МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ст. преподаватель |  |  |  | Е. О. Шумова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ОРГАНИЗАЦИИ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ ПРИ ЗАДАННЫХ КРИТЕРИЯХ |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134К |  |  |  | Н.А. Костяков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

Оглавление

[1. Постановка задачи 2](#_Toc153289453)

[1.1 Анализ предметной области 2](#_Toc153289454)

[2 Разработка классов 3](#_Toc153289455)

[2.1 Иерархия классов 3](#_Toc153289456)

[2.2 Управляющие классы: 4](#_Toc153289457)

[2.3 Интерфейсные классы: 5](#_Toc153289458)

[2.5 Рассмотрим полученную диаграмму: 5](#_Toc153289459)

[3.1 Описание интерфейсов 6](#_Toc153289460)

[3.2 Разработка методов классов 7](#_Toc153289461)

[Тестирование программы 7](#_Toc153289462)

[Приложение 11](#_Toc153289463)

# 1. Постановка задачи

Данная курсовая работа по программированию направлена на разработку системы классов, которая будет описывать Кинопрокат. Основной целью проектирования является создание программного продукта, который позволит вести учет фильмов отданных в прокат и их возврат.

# 1.1 Анализ предметной области

1.1.1. Предметной областью является управление магазина кинопроката. Основными сущностями предметной области являются фильмы, пользователи, операции аренды и возврата

1.1.2. Словарь предметной области:

- Фильм

- Автор

- Покупатель

- Аренда

- Возврат

1.1.3. Функциональные требования:

- Ведение данных о наличии фильмов для проката

- Фиксация данных об аренде фильма

- Сохранение информации о фильме: автор, год, название

- Учет данных о покупателях, их билет для доступа

- Расчет стоимости проката с учетом скидок и наценок.

- Поиск и фильтрация данных о прокате и возврата.

# 2 Разработка классов

Для разработки иерархии классов мы начнем с выделения основных сущностей предметной области и определения классов, описывающих эти сущности. Затем мы определим управляющие классы и интерфейсные классы для организации взаимодействия между ними и с внешней средой. Ниже приведена детальная разработка иерархии классов.

# 2.1 Иерархия классов

**2.1.1 Фильм**

**-**id (Идентификатор фильма)

-name (название фильма)

-автор (автор фильма)

-год (год выпуска фильма)

Методы:

- Конструктор для создания

- Методы для получения и установки значений полей

- Метод для изменения статуса

**2.1.2 Юзер**

**-**id (идентификатор пользователя)

-ФИО (Фамилия имя отчество полльзователя)

Методы:

- Конструктор для создания

- Методы для получения и установки значений полей

**2.1.3 Аренда**

-id (Идентификатор аренды)

-userId (идентификатор пользователя, который взял фильм)

-filmId (идентификатор фильма, который взяли в аренду)

-Date take (Дата взятия в аренду)

-Date return (дата, кода фильм должны вернуть)

Методы:

- Конструктор для создания

- Методы для получения и установки значений полей

- Метод для изменения статуса

# 2.2 Управляющие классы:

2.2.1. магазин (shop):

- Поля:

- Список всех доступных фильмов

- Методы:

- Методы для добавления и удаления фильмов

- Методы для поиска фильмов различным критериям (год, автор)

2.2.2. Прокат (rental):

- Поля:

- Список всех операций проката

- Методы:

- Методы для регистрации новой операции проката

- Методы для отчетности по прокатам

# 2.3 Интерфейсные классы:

Графический интерфейс (GUI):

- Методы:

- Методы для взаимодействия с пользователем, включая ввод и вывод данных.

