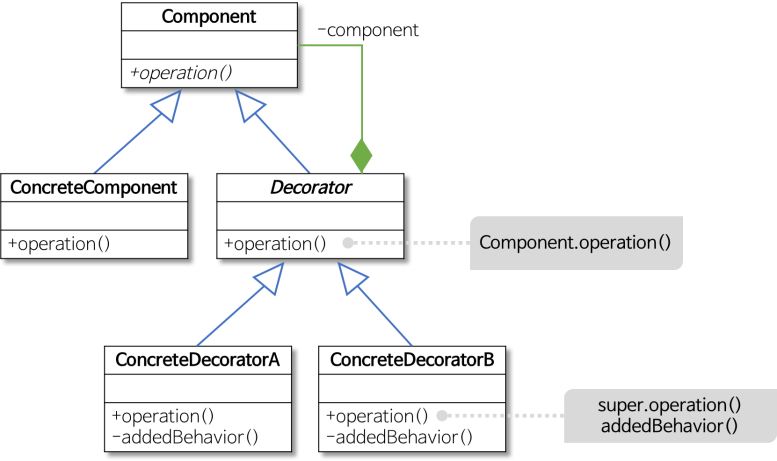
데코레이터 패턴

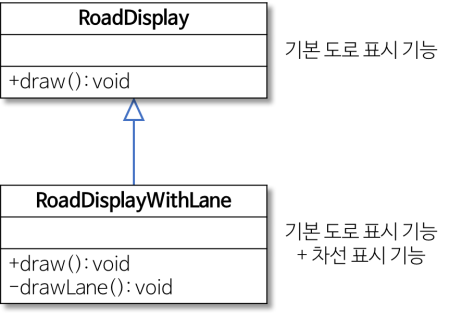
* 객체의 결합을 통해 기능을 동적으로 유연하게 확장할 수 있게 해주는 패턴
  + 즉 기본 기능에 추가할 수 있는 기능의 종류가 많은 경우에 각 추가 기능을 Decorator클래스로 정의한 후에 필요한 Decorator객체를 조합함으로써 추가 기능의 조합을 설계하는 방식
    - Ex)기본 도로 표시 기능에 차선 표시, 교통량 표시, 단속 카메라 표시의 4가지 추가 기능이 있을 때 추가 기능의 모든 조합은 15가지가 된다.
    - -> 데코레이터 패턴을 이용하여 필요 츄가 기능의 조합을 생성할 수 있다.
  + 구조 패턴의 하나



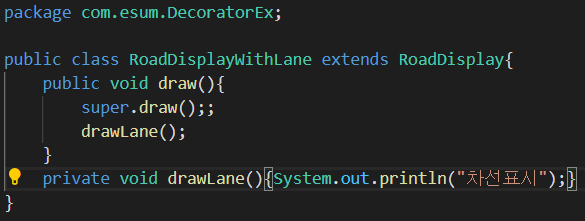
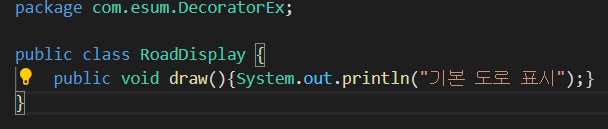
* + 기본 기능에 추가할 수 있는 많은 종류의 부가 기능에서 파생되는 다양한 조합을 동적으로 구현할 수 있는 패턴이다.
  + 역할이 수행하는 작업
    - Component
      * 기본 기능을 뜻하는 ConcreteComponent와 추가 기능을 뜻하는 Decorator의 공통 기능을 정의
      * 즉 클라이언트는 Component를 통해 실제 객체를 사용함
    - ConcreteComponent
      * 기본 기능을 구현하는 클래스
    - Decorator
      * 많은 수가 존재하는 구체적인 Decorator의 공통 기능을 제공
    - ConcreteDecoratorA, ConcreteDecoratorB
      * Decorator의 하위 클래스로 기본 기능에 추가하는 개별적인 기능을 뜻함
      * ConcreteDecorator클래스는 ConcreteComponent객체에 대한 참조가 필요한데, 이는 Decorator클래스에서 Component클래스로의 합성관계를 통해 표현된다.
  + 참조
    - 구조패턴
      * 클래스나 객체를 조합해 더 큰 구조를 만드는 패턴
      * 예를들어 서로 다른 인터페이스를 지닌 2개의 객체를 묶어 단일 인터페이스를 제공하거나 객체들을 묶어 새로운 기능을 제공하는 패턴이다.
    - 합성관계
      * 생성자에서 필드에 대한 객체를 생성하는 경우
      * 전체 객체의 라이프타임과 부분 객체의 라이프타임은 의존적이다.
      * 즉, 전체 객체가 없어지면 부분 객체도 없어진다.

