ЛАБОРАТОРНАЯ РАБОТА

по дисциплине «Programming Techniques»

направления 10.05.01 «Компьютерная безопасность»

Практическое задание по теме «Паттерны проектирования 2»

Вариант 8

Выполнил студент группы СКБ172

Федоров Сергей Андреевич

«25» Мая 2020 г.

Москва, 2020

Задача**:**

**Вариант 8.**

*Используя паттерн Observer, реализовать модель электронного аукциона. Пул товаров*

*(лотов) фиксирован, число участников определяется на этапе выполнения, при этом оно*

*может изменяться в зависимости от лота.*

**Цель лабораторной работы**

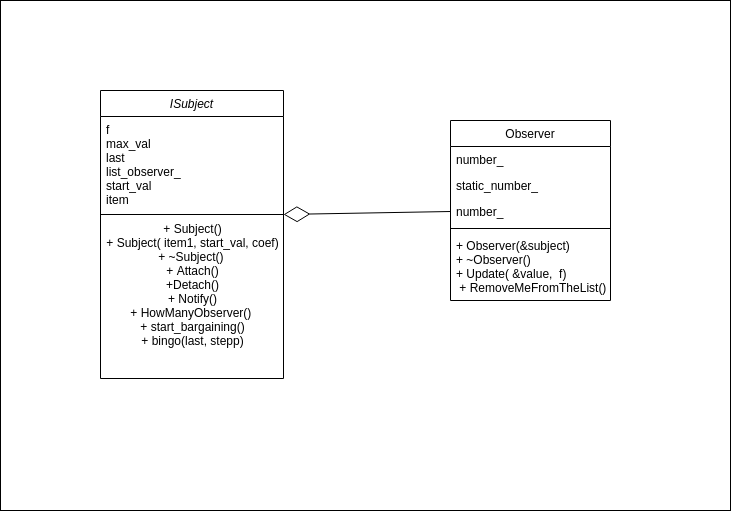
Цель лабораторной работы – познакомиться с различными типами паттернов программирования, научиться использовать их в прикладных задачах, сделать выводы о возможном использовании паттернов.

**Результаты работы**

1. Используя паттерн *Observer*, была создана симуляция аукциона товаров.

**«UML-диаграмма»**

Так же UML диаграмму можно увидеть в отчете DOXYGEN.

****

**Вывод:**

Данный паттерн удобен когда после изменения состояния одного объекта требуется что-то сделать в других, но вы не знаете наперёд, какие именно объекты должны отреагировать.

Когда одни объекты должны наблюдать за другими, но только в определённых случаях.

**Код программы:**

/\*! \mainpage <CENTER> Лабораторная работа N5 Выполнил Федоров Сергей Скб172 </CENTER>

*\* <CENTER> Вариант 8 </CENTER>*

*\* Вариант 8. Используя паттерн Observer, реализовать модель электронного аукциона. Пул товаров*

*\* (лотов) фиксирован, число участников определяется на этапе выполнения, при этом оно*

*\* может изменяться в зависимости от лота.*

*\*/*

#include <iostream>

#include <list>

#include <string>

using namespace std;

*/\*!*

*\brief Абстрактный класс объекта за которым наблюдают*

*\*/*

class IObserver {

public:

virtual ~IObserver(){};

virtual void Update(const int& value, bool f) = 0;

};

*/\*!*

*\brief Абстрактный класс наблюдателя*

*\*/*

class ISubject {

public:

virtual ~ISubject(){};

virtual void Attach(IObserver\* observer) = 0;

virtual void Detach(IObserver\* observer) = 0;

virtual void Notify() = 0;

};

*/\*\**

*\* Издатель владеет некоторым важным состоянием и оповещает наблюдателей о его*

*\* изменениях.*

*\*/*

class Subject : public ISubject {

public:

*/\*!*

*\brief конструктор по умолчанию*

*Просто выводит на экран что объект создан успешно*

*\*/*

Subject()

{

cout << "start subject " << endl;

}

*/\*!*

*\brief конструктор задающий лот и стартовую стоимость*

*\param[in] item1 название вещи*

*\param[in] start\_val начальная стоимость*

*\param[in] coef коэффициент за которого будут готовы продать вещь по умолчанию он равен 300*

*Создаем лот и стартовую цену. Шаг ставок обычно 100. И вещь продается человеку, который 1-ый достиг цены превышающей в 300 раз начальную.*

*\*/*

Subject(string item1, int start\_val, int coef = 300)

{

cout << "Now bet is: " << item1 << " and start value is " << start\_val << endl;

item = item1;

this->start\_val = start\_val;

this->max\_val = start\_val \* coef;

