Freaky Bug Public Class #01

Object-Oriented Programming (OOP)

객체지향 프로그래밍 기초

객체지향이란?

객체지향(Object-Oriented)은 컴퓨터 프로그래밍 및 소프트웨어 설계에서 사용되는 패러다임이다. 객체지향 프로그래밍(Object-Oriented Programming, OOP)은 프로그램을 개발하기 위한 방법론 중 하나로, 현실 세계에 존재하는 개체들을 모델링하여 코드로 구현하려는 시도이다.

```
*패러다임(영어: paradigm)은 어떤 한 시대 사람들의 견해나 사고를 근본적으로 규정하고 있는 테두리로서의 인식의 체계, 또는 사물에 대한 이론적인 틀이나 체계를 의미하는 개념이다.
```

객체지향의 핵심개념

객체(Object): 소프트웨어에서 모델링할 대상을 의미하며, 속성(데이터)와 메서드(동작)를 가진다. 객체는 현실 세계의 개체를 추상화한 것으로, 상태와 행동을 함께 캡슐화하여 프로그램의 구조를 표현한다.

객체지향의 핵심개념

클래스(Class): 객체를 생성하기 위한 틀로, 객체의 속성과 메서드를 정의한다. 클래스를 통해 객체들이 공통된 구조와 행동을 공유할 수 있다.

객체지향의 핵심개념

상속(Inheritance): 한 클래스가 다른 클래스의 속성과 메서드를 물려받는 것이다. 상속을 통해 코드의 중복을 줄이고, 모듈화 및 재사용성을 향상시킬 수 있다.

객체지향의 핵심개념

캡슐화(Encapsulation): 객체의 속성과 메서드를 외부로부터 숨기고, 접근 제어를 통해 객체의 상태와 행동을 보호한다. 이를 통해 코드의 안정성을 높이고, 유지보수를 용이하게 한다.

객체지향의 핵심개념

다형성(Polymorphism): 하나의 인터페이스에 여러 객체들이 서로 다른 동작을 구현할 수 있게 하는 것이다. 다형성을 통해 코드의 유연성과 확장성을 높일 수 있다.

객체지향의 핵심개념

객체지향 프로그래밍은 이러한 개념을 바탕으로 프로그램을 구조화하고 개발한다.
이를 통해 개발자들은 보다 직관적이고 모듈화된 코드를 작성할 수 있으며,
이는 유지보수와 확장성이 뛰어난 소프트웨어를 만드는데 기여한다.
주요 객체지향 프로그래밍 언어로는 자바(Java), C++, C#, 파이썬(Python) 등이 있다.

객체지향의 근본적인 원리

객체지향은 김춘수 시인의 꽃 과 같다.

꽃 - 김춘수

내가 그의 이름을 불러주기 전에는
그는 다만
하나의 몸짓에 지나지 않았다.

내가 그의 이름을 불러주었을 때,
그는 나에게로 와서
꽃이 되었다.

내가 그의 이름을 불러준 것처럼
나의 이 빛깔과 향기에 알맞는
누가 나의 이름을 불러다오.
그에게로 가서 나도
그의 꽃이 되고 싶다.

우리들은 모두
무엇이 되고 싶다.
너는 나에게 나는 너에게
잊혀지지 않는 하나의 눈짓이 되고 싶다.

객체지향

개쩌는 이제석.

```
내가 함수를 호출하기 전에는
함수는 다만
하나의 몸짓에 지나지 않았다.

내가 함수의 이름을 불러주었을 때,
함수는 나에게로 와서
return을 주었다.

...(생략)
```

객체지향의 근본적인 이야기

위에서 어려운 말들로 여러분들의 머릿속을 뒤 흔들었을 것이다.
하지만 위 시에서 보았듯 객체지향은 어렵거나. 이상한 내용이 아니다.

간단한 클레스 코드

```
public class Dog {
    private String name;
    private int age;

    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public static void main(String[] args) {
        Dog myDog = new Dog("Max", 5);
        System.out.println("My dog's name is " + myDog.getName() + " and he is " + myDog.getAge() + " years old.");
    }
}
```

```java
// 클래스와 객체에 대한 예제
class Dog {
    String breed;
    int age;

    Dog(String breed, int age) {
        this.breed = breed;
        this.age = age;
    }

    void bark() {
        System.out.println("Woof!");
    }
}
```

```java
// 상속에 대한 예제
class Labrador extends Dog {
    Labrador(int age) {
        super("Labrador", age);
    }

    // 다형성에 대한 예제: 메소드 오버라이딩
    @Override
    void bark() {
        System.out.println("Labrador barking!");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        // 객체 생성
        Dog myDog = new Dog("Poodle", 5);
        myDog.bark();  // 출력: "Woof!"

        // 상속과 다형성 확인
        Dog myLabrador = new Labrador(3);
        myLabrador.bark();  // 출력: "Labrador barking!"
    }
}
```

```java
public class Dog {
    private String breed;  // 캡슐화된 필드
    private int age;       // 캡슐화된 필드

    public Dog(String breed, int age) {
        this.breed = breed;
        this.age = age;
    }

    // breed 필드에 대한 getter 메소드
    public String getBreed() {
        return breed;
    }

    // breed 필드에 대한 setter 메소드
    public void setBreed(String breed) {
        this.breed = breed;
    }

    // age 필드에 대한 getter 메소드
    public int getAge() {
        return age;
    }

    // age 필드에 대한 setter 메소드
    public void setAge(int age) {
        this.age = age;
    }
}
```

Getter 메소드: 클래스의 필드를 읽는 데 사용되는 메소드이다. 일반적으로 get으로 시작하는 이름을 가지며, 해당 필드의 값을 반환한다. 예를 들어, 필드 age에 대한 getter 메소드는 getAge라는 이름을 가질 수 있으며, 이 메소드는 age 필드의 값을 반환한다.

Setter 메소드: 클래스의 필드를 수정하는 데 사용되는 메소드이다. 일반적으로 set으로 시작하는 이름을 가지며, 새 값을 매개변수로 받아 해당 필드에 설정한다. 예를 들어, 필드 age에 대한 setter 메소드는 setAge라는 이름을 가질 수 있으며, 이 메소드는 새로운 나이를 매개변수로 받아 age 필드에 설정한다.

```java
public class Main {
    public static void main(String[] args) {
        // Dog 클래스의 인스턴스 생성
        Dog myDog = new Dog("Poodle", 5);

        // getter 메소드를 사용하여 breed와 age를 출력
        System.out.println("My dog's breed is " + myDog.getBreed() +
        " and he is " + myDog.getAge() + " years old.");

        // setter 메소드를 사용하여 breed와 age를 변경
        myDog.setBreed("Labrador");
        myDog.setAge(3);

        // 변경된 breed와 age를 출력
        System.out.println("My dog's breed is now " + myDog.getBreed() +
        " and he is now " + myDog.getAge() + " years old.");
    }
}
```

끝.