객프 9주차 데코레이션 패턴

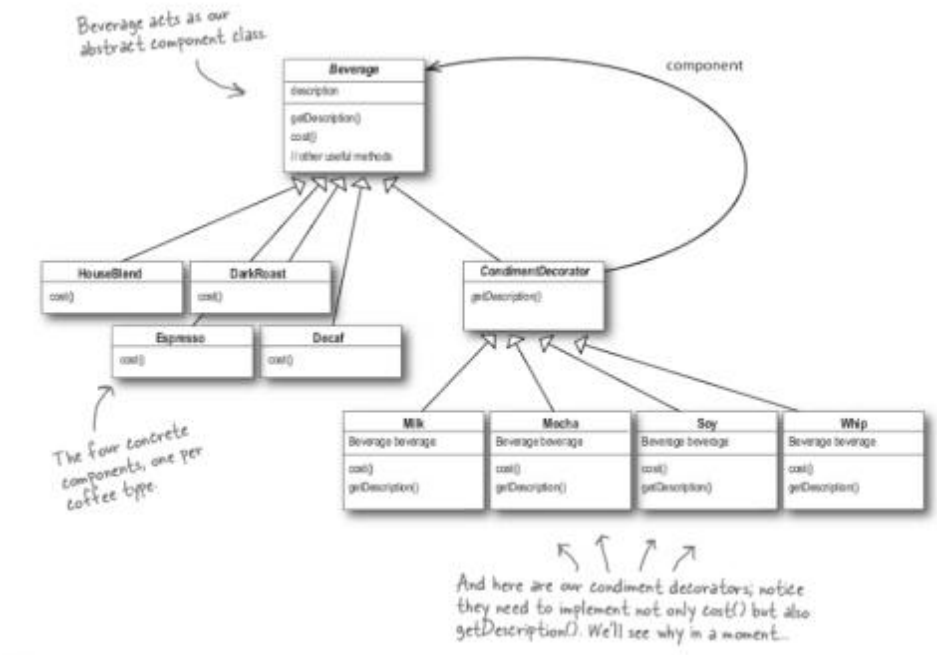
201022439 미디어학부 민준홍

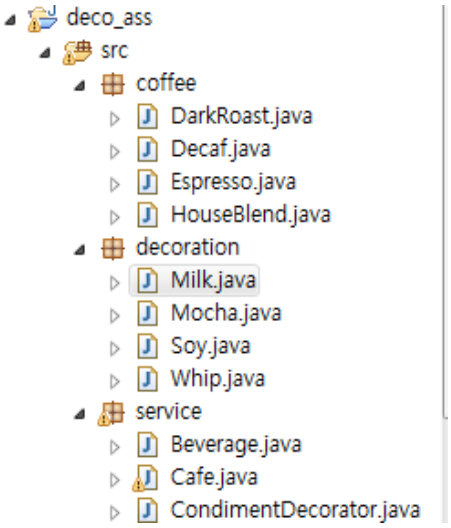
1. 과제 개요

- 개발 동기 및 필요성

Design Pattern을 통해 interface와 abstract class / constructor를 이해한다

2. 시스템 설계






```
반갑습니다. 카페에 오신 것을 환영합니다.
------------------------
커피 종류     |      가격
------------------------
Decaf                1500
HouseBlend       2000
DarkRoast        2500
Espresso           3000
CafeMocha        2700
CafeLatte         3500
------------------------
커피 토핑     |      가격
------------------------
Milk                      500
Mocha                   700
Soy                      1000
Whip                    1500
------------------------
```

3. 시스템 구현

JAVA, Eclipse Juno 사용

----------실행화면----------

1. 초기 메뉴

```
반갑습니다. 카페에 오신 것을 환영합니다.
------------------------
커피 종류     |      가격
------------------------
Decaf                1500
HouseBlend       2000
DarkRoast        2500
Espresso           3000
CafeMocha        2700
CafeLatte         3500
------------------------
커피 토핑     |      가격
------------------------
Milk                    500
Mocha                 700
Soy                   1000
Whip                 1500
------------------------
```

