Handout buổi 10 & 11: Tính trừu tượng trong lập trình hướng đối tượng

1. MỤC TIÊU BUỔI HỌC

Sau buổi học, sinh viên sẽ:

- Hiểu được khái niệm **tính trừu tượng** trong lập trình hướng đối tượng.
- Biết cách sử dụng **lớp trừu tượng (abstract class)** và **giao diện (interface)** trong Java.
- Áp dụng tính trừu tượng vào lập trình để viết mã dễ bảo trì và mở rộng.
- Thực hành xây dựng và sử dụng lớp trừu tượng, giao diện qua các bài tập cụ thể.

2. TÓM TẮT LÝ THUYẾT VÀ VÍ DỤ MINH HỌA

2.1. Tính trừu tượng là gì?

Khái niệm:

- **Tính trừu tượng (Abstraction)** là một tính chất quan trọng của lập trình hướng đối tượng (OOP).
- Nó giúp **che giấu các chi tiết thực thi**, chỉ hiển thị các chức năng cần thiết cho người dùng.
- Người sử dụng chỉ quan tâm đến **cách dùng**, không cần biết **cách hoạt động bên trong**.

Ví dụ thực tế:

- **Chiếc ô tô:**
    - Người lái chỉ cần biết: **Chạy xe, tăng tốc, phanh, rẽ trái/phải**.
    - Không cần biết: **Động cơ hoạt động thế nào, hệ thống phanh ABS xử lý ra sao**.

→ **Đây là tính trừu tượng**: Chỉ cung cấp các chức năng cần thiết và ẩn đi phần cài đặt phức tạp.

Ví dụ trong lập trình:

- Giả sử ta có một **tài khoản ngân hàng (BankAccount)**, cung cấp các chức năng:
    - deposit(double amount): Nạp tiền.

- withdraw(double amount): Rút tiền.
- showBalance(): Xem số dư.

- Nhưng cách hệ thống **quản lý số dư, xử lý giao dịch** không cần hiển thị cho người dùng.

Ví dụ trong Java:

```
abstract class BankAccount {
    protected double balance;

    public BankAccount(double balance) {
        this.balance = balance;
    }

    abstract void deposit(double amount);
    abstract void withdraw(double amount);

    public void showBalance() {
        System.out.println("Current balance: $" + balance);
    }
}
```

- Lớp BankAccount **che giấu chi tiết nội bộ**, chỉ cung cấp phương thức deposit(), withdraw().
- Các lớp con sẽ định nghĩa cụ thể cách nạp/rút tiền.

Lợi ích của tính trừu tượng:

✓ **Giúp mã nguồn dễ bảo trì**, vì thay đổi nội bộ không ảnh hưởng đến lớp sử dụng.
✓ **Tăng tính linh hoạt**, vì mỗi lớp con có thể tùy chỉnh cách hoạt động riêng.
✓ **Dễ dàng mở rộng hệ thống**, chỉ cần thêm lớp mới kế thừa BankAccount.

2.2. Lớp trừu tượng (Abstract Class)

Khái niệm:

- **Lớp trừu tượng** là lớp **không thể tạo đối tượng trực tiếp**.
- Nó có thể chứa:
  - **Phương thức trừu tượng** (không có thân, cần lớp con triển khai).
  - **Phương thức có thân** (đã có sẵn logic).
- Dùng khi **các lớp con có nhiều điểm chung nhưng vẫn cần tùy chỉnh riêng**.

Ví dụ trong lập trình:

Giả sử ta cần xây dựng hệ thống quản lý động vật:

- **Mọi động vật đều có phương thức makeSound()**, nhưng cách kêu của từng loại khác nhau.
- **Mọi động vật đều có eat()**, nhưng cách ăn có thể giống nhau.

Triển khai bằng lớp trừu tượng:

```
abstract class Animal {
    abstract void makeSound(); // Phương thức trừu tượng

    void eat() { // Phương thức có thân
        System.out.println("Eating...");
    }
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Bark");
    }
}

class Cat extends Animal {
```

```
    void makeSound() {
        System.out.println("Meow");
    }
}
```

Giải thích:

- **Lớp Animal** là **lớp trừu tượng**, chứa phương thức makeSound() chưa có thân.
- **Lớp Dog, Cat** kế thừa Animal và **tự định nghĩa makeSound()** theo cách riêng.
- Phương thức eat() đã có sẵn logic nên được kế thừa trực tiếp.

Khi nào dùng lớp trừu tượng?

✓ Khi **có quan hệ cha-con rõ ràng** (ví dụ: Animal → Dog, Cat).
✓ Khi **muốn định nghĩa chung một số hành vi** nhưng **cũng cần tùy chỉnh** ở từng lớp con.
✓ Khi **muốn tái sử dụng code** mà vẫn đảm bảo tính linh hoạt.

