МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

**Институт компьютерных технологий и информационной безопасности**

**Кафедра Математического обеспечения и применения ЭВМ**

**ОТЧЁТ**

по лабораторной работе №6

по курсу «GoF-паттерны ООП»

на тему «Паттерн Декоратор»

Выполнила:

студентка группы КТмо2-3

Куприянова А.А.

Проверил:

доцент кафедры ТОР

Максимов М.Н.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2017 г.

Таганрог 2017

**Цель работы:** Получение практических навыков работы с паттерном программирования «Декоратор».

**Задача**.

Сеть кофеин. К кофе продаются различные добавки (пенка, шоколад и т.д.). Дополнения не бесплатны, поэтому они должны быть встроены в систему оформления заказов. Также необходимо реализовать изменение цены в зависимости от объёма чашки, изменение температуры напитка и вывод меню.

**Выполнение работы.**

1. Новые два вида кофе:

class MyCoffee : public Beverage {

public:

MyCoffee() { description = "MyCoffee"; }

double cost() { return (2.99 \* volume); }

};

class MyAnotherCoffee : public Beverage {

public:

MyAnotherCoffee() { description = "MyAnotherCoffee"; }

double cost() { return (5.99 \* volume); }

};

Виды кофе характеризуются начальной ценой (ценой средней чашки кофе) и описанием.

Новые виды приправ:

class Cream : public CondimentDecorator {

public:

Cream(pBeverage bg) : CondimentDecorator(bg) {}

public:

string GetDescription() { return (beverage->GetDescription() + string(" with Cream")); }

double cost() { return 0.3 + beverage->cost(); }

};

class Milk : public CondimentDecorator {

public:

Milk(pBeverage bg) : CondimentDecorator(bg) {}

public:

string GetDescription() { return (beverage->GetDescription() + string(" with Milk")); }

double cost() { return 0.2 + beverage->cost(); }

};

Приправы добавляют цену напитку и изменяют его описание.

1. Для вывода меню разработан класс, который реализует добавление пунктов в меню и вывод меню на экран:

class Menu {

std::map<string, double> menu;

public:

void addItem(pBeverage b) { // добавление пункта меню

menu.insert(std::make\_pair<string, double>(b->GetDescription(), b->cost()));

}

void showMenu() { // вывод меню на экран

cout << "\nMenu:\n";

for (auto it : menu)

cout << it.first << "\t" << it.second << "\n";

}

};

1. Для изменения цены в зависимости от размера чашек был введён перечисляемый тип для характеристики чашки, параметр volume класса Beverage. В зависимости от размера чашки при установке размера чашки изменяется цена напитка:

void SetSize(CupSize cs) {

switch (cs) {

case normal: {

description += " Normal";

volume = 1;

break;

}

case small: {

description += " Small";

volume = 0.5;

break;

}

case big: {

description += " Big";

volume = 1.5;

break;

}

default:

break;

}

}

Изменение размера чашки также меняет описание напитка.

1. Для изменения температуры напитка был добавлен параметр t класса Beverage и функции для его задания и получения:

virtual string GetTemperature() {

return t;

}

void SetTemperature(string temp) {

description += " " + temp;

t = temp;

}

**Полный текст программы.**

#include "stdafx.h"

#include <iostream>

#include <string>

#include <map>

using namespace std;

// тип для характеристики размера чашки

enum CupSize {

small,

normal,

big

};

template <class T>

class CountedPtr {

private:

T\* ptr; // Указатель на значение

long\* count; // Количество владельцев (общие данные)

public:

// Инициализация объекта существующим указателем

// - указатель p должен быть получен в результате вызова new

explicit CountedPtr(T\* p = 0) : ptr(p), count(new long(1)) {

}

// Копирующий указатель (увеличивает счётчик владельцев)

CountedPtr(const CountedPtr<T>& p) throw() : ptr(p.ptr), count(p.count) {

++\*count;

}

// Деструктор (уничтожает объект, если владелец был последним)

~CountedPtr() throw() {

dispose();

}

// Присваивание (перевод указателя на новый объект)

CountedPtr<T>& operator= (const CountedPtr<T>& p) throw() {

if (this != &p) {

dispose();

ptr = p.ptr;

count = p.count;

++\*count;

}

return \*this;

}

// Доступ к объекту, на который ссылается указатель

T& operator\*() const throw() {

return \*ptr;

}

T\* operator->() const throw() {

return ptr;

}

private:

void dispose() {

if (--\*count == 0) {

delete count;

delete ptr;

}

}

};

class Beverage {

protected:

string description;

double volume = 1;

string t = "warm";

public:

void setDescription(string d) { description = d; }

virtual double cost() = 0;

virtual string GetDescription() {

return description;

}

virtual double GetSize() {

return volume;

