OOP, Java study

<OOP란?>

- Object-Oriented Programming (객체 지향 프로그래밍)
- 절차적 프로그래밍 방식을 개선한 방식
- 코드의 재사용성의 증가와 유지보수가 수월해짐

<OOP의 4가지 특징>

추상화 Abstraction

- 내부 구현을 숨기고 기능만 보여준다.
- 객체의 공통적인 특징을 하나의 클래스로 표현한다.
- 반복적인 코드를 줄이게 한다.
- abstract behavior의 구현을 바꿀 수 있다.
- 부분 추상화 → abstract class
- 전체 추상화 → interface

상속 Inheritance

- 자식 클래스가 부모 클래스의 특성, 기능을 상속받는다.
- 자식 클래스에서 부모 클래스의 특성, 기능을 확장, 수정할 수 있다.
- 여러 클래스의 중복되는 특성, 기능을 재사용할 수 있다.
- In Java, 여러 단계 상속이 불가능하다.
- In Java, 다중 상속이 불가능하다.

캡슐화 Encapsulation

- 내부 구현을 숨기고 기능만 보여준다.
- 데이터를 외부에서 접근하는 것을 막고 함수를 통해서 접근하도록 한다.

다형성 Polymorphism

- 서로 다른 클래스의 객체가 같은 요청을 받았을 시에 각자 다른 방식으로 동작한다.
- method overloading - 이름은 같은데 parameter-list가 다른 경우
- method overrding - 상속 시, 해당 클래스 객체에 맞게 호출

<Inheritance의 upcastin / downcasting>

upcasting

- child object →저장→ parent reference
- parent members만 접근 가능, child members는 접근 불가능

```
Parent base;
Child derived = new Child("Lee", 2021);
base = derived // upcasting
```

downcasting

- parent object →typecasting→ child object
- 묵시적으로 downcasting이 되지 않으므로 명시해주어야 한다.

```
Parent base1 = derived;
Child derived1 = base1; // compile error
Child derived1 = (Child)base2 // downcasting
```

<Interface vs Abstract Class>

Interface

- constants/abstract methods를 가질 수 있다. (오직 funtion signiture)
- In Java 8, dafault/static methods를 가질 수 있다.
- interface를 implements하는 class는 무조건 구현해야 한다.
- Java Instance의 member는 "public"이 default

Interface example

```
// IShape.java
interface IShape {
    double calcArea();
    double calcPerimeter();
}

// RectangleImp1.java
class RectangleImp1 implements IShape {
    private double width, height;

    public RectangleImp1(double width, double height) {
        this.width = width;
        this.height = height;
  }

    @Override
    public double calcArea() {
        return width * height;
  }

    @Override
    public double calcPerimeter() {
        return 2 * (width + height);
  }

    public double getHeight() { return height; }
    public double getWidth() { return width; }
}

// RectangleMain.java
class RectangleMain {
    public static void main(String[] args) {
        IShape r = new RectangleImp1(10., 20.);
        System.out.println(r.calcArea());
  }
}
```

Interface default example

default : interface에서 객체마다 고유하게 사용하는 인스턴스 메소드로 생각할 수 있다.

```
// IValue.java
interface IValue {
    default int getValue() { return 0; }
}

// ValueImpl1.java
class ValueImpl1 implements IValue {
    private String name = "ValueImpl1";

    ValueImpl1(String s) {
        name = s;
  }

    public String getName() {
        return name;
  }

    public void setName(String s) {
        name = s;
  }
}

// ValueImpl2.java
class ValueImpl2 implements IValue {
    private String name;

    ValueImpl1() {
        name = "ValueImpl2";
  }

    public String getName() {
        return name;
  }

    public void setName(String s) {
        name = s;
  }

    public int getValue() {
        return 1; // default method overriding
  }
}
```