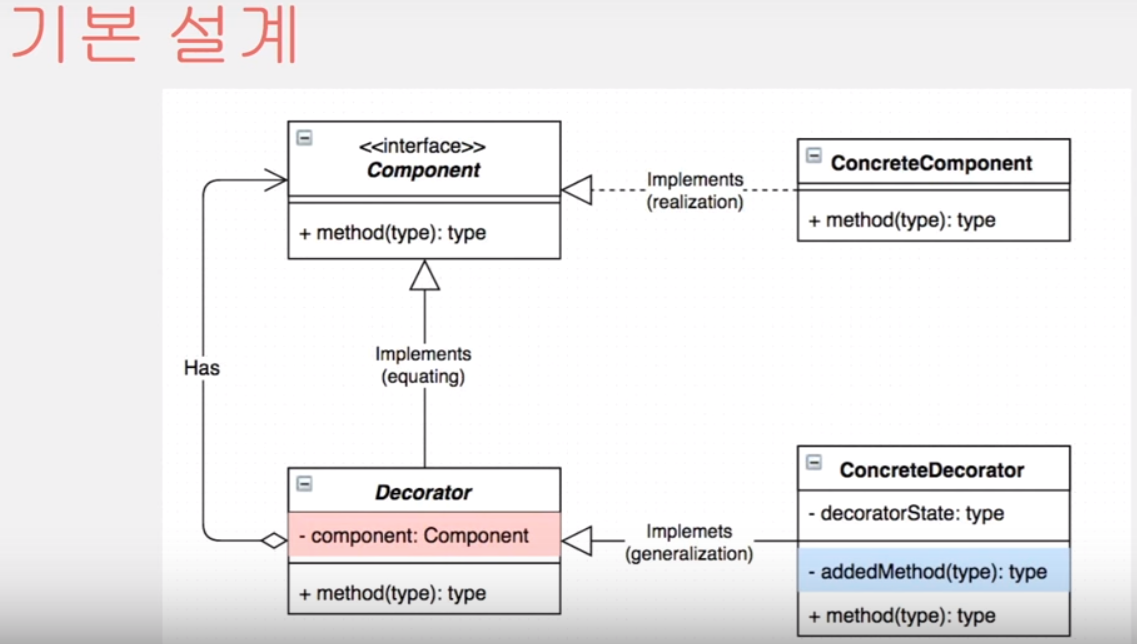
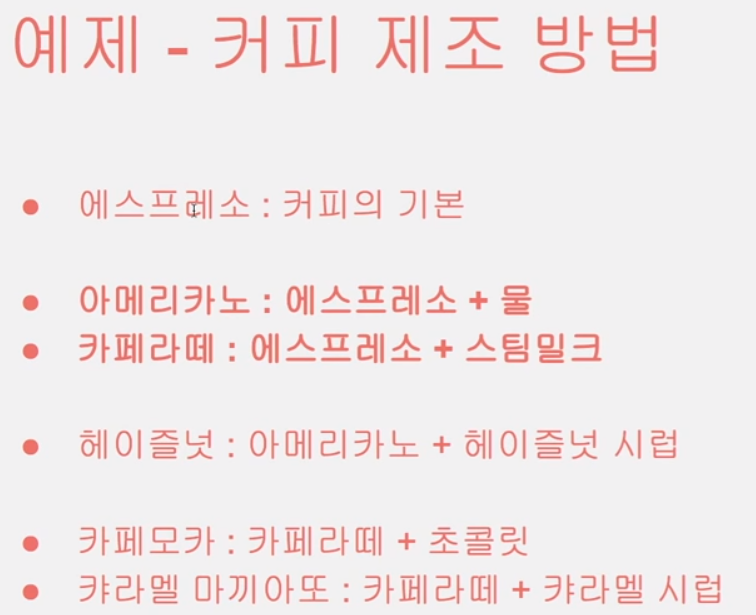
DecoratorPattern

* 동적으로 책임 추가가 필요할 때 데코레이터 패턴을 사용할 수 있다.