👉 **Tóm lại:**

- **Tính trừu tượng giúp che giấu chi tiết cài đặt, chỉ cung cấp các chức năng cần thiết.**
- **Lớp trừu tượng giúp định nghĩa các hành vi chung, nhưng vẫn cho phép tùy chỉnh cách thực hiện.**

2.3. Giao diện (Interface)

Khái niệm:

- **Giao diện (Interface)** là một cơ chế trong lập trình hướng đối tượng giúp **định nghĩa một bộ hành vi chung** mà nhiều lớp có thể thực hiện.
- **Tất cả các phương thức trong interface đều là trừu tượng hoàn toàn** (trước Java 8).
- **Một lớp có thể kế thừa nhiều interface cùng lúc** (khắc phục hạn chế của kế thừa đơn).

So sánh với lớp trừu tượng:

Đặc điểm	Lớp trừu tượng (Abstract Class)	Giao diện (Interface)
Có thể chứa	Cả phương thức trừu tượng và có thân	Chỉ chứa phương thức trừu tượng (trước Java 8)
Biến	Có thể có biến instance (có thể thay đổi)	Chỉ có hằng số (public static final)
Tính kế thừa	Một lớp chỉ có thể kế thừa **một** lớp trừu tượng	Một lớp có thể **kế thừa nhiều** interface
Mục đích	Dùng khi có quan hệ cha-con rõ ràng	Dùng để nhóm các hành vi chung cho các lớp không liên quan

Ví dụ thực tế:

- **Ổ cắm điện và thiết bị điện:**
  - Ổ cắm điện có **chuẩn giao diện chung** (220V, 2 chấu hoặc 3 chấu).
  - Nhiều thiết bị khác nhau **có thể dùng chung ổ cắm**, dù không liên quan đến nhau (máy giặt, tủ lạnh, sạc điện thoại…).
  - Điều này giống như một interface: **định nghĩa một bộ hành vi chung**, nhưng không quan trọng thiết bị là gì.

Ví dụ trong lập trình:

Giả sử chúng ta có nhiều loại động vật khác nhau, nhưng **không phải tất cả đều biết bay**.

- Ta tạo một interface Flyable, chỉ chứa phương thức fly().
- Bất kỳ động vật nào **có khả năng bay** sẽ triển khai Flyable.

```
// Interface chỉ chứa phương thức trừu tượng
interface Flyable {
    void fly();
}

// Lớp Bird có thể bay, nên triển khai Flyable
```

```
class Bird implements Flyable {
    public void fly() {
        System.out.println("Bird is flying");
    }
}

// Lớp Airplane cũng có thể bay, nhưng không liên quan đến Bird
class Airplane implements Flyable {
    public void fly() {
        System.out.println("Airplane is flying");
    }
}
```

Giải thích:

- **Flyable** là một giao diện, chỉ định nghĩa phương thức fly().
- **Bird và Airplane đều triển khai Flyable**, dù chúng không có quan hệ cha-con.
- Điều này giúp **tăng tính linh hoạt** và **cho phép nhiều lớp dùng chung một bộ hành vi**.

Kế thừa nhiều interface

Java không hỗ trợ **đa kế thừa (multiple inheritance)** với lớp, nhưng cho phép **một lớp kế thừa nhiều interface**.

Ví dụ: Một người máy (Robot) có thể **nói chuyện** như con người và **bay** như máy bay:

```
interface Speakable {
    void speak();
}

interface Flyable {
```

```
    void fly();
}

class Robot implements Speakable, Flyable {
    public void speak() {
        System.out.println("Robot is speaking");
    }

    public void fly() {
        System.out.println("Robot is flying");
    }
}
```

Giải thích:

- Lớp Robot kế thừa cả **Speakable (có thể nói) và Flyable (có thể bay)**.
- Nhờ kế thừa nhiều interface, Robot **vừa có thể bay như máy bay, vừa có thể nói như con người**.

Khi nào dùng Interface?

✓ Khi **các lớp không có quan hệ cha-con nhưng cần chung một bộ hành vi**.
✓ Khi **cần đa kế thừa**, vì Java không hỗ trợ đa kế thừa với class nhưng hỗ trợ với interface.
✓ Khi **cần định nghĩa một chuẩn chung** mà nhiều lớp có thể triển khai.

