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 №8

по курсу «GoF-паттерны ООП»

на тему «Паттерны Абстрактная фабрика и Одиночка»

Выполнила:

студентка группы КТмо2-3

Куприянова А.А.

Проверил:

доцент кафедры ТОР

Максимов М.Н.

Оценка

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

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

Таганрог 2017

**Цель работы:** Получение навыков использования паттернов «Абстрактная фабрика» и «Одиночка».

**Задание:**

1. Создайте пиццерию в чикагском стиле.
2. Доопределите класс Prise так, чтобы в нём содержалась информация о стоимости ингредиентов пиццы. Причём объект Prise должен быть общим (единственным) для всех пиццерий.
3. Выведите на экран, из каких компонентов состоит пицца, и сколько она стоит.
4. Модифицируйте код так, чтобы кроме вида пиццы можно было бы указывать и её размер.
5. Модифицируйте код так, чтобы вместе с пиццей можно было заказать и добавки к ней (декоратор).
6. Упростите программу, избавившись от класса CountedPtr.

**Выполнение работы:**

// AbstractFabric.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class Dough { protected: string name; public: string GetName()const { return name; } }; //Тесто

class ThinDough : public Dough { public: ThinDough() { name = "thin dough"; } };

class FatDough : public Dough { public: FatDough() { name = "fat dough"; } };

class Sauce { protected: string name; public: string GetName()const { return name; } };//Соус

class TomatoSauce : public Sauce { public: TomatoSauce() { name = "tomato sauce"; } };

class KetchupSauce : public Sauce { public: KetchupSauce() { name = "ketchup sauce"; } };

class Cheese { protected: string name; public: string GetName()const { return name; } };//Сыр

class HardCheese : public Cheese { public: HardCheese() { name = "hard cheese"; } };

class SoftCheese : public Cheese { public: SoftCheese() { name = "soft cheese"; } };

class Clams { protected: string name; public: string GetName()const { return name; } };//Мидии

class FreshClams : public Clams { public: FreshClams() { name = "fresh clams"; } };

class FrozenClams : public Clams { public: FrozenClams() { name = "frozen clams"; } };

//---------------------------------------------------------------------------------------------

//фабрика ингредиентов

class PizzaIngredientFactory {

public:

virtual Dough\* createDough() = 0;

virtual Sauce\* createSauce() = 0;

virtual Cheese\* createCheese() = 0;

virtual Clams\* createClams() = 0;

virtual ~PizzaIngredientFactory() {}

};

//фабрика ингредиентов в Нью-Йорке

class NYPizzaIngredientFactory : public PizzaIngredientFactory {

Dough\* createDough() override { return new ThinDough(); }

Sauce\* createSauce() override { return new TomatoSauce(); }

Cheese\* createCheese() override { return new HardCheese(); }

Clams\* createClams() override { return new FreshClams(); }

};

//фабрика ингредиентов в Чикаго

class CHPizzaIngredientFactory : public PizzaIngredientFactory {

Dough\* createDough() override { return new FatDough(); }

Sauce\* createSauce() override { return new KetchupSauce(); }

Cheese\* createCheese() override { return new SoftCheese(); }

Clams\* createClams() override { return new FrozenClams(); }

};

class PizzaMixin {

protected:

string name;

int size;

public:

PizzaMixin(int sz = 0) {

size = sz >= 1 && sz <= 3 ? sz : 1;

name = "";

}

virtual ~PizzaMixin() {}

void SetSize(int sz) { size = sz >= 1 && sz <= 3 ? sz : size; }

int getSize()const { return size; }

string GetName()const { return name; }

};

class MixinDecorator : public PizzaMixin {

protected:

PizzaMixin\* mixin;

MixinDecorator(PizzaMixin\* pc) : mixin(pc) {

SetSize(pc->getSize());

name = "";

}

public:

string GetMixinName()const { return name; }

};

class CheeseMixin : public MixinDecorator {

public:

CheeseMixin(PizzaMixin\* pc) : MixinDecorator(pc) {

name = "cheese mixin";

}

};

class SauceMixin : public MixinDecorator {

public:

SauceMixin(PizzaMixin\* pc) : MixinDecorator(pc) {

name = "sause mixin";

}

};

class Pizza {//пицца

protected:

PizzaIngredientFactory\* pingredientFactory; //Интерфейс "Абстрактная фабрика"

string name;

int size;

//Интерфейсы "Абстрактные продукты"

Dough\* pdough; //Тесто

Sauce\* psauce; //Соус

Cheese\* pcheese; //Сыр

Clams\* pclams; //Мидии

vector<PizzaMixin\*> mixins;

public:

Pizza(int sz = 1) {

size = sz >= 1 && sz <= 3 ? sz : 1;

name = "default pizza";

mixins.push\_back(new PizzaMixin());

pingredientFactory = NULL;

pdough = NULL;

psauce = NULL;

pcheese = NULL;

pclams = NULL;

}

virtual ~Pizza() {

delete pdough;

delete psauce;

delete pcheese;

delete pclams;

delete pingredientFactory;

for (auto it = mixins.begin(); it != mixins.end(); it++)

delete \*it;

mixins.clear();

}

virtual void prepare() { cout << string("Preparing ") + name << endl; }

virtual void bake() { cout << "Bake for 25 minutes at 350" << endl; }

virtual void cut() { cout << "Cutting the pizza into diagonal slices" << endl; }

virtual void box() { cout << "Place pizza in official PizzaStore box" << endl; }

void AddMixin(PizzaMixin\* pc) {

if (this->GetName() != "default pizza")

mixins.push\_back(pc);

}

void SetSize(int sz) { size = sz >= 1 && sz <= 3 ? sz : size; }

void SetName(string nm) { name = nm; }

PizzaMixin\* GetMixinStandart() { return mixins[0]; }

vector<PizzaMixin\*> getMixins()const { return mixins; }

int getSize()const { return size; }

string GetName()const { return name; }

Dough\* getDough() { return pdough; }

Sauce\* getSauce() { return psauce; }

Cheese\* getCheese() { return pcheese; }

Clams\* getClams() { return pclams; }

};

class CheesePizza : public Pizza {//пицца с сыром

public:

CheesePizza(PizzaIngredientFactory\* pF, int size = 1) : Pizza(size) {

pingredientFactory = pF;

name = "CheesePizza";

}

void prepare() override {

cout << "Preparing " + name << endl;

pdough = pingredientFactory->createDough();

psauce = pingredientFactory->createSauce();

pcheese = pingredientFactory->createCheese();

}

};

class ClamsPizza : public Pizza { //пицца с мидиями

public:

ClamsPizza(PizzaIngredientFactory\* pF, int size = 1) : Pizza(size) {

pingredientFactory = pF;

name = "ClamsPizza";

}

void prepare() override {

cout << "Preparing " + name << endl;

pdough = pingredientFactory->createDough();

psauce = pingredientFactory->createSauce();

pclams = pingredientFactory->createClams();

}

};

class PizzaStore { //пиццерия

private:

vector<Pizza\*> \_pizzas;

public:

Pizza\* orderPizza(string type, int size = 1) {

Pizza\* pizza = createPizza(type, size);

pizza->prepare();

pizza->bake();

pizza->cut();

pizza->box();

\_pizzas.push\_back(pizza);

return pizza;

}

~PizzaStore() {

for (auto it = \_pizzas.begin(); it != \_pizzas.end(); it++)

delete \*it;

\_pizzas.clear();

}

protected:

virtual Pizza\* createPizza(string type, int size = 1) = 0;

};

class NYPizzaStore : public PizzaStore {//Пиццерия в Нью-Йорке

public:

Pizza\* createPizza(string type, int size = 1) override {

if (type == "cheese")

return new CheesePizza(new NYPizzaIngredientFactory(), size);

if (type == "clams")

return new ClamsPizza(new NYPizzaIngredientFactory(), size);

return new Pizza(size);

}

};

class CHPizzaStore : public PizzaStore {

public:

Pizza\* createPizza(string type, int size = 1) override {

if (type == "cheese")

return new CheesePizza(new CHPizzaIngredientFactory(), size);

if (type == "clams")

return new ClamsPizza(new CHPizzaIngredientFactory(), size);

return new Pizza(size);

}

};

//Прайс на продукты

class Prise {

static bool fCreate;

static Prise\* price;

double costDout;

double costSauce;

double costCheese;

double costClams;

double costPizza;

double mixinTotal;

double cheeseMixinCost;

double sauceMixinCost;

Prise() {

costDout = 0;

costSauce = 0;

costCheese = 0;

costClams = 0;

costPizza = 0;

mixinTotal = 0;

cheeseMixinCost = 0;

sauceMixinCost = 0;

}

public:

static Prise\* getInstance() {

if (!fCreate) {

fCreate = true;

price = new Prise();

}

return price;

}

static void destroyInstance() {

if (!fCreate) return;

fCreate = false;

delete price;

