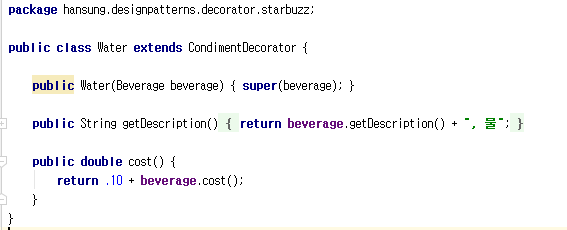
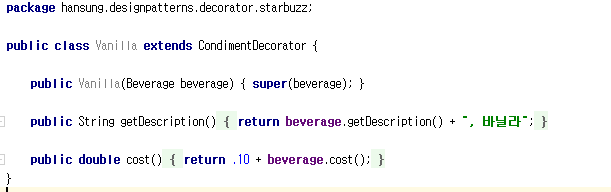
**소프트웨어 설계 패턴 (실습 3)**

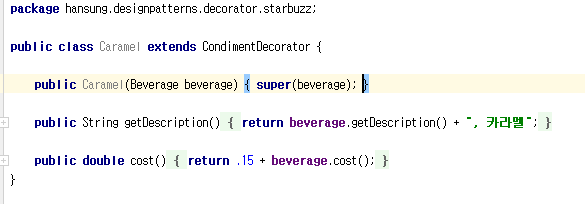
**1494067 이찬희**

이번 실습에서는 예제코드로 주어진starbuzz를 바탕으로 진행합니다.

1. 베이스 음료(에스프레소, 다크 로스트, 하우스 블렌드, 디카페인)의 첨가물로서 물, 바닐라, 카라멜 시럽을 추가합니다. 이를 위해서 코드의 어떠한 부분을 수정 및 확장해야 하는지, 수정 및 확장이 얼마나 쉬운지 어려운지를 토의합니다.







CondimentDecorator 클라스를 상속받는 Water, Vanilla, Caramel 클래스를 생성한다. Beverage 타입의 변수를 사용하고 getDiscription() 메소드로 값을 넘겨준다. Cost() 메소드로 가격을 넘겨준다. 총 3개를 추가한다.

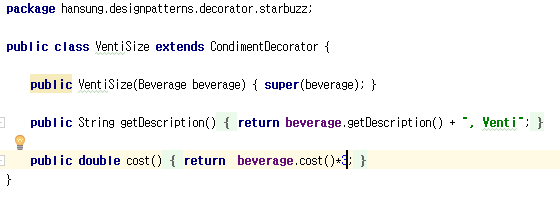
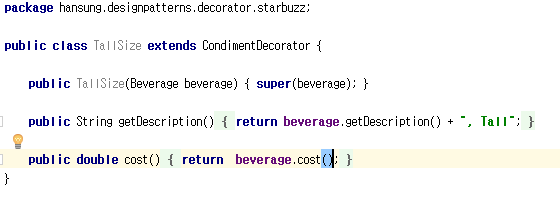
확장 같은 경우는 추가되는 재료를 클래스화 시키면 되기 때문에 간단하고,

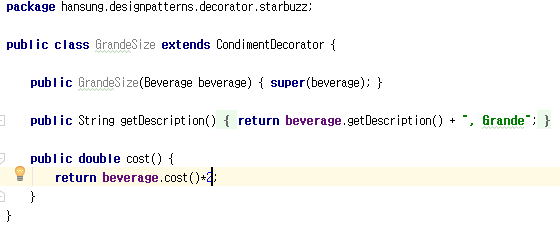
가격 수정 시에도 각 클래스 안의 값을 수정하면 된다.

1. 음료 주문시에 베이스 음료를 먼저 선택하고, 사이즈 (톨, 그랑데, 벤티)를 선택한 후, 기타 첨가물을 선택한다고 가정합니다. 예를 들면, “에스프레소 -> 톨 사이즈 -> 물” 을 순서대로 선택하거나, “다크 로스트 -> 벤티 사이즈 -> 더블 모카 -> 위핑” 을 선택할 수 있습니다.

사이즈의 선택에 따라 음료의 가격이 다르게 책정됩니다. 톨 사이즈는 베이스 음료의 1샷, 그랑데 사이즈는 2샷, 벤티 사이즈는 3샷 가격으로 책정된다고 가정합니다. 단, 사이즈에 따른 첨가물의 가격 차이는 없다고 가정합니다.

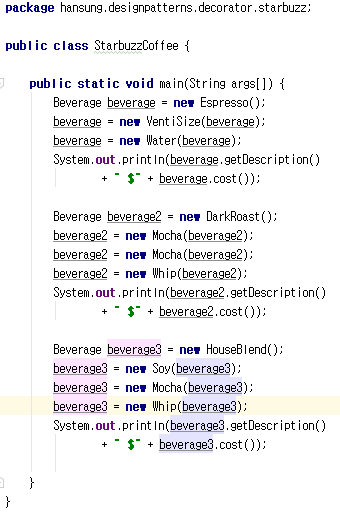
사이즈 선택 기능을 추가하기 위해서, 코드를 어떻게 변경하였는, 수정 및 확장이 얼마나 쉬운지 어려운지를 토의합니다.





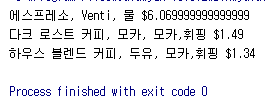
CondimentDecorator 클래스를 상속받는 TallSize, VentiSize, GrandeSize 클래스를 만들어 준다. 가격 측정 시에는 Tallsize는 그대로 return, GrandeSize는 가격을 2배 하여 return, VentiSize는 가격을 3배 하여 return하는 방식으로 구성하여 코드의 추가나 수정이 어렵지 않다.

StarBuzzCoffee.class



Beverage타입의 beverage객체에 venti사이즈 가격 넣음

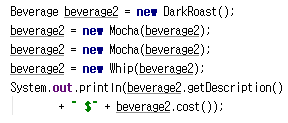
결과출력은



다음과 같이 venti 를 출력하고 알맞은 값을 리턴함

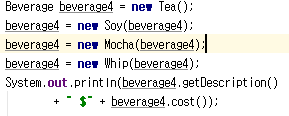
1. 이번에는 베이스 음료의 종류로 티(Tea)를 추가하고자 합니다. 티의 경우 일부 첨가물 (예, 모카, 휘핑)를 선택할 수 없다고 한다면, 이것이 현재 디자인에서 수용가능한지 그렇지 않은지에 대해서 토의합니다.

StarBuzzCoffee의 일부 코드를 보게 되면



DarkRoast를 beverage2에 넣고 모카를 다시 beverage2에 넣고.. 를 반복하는 형태로 되어있다. 해당 가격을 위임해 주는 식으로 되어 있다. Whip이 Mocha를 감싸고 있고, Mocha는 또 Mocha를 감싸는 식으로 되어 있는데 감싸고 있다고 해서 바로 앞에 무엇이 있는지 정확히 인식이 되어 있는 코드가 아니다.

예를 들어



이러한 코드를 StarBuzzCoffee.class에 코딩했다고 하고

Tea를 선택할 시 MoCha를 추가하면 오류가 발생하게 구현한다고 한다. 그렇다면 Mocha Class에서는 cost()메소드를 0으로 리턴시킨 후 getDiscription()메소드는 예를들어 “오류”라는 텍스트를 출력시키게 만들어야 한다.(베이스 음료가 구분된다고 가정하면) 이 부분은 Mocha Class에서 구현할 수 있지만 beverage4라는 객체에 Tea가 들어갔는지, 정확히 어떠한 베이스 음료가 들어갔는지 구분할 방법이 이 패턴에서는 존재하지 않는다.