oop(객체지향 프로그래밍),AOP,FP

객체지향 프로그래밍

객체 지향 프로그래밍은 OOP(Object Oriented Programming)이라고도 한다.

프로그래밍에서 필요한 데이터를 추상화시켜 상태(속성, 어트리뷰트)와 행위(메서드)를 가진 객체를 만들고 그 객체들 간의 유기적인 상호작용을 통해 로직을 구성하는 프로그래밍 방법이다.

```
class Calculator {
    static int result = 0;

    static int add(int num) {
        result += num;
        return result;
    }
}


public class Sample {
    public static void main(String[] args) {
        System.out.println(Calculator.add(3));
        System.out.println(Calculator.add(4));
    }
}
```

```
class Calculator1 {
    static int result = 0;

    static int add(int num) {
        result += num;
        return result;
    }
}

class Calculator2 {
```

```
    static int result = 0;

    static int add(int num) {
        result += num;
        return result;
    }
}


public class Sample {
    public static void main(String[] args) {
        System.out.println(Calculator1.add(3));
        System.out.println(Calculator1.add(4));

        System.out.println(Calculator2.add(3));
        System.out.println(Calculator2.add(7));
    }
}
```

```
class Calculator {
    int result = 0;

    int add(int num) {
        result += num;
        return result;
    }
}


public class Sample {
    public static void main(String[] args) {
        Calculator cal1 = new Calculator();  // 계산기1 객체를 생
        Calculator cal2 = new Calculator();  // 계산기2 객체를 생

        System.out.println(cal1.add(3));
        System.out.println(cal1.add(4));

        System.out.println(cal2.add(3));
```

```
        System.out.println(cal2.add(7));
    }
}
```

[장점]

- 코드의 재사용성이 높다.
  - 누군가가 만든 클래스를 가져와 사용할 수 있고 상속을 통해 확장할 수도 있다.
- 유지보수가 쉽다.
  - 수정해야 할 부분이 클래스 내부에 멤버 변수 혹은 메소드로 존재하기 때문에 해당 부분만 수정하면 된다.
- 대형 프로젝트에 적합하다.
  - 클래스 단위로 모듈화시켜서 개발할 수 있으므로 업무 분담하기가 쉽다.

[단점]

- 처리 속도가 상대적으로 느리다.
  - OOP는 객체 간의 상호 작용을 통해 프로그램이 동작을 하여 메소드 호출과 같은 추가적인 작업이 필요하여 느립니다.
- 객체가 많으면 용량이 커질 수 있다.
  - 각각의 객체는 메소드와 속성을 가지기에 이들은 메모리를 차지합니다.
- 설계 시 많은 노력과 시간이 필요하다.
  - 객체 지향 프로그래밍은 객체 간의 관계와 역할, 책임을 정의하는 것이 중요해서 설계에 많은 시간을 투자하는데 이것이 코딩 작업보다 복잡하고 시간이 많이 소요될 수 있습니다.

객체 지향 프로그래밍의 특징

- 추상화, 캡슐화, 상속, 다형성, 클래스, 인스턴스
- 추상화

복잡한 시스템을 단순화시켜 필요한 정보만을 강조하고, 불필요한 정보는 숨기고 필요한 정보만을 표현함으로써 공통의 속성이나 기능을 묶어 이름을 붙이는 것이다.

이를 통해 코드의 복잡성을 줄이고, 필요한 정보에만 집중할 수 있게 되어 코드의 가독성과 유지 보수성이 향상됩니다.

```
public abstract class 차량 {
    //속성
    private String 차량번호;
    private String 차량색상;

    //메소드
    public abstract void 가속하기();
    public abstract void 정지하기();
}

public class 자동차 extends 차량 {
    //속성
    private int 현재속도;

    //메소드
    @Override
    public void 가속하기() {
        현재속도 += 10;
        System.out.println("현재 속도: " + 현재속도);
    }

    @Override
    public void 정지하기() {
        현재속도 = 0;
        System.out.println("차량이 정지했습니다.");
    }
}
```

- 캡슐화

코드를 수정없이 재활용 하는 것을 목적으로 함. 캡슐화는 객체의 속성과 메서드를 하나로 묶는 것을 말합니다. 이를 통해 객체의 내부 구현을 숨기고, 외부에서는 노출된 메서드를 통해서만 접근이 가능하게 함으로써 데이터 보호와 코드의 안정성을 높일 수 있습니다.