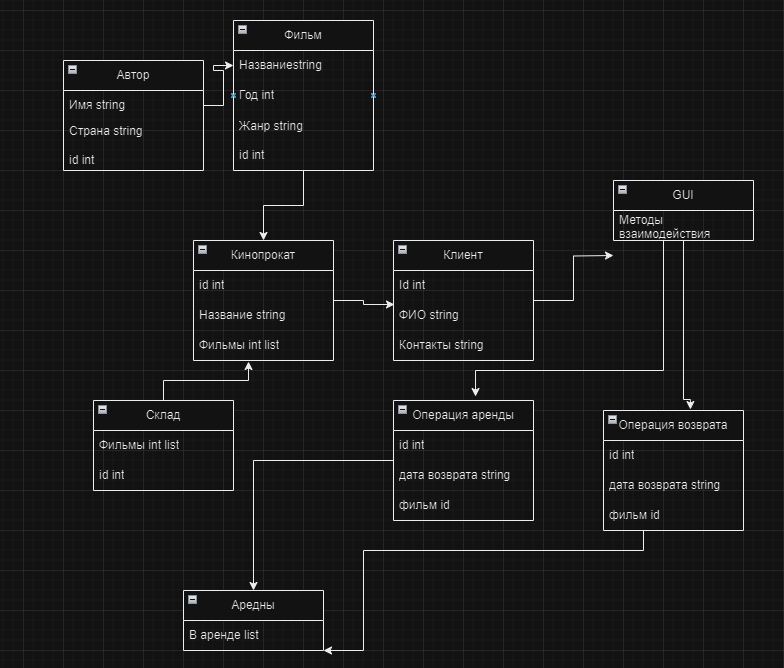
**Обоснование проектных решений:**

- Использование классов сущностей для описания базовых объектов предметной области позволяет удобно хранить и обрабатывать информацию о фильмах в наличии, авторах, покупателях и операциях проката.

- Управляющие классы, такие как Склад и Прокат, позволяют организовать управление данными и операциями, связанными с фильмами и прокатом

- Интерфейсный класс GUI обеспечивает взаимодействие пользователя с программой через графический интерфейс, что делает использование приложения более удобным для конечных пользователей.

# 2.5 Рассмотрим полученную диаграмму:



Эта иерархия классов позволит эффективно управлять данными и операциями, связанными с кинопрокатом, а также обеспечит удобный интерфейс для пользователей.

# 3.1 Описание интерфейсов

Интерфейс представляет из себя набор GroupBox которые становятся прозрачными при определенных событиях и непрозрачными при дргугих. Таким образом все действия происходят в одном окне и элементы взаимозаменяют друг друга.

ui->groupBox\_registration->setHidden(1);

Разработались несколько форм для регистрации, которая состоит из нескольких полей ввода и кнопки для проверки, форма для того, чтобы выбрать фильм, которая наполняется в зависимости от выбранного магазина, форма возврата, аналогичная форме выбора фильма и навигационная панель, которая размещает на себе дополнительный функционал, например регистрация, выбор магазина и просмотра логов

# 3.2 Разработка методов классов

Логикой управляют несколько классов, которые объединяют в себе общие методы для работы программы

Так класс shop, который представляет физический магазин, в себе имеет класс movies, который играет роль каталога. Так каждый такой магазин может сам для себя определить набор фильмов, которыми торгует

Rentalstrategy определяет, по какой цене будет выдан фильм покупателю, так исходя из сценариев, в программе видно, сколько стоит прокат.

Subscribers – таблица пользователей, которые взаимодействуют с ПО

Rentalmovies – адаптер лдя класса rental, который собирает нужную информацию о фильме и пользователе и покует ее в таблицу взятых в аренду фильмов

