객체지향개발론및실습

Laboratory 3. Decorator 패턴: StarBuzz 커피숍 응용

1. 목적

- 객체에 동적으로 새로운 책임(기능, 상태)를 추가할 수 있도록 해주는 decorator 패턴에 대해 숙제를 포함하여 다음 3가지를 실습해본다.
  - Decorator 패턴에서 추가된 책임을 동적으로 제거하는 방법을 연구해본다.
  - Decorator 패턴을 사용하지 않고 첨가물을 리스트에 유지하는 방법을 구현하여 비교해본다.
  - Decorator 패턴을 이용하여 장식한 객체를 서로 비교하는 방법을 연구해본다.

2. 개요

- 커피를 판매할 때 고객들은 다양한 첨가물(스팀우유, 두유, 모카, 크림 등)을 추가할 수 있다. 이에 대한 가격을 유연하게 계산하기 위해 이들을 장식자 패턴을 통해 모델링하여 구현할 수 있다.
- 이미 StarBuzz 커피숍 응용은 이론 시간에 설명 및 실습하였다.
- 실습 1에서는 추가된 책임을 제거하는 방법을 실습해보고, 실습 2에서는 장식자 패턴을 사용하지 않고 리스트를 이용하여 구현한 다음 decorator 패턴을 사용한 코드와 비교해본다.

3. 실습 1

- 수업시간에 완성한 첫 번째 StarBuzz 커피숍 응용을 복사하여 새 프로젝트를 만들고 수정하시오.
- 맨 마직막으로 추가한 첨가물을 제거하는 방법을 생각하여 보자.
- 테스트 프로그램을 통해 알 수 있듯이 커피 음료는 항상 Beverage 타입으로 유지된다. 따라서 removeCondiment라는 메소드는 CondimentDecorator 타입에서만 필요한 것이지만 Beverage 타입에 추가되어야 사용이 용이하다.
- 예)

```
Beverage beverage = new DarkRoast();
beverage = new Mocha(beverage);
beverage = new Mocha(beverage);
beverage = new Whip(beverage);
System.out.printf("%s: %,d원%n", beverage.getDescription(), beverage.cost());
beverage = beverage.removeCondiment();
System.out.printf("%s: %,d원%n", beverage.getDescription(), beverage.cost());
```

- 이 실습에서는 Beverage 클래스와 각 Concrete Decorator 클래스에 다음 메소드를 추가한다.
  - `public Beverage removeCondiment()`
    - 사후조건: 현재 클래스의 타입이 `Beverage`이면 `this`를 반환하고, `CondimentDecorator` 타입이면 장식한 `beverage`를 반환한다.
- Decorator를 만들면서 그것이 장식할 클래스를 수정해야 하는 것은 바람직하지 않다. 실제로 Beverage 클래스에 `removeCondiment` 메소드를 추가하지 않고 `CondimentDecorator` 타입에만 추가하여 사용할 수 있다. 이 경우 테스트 프로그램은 다음과 같이 변경이 필요하다.

```
Beverage beverage = new DarkRoast();
beverage = new Mocha(beverage);
beverage = new Mocha(beverage);
beverage = new Whip(beverage);
System.out.printf("%s: %,d원%n", beverage.getDescription(), beverage.cost());
if(beverage instanceof CondimentDecorator){
    beverage = ((CondimentDecorator)beverage).removeCondiment();
}
System.out.printf("%s: %,d원%n", beverage.getDescription(), beverage.cost());
```

- 이 처럼 구현하면 각 첨가물에 중복 코드가 존재하게 된다. 이를 제거하는 방법은?

4. 검토사항

- 실습 1에서 문제 제기한 decorator 패턴에서 장식된 것을 제거하는 기능을 제공하는 것이 논리적이지 않고 필요없는 기능이라고 주장되는 경우도 있다. 동적으로 추가된 장식이 제거된 어떤 객체가 필요하면 기존 객체를 이용하지 않고 새로 필요한 형태로 만들 수 있다.
- 특히, 제거 기능을 제공하기 위해 사용된 기법은 기술적으로는 가능한 기법이지만 기본 타입에 원래 필요없는 메소드가 추가되어야 하기 때문에 논리적이라고 보기도 어렵다.