}

void SetSize(CupSize cs) {

switch (cs) {

case normal: {

description += " Normal";

volume = 1;

break;

}

case small: {

description += " Small";

volume = 0.5;

break;

}

case big: {

description += " Big";

volume = 1.5;

break;

}

default:

break;

}

}

virtual string GetTemperature() {

return t;

}

void SetTemperature(string temp) {

description += " " + temp;

t = temp;

}

};

typedef CountedPtr<Beverage> pBeverage;

class CondimentDecorator : public Beverage {

protected:

pBeverage beverage;

CondimentDecorator(pBeverage pBg) : beverage(pBg) {}

};

class EmptyCup : public Beverage {

public:

EmptyCup() { description = ""; }

double cost() { return 0; }

};

class Espresso : public Beverage {

public:

Espresso() { description = "Espresso"; }

double cost() { return (1.99 \* volume); }

};

class HouseBlend : public Beverage {

public:

HouseBlend() { description = "HouseBlend"; }

double cost() { return (0.89 \* volume); }

};

class MyCoffee : public Beverage {

public:

MyCoffee() { description = "MyCoffee"; }

double cost() { return (2.99 \* volume); }

};

class MyAnotherCoffee : public Beverage {

public:

MyAnotherCoffee() { description = "MyAnotherCoffee"; }

double cost() { return (5.99 \* volume); }

};

class Mocha : public CondimentDecorator {

public:

Mocha(pBeverage bg) : CondimentDecorator(bg) {}

public:

string GetDescription() { return (beverage->GetDescription() + string(" Mocha")); }

double cost() { return 0.20 + beverage->cost(); }

};

class Soy : public CondimentDecorator {

public:

Soy(pBeverage bg) : CondimentDecorator(bg) {}

public:

string GetDescription() { return (beverage->GetDescription() + string(" with Soy")); }

double cost() { return 0.16 + beverage->cost(); }

};

class Cream : public CondimentDecorator {

public:

Cream(pBeverage bg) : CondimentDecorator(bg) {}

public:

string GetDescription() { return (beverage->GetDescription() + string(" with Cream")); }

double cost() { return 0.3 + beverage->cost(); }

};

class Milk : public CondimentDecorator {

public:

Milk(pBeverage bg) : CondimentDecorator(bg) {}

public:

string GetDescription() { return (beverage->GetDescription() + string(" with Milk")); }

double cost() { return 0.2 + beverage->cost(); }

};

class Menu {

std::map<string, double> menu;

public:

void addItem(pBeverage b) {

menu.insert(std::make\_pair<string, double>(b->GetDescription(), b->cost()));

}

void showMenu() {

cout << "\nMenu:\n";

for (auto it : menu)

cout << it.first << "\t" << it.second << "\n";

}

};

int main()

{

pBeverage pB = pBeverage(new Espresso());

cout << pB->GetDescription() + " $" << pB->cost();

cout << "\n";

// изменение размера чашки пропорционально меняет цену продукта

pB->SetSize(small);

cout << pB->GetDescription() + " $" << pB->cost();

cout << "\n";

// изменение температуры не влияет на цену, нов лияет на описание

pB->SetTemperature("hot");

cout << pB->GetDescription() + " $" << pB->cost();

cout << "\n";

// добавление добавки прибавляет её цену к цене продукта

pB = pBeverage(new Milk(pB));

cout << pB->GetDescription() + " $" << pB->cost();

cout << "\n";

// другой вид кофе

pBeverage pB2 = pBeverage(new HouseBlend());

pB2->SetSize(small);

pB2 = pBeverage(new Milk(pB2));

cout << pB2->GetDescription() + " $" << pB2->cost();

cout << "\n";

Menu menu;

pBeverage cup = pBeverage(new EmptyCup());

menu.addItem(pBeverage(new Espresso()));

menu.addItem(pBeverage(new HouseBlend()));

menu.addItem(pBeverage(new MyCoffee()));

menu.addItem(pBeverage(new MyAnotherCoffee()));

menu.addItem(pBeverage(new Cream(cup)));

menu.addItem(pBeverage(new Milk(cup)));

menu.addItem(pBeverage(new Mocha(cup)));

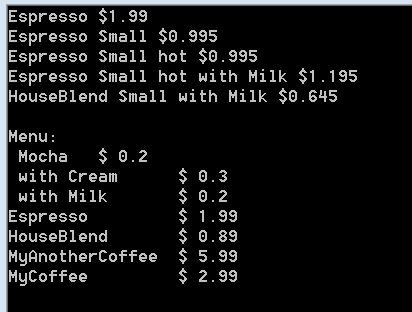
menu.showMenu();

getchar();

return 0;

}

**Результат работы программы.**



При изменении размера чашки, а также при добавлении добавок меняется цена и описание кофе, изменение температуры напитка не меняет его цену, но меняет описание.