2. Decaf + Milk + Whip 입력시

```
----------------------
커피 종류를 선택하세요 : Decaf
커피 토핑을 입력하세요.
주문을 마치시면 'Exit'를 입력하세요.
커피 토핑을 제거하실 때는 앞에 Remove를 입력하시면 됩니다.(예)RemoveMilk)
중복 토핑이 있을 시 하나만 삭제됩니다.(Milk를 2회 선택 후 RemoveMilk를 1회 경우 Milk 하나는 남아있게 됩니다.)
Milk
Whip
Exit
주문하신 커피는 Decaf+Milk+Whip 입니다.
총 금액은 3500원입니다.
```

3. Espresso + Milk 입력 (해당 메뉴가 CafeLatte로 출력이 되는지 확인)

```
-----------------------
커피 종류를 선택하세요 : Espresso
커피 토핑을 입력하세요.
주문을 마치시면 'Exit'를 입력하세요.
커피 토핑을 제거하실 때는 앞에 Remove를 입력하시면 됩니다.(예)RemoveMilk)
중복 토핑이 있을 시 하나만 삭제됩니다.(Milk를 2회 선택 후 RemoveMilk를 1회 경우 Milk 하나는 남아있게 됩니다.)
Milk
Exit
주문하신 커피는 CAFELATTE 입니다.
총 금액은 3500원입니다.
```

4. CafeLatte 입력 (라떼 입력시 라떼가 출력되는지 확인)

```
-----------------------
커피 종류를 선택하세요 : CafeLatte
커피 토핑을 입력하세요.
주문을 마치시면 'Exit'를 입력하세요.
커피 토핑을 제거하실 때는 앞에 Remove를 입력하시면 됩니다.(예)RemoveMilk)
중복 토핑이 있을 시 하나만 삭제됩니다.(Milk를 2회 선택 후 RemoveMilk를 1회 경우 Milk 하나는 남아있게 됩니다.)
Exit
주문하신 커피는 CAFELATTE 입니다.
총 금액은 3500원입니다.
```

5. CafeLatte + RemoveMilk 입력 (라떼에서 Milk를 제외할 때 에스프레소가 뜨는지 확인)

```
-----------------------
커피 종류를 선택하세요 : CafeLatte
커피 토핑을 입력하세요.
주문을 마치시면 'Exit'를 입력하세요.
커피 토핑을 제거하실 때는 앞에 Remove를 입력하시면 됩니다.(예)RemoveMilk)
중복 토핑이 있을 시 하나만 삭제됩니다.(Milk를 2회 선택 후 RemoveMilk를 1회 경우 Milk 하나는 남아있게 됩니다.)
RemoveMilk
Exit
주문하신 커피는 Espresso 입니다.
총 금액은 3000원입니다.
```

6. CafeLatte + RemoveMilk + Whip + Mocha입력

(라떼에서 Milk를 제외하여 에스프레소가 된 상태에서 whip과 mocha가 입력되는지 확인)

```
-----------------------
커피 종류를 선택하세요 : CafeLatte
커피 토핑을 입력하세요.
주문을 마치시면 'Exit'를 입력하세요.
커피 토핑을 제거하실 때는 앞에 Remove를 입력하시면 됩니다.(예)RemoveMilk)
중복 토핑이 있을 시 하나만 삭제됩니다.(Milk를 2회 선택 후 RemoveMilk를 1회 경우 Milk 하나는 남아있게 됩니다.)
RemoveMilk
Whip
Mocha
Exit
주문하신 커피는 Espresso+Whip+Mocha 입니다.
총 금액은 5200원입니다.
```

-코드설명

클래스를 유지하기 위해서 메인함수에서 배열을 써서 구현했습니다.

먼저 입력되는 토핑들을 배열에 저장한후 배열에 있는값을 하나하나 불러오면서 토핑을 추가한다.

리무브를 할 경우 배열에 있는 값을 삭제하기 때문에 아직 음료(샷)에는 토핑이 추가되지않은 상태에서 배열값만 삭제가 된다.
마지막으로 입력이 끝나면 배열값을 불러와 음료를 조제한다.

이때 특수한 조합이 있는경우(espresso+milk) 리턴값에 조건문을 걸어서 조합된음료(CAFELATTE)가 출력이 된다.