예시

도로 표시 방법 조합하기



* 네비게이션 sw에서 도로를 표시하는 기능
  + 도로를 간단한 선으로 표시하는 기능(기본 기능)
  + 네비게이션 sw에 따라 도로의 차선을 표시하는 기능(추가기능)

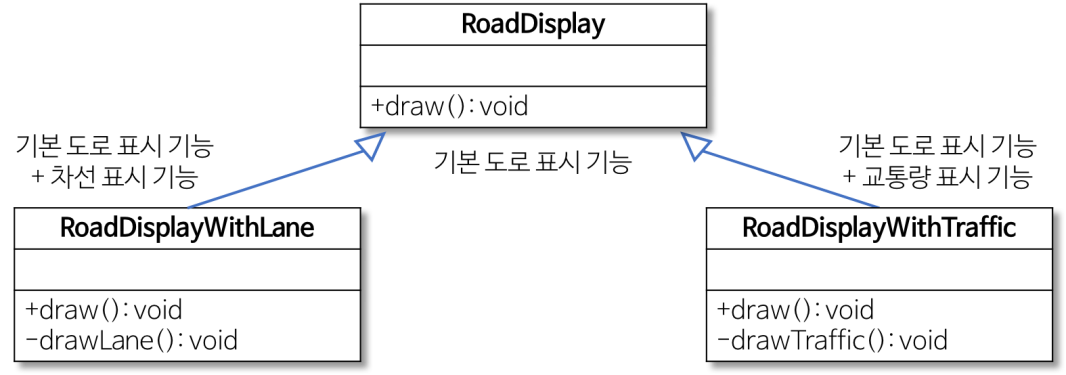


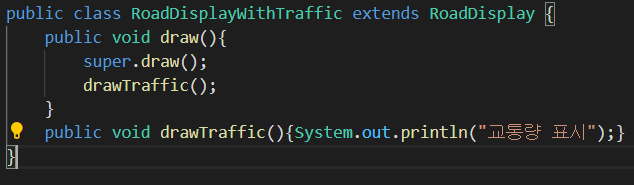


* + RoadDisplay 클래스에는 기본 도로 표시 기능을 실행하기 위한 draw메소드를 구현한다
  + RoadDisplayWithLane 클래스에는 자신 표시 기능을 추가하기 위해 상속받은 draw메서드를 오버라이드 한다.
    - 기본 도로 표시 기능 : 상위 클래스(RoadDisplay)의 draw 메서드 호출
    - 차선 표시 기능 : 자신의 drawLane메서드 호출

