lớp con SavingAccount và CheckingAccount kế thừa BankAccount.

- Ghi đè `displayInfo()` để hiển thị thông tin tài khoản.
- Trong `main()`, tạo danh sách tài khoản và gọi `displayInfo()`.

Hướng dẫn:

1. Tạo lớp **BankAccount**

- Khai báo các thuộc tính `accountNumber`, `balance`.
- Viết constructor để khởi tạo tài khoản.
- Viết phương thức `displayInfo()` để in thông tin tài khoản.

2. Tạo lớp **SavingAccount** và **CheckingAccount** kế thừa **BankAccount**

- Sử dụng `extends` để kế thừa.
- Thêm các thuộc tính riêng như `interestRate` cho `SavingAccount`.
- Dùng từ khóa `super` để gọi constructor của lớp cha.
- Ghi đè `displayInfo()` để hiển thị thông tin tài khoản phù hợp.

3. Viết `main()` để kiểm tra chương trình

- Tạo danh sách tài khoản `SavingAccount` và `CheckingAccount`.
- Gọi `displayInfo()` để kiểm tra kết quả.

BÀI TẬP LUYỆN TẬP

Bài 6: Kế thừa đơn giản - Hệ thống thư viện

Đề bài:

- Tạo lớp `Book` có các thuộc tính: `title`, `author` và phương thức `displayInfo()`.
- Tạo hai lớp con `EBook` và `PrintedBook` kế thừa `Book`.
- Ghi đè phương thức `displayInfo()` để hiển thị:
 - `EBook`: Thêm thông tin về `fileSize`.
 - `PrintedBook`: Thêm thông tin về `pageCount`.
- Trong `main()`, tạo danh sách sách và gọi `displayInfo()`.

Hướng dẫn:

1. Tạo lớp **Book**

- Khai báo các thuộc tính title, author.
- Viết phương thức displayInfo() để hiển thị thông tin sách.

2. Tạo lớp EBook và PrintedBook kế thừa Book

- Sử dụng extends để kế thừa.
- Thêm thuộc tính fileSize cho EBook và pageCount cho PrintedBook.
- Ghi đè phương thức displayInfo() để hiển thị đầy đủ thông tin sách.

3. Viết main() để kiểm tra chương trình

- Tạo danh sách sách, bao gồm cả EBook và PrintedBook.
- Gọi displayInfo() để kiểm tra kết quả.

Bài 7: Kế thừa với super - Quản lý sinh viên

Đề bài:

- Tạo lớp Person có các thuộc tính name, age, và phương thức displayInfo().
- Tạo lớp con Student kế thừa Person, thêm thuộc tính studentID.
- Ghi đè phương thức displayInfo() để hiển thị thêm studentID.
- Trong main(), tạo danh sách sinh viên và gọi displayInfo().

Hướng dẫn:

1. Tạo lớp Person

- Khai báo thuộc tính name, age.
- Viết phương thức displayInfo() để hiển thị thông tin cá nhân.

2. Tạo lớp Student kế thừa Person

- Sử dụng extends để kế thừa.
- Thêm thuộc tính studentID.
- Dùng từ khóa super để gọi constructor của lớp cha.
- Ghi đè phương thức displayInfo() để hiển thị đầy đủ thông tin.

3. Viết main() để kiểm tra chương trình

- Tạo danh sách sinh viên và gọi displayInfo().

Bài 8: Kế thừa và ghi đè - Quản lý tài khoản ngân hàng nâng cao

Đề bài:

- Tạo lớp BankAccount có accountNumber, balance, và phương thức withdraw(amount).
- Tạo lớp con CheckingAccount kế thừa BankAccount, thêm overdraftLimit.
- Ghi đè phương thức withdraw() để kiểm tra giới hạn thấu chi.
- Trong main(), tạo tài khoản và kiểm tra tính năng rút tiền.

Hướng dẫn:

1. Tạo lớp BankAccount

- Khai báo thuộc tính accountNumber, balance.
- Viết phương thức withdraw(amount) để rút tiền.

2. Tạo lớp CheckingAccount kế thừa BankAccount

- Thêm thuộc tính overdraftLimit.
- Ghi đè phương thức withdraw() để kiểm tra giới hạn thấu chi trước khi rút tiền.

3. Viết main() để kiểm tra chương trình

- Tạo tài khoản CheckingAccount và kiểm tra tính năng rút tiền.

Bài 9: Kế thừa và ghi đè - Quản lý động vật nâng cao

Đề bài:

- Tạo lớp Animal có phương thức makeSound() và thuộc tính name.
- Tạo lớp Dog kế thừa Animal, thêm thuộc tính breed.
- Ghi đè phương thức makeSound() để hiển thị "Woof! Woof!".
- Viết lớp Cat kế thừa Animal, thêm thuộc tính color.
- Ghi đè phương thức makeSound() để hiển thị "Meow! Meow!".
- Trong main(), tạo danh sách động vật, gọi makeSound().

Hướng dẫn:

1. Tạo lớp Animal

- Khai báo thuộc tính name.
- Viết phương thức makeSound(), mặc định in "Some sound...".

2. Tạo lớp Dog và Cat kế thừa Animal

- Dog thêm thuộc tính breed, ghi đè makeSound() để in "Woof! Woof!".
- Cat thêm thuộc tính color, ghi đè makeSound() để in "Meow! Meow!".

3. Viết main() để kiểm tra chương trình

- Tạo danh sách động vật, gọi makeSound().