👉 **Tóm lại:**

- **Interface giúp định nghĩa một tập hợp hành vi chung mà nhiều lớp có thể triển khai.**
- **Một lớp có thể kế thừa nhiều interface, giúp tăng tính linh hoạt trong thiết kế phần mềm.**

2.4. So sánh Lớp Trừu Tượng (Abstract Class) và Giao Diện (Interface)

Lớp trừu tượng (abstract class) và giao diện (interface) đều được sử dụng để thực hiện **tính trừu tượng** trong Java, nhưng chúng có cách hoạt động khác nhau. Dưới đây là bảng so sánh chi tiết:

Bảng so sánh Lớp Trừu Tượng và Giao Diện

Tiêu chí	Lớp trừu tượng (abstract class)	Giao diện (interface)
Mục đích	Dùng khi có **quan hệ cha-con** rõ ràng giữa các lớp.	Dùng khi các lớp **không liên quan** nhưng cần chung một bộ hành vi.
Tính kế thừa	Một lớp **chỉ có thể kế thừa một lớp trừu tượng**.	Một lớp **có thể kế thừa nhiều interface**.
Chứa phương thức có thân	**Có thể** có phương thức đã triển khai.	Trước Java 8: **Không** có phương thức có thân. Từ Java 8+: **Có thể có** phương thức mặc định (default method).
Chứa phương thức trừu tượng	**Có thể có** phương thức trừu tượng và phương thức có thân.	**Chỉ chứa** phương thức trừu tượng (trước Java 8).
Chứa biến	**Có thể có** biến thông thường (có thể thay đổi giá trị).	**Chỉ có hằng số** (public static final).
Truy cập mặc định của phương thức	public, protected, hoặc default.	Mặc định là public (các phương thức luôn công khai).
Tốc độ thực thi	Nhanh hơn do hỗ trợ phương thức có thân.	Chậm hơn do phải thực hiện qua nhiều lớp triển khai.
Khi nào dùng?	Khi **cần định nghĩa chung** và có **các phương thức mặc định** cho lớp con.	Khi **cần tạo chuẩn chung** cho các lớp không liên quan nhưng có cùng hành vi.

Ví dụ thực tế

1. Dùng Lớp Trừu Tượng (abstract class) khi có quan hệ cha-con rõ ràng

Giả sử chúng ta có hệ thống xe cộ (Vehicle), trong đó **tất cả phương tiện đều có phương thức startEngine() nhưng cách triển khai khác nhau**.

- Car khởi động bằng **nút bấm**.
- Motorcycle khởi động bằng **đạp cần**.

Triển khai bằng lớp trừu tượng:

```
abstract class Vehicle {
    abstract void startEngine(); // Phương thức trừu tượng

    void stopEngine() { // Phương thức có thân
        System.out.println("Engine stopped.");
    }
}

class Car extends Vehicle {
    void startEngine() {
        System.out.println("Car starts with a button.");
    }
}

class Motorcycle extends Vehicle {
    void startEngine() {
        System.out.println("Motorcycle starts with a kick.");
    }
}
```

Tại sao dùng abstract class?

✓ Vì Car và Motorcycle **có quan hệ cha-con** (đều là Vehicle).

✓ **Phương thức stopEngine() có thể dùng chung** nên không cần lớp con triển khai lại.

2. Dùng Interface (interface) khi các lớp không liên quan nhưng cần hành vi chung

Giả sử chúng ta có nhiều loại **động vật, máy bay, robot có thể bay**.

- **Chim (Bird)** là động vật nhưng có thể bay.
- **Máy bay (Airplane)** không liên quan đến chim nhưng cũng có thể bay.
- **Robot (Drone)** cũng có thể bay.

→ Chúng không có quan hệ cha-con nhưng đều có chung hành vi "bay".

Triển khai bằng Interface:

```
interface Flyable {
    void fly();
}

class Bird implements Flyable {
    public void fly() {
        System.out.println("Bird is flying using wings.");
    }
}

class Airplane implements Flyable {
    public void fly() {
        System.out.println("Airplane is flying using engines.");
    }
}

class Drone implements Flyable {
    public void fly() {
        System.out.println("Drone is flying using propellers.");
    }
```

```
}
```

Tại sao dùng interface?
✓ Vì Bird, Airplane, Drone **không có quan hệ cha-con**, nhưng **đều có chung hành vi bay**.
✓ Giúp các lớp **không liên quan cũng có thể triển khai Flyable** mà không cần kế thừa từ cùng một lớp cha.

Khi nào dùng Lớp Trừu Tượng và khi nào dùng Interface?

✅ **Dùng abstract class nếu:**

- Các lớp **có quan hệ cha-con** rõ ràng.
- Cần **cung cấp một số phương thức mặc định** mà lớp con có thể kế thừa trực tiếp.
- Cần sử dụng **biến instance** (không chỉ là hằng số).

✅ **Dùng interface nếu:**

- Cần **định nghĩa một bộ hành vi chung** mà nhiều lớp không liên quan có thể triển khai.
- Cần **đa kế thừa** (vì Java không hỗ trợ đa kế thừa với class nhưng hỗ trợ với interface).
- Cần đảm bảo **tính linh hoạt cao** cho thiết kế hệ thống.

Tóm lại:

- Lớp trừu tượng giúp chia sẻ code chung giữa các lớp có quan hệ cha-con.
- Interface giúp định nghĩa một bộ hành vi chung mà nhiều lớp không liên quan có thể sử dụng.
- Dùng lớp trừu tượng khi cần kế thừa, dùng interface khi cần đa kế thừa.

