HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO

THỰC TẬP CƠ SỞ

Giảng viên: Kim Ngọc Bách

Tuần 1: 15/3/2025

Sinh viên: Nguyễn Khắc Trường B22DCCN884

Hà Nội, 3/2024

MỤC LỤC

Giới Thiệu

Dự án mà em định triển khai sẽ có phần BackEnd sử dụng công nghệ SpringBoot của java và java là ngôn ngữ code thuần hướng đối tượng, do vậy đầu tiên em cần ôn lại chắc nền tảng OOP. Do vậy ở tuần này em sẽ học lại kiến thức về Class, Abstract Class và Interface**.**

Phần 1: Khái niệm về Class, Abstract Class và Interface

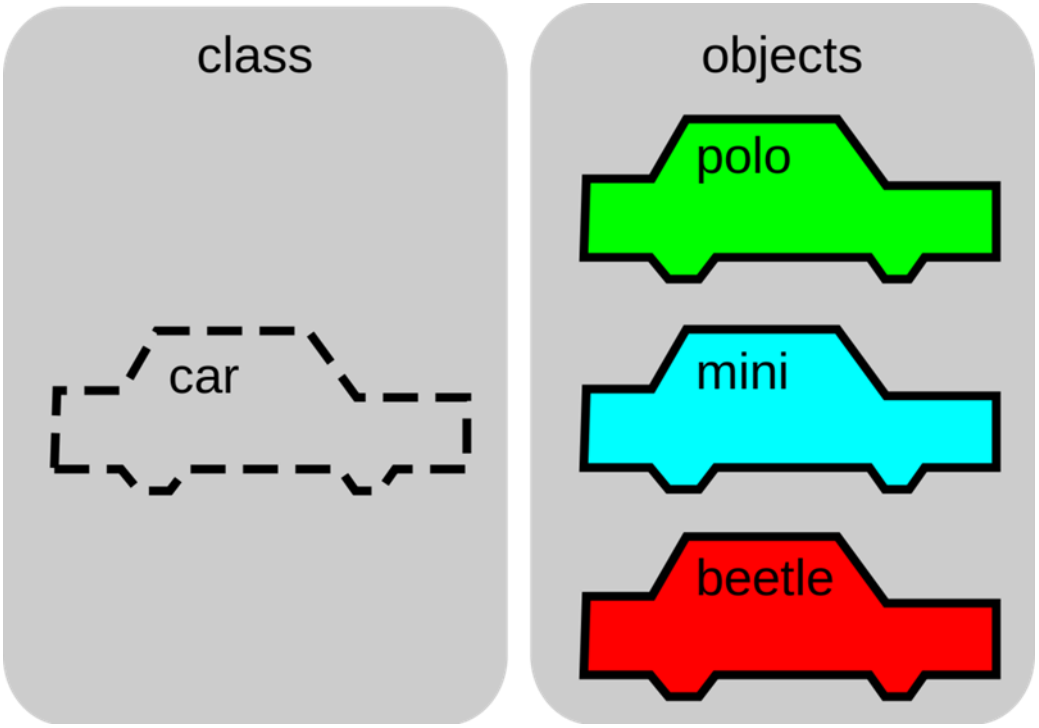
Lập trình hướng đối tượng (Object-Oriented Programming - OOP) là một phương pháp lập trình dựa trên các khái niệm về đối tượng (objects), cho phép việc tổ chức và quản lý mã nguồn một cách trực quan và hiệu quả. Trong OOP, mọi thứ đều được mô hình hóa thông qua các đối tượng, và các đối tượng này đại diện cho các thực thể trong thế giới thực, có thuộc tính (attributes) và hành vi (methods).

Các khái niệm cơ bản trong OOP bao gồm lớp (class), lớp trừu tượng (abstract class), và giao diện (interface), mỗi yếu tố này đều đóng vai trò quan trọng trong việc xây dựng hệ thống phần mềm theo mô hình hướng đối tượng.

1.1. Class

Đầu tiên để hiểu về class thì ta cần biết đến object trước, đối tượng (object) là một thực thể có thể đại diện cho các vật thể, khái niệm, hoặc sự kiện trong thế giới thực. Để dễ hiểu, hãy tưởng tượng một chiếc xe hơi cụ thể, như Honda Civic, là một đối tượng. Xe Honda Civic có các thuộc tính như trọng lượng 1.200kg và màu sắc đen, và các phương thức như lái và phanh.

