태그가 달린 클래스보다는 클래스보다는 클래스 계층구조를 활용하라

최 혁

```
class Figure {
    enum Shape { RECTANGLE, CIRCLE };
    fianl Shape shape;
    double lengh;
    double width;
    double radius;
    Figure(double radius) {
        shape = Shape.CIRCLE;
        this.radius = radius;
    }
    Figure(double length, double width) {
        shape = Shape.RECTANLE;
        this.length = length; this.width = width;
    }
    double area() {
        switch(shape) {
            case RECTANGLE: return length*width;
            case CIRCLE: rethrn Math.PI*(radius*radius)
        }
    }
}
```

태그가 달린 클래스의 문제점

- 여러 구현이 한 클래스에 혼합되어 있다.
- 메모리도 많이 사용한다(쓸데없는 필드들 생성 - 응집도가 낮아짐)
- final 필드로 만들면 필요없는 필드까지 초기화해야 한다.
- 다른 의미의 태그를 추가할 때마다 코드 수정이 발생한다.(switch문)
- 인스턴스의 타입만으로 현재 나타내는 의미를 알 길이 없다.

클래스 계층 구조를 활용하여 바꾸어보자!

1. 공통된 구조를 추상 클래스에 정의하고, 태그 값에 따라 달라지는 동작을 추상 메서드로 선언한다.
2. 태그 값에 상관없이 동작이 일정한 메서드들을 루트 클래스에 일반 메서드로 추가한다.
3. 모든 하위 클래스에서 공통으로 사용하는 필드를 루트 클래스로 올린다.
4. 루트 클래스를 확장한 구체 클래스를 의미별로 하나씩 정의한다.

```
abstract class Figure {
    abstract double area();
}
class Circle extends Figure {
    final double radius;
    Circle(double radius) { this.radius = radius; }
    @Override double area() { return Math.PI * (radius * radius); }
}
class Rectangle extends Figure {
    final double length;
    final double width;
    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }
    @Override double area() { return length * width; }
}
```

바뀐점

- 쓸데없는 코드들이 모두 사라져 간결하고 명확해졌다.
- 각 클래스의 생성자가 모든 필드를 남김없이 초기화하고 추상 메서드를 모두 구현했는지 컴파일러가 확인해준다.
- 다형성을 활용할 수 있다.

클래스 계층 구조의 확장성

```
class Square extends Rectangle {
    Square(double side) {
        super(side, side);
    }
}
```

직사각형을 확장하여 정사각형을 구현하면 정사각형이 직사각형의 특별한 형태라는 우리의 직관에 부합하게 되어 가독성이 올라간다.

느낀점

결론은 응집도 높은 클래스를 작성하기 위한 구체적인 방법을 제시

(태그가 달린 클래스로 코드를 짜는 사람이 있을까요..?)