}

void getPrises(Pizza\* pizza) {

cout << endl;

costDout = 0;

costSauce = 0;

costCheese = 0;

costClams = 0;

costPizza = 0;

mixinTotal = 0;

cheeseMixinCost = 0;

sauceMixinCost = 0;

Dough\* dough = pizza->getDough();

Sauce\* sauce = pizza->getSauce();

Cheese\* cheese = pizza->getCheese();

Clams\* clams = pizza->getClams();

auto mixins = pizza->getMixins();

int size = pizza->getSize();

string sizeName;

if (size == 1) {

sizeName = "small";

}

else if (size == 2) {

sizeName = "medium";

}

else if (size == 3) {

sizeName == "large";

}

cout << sizeName << " " << pizza->GetName() << ": $" << getCostPizza(pizza) << endl;

if (dough != NULL || sauce != NULL || cheese != NULL || clams!=NULL)

cout << "Ingredients:\n";

if (dough != NULL || sauce != NULL || cheese != NULL || clams != NULL) {

if (dough != NULL)

cout << dough->GetName() << "\n";

if (sauce != NULL)

cout << sauce->GetName() << "\n";

if (cheese != NULL)

cout << cheese->GetName() << "\n";

if (clams != NULL)

cout << clams->GetName() << "\n";

cout << endl;

}

// добавочки

if (mixins.size() > 1) {

getCheeseMixinCost(pizza);

getSauceMixinCost(pizza);

}

cout << "Pizza price: $" << getWholePizza(pizza) << endl;

}

double getWholePizza(Pizza\* pizza) {

int size = pizza->getSize();

return getCostPizza(pizza) + costDout + costSauce + costCheese

+ costClams + cheeseMixinCost \* size + sauceMixinCost \* size;

}

double getCostPizza(Pizza\* pizza) {

if (pizza->GetName() == "CheesePizza")

costPizza = 1.3;

else if (pizza->GetName() == "ClamsPizza")

costPizza = 3.25;

return costPizza;

}

double getDoughCost(Dough\* dough) {

//cout << "dough=" << dough;

if (dough == NULL)

return costDout;

if (dough->GetName() == "thin")

costDout = 0.5;

if (dough->GetName() == "fat")

costDout = 0.65;

return costDout;

}

double GetcostSauce(Sauce\* sauce) {

if (sauce == NULL)

return costSauce;

else if (sauce->GetName() == "sauce1")

costSauce = 2.3;

else if (sauce->GetName() == "sauce2")

costSauce = 2.3;

return costSauce;

}

double GetcostCheese(Cheese\* cheese) {

if (cheese == NULL)

return costCheese;

else if (cheese->GetName() == "cheese1")

costCheese = 2.1;

if (cheese->GetName() == "cheese2")

costCheese = 1.2;

return costCheese;

}

double GetcostClams(Clams\* clams) {

if (clams == NULL)

return costClams;

else if (clams->GetName() == "clams1")

costClams = 3.2;

else if (clams->GetName() == "clams2")

costClams = 1.5;

return costClams;

}

double getCheeseMixinCost(Pizza\* pizza) {

auto mixins = pizza->getMixins();

for (auto it = mixins.begin(); it != mixins.end(); it++)

if ((\*it)->GetName() == "cheese mixin") {

cheeseMixinCost = 0.1;

cout << (\*it)->GetName() << ": $" << cheeseMixinCost \* pizza->getSize() << "\n";

}

return cheeseMixinCost;

}

double getSauceMixinCost(Pizza\* pizza) {

auto mixins = pizza->getMixins();

for (auto it = mixins.begin(); it != mixins.end(); it++)

if ((\*it)->GetName() == "sauce mixin") {

sauceMixinCost = 0.2;

cout << (\*it)->GetName() << ": $" << sauceMixinCost \* pizza->getSize() << "\n";

}

return sauceMixinCost;

}

};

bool Prise::fCreate = false;

Prise\* Prise::price;

//--------------------------------------------------------------------------------------

int main()

{

NYPizzaStore nyp;

CHPizzaStore chp;

Prise\* pr = Prise::getInstance();

Pizza\* nyCheesePizza = nyp.orderPizza("cheese", 2);

cout << "\n\n";

nyp.orderPizza("clams");

cout << "\n\n";

nyp.orderPizza("");

cout << "\n\n";

Pizza\* chClamsPizza = chp.orderPizza("clams");

cout << "\n\n";

nyCheesePizza->AddMixin(new CheeseMixin(nyCheesePizza->GetMixinStandart()));

pr->getPrises(nyCheesePizza);

pr->getPrises(chClamsPizza);

Prise::destroyInstance();

system("pause");

return 0;

}

**Результаты работы программы:**