3. BÀI TẬP CÓ LỜI GIẢI

Bài 1: Sử dụng Abstract Class - Quản lý phương tiện giao thông

Đề bài:

Một hệ thống quản lý phương tiện giao thông cần phân loại các phương tiện như ô tô, xe máy. Hệ thống cần đảm bảo rằng mọi phương tiện đều có thể hiển thị thông tin cơ bản và tính thuế trước bạ.

- Mỗi phương tiện đều có biển số, hãng sản xuất và giá gốc.
- Ô tô có thêm số chỗ ngồi và loại nhiên liệu.
- Xe máy có thêm dung tích động cơ.
- Thuế trước bạ được tính như sau:
    - Ô tô: 10% giá gốc
    - Xe máy:
        - Dung tích < 100cc: 2% giá gốc
        - Dung tích >= 100cc: 5% giá gốc

Phân tích bài toán:

- **Abstract class** Vehicle để chứa các thông tin và phương thức chung của phương tiện.
- Các lớp Car và Motorbike kế thừa từ Vehicle, triển khai phương thức calculateTax().

Code chương trình

```
// Abstract class đại diện cho phương tiện giao thông
abstract class Vehicle {
    protected String licensePlate;
    protected String manufacturer;
    protected double price;

    public Vehicle(String licensePlate, String manufacturer, double price) {
        this.licensePlate = licensePlate;
```

```
        this.manufacturer = manufacturer;
        this.price = price;
    }

    public abstract double calculateTax(); // Phương thức trừu tượng

    public void displayInfo() {
        System.out.println("Biển số: " + licensePlate);
        System.out.println("Hãng sản xuất: " + manufacturer);
        System.out.println("Giá gốc: " + price);
        System.out.println("Thuế trước bạ: " + calculateTax());
    }
}

// Lớp Car kế thừa từ Vehicle
class Car extends Vehicle {
    private int seatNumber;
    private String fuelType;

    public Car(String licensePlate, String manufacturer, double price, int seatNumber,
String fuelType) {
        super(licensePlate, manufacturer, price);
        this.seatNumber = seatNumber;
        this.fuelType = fuelType;
    }

    @Override
    public double calculateTax() {
```

```
        return price * 0.10; // 10% giá gốc
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Số chỗ ngồi: " + seatNumber);
        System.out.println("Loại nhiên liệu: " + fuelType);
    }
}

// Lớp Motorbike kế thừa từ Vehicle
class Motorbike extends Vehicle {
    private int engineCapacity;

    public Motorbike(String licensePlate, String manufacturer, double price, int
engineCapacity) {
        super(licensePlate, manufacturer, price);
        this.engineCapacity = engineCapacity;
    }

    @Override
    public double calculateTax() {
        return (engineCapacity < 100) ? price * 0.02 : price * 0.05;
    }

    @Override
    public void displayInfo() {
```

```
        super.displayInfo();
        System.out.println("Dung tích động cơ: " + engineCapacity + "cc");
    }
}

// Chương trình chính
public class Main {
    public static void main(String[] args) {
        Vehicle car = new Car("30A-12345", "Toyota", 500000000, 5, "Xăng");
        Vehicle bike = new Motorbike("29B-67890", "Honda", 20000000, 110);

        car.displayInfo();
        System.out.println("----------------------");
        bike.displayInfo();
    }
}
```

Phân tích lời giải:

- Vehicle là abstract class có phương thức calculateTax() trừu tượng.
- Car và Motorbike kế thừa Vehicle và triển khai calculateTax().
- Car áp dụng thuế 10% giá gốc.
- Motorbike tính thuế theo dung tích động cơ.
- displayInfo() giúp hiển thị thông tin phương tiện.

Bài 2: Sử dụng Interface - Hệ thống thanh toán

Đề bài:

Một hệ thống cần xử lý thanh toán qua nhiều phương thức như CreditCard và PayPal.

- Mọi phương thức thanh toán phải có khả năng thanh toán và hoàn tiền.

- Thanh toán qua CreditCard tính phí 1.5%.
- Thanh toán qua PayPal tính phí 2%.

Phân tích bài toán:

- **Interface PaymentMethod** định nghĩa pay() và refund().
- **Lớp CreditCard và PayPal** triển khai PaymentMethod.