Lớp (Class) giống như một bộ khung tạo ra đối tượng, hoặc là "bản thiết kế" để tạo ra các đối tượng. Khi một lớp được định nghĩa, nó không chiếm giữ bộ nhớ cho đến khi một đối tượng (instance) được tạo ra từ lớp đó. Lớp quy định các thuộc tính và phương thức mà đối tượng sẽ có, tạo ra sự nhất quán giữa các đối tượng cùng loại. Ví dụ, lớp "Xe hơi" sẽ quy định rằng mọi chiếc xe được tạo ra đều có các thuộc tính như màu sắc, trọng lượng và phương thức như lái, phanh.



Nhắc đến class ta nhắc đến 3 nguyên tắc quan trọng đó là tính đóng gói, kế thừa và đa hình:

+ Đóng gói (Encapsulation): Ẩn giấu các chi tiết triển khai của một đối tượng khỏi những phần bên ngoài sử dụng đối tượng đó. Việc đóng gói đối tượng cung cấp tính trừu tượng, vì chúng ta có thể sử dụng đối tượng mà không cần biết cách nó hoạt động bên trong.

+ Đa hình (Polymorphism): Cho phép các đối tượng thuộc các lớp khác nhau được xử lý bằng cách sử dụng cùng một phương thức hoặc giao diện. Tính đa hình cung cấp tính linh hoạt, vì cùng một phương thức có thể hoạt động theo nhiều cách khác nhau tùy thuộc vào đối tượng đang sử dụng nó.

+ Kế thừa (Inheritance): Là một khái niệm trong lập trình hướng đối tượng, cho phép một lớp kế thừa các thuộc tính và phương thức từ một lớp khác, giúp tái sử dụng mã nguồn và giảm thiểu sự lặp lại trong code. Khi mối quan hệ kế thừa được thiết lập giữa hai lớp, lớp con (subclass) có thể kế thừa toàn bộ trạng thái (các thuộc tính) và hành vi (các phương thức) từ lớp cha (superclass), từ đây ta có khái niệm về Abstract Class.

1.2. Abstract Class

Lớp trừu tượng (abstract class) là một lớp không thể được khởi tạo trực tiếp mà phải được kế thừa bởi một lớp con khác để có thể sử dụng các thuộc tính và phương thức của nó. Lớp trừu tượng đóng vai trò như một khuôn mẫu, cho phép bạn định nghĩa những phương thức chưa hoàn chỉnh mà các lớp con bắt buộc phải triển khai.

Lớp trừu tượng là một thành phần quan trọng trong lập trình hướng đối tượng, giúp tạo ra sự linh hoạt trong thiết kế hệ thống. Nó cho phép bạn xây dựng những lớp cơ sở chung với những đặc điểm và hành vi chung, trong khi vẫn để các lớp con tùy chỉnh và hoàn thiện chi tiết cụ thể. Điều này giúp giảm thiểu việc lặp lại mã nguồn và tạo ra cấu trúc kế thừa rõ ràng, dễ bảo trì.

Lớp trừu tượng được sinh ra là để kế thừa, nó không có hàm khởi tạo đồng nghĩa không thể khởi tạo chính nó.

Vậy ta đặt câu hỏi rằng lớp trừu tượng sinh ra để làm gì khi lớp thường cũng có thể được kế thừa và nó còn có cả lớp khởi tạo? Câu trả lời đó là tính Đa hình (Polymorphism). Giả sử luật sư, thư ký, giám đốc điều hành đều có những điểm mà chúng ta muốn biết, vậy chẳng nhẽ cả ba lớp đó đều phải khai báo những phương thức và thuộc tính giống hệt nhau? Thay vào đó, chúng ta có thể tạo ra một lớp cha (ví dụ, lớp Employee) với các phương thức và thuộc tính chung cho tất cả.

Các lớp con như Luật sư, Thư ký, và Giám đốc điều hành sẽ kế thừa từ lớp Employee, sử dụng lại các phương thức và thuộc tính chung, đồng thời có thể mở rộng hoặc ghi đè các phương thức riêng của mình.