Bài 10: Kế thừa và super - Quản lý nhân viên nâng cao

Đề bài:

- Tạo lớp Employee có các thuộc tính name, id, salary và phương thức calculateSalary().
- Tạo lớp Manager kế thừa Employee, thêm thuộc tính bonus.
- Ghi đè calculateSalary() để tính tổng lương: salary + bonus.
- Tạo lớp Developer kế thừa Employee, thêm thuộc tính overtimeHours.
- Ghi đè calculateSalary() để tính tổng lương: salary + (overtimeHours * 200000).
- Trong main(), tạo danh sách nhân viên, gọi calculateSalary().

Hướng dẫn:

1. Tạo lớp Employee

- Khai báo thuộc tính name, id, salary.
- Viết phương thức calculateSalary(), mặc định trả về salary.

2. Tạo lớp Manager và Developer kế thừa Employee

- Manager thêm thuộc tính bonus, ghi đè calculateSalary() để tính tổng lương.
- Developer thêm thuộc tính overtimeHours, ghi đè calculateSalary() để tính lương theo giờ làm thêm.

3. Viết main() để kiểm tra chương trình

- Tạo danh sách nhân viên Manager, Developer.
- Gọi calculateSalary() để kiểm tra kết quả.

BÀI TẬP DỰ ÁN NHÓM

Dự án 1: Quản lý nhân sự trong công ty

Lợi ích:

- ✓ Giúp sinh viên thực hành kế thừa và ghi đè phương thức trong bài toán thực tế.
- ✓ Hiểu cách tổ chức mã nguồn trong hệ thống lớn hơn.

Đề bài:

- Xây dựng hệ thống quản lý nhân sự với các lớp sau:
 - Employee (Lớp cha) có các thuộc tính chung: name, id, baseSalary.
 - Manager (Lớp con) có thêm thuộc tính bonus.
 - Developer (Lớp con) có thêm thuộc tính overtimeHours.
 - Intern (Lớp con) có thêm thuộc tính unpaidHours.
- Ghi đè phương thức calculateSalary() trong từng lớp để tính lương:
 - **Nhân viên thường:** salary = baseSalary
 - **Quản lý:** salary = baseSalary + bonus
 - **Lập trình viên:** salary = baseSalary + (overtimeHours * 200000)
 - **Thực tập sinh:** salary = baseSalary - (unpaidHours * 100000)
- Viết chương trình main():
 - Tạo danh sách nhân viên thuộc nhiều loại khác nhau.
 - Hiển thị thông tin nhân viên và lương của họ.

Hướng dẫn:

1. Tạo lớp Employee (Lớp cha)

- Khai báo thuộc tính name, id, baseSalary.
- Viết phương thức calculateSalary(), mặc định trả về baseSalary.

- Viết phương thức `displayInfo()` để hiển thị thông tin nhân viên.

2. Tạo các lớp **Manager**, **Developer**, **Intern** kế thừa **Employee**

- **Manager**: Thêm thuộc tính `bonus`, ghi đè `calculateSalary()`.
- **Developer**: Thêm thuộc tính `overtimeHours`, ghi đè `calculateSalary()`.
- **Intern**: Thêm thuộc tính `unpaidHours`, ghi đè `calculateSalary()`.

3. Viết `main()` để kiểm tra chương trình

- Tạo danh sách nhân viên **Manager**, **Developer**, **Intern**.
- Gọi `displayInfo()` để kiểm tra kết quả.

Dự án 2 (Mức độ khó): Quản lý cửa hàng và sản phẩm

Lợi ích:

- ✓ Giúp sinh viên hiểu cách kế thừa và ghi đè phương thức trong bài toán thương mại điện tử.
- ✓ Rèn luyện kỹ năng tính toán giá trị động dựa trên thuộc tính sản phẩm.

Đề bài:

- Xây dựng hệ thống quản lý cửa hàng với các lớp sau:
 - **Product** (Lớp cha) có các thuộc tính: `productId`, `name`, `price`.
 - **Electronics** (Lớp con) có thêm thuộc tính `warrantyPeriod`.
 - **Clothing** (Lớp con) có thêm thuộc tính `size` và `material`.
- Viết phương thức `applyDiscount(double percent)` trong lớp cha **Product**, cho phép giảm giá sản phẩm.
- Ghi đè `applyDiscount()` trong **Electronics** để áp dụng thêm giảm giá đặc biệt cho sản phẩm có bảo hành trên 1 năm.
- Viết phương thức `calculateFinalPrice()` trong lớp cha **Product** để tính giá sau khi giảm giá.
- Viết chương trình `main()`:
 - Tạo danh sách sản phẩm thuộc nhiều loại khác nhau.
 - Áp dụng giảm giá và hiển thị thông tin sản phẩm sau khi giảm giá.

Hướng dẫn:

1. Tạo lớp Product (Lớp cha)

- Khai báo các thuộc tính productId, name, price.
- Viết phương thức applyDiscount(double percent) để giảm giá.
- Viết phương thức calculateFinalPrice() để tính giá sau giảm giá.

2. Tạo lớp Electronics và Clothing kế thừa Product

- Electronics: Thêm thuộc tính warrantyPeriod, ghi đè applyDiscount() để áp dụng thêm ưu đãi.
- Clothing: Thêm thuộc tính size, material, kế thừa phương thức giảm giá từ Product.

3. Viết main() để kiểm tra chương trình

- Tạo danh sách sản phẩm thuộc các loại Electronics và Clothing.
- Áp dụng giảm giá cho từng sản phẩm.
- Hiển thị thông tin sản phẩm sau khi giảm giá.