Code chương trình

```
// Interface PaymentMethod đại diện phương thức thanh toán
interface PaymentMethod {
    void pay(double amount);
    void refund(double amount);
}

// Lớp CreditCard triển khai PaymentMethod
class CreditCard implements PaymentMethod {
    private String cardNumber;

    public CreditCard(String cardNumber) {
        this.cardNumber = cardNumber;
    }

    @Override
    public void pay(double amount) {
        double fee = amount * 0.015;
        System.out.println("Thanh toán bằng thẻ tín dụng: " + (amount + fee) + " VND (bao gồm phí 1.5%)");
    }
```

```
    @Override
    public void refund(double amount) {
        System.out.println("Hoàn tiền vào thẻ tín dụng: " + amount + " VND");
    }
}

// Lớp PayPal triển khai PaymentMethod
class PayPal implements PaymentMethod {
    private String email;

    public PayPal(String email) {
        this.email = email;
    }

    @Override
    public void pay(double amount) {
        double fee = amount * 0.02;
        System.out.println("Thanh toán bằng PayPal: " + (amount + fee) + " VND (bao gồm phí 2%)");
    }

    @Override
    public void refund(double amount) {
        System.out.println("Hoàn tiền vào tài khoản PayPal: " + amount + " VND");
    }
}

// Chương trình chính
```

```
public class Main {
    public static void main(String[] args) {
        PaymentMethod creditCard = new CreditCard("1234-5678-9876-5432");
        PaymentMethod payPal = new PayPal("user@example.com");

        creditCard.pay(1000000);
        payPal.pay(1000000);

        System.out.println("----------------------");

        creditCard.refund(500000);
        payPal.refund(500000);
    }
}
```

Phân tích lời giải:

- PaymentMethod là một interface với pay() và refund().
- CreditCard và PayPal triển khai PaymentMethod, định nghĩa cách tính phí riêng.
- Phí thanh toán:
    - CreditCard: 1.5%
    - PayPal: 2%
- Chương trình minh họa việc sử dụng cả hai phương thức thanh toán.

Tổng kết:

- **Abstract class** phù hợp khi có **thuộc tính chung** và một số **hành vi chung** cần triển khai một phần.

- **Interface** phù hợp khi có nhiều **hành vi chung** nhưng cách triển khai hoàn toàn khác nhau.
- Bài 1 sử dụng **abstract class** để phân loại phương tiện.
- Bài 2 sử dụng **interface** để trừu tượng hóa thanh toán.

4. TRẮC NGHIỆM

Phần 1: Tổng quan về Tính trừu tượng

1. **Tính trừu tượng trong lập trình hướng đối tượng giúp:**
A. Giảm số lượng dòng code cần viết
B. Che giấu chi tiết thực thi và chỉ hiển thị các chức năng cần thiết
C. Tăng tốc độ thực thi chương trình
D. Loại bỏ hoàn toàn sự kế thừa

2. **Lợi ích chính của tính trừu tượng là gì?**
A. Cho phép tạo ra nhiều lớp con giống nhau
B. Giúp mã nguồn dễ bảo trì và mở rộng hơn
C. Giúp tăng tốc độ chạy chương trình
D. Loại bỏ sự cần thiết của các phương thức

3. **Trong lập trình hướng đối tượng, tính trừu tượng chủ yếu được thể hiện thông qua:**
A. Lớp kế thừa
B. Phương thức trừu tượng và interface
C. Định nghĩa biến toàn cục
D. Câu lệnh điều kiện

4. **Tính trừu tượng giúp lập trình viên:**
A. Giảm số lượng class trong chương trình
B. Tăng độ phức tạp của mã nguồn
C. Giấu đi cách thức hoạt động bên trong của một đối tượng
D. Loại bỏ sự cần thiết của constructor

5. **Ví dụ nào sau đây KHÔNG phải là ứng dụng của tính trừu tượng?**
A. Một chiếc ô tô có vô lăng, bàn đạp nhưng không cần biết chi tiết động cơ hoạt động như thế nào
B. Một máy ATM cung cấp giao diện rút tiền mà không cần biết cách xử lý giao dịch bên trong
C. Một phương thức toán học như Math.sqrt() có thể dùng trực tiếp mà không cần biết thuật toán bên trong
D. Một biến toàn cục chứa danh sách tất cả các đối tượng trong chương trình

Phần 2: Lớp trừu tượng

6. **Lớp trừu tượng trong Java:**
A. Không thể chứa bất kỳ phương thức nào có thân
B. Có thể chứa cả phương thức có thân và phương thức trừu tượng
C. Chỉ có thể chứa phương thức trừu tượng
D. Không thể có biến thành viên

7. **Phát biểu nào sau đây là đúng về lớp trừu tượng?**
A. Một lớp trừu tượng có thể được khởi tạo trực tiếp
B. Một lớp trừu tượng bắt buộc phải có ít nhất một phương thức trừu tượng
C. Một lớp con kế thừa lớp trừu tượng phải triển khai tất cả các phương thức trừu tượng của lớp cha
D. Một lớp trừu tượng không thể chứa biến thành viên

8. **Phương thức trừu tượng là gì?**
A. Là phương thức không có phần thân, chỉ có khai báo
B. Là phương thức có thể bị ghi đè
C. Là phương thức chỉ có thể sử dụng trong lớp cha
D. Là phương thức không thể bị ghi đè