1.3, Interface

Interface là một cách để chỉ định những khả năng mà các đối tượng có thể thực hiện, mà không chỉ rõ cách thức thực hiện. Nó định nghĩa một tập hợp các phương thức mà các lớp phải triển khai, nhưng không cung cấp cài đặt cụ thể cho những phương thức đó.

Các biến kiểu interface có thể gọi bất kỳ phương thức nào được liệt kê trong interface, nhưng kết quả sẽ phụ thuộc vào lớp cụ thể mà chúng tham chiếu. Điều này có nghĩa là các biến kiểu interface cho phép đa hình (polymorphism).

Nghe interface có vẻ rất giống với abstract class chỉ là abstract class có những phương thức đã được gán logic code sẵn, vậy tại sao không dùng abstract class? Chúng đều có tính đa hình cơ mà?

Vậy ta thử nghĩ đến trường hợp rằng ta có 2 lớp là nhà thám hiểm và nhà bác học đều có thuộc tính là "nghiên cứu" nhưng nội dung của có hoàn toàn khác nhau, 1 bên là nghiên cứu khoa học, 1 bên là nghiên cứu xem hang động đó có gì, cùng là 1 cái tên nhưng nội dung hoàn toàn khác nhau và thuộc tính này nằm trong interface "Công việc", có thể thấy "Công việc" chỉ là điểm chung của cả 2 class kể trên chứ nó không thể là cha, quan hệ isA được vì 1 bên là 1 người 1 bên là khái niệm. Vì vậy đây cũng là 1 phần lý do sinh ra interface.

Phần 2: Điểm giống và khác nhau của class và abstract class

2.1. Điểm giống nhau

- Class và abstract class đều là 1 kiểu dữ liệu mô tả một nhóm các đối tượng có các đặc điểm chung

- Đều có các thành phần chính như: các thuộc tính (các fields, biến instance) chứa trạng thái của đối tượng và các method mô tả hành vi của đối tượng.

- Các method trong cả class và abstract class đều có 3 mức độ truy nhập (public, private, protected) và có thể có các phương thức tĩnh (static).

2.2. Điểm khác nhau

- Khai báo:
    + Class: dùng từ khóa class
    + Abstract class: cần có từ khóa abstract
- Khởi tạo đối tượng:
    + Class: Có thể khởi tạo trực tiếp đối tượng từ lớp
    + Abstract class: không thể khởi tạo đối tượng trực tiếp từ lớp trừu tượng
- Các method:

+ Class: Chỉ chứa phương thức thông thường (có định nghĩa, có phần thân). Không thể chứa phương thức trừu tượng

```
class Animal {
    void sound() {
        System.out.println("This is an animal sound.");
    }
}
```

+ Abstract class: Có thể chứa cả phương thức trừu tượng (chỉ có tên phương thức, không có phần thân) và phương thức thông thường

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

    void sleep() {  // Phương thức thông thường
        System.out.println("This animal is sleeping.");
    }
}
```

- Về tính thừa kế:

+ Class: Khi một lớp con kế thừa một lớp thông thường, nó không bắt buộc phải override bất kỳ phương thức nào

+ Abstract class: Một lớp kế thừa từ lớp trừu tượng (subclass – lớp con) không cần phải implement non-abstract methods, nhưng những method nào có abstract thì bắt buộc phải override. Trừ khi subclass cũng là abstract