은닉화와의 차이 - 은닉화는 캡슐화의 일부라고 볼 수 있으며, 목적으로 묶인 캡슐 안을 사용자는 볼 수 없다는 것이 은닉화.

```
public class 사람 {
    // private 속성을 통해 외부 접근을 제한
```

```
    private String 이름;
    private int 나이;

    // getter, setter 메소드를 통해 속성에 접근
    public String get이름() {
        return 이름;
    }

    public void set이름(String 이름) {
        this.이름 = 이름;
    }

    public int get나이() {
        return 나이;
    }

    public void set나이(int 나이) {
        if (나이 > 0) { // 유효성 검사를 통해 나이가 0보다 큰 경우에만
            this.나이 = 나이;
        }
    }
}
```

- 상속

부모 클래스의 속성과 기능을 그대로 이어 받아 사용할 수 있게 하고 기능의 일부분을 변경해야 할 경우, 상속 받은 자식 클래스에서 해당 기능만 다시 수정(정의)하여 사용할 수 있게 하는 것이다.

상속을 통해서 클래스를 작성하면 보다 적은 양의코드로 새로운 클래스를 작성할 수 있다.

또한, 코드를 공통적으로 관리하여 코드 추가 및 변경이 용이하다.

```
public class 동물 {
    // 속성
    private String 이름;

    // 메소드
    public void 먹다() {
        System.out.println(이름 + "가 먹습니다.");
```

```
    }

    // getter, setter
    public String get이름() {
        return 이름;
    }

    public void set이름(String 이름) {
        this.이름 = 이름;
    }
}

public class 강아지 extends 동물 {
    // 추가적인 속성
    private String 종;

    // 추가적인 메소드
    public void 짖다() {
        System.out.println(get이름() + "가 짖습니다.");
    }

    // getter, setter
    public String get종() {
        return 종;
    }

    public void set종(String 종) {
        this.종 = 종;
    }
}
```

- 다형성

하나의 변수명, 함수명 등이 상황에 따라서 다른 의미로 해석될 수 있는 것이다.

즉, 오버라이딩, 오버로딩이 가능하다.

> 오버라이딩(Overriding): 상위 클래스에서 정의한 메소드를 하위 클래스에서 재정의하여 사용하는 것을 의미합니다. 이를 통해 상위 클래스의 메소드를 하위 클래스에서 필요에 맞게 변경하여 사용할 수 있습니다.

오버로딩(Overloading): 같은 이름의 메소드를 여러 개 정의하되, 매개변수의 타입이나 개수를 다르게 하는 것을 의미합니다. 이를 통해 같은 이름의 메소드를 다양한 방식으로 사용할 수 있습니다.

- 클래스

어떤 문제를 해결하기 위한 데이터를 만들기 위해 추상화를 거쳐 집단에 속하는 속성과 행위를 변수와 메서드로 정의한 것으로 객체를 만들기 위한 메타정보라고 볼 수 있다.

현실 세계의 객체를 추상화시켜, 속성과 메서드로 정의한 것(논리적 개념)

- 인스턴스

클래스에서 정의한 것을 토대로 실제 메모리에 할당된것으로 실제 프로그램에서 사용되는 데이터

클래스는 객체를 생성하기 위한 템플릿으로, 객체의 초기 상태(속성)와 가능한 행동(메서드)을 정의합니다. 반면, 인스턴스는 클래스를 기반으로 생성된 실체로, 메모리에 할당된 객체를 의미합니다. 즉, 클래스는 설계도와 같고, 인스턴스는 그 설계도를 바탕으로 만들어진 하나의 실체를 나타냅니다.