Методы класса movies

void add(int y, std::string auth, std::string n); - добавление

movie\* get(int index); -получить фильм по индексу

int get\_length(){return count;} – получить количество фильмов

**методы класса subscriber**

int get\_length(); - получить длинну

void registers(std::string name); - зарегистрировать пользователя

subscriber\* find(std::string name); - найти пользователя по имени

subscriber\* get(int index); - найти пользователя по индексу

Адаптер rentalmovies

void add(info \*input); - добавить в аренду

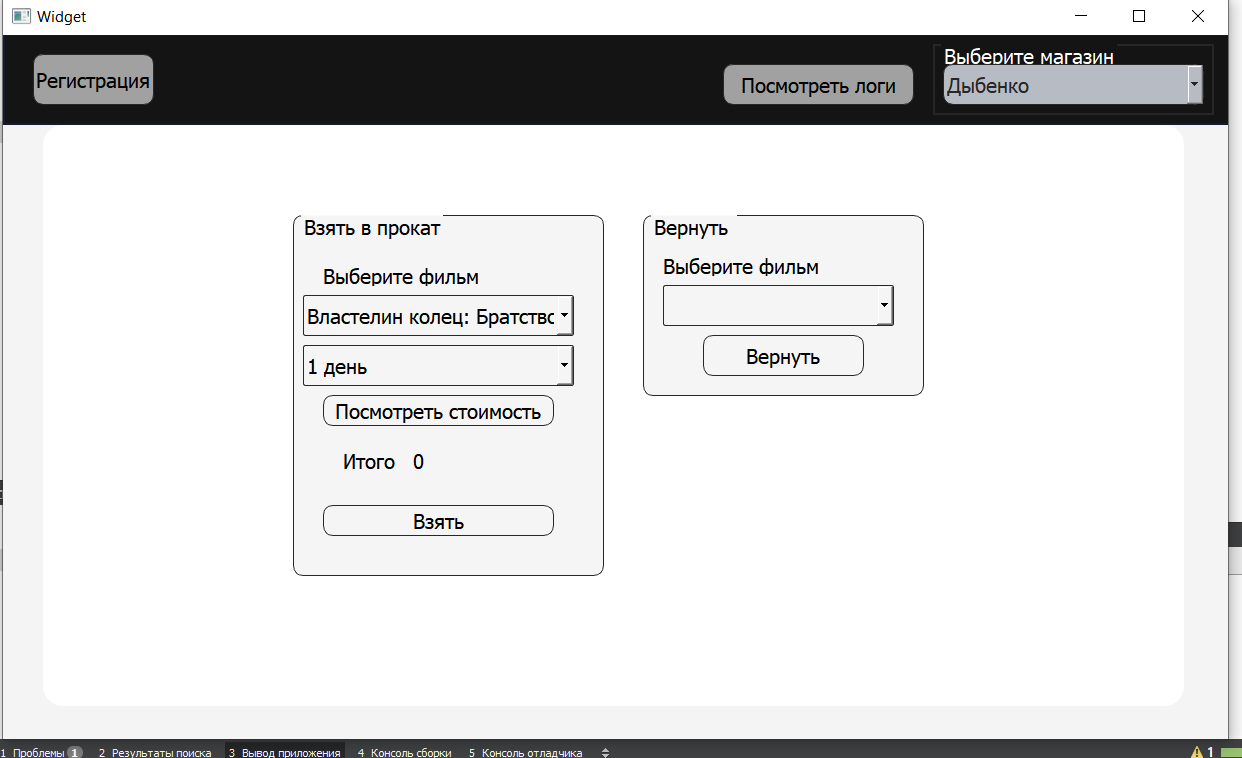
info\* find(std::string name, int sub\_id); - поиск операции по имени и id

void return\_film(std::string name, subscriber\* sub); -операция возврата фильма

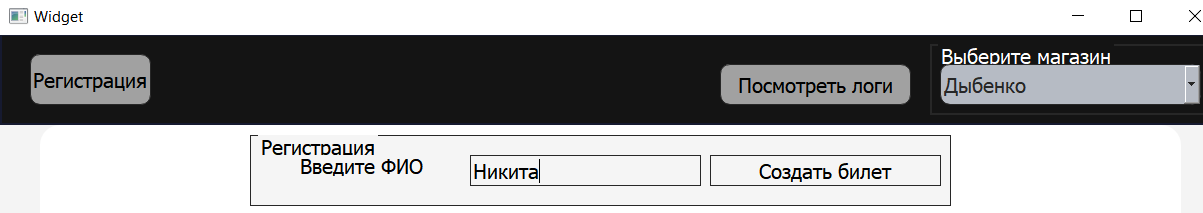
std::string\* get\_all\_from\_sub(subscriber\* sub); - показать все операции