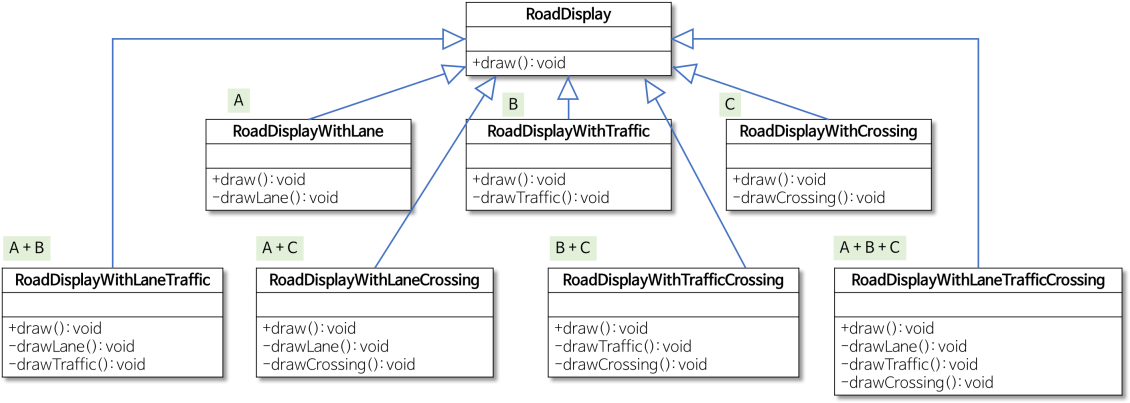
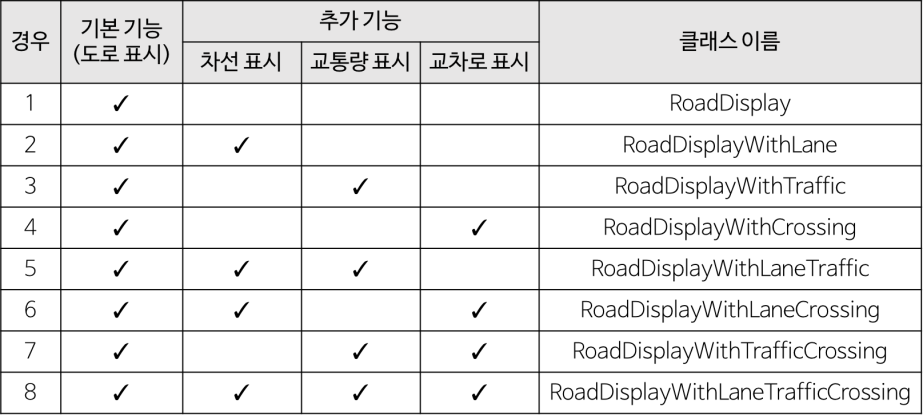
문제점

1. 또 다른 도로 표시 기능을 추가로 구현하는 경우
   1. 기본 도로 표시에 교통량을 표시하고 싶다면?





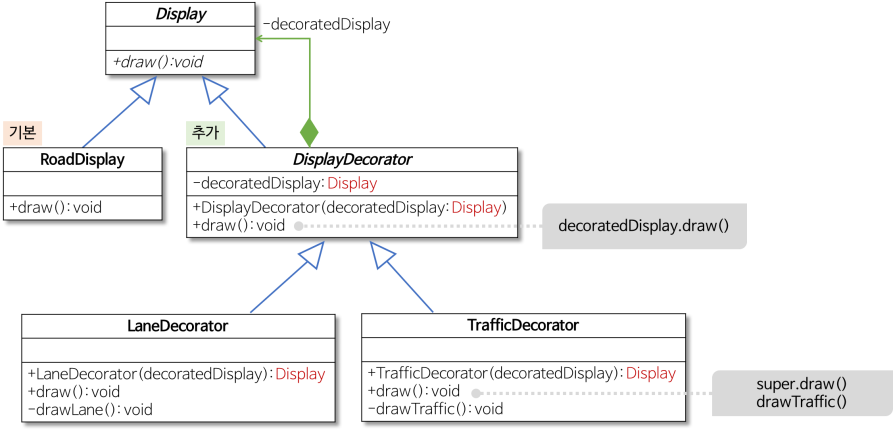
1. 여러가지 추가 가능을 조합해야 하는 경우
   1. 기본 도로 표시에 차선 표시 기능과 교통량 표시 기능을 함께 제공하고 싶다면



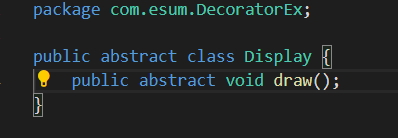
* + - * 위와 같이 상속을 통해 조합의 각 경우를 설계한다면, 각 조합별로 하위 클래스(7)개를 구현해야 한다.
      * 즉, 다양한 기능의 조합을 고려해야 하는 경우 상속을 통한 기능의 확장은 각 기능별로 클래스를 추가해야 한다는 단점이 있다.