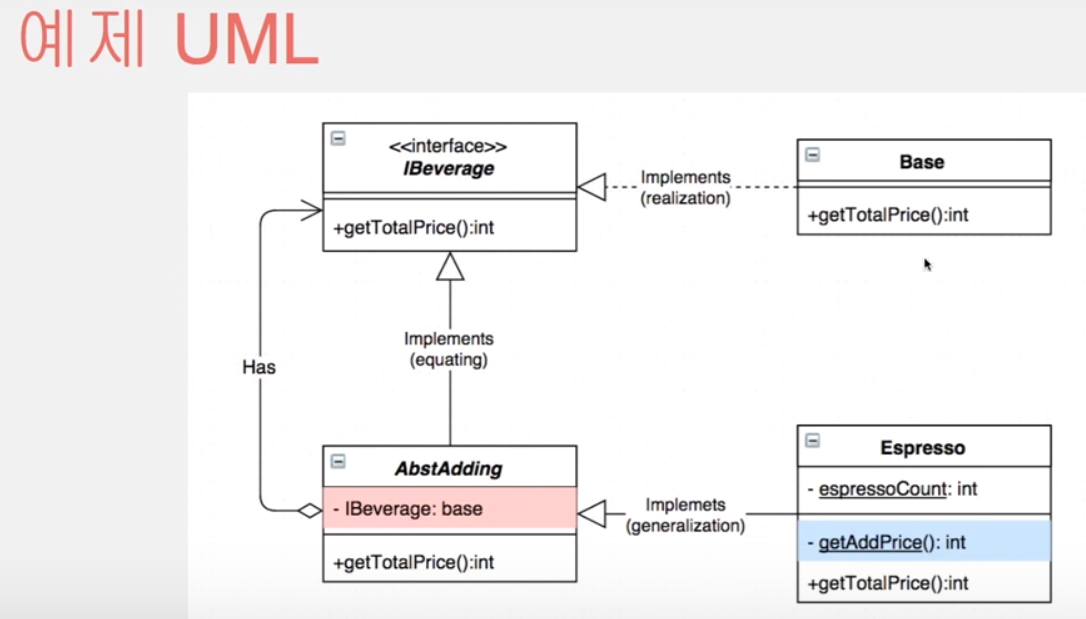
⇒ 실시간 책임 추가



예제



커피의 값을 측정하는 프로그램



소스코드 내용

package designpattern;

import java.util.Scanner;

public class DecoratorPattern {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

IBeverage beverage = new Base();

boolean done = false;

while (!done) {

System.out.println("음료 현재가격" + beverage.getTotalPrice());

System.out.println("선택 : 1샷 추가 / 2: 우유 추가");

switch (scan.nextInt()) {

case 0:

done = true;

break;

case 1:

beverage = new Espresso(beverage);

break;

case 2:

beverage = new Milk(beverage);

break;

}

}

System.out.println("음료가격 : "+ beverage.getTotalPrice());

scan.close();

}

}

interface IBeverage {

// 총 가격

int getTotalPrice();

}

class Base implements IBeverage {

@Override // 실질적인 책임의 주체

public int getTotalPrice() {

return 0; // 아무것도 없는 상태에서는 가격이 산출되지 않음

}

}

abstract class AbstAding implements IBeverage {

private IBeverage base; // 컴포턴트워 장식을 동시에함

public AbstAding(IBeverage base) {

super();

this.base = base;

}

protected IBeverage getBase() {

return base;

}

@Override

public int getTotalPrice() {

return base.getTotalPrice();

}

}

class Milk extends AbstAding {

public Milk(IBeverage meterial) {

super(meterial);

}

public int getTotalPrice() {

return super.getTotalPrice() + 50;

}

}

class Espresso extends AbstAding {

static protected int espressoCount = 0;

public Espresso(IBeverage base) {

super(base);

}

@Override

public int getTotalPrice() {

return super.getTotalPrice() + getAddPrice();

}

private static int getAddPrice() {

espressoCount += 1;

int addPrice = 100;

if (espressoCount > 1) {

addPrice = 70;

}

return addPrice;

}

}