# Тестирование программы

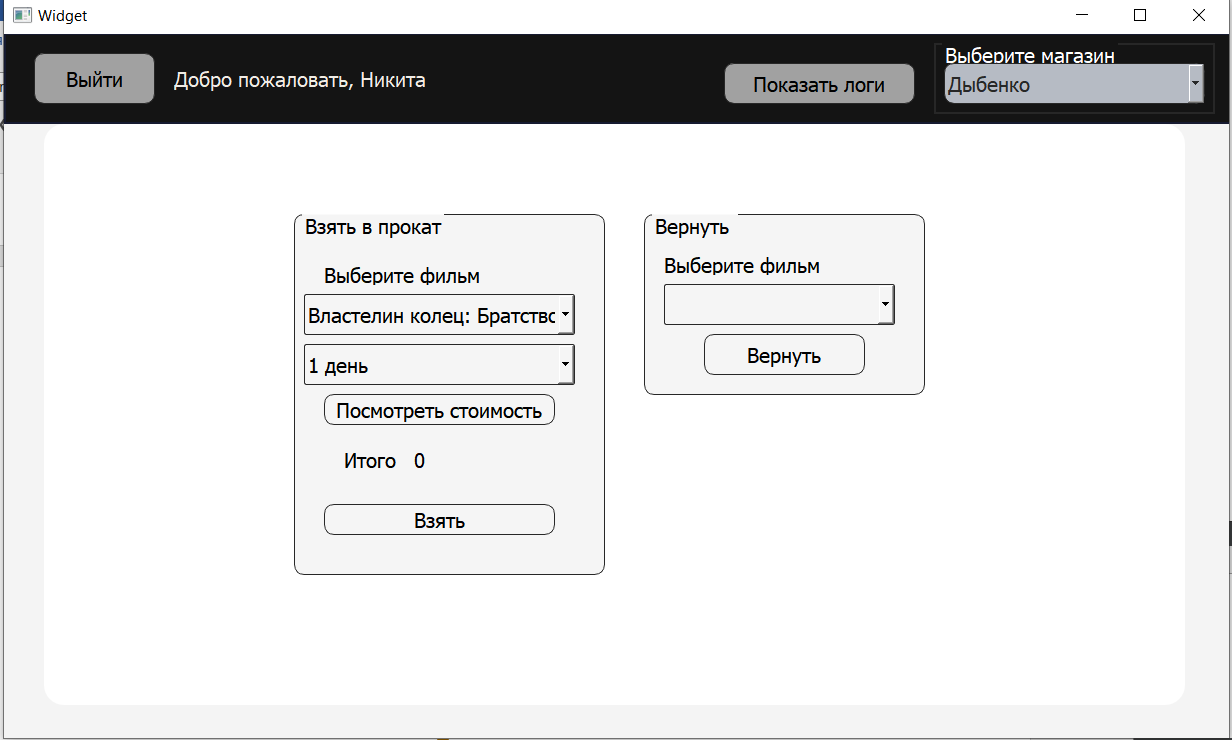
Главное меню

****

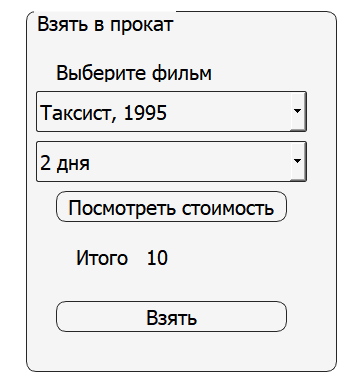
Регистрация в системе



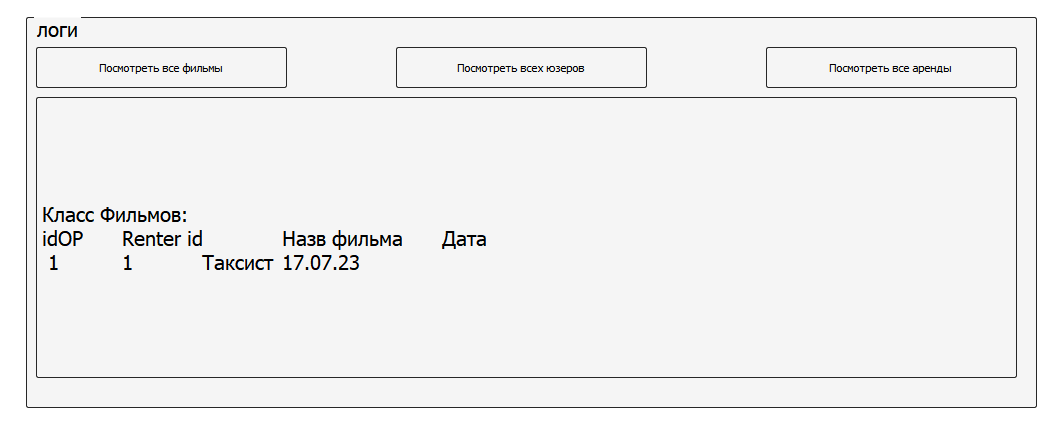
Когда мы авторизовались, мы получаем доступ к программе



