

Item 23

태그 달린 클래스보다는 클래스 계층구조를 활용하라

목차

1

태그 달린 클래스

2

태그 달린 클래스 단점

3

클래스 계층 구조

4

태그 달린 클래스->클래스 계층 구조

5

클래스 계층 구조 장점

목차

1

태그 달린 클래스

2

태그 달린 클래스 단점

3

클래스 계층 구조

4

태그 달린 클래스->클래스 계층 구조

5

클래스 계층 구조 장점

태그 달린 클래스

두 가지 이상의 의미 표현 가능 클래스

여러 의미 중 현재 표현하는 의미 **태그 필드로 알려줌**



태그 달린 클래스

1. 필드

// 코드 23-1 태그 달린 클래스 - 클래스 계층구조보다 훨씬 나쁘다! (142-143쪽)

```
class Figure {  
    enum Shape { RECTANGLE, CIRCLE };
```

```
    // 태그 필드 - 현재 모양을 나타낸다.  
    final Shape shape;
```

// 다음 필드들은 모양이 사각형(RECTANGLE)일 때만 쓰인다.

```
    double length;  
    double width;
```

// 다음 필드는 모양이 원(CIRCLE)일 때만 쓰인다.

```
    double radius;
```

원, 사각형 표현 클래스
shape 필드가 태그 필드

2. 메서드

// 원용 생성자

```
Figure(double radius) {  
    shape = Shape.CIRCLE;  
    this.radius = radius;  
}
```

// 사각형용 생성자

```
Figure(double length, double width) {  
    shape = Shape.RECTANGLE;  
    this.length = length;  
    this.width = width;  
}
```

```
double area() {  
    switch(shape) {  
        case RECTANGLE:  
            return length * width;  
        case CIRCLE:  
            return Math.PI * (radius * radius);  
        default:  
            throw new AssertionError(shape);  
    }  
}
```

목차

1

태그 달린 클래스

2

태그 달린 클래스 단점

3

클래스 계층 구조

4

태그 달린 클래스->클래스 계층 구조

5

클래스 계층 구조 장점

태그 달린 클래스 단점

1. 태그 필드, switch문 등 **쓸데 없는 코드 많음**
2. 여러 구현이 한 클래스에 혼합되어 **가독성 나쁨**
3. 다른 의미 위한 코드 함께 있어 **메모리 많이 사용**



태그 달린 클래스 단점

1. 필드

// 코드 23-1 태그 달린 클래스 - 클래스 계층구조보다 훨씬 나쁘다! (142-143쪽)

```
class Figure {  
    enum Shape { RECTANGLE, CIRCLE };
```

```
    // 태그 필드 - 현재 모양을 나타낸다.  
    final Shape shape;
```

// 다음 필드들은 모양이 사각형(RECTANGLE)일 때만 쓰인다.

```
    double length;  
    double width;
```

// 다음 필드는 모양이 원(CIRCLE)일 때만 쓰인다.

```
    double radius;
```

**쓸데 없는 코드 많아
가독성 나쁘고 메모리 낭비**

2. 메서드

// 원용 생성자

```
Figure(double radius) {  
    shape = Shape.CIRCLE;  
    this.radius = radius;  
}
```

// 사각형용 생성자

```
Figure(double length, double width) {  
    shape = Shape.RECTANGLE;  
    this.length = length;  
    this.width = width;  
}
```

```
double area() {
```

```
    switch(shape) {
```

```
        case RECTANGLE:
```

```
            return length * width;
```

```
        case CIRCLE:
```

```
            return Math.PI * (radius * radius);
```

```
        default:
```

```
            throw new AssertionError(shape);
```

```
    }
```

```
}
```

```
}
```