```
abstract class Animal {
    abstract void sound();
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Woof!");
    }
}
```

- Mức độ trừu tượng:

+ Class: không hỗ trợ bất kỳ tính trừu tượng nào, mọi phương thức đều phải có phần thân

+ Abstract class: có thể có cả tính trừu tượng và không trừu tượng, tùy vào thiết kế của lớp

- Mục đích sử dụng:

+ Class: được sử dụng để tạo các đối tượng và mô tả hành vi cụ thể của đối tượng đó.

Ví dụ: Một lớp Dog có thể mô tả tất cả các hành vi cụ thể của một con chó.

+ Abstract class: dùng để cung cấp 1 khuôn mẫu (template) cho các lớp con. Nó mô tả những hành vi mà các lớp con phải triển khai, nhưng không chỉ ra cách thức thực hiện chi tiết (đối với các phương thức trừu tượng).

Ví dụ: Lớp Animal có thể khai báo một phương thức trừu tượng sound(), yêu cầu các lớp con phải triển khai hành vi của phương thức này.

Phần 3: Điểm giống và khác nhau của abstract class và interface

3.1. Điểm giống nhau

- Định nghĩa phương thức mà không có thân hàm:

Trong cả abstract class và interface có thể khai báo các phương thức mà không cung cấp cài đặt. Điều này yêu cầu các lớp con hoặc lớp triển khai phải cài đặt phương thức đó.

- Không thể khởi tạo trực tiếp:

Cả abstract class và interface đều không thể khởi tạo trực tiếp thành đối tượng. Các lớp con có thể kế thừa chúng và cài đặt các phương thức được khai báo bên trong lớp con.

- Dùng để hỗ trợ tính đa hình (polymorphism):

Cả abstract class và interface đều được sử dụng để hỗ trợ tính đa hình. Có thể tham chiếu đến các lớp con thông qua một biến kiểu của lớp trừu tượng hoặc giao diện. Điều này cho phép linh hoạt trong việc xử lý các đối tượng theo cùng một cách, mặc dù chúng có thể thuộc các lớp khác nhau.

3.2. Điểm khác nhau

- Khái niệm: Abstract class là một lớp nhưng interface thì không phải là một lớp mà là một khung mẫu.

- Phương thức:

+ Abstract class có thể chứa abstract method và phương thức cụ thể.

```
// Abstract class
abstract class Animal {
    public int n = 10;
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}
```

+ Interface chỉ có thể chứa các phương thức không có phần thân.

```
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}
```

+ Đa kế thừa: Một lớp có thể kế thừa nhiều interface nhưng chỉ có thể kế thừa một abstract class.

+ Chế độ truy cập: Mặc định, trong interface, tất cả các phương thức ở chế độ public hoặc abstract, thuộc tính ở chế độ public, static và final phải là public. Còn trong abstract class, phương thức và thuộc tính có thể sử dụng bất kỳ chế độ truy cập nào (public, protected, private)

+ Thuộc tính: interface và abstract class đều chứa biến. Tuy nhiên, lớp con không thể khai báo biến cùng tên với biến ở abstract class nhưng có thể khai báo lại biến cùng tên biến của interface (cần chú ý khi dùng hằng số này). Biến được khai báo trong interface là hằng số còn trong abstract class biến có thể thay đổi.

+ Mục đích sử dụng: interface thường được dùng để định nghĩa hành vi mà các lớp khác nhau có thể thực hiện (mang tính mô tả), trong khi abstract class được dùng để làm cơ sở cho các lớp cùng bản chất.

→ Do đó:

- Khi một nhóm đối tượng có cùng bản chất kế thừa từ một class thì sử dụng abstract class.
- Khi một nhóm đối tượng không có cùng bản chất nhưng chúng có hành động giống nhau thì sử dụng interface.

Phần 4: Điểm giống và khác nhau của class và interface

4.1. Điểm giống nhau

- Đều là các kiểu dữ liệu: Cả interface và class đều là các kiểu dữ liệu trong Java, dùng để định nghĩa các thành phần mà lập trình viên có thể sử dụng.

- Đều có thể chứa phương thức: Cả interface và class đều có thể chứa các phương thức (method), nhưng cách thức định nghĩa và sử dụng khác nhau.

- Tính trừu tượng: Cả interface và class đều có thể được sử dụng để trừu tượng hóa một nhóm các hành vi (behaviors) trong lập trình hướng đối tượng.

4.2. Điểm khác nhau

Đặc điểm	**Interface**	**Class**
Cấp độ trừu tượng	Hoàn toàn trừu tượng (trước Java 8, chỉ chứa phương thức trừu tượng). Sau Java 8, có thể chứa phương thức mặc định (default methods) và phương thức tĩnh (static methods).	Có thể là lớp trừu tượng hoặc không trừu tượng. Chứa cả phương thức cụ thể và trừu tượng.
Kế thừa	Một lớp có thể implements nhiều interface.	Một lớp chỉ có thể extends một class.

Định nghĩa phương thức	Không chứa phần thân phương thức (ngoại trừ phương thức mặc định và tĩnh).	Có thể chứa cả phần thân phương thức.
Thuộc tính	Các biến trong interface mặc định là public, static, và final.	Các biến có thể là bất kỳ loại truy cập nào (private, protected, public) và có thể thay đổi giá trị.
Mục đích sử dụng	Dùng để định nghĩa một tập hợp các hành vi mà nhiều lớp có thể thực hiện, không quan trọng chúng nằm trong cùng cây kế thừa.	Dùng để định nghĩa một đối tượng cụ thể hoặc nhóm đối tượng có liên quan, thường dựa trên cấu trúc cây kế thừa.
Từ khóa sử dụng	Sử dụng từ khóa implements khi một lớp triển khai interface.	Sử dụng từ khóa extends khi một lớp kế thừa từ lớp khác.
Constructor	Không có constructor.	Có constructor, dùng để khởi tạo đối tượng.
Access Modifier	Tất cả các phương thức trong interface mặc định là public.	Các phương thức có thể có các access modifier khác nhau (private, protected, public).

- Ví dụ minh họa

+ Ví dụ về interface:

```
// Định nghĩa một interface
public interface Animal {
    void sound(); // Phương thức trừu tượng
    default void sleep() { // Phương thức mặc định
```

```
        System.out.println("Sleeping...");
    }
}

// Một lớp Dog triển khai interface Animal
public class Dog implements Animal {
    @Override
    public void sound() {
        System.out.println("Woof Woof");
    }
}

// Một lớp Cat triển khai interface Animal
public class Cat implements Animal {
    @Override
    public void sound() {
        System.out.println("Meow Meow");
    }
}
```

+ Ví dụ về class:

```
// Định nghĩa một lớp trừu tượng
public abstract class Vehicle {
    // Phương thức trừu tượng
    abstract void move();
```

```
    // Phương thức cụ thể
    public void fuel() {
        System.out.println("Fueling the vehicle...");
    }
}

// Một lớp Car kế thừa lớp Vehicle
public class Car extends Vehicle {
    @Override
    public void move() {
        System.out.println("Car is moving");
    }
}

// Một lớp Bike kế thừa lớp Vehicle
public class Bike extends Vehicle {
    @Override
    public void move() {
        System.out.println("Bike is moving");
    }
}
```

→ Tóm lại:

Interface: Phù hợp khi bạn cần nhiều lớp có thể có những hành vi chung nhưng không có mối quan hệ kế thừa rõ ràng, ví dụ: Animal, Vehicle, Drawable.

Class: Phù hợp khi bạn muốn tổ chức các lớp theo cấu trúc cây kế thừa, ví dụ: Vehicle -> Car và Bike.

Phần 5: Kết luận

Trong lập trình hướng đối tượng (OOP), các khái niệm về class, abstract class và interface đều đóng vai trò quan trọng trong việc thiết kế và phát triển phần mềm. Class giúp chúng ta mô hình hóa đối tượng, hỗ trợ tính đóng gói, kế thừa và đa hình, từ đó giúp tạo ra các đối tượng với các hành vi và thuộc tính cụ thể. Abstract class giúp định nghĩa những khuôn mẫu chung mà các lớp con phải kế thừa và triển khai, cung cấp sự linh hoạt và tối ưu hóa trong việc tổ chức mã nguồn. Interface giúp định nghĩa các phương thức trừu tượng, cho phép nhiều lớp khác nhau triển khai các hành vi tương tự mà không cần có mối quan hệ cha-con trực tiếp.

Mặc dù class và abstract class có nhiều điểm tương đồng, nhưng abstract class phù hợp hơn khi cần tạo ra sự trừu tượng hóa với các phương thức chung và định nghĩa hành vi mà các lớp con phải thực hiện. Trong khi đó, interface cho phép đa kế thừa, giúp các lớp không liên quan về bản chất có thể chia sẻ hành vi giống nhau. Tùy vào yêu cầu cụ thể của từng bài toán, chúng ta cần biết cách chọn lựa giữa class, abstract class và interface để có được thiết kế phần mềm hiệu quả, linh hoạt và dễ mở rộng.