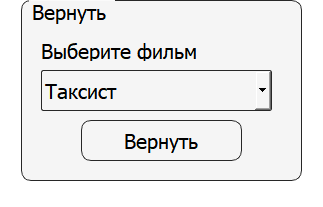
Выбираю себе фильм



В логах появилась запись

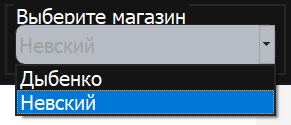


Также фильм можно вернуть

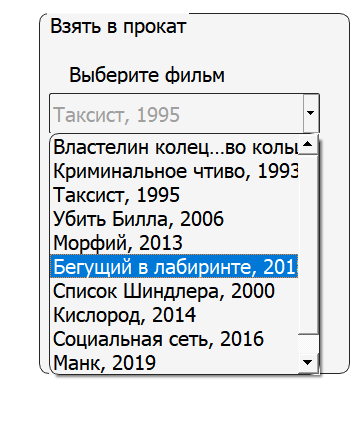
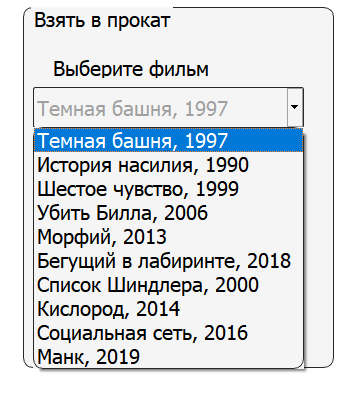


В разных магазинах разные фильмы:

Выбираем интересующий



И смотрим на каталог

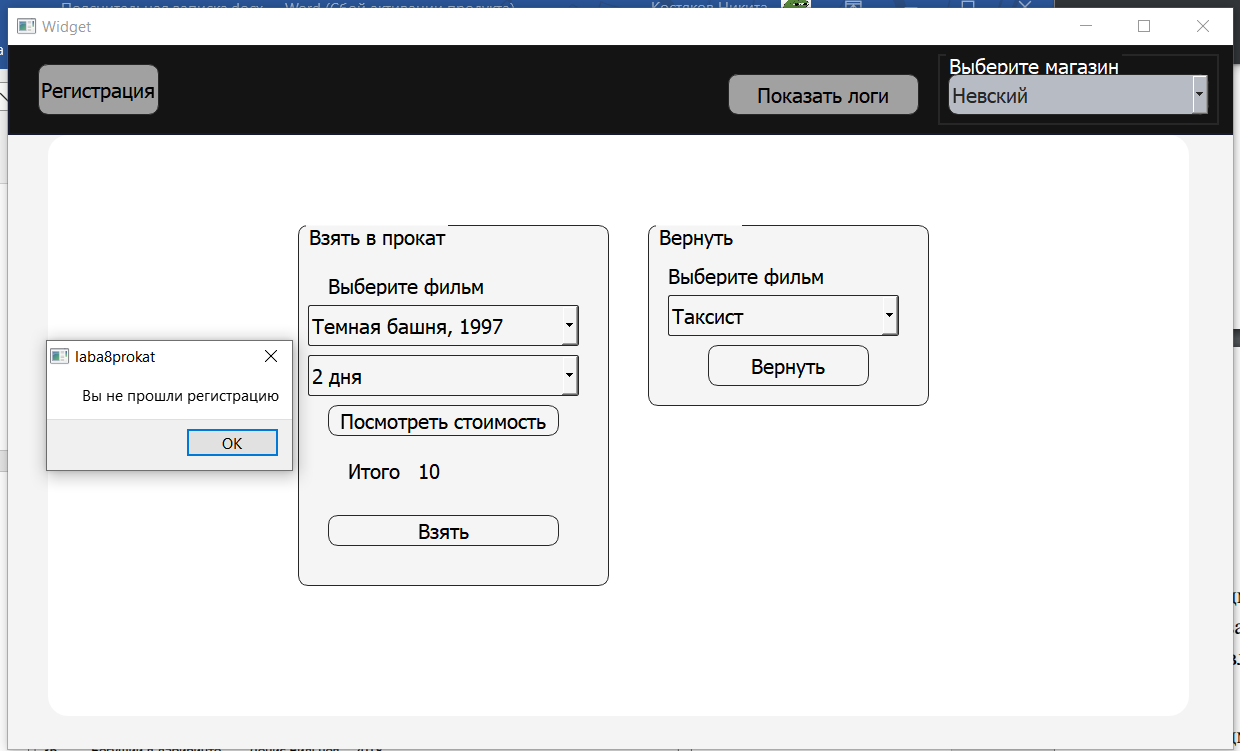
 

В логах тоже можно посмотреть на информацию





Если неавторизованный пользователь попробует что-то сделать, он получит сообщение



# Приложение

Листинг программы

dailyrentalstrategy.h

#ifndef DAILYRENTALSTRATEGY\_H

#define DAILYRENTALSTRATEGY\_H

#include *"rentalstrategy.h"*

**class** **DailyRentalStrategy** : **public** IRentalStrategy {

**public**:

double CalculateRentalCost(int rentalTime) **override** {

**return** rentalTime \* 5.0;

}

};

#endif *// DAILYRENTALSTRATEGY\_H*

dailyrentalstrategyh.cpp

#include *"dailyrentalstrategyh.h"*

dailyrentalstrategyh::dailyrentalstrategyh()

{

}

main.cpp

#include *"widget.h"*

#include *<QApplication>*

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

Widget w;

w.show();

**return** a.exec();

}

movies.h

#ifndef MOVIES\_H

#define MOVIES\_H

#include *<string>*

**class** **movie**{

**public**:

int number;

int id, year;

std::string author, name;

bool state; *//0 - на складе, 1 - выдан*

movie\* next= **nullptr**;

movie(int num=0, int i=0, int y=0, std::string auth=0, std::string n=0, bool stat =1){number=num;id=i; year = y; author = auth; name = n; state = stat;}

};

**class** **movies**{

**private**:

int count;

movie\* head = **nullptr**;

**public**:

void add(int y, std::string auth, std::string n);

movie\* get(int index);

int get\_length(){**return** count;}

movies(){

}

};

movie\* movies::get(int index){

**if**(count<index) **return** **nullptr**;

movie\* current = head;

int i = 0;

**while** (i<index){

current=current->next;

i++;

}

**return** current;

}

void movies::add(int y, std::string auth, std::string n){

int number = get\_length();

int id =1+ number\*7;

movie\* sub = **new** movie(number, id, y, auth, n ,1);