객체 지향 설계 원칙

- SOLID

1. SRP(Single Responsibility Principle) : 단일 책임 원칙

클래스는 단 하나의 책임을 가져야 하며, 클래스를 변경하는 이유는 단 하나의 이유여야 한다.

1. OCP(Open Closed Principle) : 개방 폐쇄 원칙

각 클래스는 클래스에 대한 수정을 폐쇄하고, 확장에 대해 개방해야 한다. 이는 기존의 코드를 변경하지 않고 기능을 추가할 수 있도록 설계가 되어야 함을 말한다.

즉 클래스를 수정해야 한다면 그 클래스를 상속, 즉 확장하여 수정한다.

1. LSP(Liskov Substitution Principle) : 리스코프 치환 원칙

자식 클래스를 사용 중일때, 거기에 부모 클래스로 치환하여도 문제가 없어야 한다.

즉 서브 타입은 언제나 그것의 상위 타입으로 교체할 수 있어야 한다.

1. ISP(Interface Segregation Principle) : 인터페이스 분리 원칙

인터페이스는 그 인터페이스를 사용하는 클라이언트를 기준으로 분리해야 한다.

핸드폰을 예로 들면, 전화를 하는데 핸드폰 카메라가 방해가 되면 안된다는 말.

1. DIP(Dependency Inversion Principle) : 의존 역전 원칙

상위 클래스가 하위 클래스에 의존하면 안된다는 법칙. 즉 기본적인 공통되는 속성을 하위 클래스에 의존하면 안된다.

나머지

- AOP (관점지향 프로그래밍)

  스프링 프레임워크의 핵심 요소 중 하나. 비즈니스 로직과 공통 모듈로 분리하고, 핵심 로직 사이사이에 공통 모듈을 잘 끼워 넣는 것을 말함. 이때 공통 모듈을 코드 밖에서 설정된다는 것이 핵심. 인증, 로깅, 트랜잭션 처리에 용이.

- FP (함수형 프로그래밍)

  함수형 프로그래밍은 선언형 프로그래밍으로, 어떻게(How)가 아닌 무엇(What)을 정의한다. C, Java등의 언어는 명령형 프로그래밍이며, 알고리즘을 기술하고 목적은 기술하지 않는다. 선언형은 반대로 알고리즘은 기술하지 않고 목적 위주로 기술하며, 데이터의 입력이 주어지고 데이터의 흐름을 추상적을 정의하는 방식.

> 💡 객체지향은 동작하는 부분을 캡슐화하여 이해를 돕고, 함수형은 동작하는 부분을 최소화하여 이해를 돕는다.

절차지향과 객체지향의 차이점은?

오버로딩과 오버라이딩은 무엇인가요?

캡슐화의 protect,default는 무엇인가요?

get과 post의 차이

출처비교는 어디서 하나요?

sop와 cors에 대해 설명을 해주세요

테스트를 진행할 때 postman에서 되는데 웹 브라우저에서는 안되는 이유에 대해 설명해주세요

출처가 다른 open API는 어떻게 가져왔나요?

CORS정책을 허용하는 리소스에 대해선 다른 출처라도 받아 드리기로 함

CORS 동작 방식은 뭔가요?

CORS 동작 3가지 시나리오는?

CORS를 사용할 때 nginx를 사용을 했는데 어떻게 진행을 하는지

테스트코드는 왜 사용하나요?

클린코드는 왜 사용하나요?

프로젝트에서 클린코드를 사용하고자하면 어떻게 사용하나요?

TDD의 개발주기에 대해 설명해주세요

식별자의 특징에 대해 설명해주세요

DB 외래키는 언제 사용하나요?

DB 기본키는 언제 사용하나요?