태그 달린 클래스 단점

4. **쓰이지 않는 필드까지** 생성자에서 초기화 해야 함

5. 생성자에서 엉뚱한 필드 초기화 해도 **런타임에 문제 드러남**

태그 달린 클래스 단점

필드 **final**이라면 쓰이지 않는
필드까지 생성자에서 초기화

엉뚱한 필드 초기화 해도
런타임에 문제 드러남

```
public class Figure {
    enum Shape {RECTANGLE, CIRCLE}

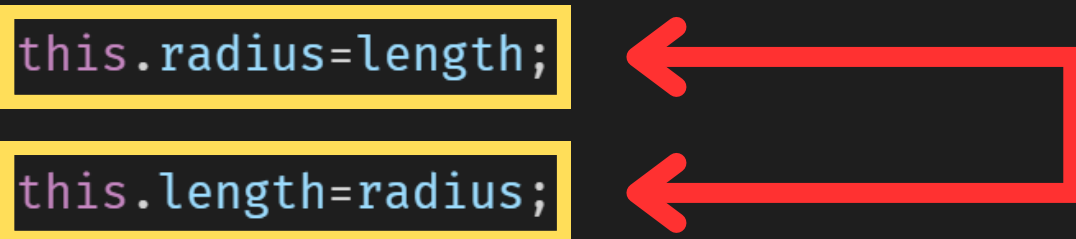
    private final Shape shape;
    private final double length;
    private final double width;
    private final double radius;

    // CIRCLE 전용 생성자
    public Figure(double radius, double length, double width){
        shape=Shape.CIRCLE;

        this.radius=length;
        this.length=radius;
        this.width=width;
    }

    // RECTANGLE 전용 생성자
    public Figure(double radius, double length, double width){
        ...
    }

    ...
}
```



태그 달린 클래스 단점

6. 다른 의미 추가하려면 **코드 수정 필요**

7. 인스턴스 타입만으로는 **현재 나타내는 의미 알 수 없음**

태그 달린 클래스 단점

다른 의미 추가하려면
코드 수정 필요

```
public class Figure {  
    enum Shape {RECTANGLE, CIRCLE, TRIANGLE}  
    ...  
  
    private double area() {  
        switch (shape) {  
            case RECTANGLE:  
                return length + width;  
            case CIRCLE:  
                return Math.PI * (radius * radius);  
            case TRIANGLE:  
                return .....;  
            default:  
                throw new AssertionError(shape);  
        }  
    }  
}
```

태그 달린 클래스 단점

태그 달린 클래스는 **장황**하고,
오류 내기 쉽고, **비효율적**!!