9. **Một lớp trừu tượng có thể chứa:**
A. Cả phương thức tĩnh và phương thức trừu tượng
B. Chỉ phương thức tĩnh
C. Chỉ phương thức trừu tượng
D. Chỉ phương thức có thân

10. Một lớp có thể kế thừa từ một lớp trừu tượng bằng cách nào?
A. Sử dụng từ khóa abstract
B. Sử dụng từ khóa extends
C. Sử dụng từ khóa implements
D. Sử dụng từ khóa override

Phần 3: Giao diện (Interface)

11. **Một interface trong Java có thể chứa:**
A. Biến thành viên có giá trị thay đổi
B. Phương thức có thân
C. Chỉ phương thức trừu tượng và hằng số
D. Constructor

12. **Điểm khác biệt giữa abstract class và interface là gì?**
A. Interface có thể chứa phương thức có thân, còn abstract class thì không

B. Một lớp có thể kế thừa nhiều abstract class nhưng chỉ có thể thực hiện một interface
C. Interface chỉ có phương thức trừu tượng hoàn toàn, còn abstract class có thể chứa cả phương thức có thân và phương thức trừu tượng
D. Một lớp con không thể kế thừa từ abstract class

13. **Từ khóa nào được sử dụng để một lớp thực hiện một interface?**
A. extends
B. implements
C. abstract
D. interface

14. **Một lớp có thể thực hiện nhiều interface không?**
A. Có
B. Không
C. Chỉ nếu lớp đó là abstract
D. Chỉ nếu interface đó không có phương thức

15. **Interface giúp ích gì trong lập trình hướng đối tượng?**
A. Cho phép nhiều lớp không liên quan chia sẻ chung một tập hành vi
B. Giảm số lượng lớp trong chương trình
C. Ngăn không cho lớp cha bị ghi đè
D. Tăng tốc độ thực thi chương trình

Phần 4: Ứng dụng thực tế và bài toán lập trình

16. **Tình huống nào sau đây phù hợp để sử dụng abstract class?**
A. Khi có một nhóm đối tượng có thuộc tính chung và một số phương thức có triển khai chung
B. Khi cần đảm bảo tất cả các lớp con đều triển khai một số phương thức theo một cách giống nhau
C. Khi cần cho phép đa kế thừa
D. Khi cần tạo ra các đối tượng tĩnh

17. **Tình huống nào phù hợp để sử dụng interface?**
A. Khi các lớp không có quan hệ trực tiếp nhưng cần có chung một tập hành vi
B. Khi cần khai báo các thuộc tính chung
C. Khi muốn triển khai một phương thức mặc định cho các lớp con
D. Khi muốn sử dụng một constructor chung cho tất cả các lớp

18. **Trong bài toán quản lý phương tiện giao thông, tại sao nên sử dụng abstract class?**
A. Vì tất cả các phương tiện có chung một số thuộc tính và hành vi
B. Vì tất cả các phương tiện đều có cấu trúc giống nhau
C. Vì interface không thể chứa biến thành viên
D. Vì abstract class nhanh hơn interface

19. **Trong bài toán hệ thống thanh toán, tại sao nên sử dụng interface?**
A. Vì các phương thức thanh toán không có quan hệ trực tiếp với nhau
B. Vì mỗi phương thức thanh toán có cách triển khai riêng
C. Vì cần đảm bảo mọi phương thức thanh toán đều có chung tập hành vi
D. Cả A, B và C đều đúng

20. **Điều nào sau đây KHÔNG đúng khi sử dụng abstract class và interface?**
A. Interface có thể chứa phương thức tĩnh
B. Một lớp có thể kế thừa từ nhiều abstract class
C. Một interface có thể kế thừa từ nhiều interface khác
D. Một lớp có thể thực hiện nhiều interface

5. BÀI TẬP LUYỆN TẬP

BÀI 1: Hiểu về Tính Trừu Tượng

Yêu cầu:

- Giải thích khái niệm tính trừu tượng trong lập trình hướng đối tượng.
- Cho một ví dụ minh họa (không cần viết mã nguồn).

Phân tích:

Tính trừu tượng giúp ẩn đi chi tiết triển khai bên trong và chỉ hiển thị những gì cần thiết cho người sử dụng. Điều này giúp đơn giản hóa việc sử dụng và bảo trì chương trình.

- Ví dụ thực tế: Khi lái xe ô tô, người dùng chỉ cần biết cách điều khiển vô lăng và bàn đạp mà không cần biết chi tiết về động cơ hoạt động thế nào.
- Trong lập trình, ta có thể sử dụng **lớp trừu tượng (abstract class)** hoặc **giao diện (interface)** để định nghĩa các hành vi chung và ẩn đi các chi tiết triển khai.