5. 실습 2

- 커피숍의 가격 문제는 decorator 패턴을 사용하지 않고도 해결할 수 있다. 항상 모든 문제는 여러 가지 방법으로 해결할 수 있다. 좋은 개발자는 이 중에 가장 좋은 방법을 선택할 수 있어야 한다.
- 이 실습 역시 수업시간에 완성한 첫 번째 StarBuzz 커피숍 응용을 복사하여 새 프로젝트를 만들고 수정하시오.
- Beverage 클래스에 다음이 추가되어야 함
  - ArrayList를 이용하여 condiment들을 유지할 수 있는 멤버 `condiments`를 추가하시오.
  - `public void addCondiment(Condiment condiment)`
    - 사전조건: `condiment`가 **null**이 아니어야 함. 검사하지 않음.
    - 사후조건: `condiments`에 주어진 `condiment`를 추가한다.
- 기존 `getDescription` 메소드도 첨가물들이 출력될 수 있도록 수정되어야 함.
- `Condiment` **interface**를 새롭게 정의함. 이 **interface**에는 수업시간에 보여준 전략패턴 적용 예처럼 `getDescription`과 `cost` 메소드가 선언되어 있어야 함
- 기존 첨가물 클래스들은 `CondimentDecorator`를 상속받는 대신에 `Condiment` **interface**를 구현하도록 바꾸고, 유지하고 있는 `Beverage` 멤버와 생성자는 더 이상 필요가 없음. 수업시간에 전략패턴 적용 예와 동일함.

6. 실습 3

- 자바의 reflection 클래스들을 이용하여 장식자를 이용한 객체 생성 부분을 추상화할 수 있다. 우리가 사용하고자 하는 형태는 다음과 같다.

```
Beverage beverage = Beverage.getInstance(new HouseBlend(), "Mocha", "Whip","Mocha");
```

- 위 **static** 메소드를 구현하시오. 이를 위해 Class 클래스의 forName 메소드와 Constructor 클래스의 getConstructor 메소드, newInstance 메소드의 사용이 필요함

7. 숙제

- 장식된 객체의 등가 여부를 비교하는 기능을 추가하여 보자. 자바에서는 equals 메소드를 통해 두 객체가 같은지 여부를 확인한다.
- equals 메소드를 `Beverage` 클래스, 모든 concrete 클래스, 모든 장식 클래스에 추가하여야 한다. 이 점이 번거롭다. 하지만 커피 예제의 경우 모든 concrete 클래스에 구현할 필요가 없다. concrete 클래스의 경우 `Beverage` 클래스에만 정의하여도 우리가 원하는 기능을 얻을 수 있다. 장식 클래스도 마찬가지이다. 각 장식 클래스에 구현하지 않고 `CondimentDecorator` 클래스에만 구현하여도 원하는 기능을 얻을 수 있다. 하지만 이를 위해서는 각 장식 클래스에 유지하였던 beverage를 멤버를 CondimentDecorator에 다음과 같이 유지하여야 한다.

```
public abstract class CondimentDecorator extends Beverage {
    private Beverage beverage;
    protected void setDecoratee(Beverage beverage){
        this.beverage = beverage;
    }
    protected Beverage getBeverage(){
        return beverage;
    }
    public abstract String getDescription();
}
```

- 이 예제에서 각 객체를 비교할 때 비교에 사용할 데이터가 없다. 가격정보가 멤버로 유지되는 것이 아니라 **return** 문 코드에 유지되고 있다. 이 경우 클래스(getClass())나 클래스 이름(getClass().getName())을 이용하여 비교하는 것으로 충분한 비교가 된다.
- 주의사항. 장식된 순서가 다르더라도 첨가물이 같고, 커피가 같은 종류이면 **true**를 얻을 수 있어야 한다.