}

*/\*!*

*\brief деструктор*

*Просто пишет Прощание и выводит имя свеого продаваемого товара, цену по которой товар был продан.*

*И номер человека которому был продан товар. Если N человека =0 то этот товар не был продан.*

*\*/*

virtual ~Subject()

{

cout << "Goodbye, I was the Subject. and i sold " << item << " for price: " << start\_val << " person number " << last << endl;

}

*/\*\**

*\* Методы управления подпиской.*

*\*/*

*/\*!*

*\brief добавить наблюдателя*

*\*/*

void Attach(IObserver\* observer) override

{

list\_observer\_.push\_back(observer);

}

*/\*!*

*\brief Исключить наблюдателя*

*\*/*

void Detach(IObserver\* observer) override

{

list\_observer\_.remove(observer);

}

*/\*!*

*\brief Уведомить всех подписчиков*

*\*/*

void Notify() override

{

std::list<IObserver\*>::iterator iterator = list\_observer\_.begin();

HowManyObserver();

while (iterator != list\_observer\_.end()) {

(\*iterator)->Update(start\_val, f);

++iterator;

}

}

*/\*!*

*\brief Выводит кол-во участников в торгах*

*\*/*

void HowManyObserver()

{

cout << "There are " << list\_observer\_.size() << " observers in the list.**\n**";

}

*/\*!*

*\brief Начало торгов и сами торги*

*Выводит на экран нынешнюю стоимость товара и оповещает всех подписчиков, о нынешней стоимости товара.*

*\*/*

void start\_bargaining()

{

cout << "-------------------- " << endl;

cout << "Now price is " << start\_val << endl;

Notify();

}

*/\*!*

*\brief Проверка и повышение стоимости*

*Провека не продан ли был товар. Установление новой стоимости на товар (по шагу, шаг по умолчанию 100).*

*И тут же определяется купивший товар человек.*

*\*/*

void bingo(int last, int stepp = 100)

{

if (max\_val <= start\_val) {

f = 1;

cout << "===================SOLD=================" << endl;

}

else {

start\_val += stepp;

this->last = last;

start\_bargaining();

}

}

private:

bool f = 0; // если 0, то товар не продан, 1 товар продан

int max\_val; // максимальная стоимость

int last = 0; // человек которому было продано, по умолчанию никому

list<IObserver\*> list\_observer\_; // список наблюдателей

string item = ""; // название товара

int start\_val = 0; // начальная стоимость

};

*/\*!*

*\brief Класс наблюдателя*

*Тут реализована логика подписчика.*

*\*/*

class Observer : public IObserver {

public:

*/\*!*

*\brief Конструктор наблюдателя*

*Устанавливает за кем наблюдает.*

*\*/*

Observer(Subject& subject)

: subject\_(subject)

{

this->subject\_.Attach(this);

cout << "Hi, I'm the Observer **\"**" << ++Observer::static\_number\_ << "**\"**.**\n**";

this->number\_ = Observer::static\_number\_;

}

*/\*!*

*\brief Деструктор наблюдателя*

*Просто прописывает прощание учвастника и его номер.*

*\*/*

virtual ~Observer()

{

cout << "Goodbye, I was the Observer **\"**" << this->number\_ << "**\"**.**\n**";

}

*/\*!*

*\brief Будет ли он повышать?*

*Тут проверяется повысит ли ставку участник или нет..*

*\*/*

void Update(const int& value, bool f) override

{

if (!f) {

int will\_I = rand() % 2;

if (will\_I) {

cout << "i'm " << this->number\_ << " and yes " << will\_I << endl;

subject\_.bingo(this->number\_);

}

else {

cout << "i'm " << this->number\_ << " and no " << will\_I << endl;

}

}

}

*/\*!*

*\brief Выходит из участия за лот*

*Участник уходит из торгов*

*\*/*

void RemoveMeFromTheList()

{

subject\_.Detach(this);

cout << "Observer **\"**" << number\_ << "**\"** removed from the list.**\n**";

}

private:

Subject& subject\_;

static int static\_number\_;

int number\_;

};

int Observer::static\_number\_ = 0;

*/\*!*

*\brief Симулиция клиентского запроса*

*\*/*

int main()

{

srand(time(0));

Subject subject("xxx", 1100);

Observer observer1(subject);

Observer observer2(subject);

// Observer observer3(subject);

subject.start\_bargaining();

// cout << "================== ROUND 2 ===============" <<endl;

// Subject subject1("SOME", 2100);

// Observer observer3(subject1);

// Observer observer4(subject1);

// Observer observer5(subject1);

// subject1.start\_bargaining();

cout << " ================== end ======================" << endl;

return 0;

}