BÀI 2: Xây dựng Lớp Trừu Tượng Cho Hình Học

Yêu cầu:

- Viết một lớp trừu tượng Shape có phương thức calculateArea() và calculatePerimeter().
- Tạo hai lớp Circle và Rectangle kế thừa từ Shape và cài đặt các phương thức trên.

Phân tích:

1. **Lớp trừu tượng Shape**
    - Định nghĩa các phương thức calculateArea() và calculatePerimeter() nhưng không triển khai chi tiết.
    - Không thể tạo đối tượng từ Shape, chỉ có thể dùng nó làm lớp cha cho các hình cụ thể.
2. **Lớp Circle và Rectangle**
    - Kế thừa từ Shape và triển khai phương thức calculateArea() và calculatePerimeter().
    - Circle cần bán kính để tính diện tích và chu vi.
    - Rectangle cần chiều dài và chiều rộng để tính diện tích và chu vi.

BÀI 3: Tạo Giao Diện Cho Động Vật

Yêu cầu:

- Tạo một interface Animal với phương thức makeSound().
- Viết các lớp Dog và Cat triển khai interface này.

Phân tích:

1. **Giao diện Animal**
    - Chỉ chứa phương thức makeSound(), không có thân hàm.
    - Các lớp triển khai phải cung cấp cách thực thi cụ thể.
2. **Lớp Dog và Cat**
    - Dog có thể in ra tiếng "Woof!".
    - Cat có thể in ra tiếng "Meow!".

- Một lớp khác có thể triển khai Animal mà không cần sửa đổi giao diện.

BÀI 4: So sánh Abstract Class và Interface

Yêu cầu:

- So sánh sự khác nhau giữa abstract class và interface trong Java.
- Khi nào nên sử dụng mỗi loại?

Phân tích:

Tiêu chí	Abstract Class	Interface
Có thể chứa phương thức có thân không?	Có thể	Không (trừ khi dùng default)
Hỗ trợ đa kế thừa?	Không	Có
Khi nào sử dụng?	Khi các lớp con có nhiều điểm chung	Khi cần định nghĩa hành vi chung cho các lớp không liên quan

Ví dụ:

- Dùng **abstract class** cho Vehicle (có cả thuộc tính và phương thức).
- Dùng **interface** cho Flyable (mô tả hành vi có thể bay).

BÀI 5: Xây dựng Hệ Thống Phương Tiện Giao Thông

Yêu cầu:

- Tạo lớp trừu tượng Vehicle với phương thức startEngine() và stopEngine().
- Viết hai lớp Car và Motorcycle kế thừa Vehicle.

Phân tích:

1. **Lớp Vehicle**
    - Định nghĩa các phương thức trừu tượng startEngine() và stopEngine().
    - Có thể chứa thuộc tính như brand, model.
2. **Lớp Car và Motorcycle**
    - Car có thể mô tả cách khởi động bằng chìa khóa.
    - Motorcycle có thể mô tả cách khởi động bằng nút bấm.

BÀI 6: Mô phỏng Công Việc Con Người

Yêu cầu:

- Tạo interface Person với phương thức work().
- Viết hai lớp Teacher và Doctor triển khai interface này.

Phân tích:

1. **Interface Person**
    - Định nghĩa phương thức work().
2. **Lớp Teacher và Doctor**
    - Teacher triển khai work() với nội dung giảng dạy.
    - Doctor triển khai work() với nội dung khám chữa bệnh.

BÀI 7: Quản lý Thiết Bị Điện Tử

Yêu cầu:

- Tạo lớp trừu tượng ElectronicDevice.
- Viết hai lớp Smartphone và Laptop kế thừa nó.

Phân tích:

1. **Lớp ElectronicDevice**
    - Định nghĩa turnOn() và turnOff().
2. **Lớp Smartphone và Laptop**
    - Smartphone mô tả bật nguồn bằng cách nhấn nút.
    - Laptop mô tả bật nguồn khi mở nắp.

BÀI 8: Xây dựng Hệ Thống Ngân Hàng

Yêu cầu:

- Viết lớp trừu tượng BankAccount.
- Tạo hai lớp SavingsAccount và CheckingAccount kế thừa nó.

Phân tích:

1. **Lớp BankAccount**

- Định nghĩa deposit() và withdraw().

2. **Lớp SavingsAccount và CheckingAccount**
    - SavingsAccount có giới hạn rút tiền.
    - CheckingAccount có hỗ trợ thấu chi.

BÀI 9: Thiết Kế Nhân Vật Trong Game

Yêu cầu:

- Tạo lớp trừu tượng Character.
- Viết các lớp Warrior và Mage kế thừa nó.

Phân tích:

1. **Lớp Character**
    - Định nghĩa phương thức attack(), defend().
2. **Lớp Warrior và Mage**
    - Warrior tấn công bằng vũ khí cận chiến.
    - Mage tấn công bằng phép thuật.