해결책

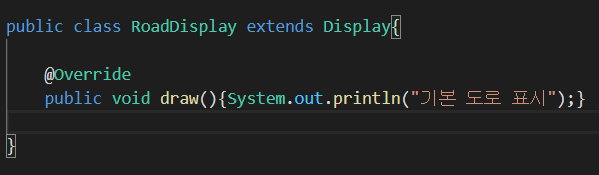
* 문제를 해결하기 위해서는 각 추가 기능별로 개별적인 클래스를 설계하고 기능을 조합하 때 각 클래스의 객체 조합을 이용하면 된다.



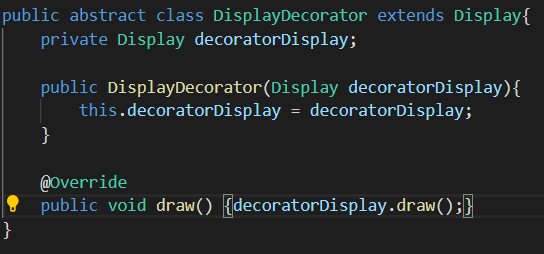
* + 도로를 표시하는 기본 기능만 필요한 경우 RoadDisplay 객체를 이용한다.
  + 차선을 표시하는 추가 기능도 필요한 경우 RoadDisplay 객체와 LaneDecorator객체를 이용한다.
    - LaneDecorator에서는 차선 표시 기능만 직접 제공한다.
    - 도로 표시 기능은 RoadDisplay클래스의 draw 메서드를 호출 : super.draw()
      * (DisplayDecorator 클래스에서 Display 클래스로의 합성(composition) 관계를 통해 RoadDisaply 객체에 대한 참조)
  + Display 클래스



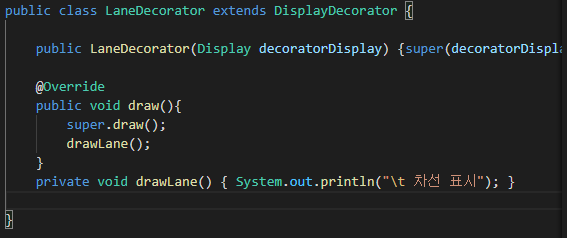
* + RoadDisplay클래스

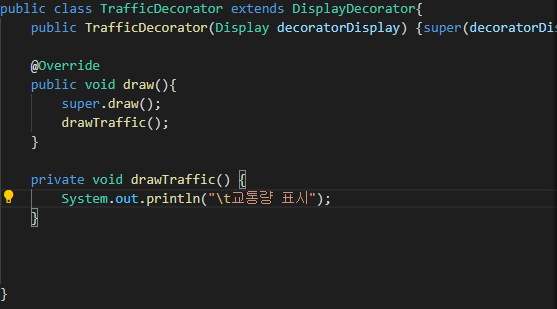


* + DisplayDecorator클래스



* + LaneDecorator, TrafficDecorator클래스





* + Client에서 사용

’

* + 각 road roadWithLane roadWithTraffic객체의 접근이 모두 display 클래스를 통해 이루어진다.
  + 즉 어떤 기능을 추가하느냐 관계없이 client 클래스는 동일한 display클래스만을 통해 일관성 있는 방식으로 도로 정보를 표시할 수 있다.
  + 이렇게 decorator 패턴을 이용하면 추가 기능 조합별로 별도의 클래스를 구현하는 대신 각 추가 기능에 해당하는 클래스의 객체를 조합해 추가 기능의 조합을 구현할 수 있게 된다.
    - 이 설계는 추가 기능의 수가 많을수록 효과가 크다.

