JAVA 객체지향 디자인 패턴

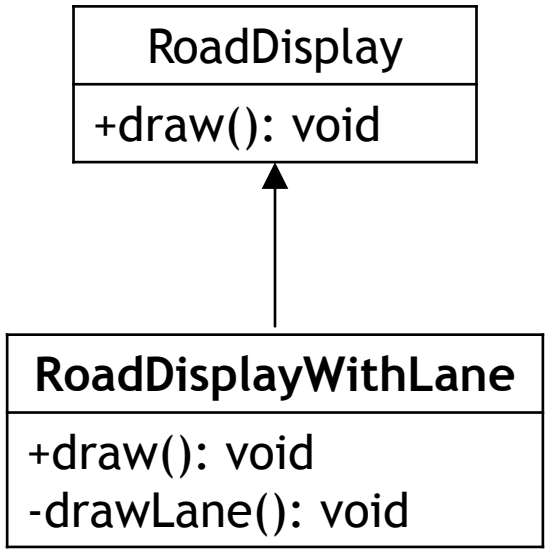
10장 데커레이터 패턴

목차

- 도로 표시 방법 조합하기
- 문제점
  - 또 다른 도로 표시 기능을 추가로 구현하는 경우
  - 여러 가지 추가 기능을 조합해야 하는 경우
- 해결책
- 데커레이터 패턴

도로 표시 방법 조합하기

- 내비게이션 SW 도로 표시 기능
  - 가장 기본적인 기능은 도로를 간단한 선으로 표시하는 것



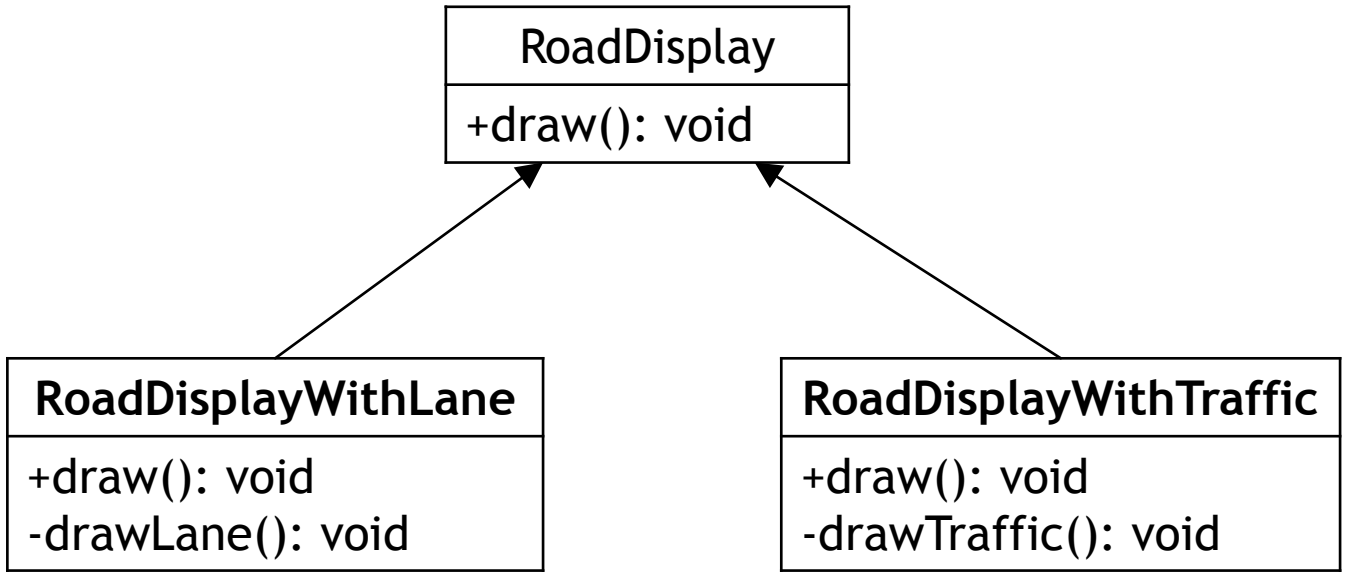

```
Public class RoadDisplay{
        public void draw(){
                System.out.println("기본 도로 표시");
        }
}
Public class RoadDisplayWithLane extends RoadDisplay{
        public void draw(){
                super.draw();
                drawLane();
        }
        private void drawLane(){
                System.out.println("차선 표시");
        }
}
```

문제점

- 또 다른 도로 표시 기능을 추가로 구현하고 싶다면 어떻게 해야 하는가?
  - ex) 기본 도로 표시에 교통량을 표시하고 싶다면?
- 여러 가지 추가 기능을 조합해 제공하고 싶다면 어떻게 해야 하는가?
  - ex) 기본 도로 표시에 차선 표시 기능과 교통량 표시 기능을 함께 제공하고 싶다면?