목차

1

태그 달린 클래스

2

태그 달린 클래스 단점

3

클래스 계층 구조

4

태그 달린 클래스->클래스 계층 구조

5

클래스 계층 구조 장점

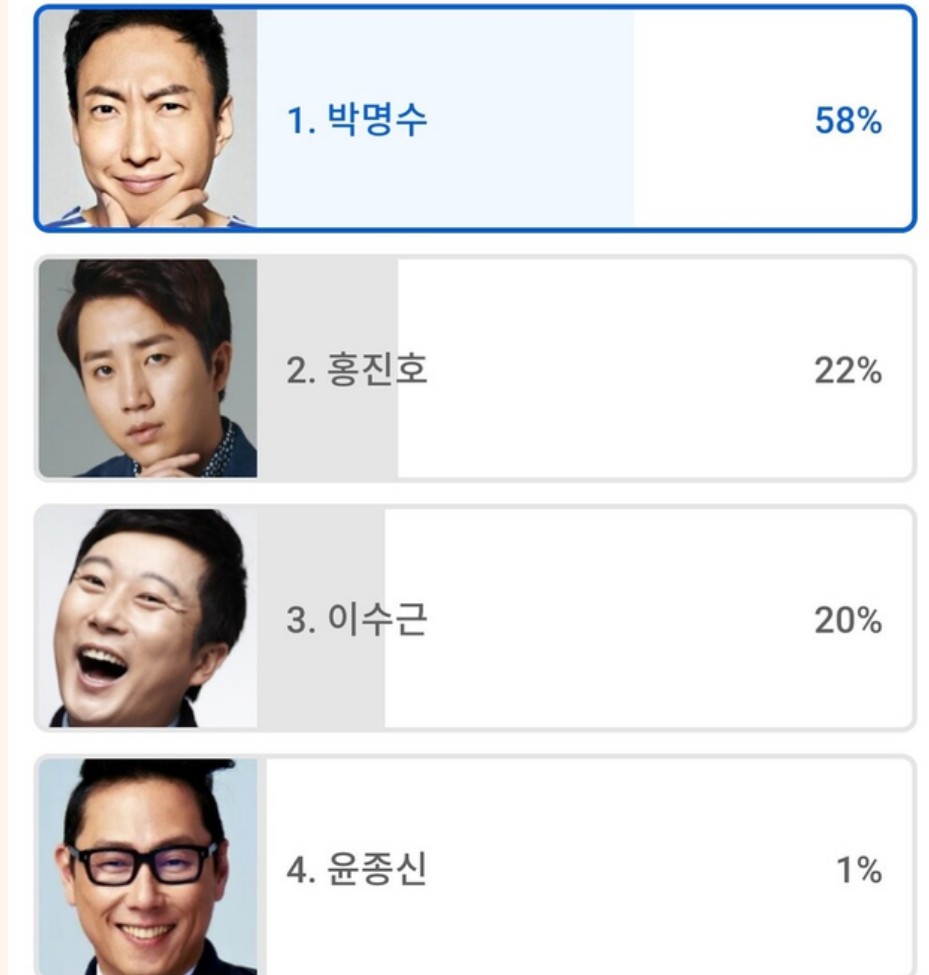
클래스 계층 구조

여러 의미 표현 어떻게 해야 하는가??



클래스 계층 구조 활용하는 서브 타이핑

2인자 원탑은?



25만명 투표

1.3천

688

클래스 계층 구조

서브 타이핑

상속의 용도

1. 타입 계층을 구현 -> 서브 타이핑
2. 코드 재사용 -> 서브 클래싱

클래스 계층 구조

서브 타이핑

상속은 **서브 타이핑 용도**로 사용

서브 클래싱 용도는 **상속보다 Composition** 사용

서브 타이핑 : 상속으로 클래스 계층 구조 표현하란 의미

클래스 계층 구조

태그 달린 클래스보다 클래스 계층 구조 활용한

상속으로 여러 의미를 표현하자!

목차

1

태그 달린 클래스

2

태그 달린 클래스 단점

3

클래스 계층 구조

4

태그 달린 클래스->클래스 계층 구조

5

클래스 계층 구조 장점

태그 달린 클래스 -> 클래스 계층 구조

1. 계층 구조의 **root** 될 추상 클래스 정의

2. 태그 값에 따라 동작 달라지는 메서드 **root** 클래스 추상 메서드로 선언

태그 달린 클래스 -> 클래스 계층 구조

1, 2번 단계

```
class Figure {  
    ...  
  
    double area() {  
        switch(shape) {  
            case RECTANGLE:  
                return length * width;  
            case CIRCLE:  
                return Math.PI * (radius * radius);  
            default:  
                throw new AssertionError(shape);  
        }  
    }  
}
```

```
abstract class Figure {  
    abstract double area();  
}
```