<음료주문> main

```
        ArrayList<String> toppingList=new ArrayList<String>();
        //토핑 목록이 저장될 ArrayList
                do{
                        String coffeeType=s.next();//커피 종류 입력
                        if(coffeeType.equals("Decaf")){//Decaf
                                beverage=new Decaf();
                                typeFlag=false;
                        }else if(coffeeType.equals("HouseBlend")){//HouseBlend
                                beverage=new HouseBlend();
                                typeFlag=false;
                        }else if(coffeeType.equals("DarkRoast")){//DarkRoast
                                beverage=new DarkRoast();
                                typeFlag=false;
                        }else if(coffeeType.equals("Espresso")){//Espresso
                                beverage=new Espresso();
                                typeFlag=false;
```

```
                }else if(coffeeType.equals("CafeMocha")){//HouseBlend+Mocha
                        beverage=new HouseBlend();
                        toppingList.add("Mocha");//toppingList에 Mocha 추가
                        typeFlag=false;
                }else if(coffeeType.equals("CafeLatte")){//Espresso+Milk
                        beverage=new Espresso();
                        toppingList.add("Milk");//toppingList에 Milk 추가
                        typeFlag=false;
                }else{  typeFlag=true; }
```

<토핑의 추가/삭제 : 클래스 구조를 유지하기 위해 배열사용>

```
        do{
                coffeeTopping=s.next();//커피 토핑 주문 입력
                        if(coffeeTopping.equals("Exit"){//커피 주문을 마치면,
                        toppingFlag=false;//주문 여부 변경
                        }else if(coffeeTopping.equals("Milk") ||
                        coffeeTopping.equals("Mocha") ||
                        coffeeTopping.equals("Soy") ||
                        coffeeTopping.equals("Whip")){//커피 토핑 선택
                        toppingList.add(coffeeTopping);
                        //입력한 커피 토핑을 toppingList에 추가

                        //배열에서 토핑 삭제
                }else if(coffeeTopping.equals("RemoveMilk")){
                        removeTopping(toppingList,"Milk");
                }else if(coffeeTopping.equals("RemoveMocha")){
                        removeTopping(toppingList,"Mocha");
                }else if(coffeeTopping.equals("RemoveSoy")){
                        removeTopping(toppingList,"Soy");
                }else if(coffeeTopping.equals("RemoveWhip")){
                        removeTopping(toppingList,"Whip");
                }else{
                        System.out.println("잘못된 커피 토핑 주문입니다.");
                }
```

<최종주문>

```
        //최종 주문 상태를 반영
        for(int i=0;i<toppingList.size();i++){
        //toppingList 배열 내의 항목의 목록과 비용을 모두 더한다.
                        if(toppingList.get(i).equals("Milk")){
                        beverage=new Milk(beverage);
                }else if(toppingList.get(i).equals("Mocha")){
                        beverage=new Mocha(beverage);
                }else if(toppingList.get(i).equals("Soy")){
                        beverage=new Soy(beverage);
                }else if(toppingList.get(i).equals("Whip")){
                        beverage=new Whip(beverage);
                }
        }
        System.out.println
```

```
("주문하신 커피는 "+beverage.getDescription()+" 입니다.");
System.out.println
("총 금액은 "+beverage.cost()+"원입니다.");
```

```
//조합하는 음료의 이름 변경 (houseblend+mocha -> CAFEMOCHA)
class Mocha extends CondimentDecorator {
…
…

@Override
	public String getDescription() {
		if (beverage.getDescription().equals("HouseBlend"))
		{
			return this.description = "CAFEMOCHA";
		}
		return beverage.getDescription() + "+Mocha";
	}

}
```

모카가 추가될 때 beverage.getDescription()에 하우스블렌드가 존재하면 CAFEMOCHA로 재입력 라떼는 Milk클래스 getDescripion에 존재.

4. 결론

UML을 지키면서 코드를 짜는 것이 생각보다 어려웠다. 그 과정속에서 abstact와 interface의 차이를 알수있게 되었고, this를 쓰는 것에 익숙해졌다. 삭제부분에서 많은 생각을 하다가 결국 배열을 쓰는 방법을 택했는데, 데코레이션 패턴을 잘 활용한건지는 모르겠다. 어찌됐든 UML대로 구현해서 마음이 편하다.