또 다른 도로 표시 기능을 추가로 구현하는 경우

- 도로에 교통량을 추가로 표시하는 클래스를 구현한다면?
  - RoadDisplayWithLane 클래스에서 정의한 것과 동일한 방식으로 기능 추가



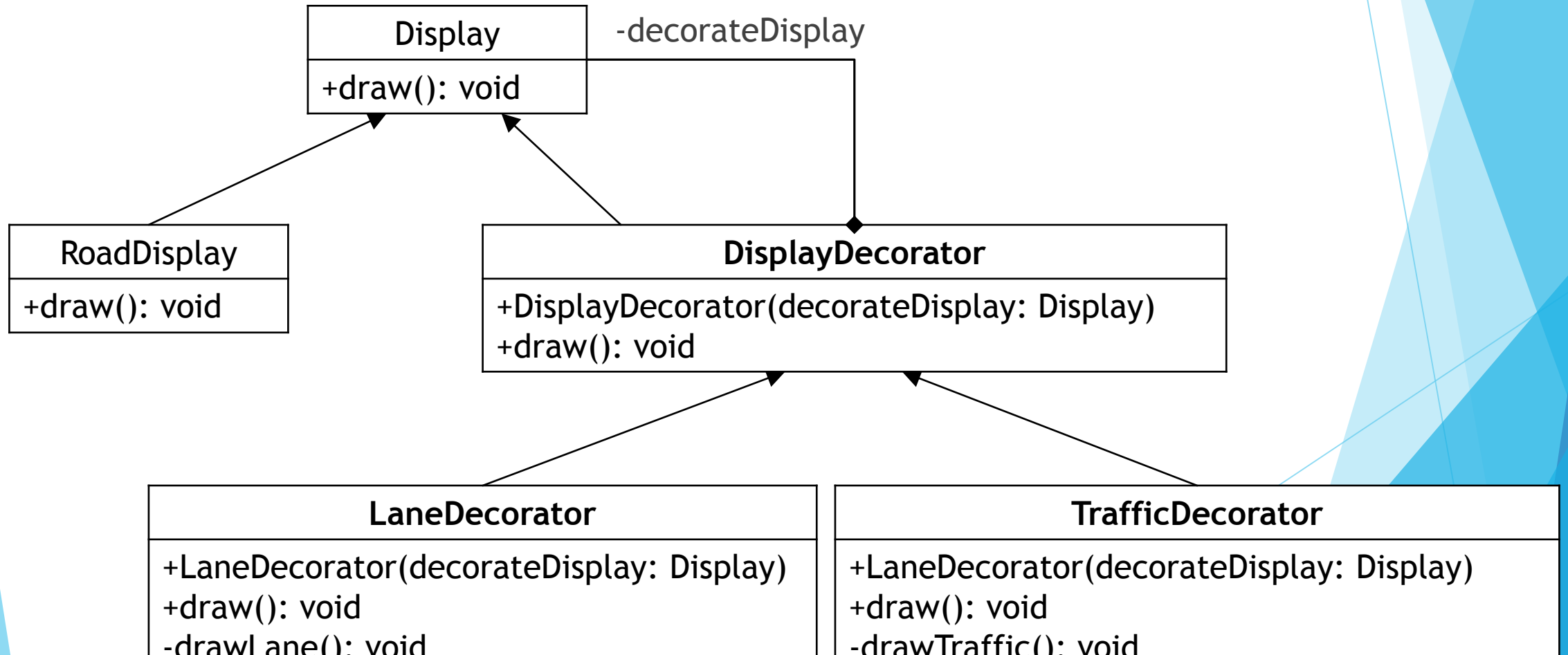
여러 가지 추가 기능을 조합해야 하는 경우

- RoadDisplay 클래스의 하위 클래스로 도로 표시 기능을 추가할 수도 있다.
  - 차선 표시, 교통량 표시, 교차로 표시를 추가로 제공해야 한다면 총 8가지 조합이 가능

경우	기본 기능 (도로 표시)	추가기능			클래스 이름
		차선 표시	교통량 표시	교차로 표시	
1	ok				RoadDisplay
2	ok	ok			RoadDisplayWithLane
3	ok		ok		RoadDisplayWithTraffic
4	ok			ok	RoadDisplayWithCrossing
5	ok	ok	ok		RoadDisplayWithLaneTraffic
6	ok	ok		ok	RoadDisplayWithLaneCrossing
7	ok		ok	ok	RoadDisplayWithTrafficCrossing
8	ok	ok	ok	ok	RoadDisplayWithLaneTrafficCrossing

해결책

- 각 추가 기능별로 개별적인 클래스를 설계하고 기능을 조합할 때 각 클래스의 객체 조합을 이용



데커레이션 패턴

- 기본 기능에 추가할 수 있는 기능의 종류가 많은 경우에 각 추가 기능을 Decorator 클래스로 정의한 후 필요한 Decorator 객체를 조합함으로써 추가 기능의 조합을 설계하는 방식
  - 자신의 addedBehavior 메서드를 먼저 호출한 후 Component의 operation 메서드를 호출하는 방식으로 구현 가능