1. 계층 구조의 root 될 추상 클래스 정의
2. 태그 값에 따라 동작 달라지는 메서드 root 클래스 추상 메서드로 선언

태그 달린 클래스 -> 클래스 계층 구조

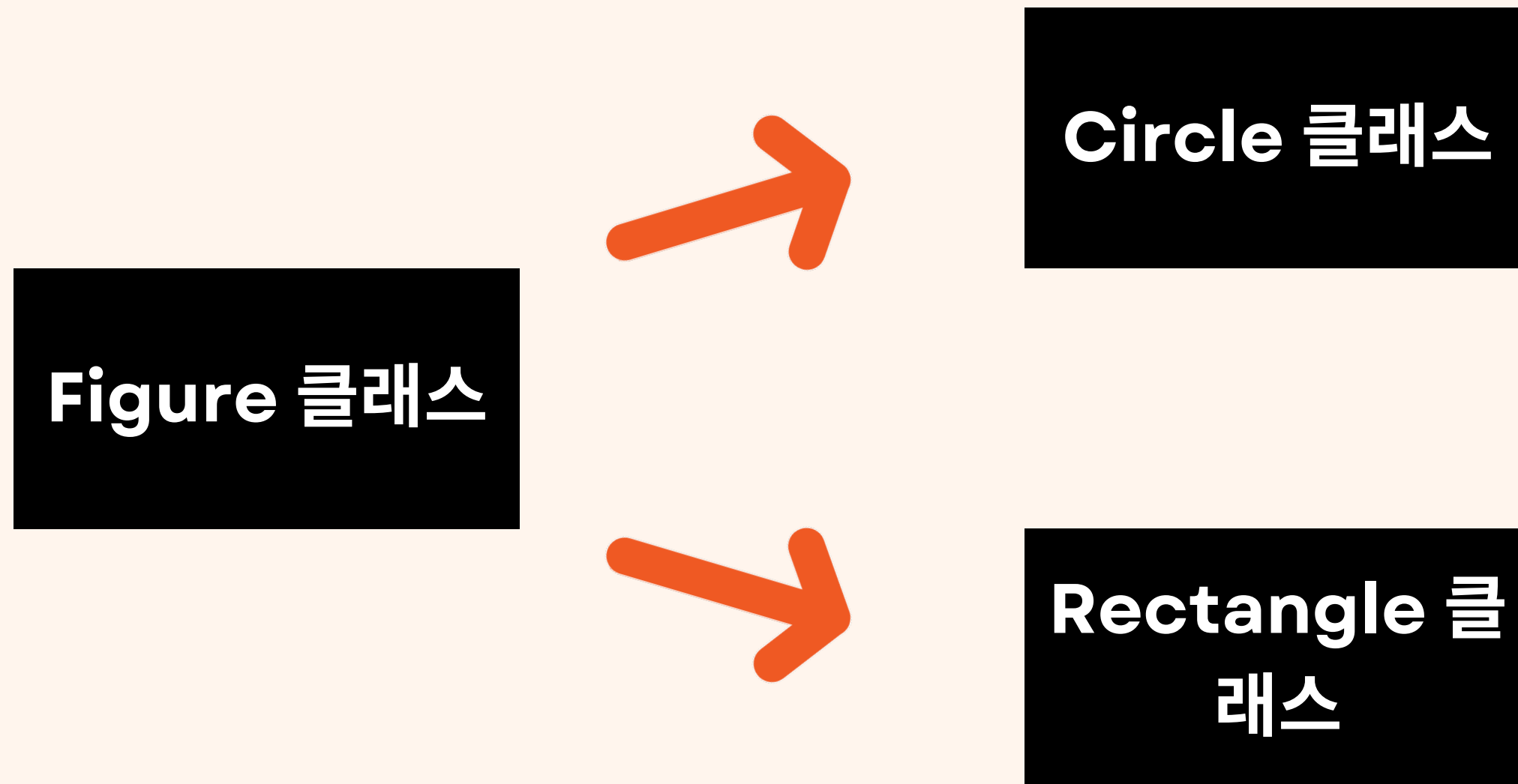
- 3. 태그 값에 상관 없이 **동작 일정한 메서드** root 클래스 **일반 메서드**로 추가
- 4. 모든 하위 클래스에서 **공통 사용하는 데이터 필드들** root 클래스에 추가

태그 달린 클래스 -> 클래스 계층 구조

5. root 클래스 확장한 **구체 클래스 의미별로** 하나씩 정의

태그 달린 클래스 -> 클래스 계층 구조

5번 단계



5. root 클래스 확장한 구체 클래스 의미별로 하나씩 정의

태그 달린 클래스 -> 클래스 계층 구조

6. 각 하위 클래스에 각자의 의미 해당하는 데이터 필드 추가

7. root 클래스가 정의한 추상 메서드 각자의 의미에 맞게 구현

태그 달린 클래스 -> 클래스 계층 구조

6, 7번 단계

```
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;  
    }  
}
```

6. 각 하위 클래스에 각자의 의미 해당하는 데이터 필드 추가
7. root 클래스가 정의한 추상 메서드 각자의 의미에 맞게 구현

목차

1

태그 달린 클래스

2

태그 달린 클래스 단점

3

클래스 계층 구조

4

태그 달린 클래스->클래스 계층 구조

5

클래스 계층 구조 장점

클래스 계층 구조 장점

코드 간결, 명확해진다.

쓸데 없는 코드 모두 사라진다.

살아남은 필드 모두 `final`이다.

독립적으로 계층 구조 확장 할 수 있다.

....

클래스 계층 구조 장점

클래스 계층 구조로 바꾸니

태그 달린 클래스의 단점 모두 사라진다!

태그 달린 클래스

클래스 계층 구조



결론!

태그 달린 클래스보다는
클래스 계층구조를 활용하라