BÀI 10: Hệ Thống Thanh Toán

Yêu cầu:

- Tạo interface Payment.
- Viết các lớp CreditCardPayment và PayPalPayment.

Phân tích:

1. **Interface Payment**
    - Định nghĩa processPayment().
2. **Lớp CreditCardPayment và PayPalPayment**
    - CreditCardPayment xử lý thanh toán qua thẻ.
    - PayPalPayment xử lý thanh toán qua tài khoản PayPal.

6. BÀI DỰ ÁN

DỰ ÁN 1: HỆ THỐNG QUẢN LÝ THÚ CƯNG

Mô tả bài toán:

Một cửa hàng thú cưng cần một hệ thống để quản lý thông tin về các loại thú cưng, bao gồm chó, mèo và cá cảnh. Hệ thống này cần cho phép:

- Thêm mới thú cưng vào danh sách.
- Hiển thị thông tin cơ bản của từng thú cưng.
- Mô phỏng âm thanh mà thú cưng phát ra (đối với chó, mèo).

Yêu cầu kỹ thuật:

1. **Sử dụng lớp trừu tượng (Abstract Class)**
    - Tạo một lớp Pet làm lớp trừu tượng với các thuộc tính chung như name, age, species và phương thức displayInfo().
    - Định nghĩa phương thức makeSound() nhưng không cài đặt.
2. **Xây dựng các lớp con kế thừa từ Pet**
    - Dog: Cài đặt phương thức makeSound() để in ra "Woof!"
    - Cat: Cài đặt phương thức makeSound() để in ra "Meow!"
    - Fish: Không có âm thanh nên có thể bỏ qua makeSound().
3. **Sử dụng giao diện (Interface)**
    - Tạo interface Trainable cho các loài có thể huấn luyện (chó, mèo).
    - Trong interface này, định nghĩa phương thức doTrick().
4. **Chức năng chính của hệ thống:**
    - Thêm thú cưng vào danh sách.
    - Hiển thị danh sách thú cưng.
    - Gọi phương thức makeSound() nếu có.

Hướng dẫn thực hiện:

- Sinh viên 1 thiết kế và triển khai lớp trừu tượng Pet và các lớp con.
- Sinh viên 2 thiết kế giao diện Trainable và cài đặt các phương thức liên quan.
- Cùng nhau kiểm thử chương trình và viết báo cáo ngắn về cách sử dụng tính trừu tượng trong bài toán này.

DỰ ÁN 2: HỆ THỐNG QUẢN LÝ PHƯƠNG TIỆN GIAO THÔNG

Mô tả bài toán:

Một công ty cho thuê xe muốn xây dựng hệ thống quản lý các phương tiện, bao gồm ô tô, xe máy và xe tải. Hệ thống cần có khả năng:

- Quản lý thông tin phương tiện như biển số, loại xe, số chỗ ngồi.
- Mô phỏng cách khởi động của từng loại xe.
- Xác định các loại xe có thể chở hàng.

Yêu cầu kỹ thuật:

1. **Sử dụng lớp trừu tượng (Abstract Class)**
    - Tạo lớp Vehicle với thuộc tính licensePlate, model và phương thức startEngine().
    - Định nghĩa phương thức getInfo().
2. **Xây dựng các lớp con kế thừa từ Vehicle**
    - Car: Triển khai startEngine() với mô tả "Bật khóa xe để khởi động".
    - Motorcycle: Triển khai startEngine() với mô tả "Nhấn nút khởi động".
    - Truck: Triển khai startEngine() với mô tả "Khởi động động cơ Diesel".
3. **Sử dụng giao diện (Interface)**
    - Tạo interface CargoVehicle để định nghĩa phương thức loadCargo().
    - Lớp Truck sẽ triển khai giao diện này để mô tả cách chở hàng.
4. **Chức năng chính của hệ thống:**
    - Thêm phương tiện vào danh sách quản lý.
    - Hiển thị thông tin phương tiện.
    - Mô phỏng cách khởi động của từng loại xe.
    - Xác định xe có khả năng chở hàng.

Hướng dẫn thực hiện:

- Sinh viên 1 thiết kế và triển khai lớp trừu tượng Vehicle và các lớp con.
- Sinh viên 2 thiết kế giao diện CargoVehicle và cài đặt phương thức liên quan.

- Cùng nhau kiểm thử chương trình, tối ưu mã nguồn và viết báo cáo ngắn phân tích tính trừu tượng trong hệ thống này.

---

5. KẾT LUẬN

- **Lớp trừu tượng** dùng khi có quan hệ kế thừa, cần dùng cả phương thức có thân và không thân.
- **Giao diện** dùng khi cần đảm bảo nhiều lớp có chung hành vi mà không cần quan hệ kế thừa.
- Áp dụng đúng giúp mã sạch, dễ mở rộng.