**if**(head==**nullptr**){

head = sub;

count++;

**return**;

}

movie\* current = head;

**while**(current->next!=**nullptr**){

current = current->next;

}

current->next = sub;

count++;

**return**;

}

#endif *// MOVIES\_H*

rental.h

#ifndef RENTAL\_H

#define RENTAL\_H

#include *"rentalstrategy.h"*

#include *"weeklystrategy.h"*

#include *"dailyrentalstrategy.h"*

**class** **Rental** {

**private**:

IRentalStrategy\* \_rentalStrategy;

**public**:

Rental(IRentalStrategy\* rentalStrategy) : \_rentalStrategy(rentalStrategy) {}

double CalculateRentalCost(int rentalTime) {

**return** \_rentalStrategy->CalculateRentalCost(rentalTime);

}

};

*//int main() {*

*//auto dailyRental = new Rental(new DailyRentalStrategy());*

*//std::cout << "Daily rental cost for 3 days: " << dailyRental->CalculateRentalCost(3) << std::endl;*

*// auto weeklyRental = new Rental(new WeeklyRentalStrategy());*

*// std::cout << "Weekly rental cost for 2 weeks: " << weeklyRental->CalculateRentalCost(2) << std::endl;*

*// delete dailyRental;*

*// delete weeklyRental;*

*// return 0;*

*//}*

#endif *// RENTAL\_H*

rentalmovieadapter.h

#ifndef RENTALMOVIEADAPTER\_H

#define RENTALMOVIEADAPTER\_H

#include *"rentalmovies.h"*

#include *"movies.h"*

#include *"subscriber.h"*

**class** **RentalMoviesAdapter** : **public** RentalMovies {

**private**:

movie movieInfo;

subscriber customerInfo;

**public**:

RentalMoviesAdapter(movie movieInfo, subscriber customerInfo) {

**this**->movieInfo = movieInfo;

**this**->customerInfo = customerInfo;

}

void rentMovie(string movieName) **override** {

}

void returnMovie(string movieName) **override** {

cout << "The movie " << movieName << " has been returned by " << customerInfo.name << "." << endl;

}

};

#endif *// RENTALMOVIEADAPTER\_H*

rentalmovies.h

#ifndef ARENDA\_H

#define ARENDA\_H

#include *<string>*

#include *"movies.h"*

#include *"subscriber.h"*

#include *<iostream>*

**class** **info**{

**public**:

int renterId;

int filmId;

int OperId;

std::string date\_take;

std::string date\_return;

std::string film\_name;

info\* next = **nullptr**;

info(movie\* mov, subscriber\* sub){

renterId = sub->id;

filmId = mov->id;

film\_name = mov->name;

date\_take = "16.07.23";

date\_return = "17.07.23";

}

};

**class** **RentalMovies** {

**private**:

int count;

**public**:

info\* head= **nullptr**;

int get\_length(){**return** count;}

std::string show();

void add(info \*input);

info\* find(std::string name, int sub\_id);

void return\_film(std::string name, subscriber\* sub);

std::string\* get\_all\_from\_sub(subscriber\* sub);

info\* get(int index);

};

info\* RentalMovies::get(int index){

**if**(count<index) **return** **nullptr**;

info\* current = head;

int i = 0;

**while** (i<index){

current=current->next;

i++;

}

**return** current;

}

void RentalMovies::return\_film(std::string name, subscriber\* sub){

info\* to\_remove = find(name, sub->id);

info\* current = head;

**if**(current == to\_remove){

head = head->next;

count--;

**return**;

}

**while**(current->next!= to\_remove){

current = current->next;

}

current->next=current->next->next;

count--;

}

info\* RentalMovies::find(std::string name, int sub\_id){

info\* current = head;

**while** (current!=**nullptr**){

**if**(current->film\_name==name && current->renterId == sub\_id) **return** current;

current= current->next;

}

**return** **nullptr**;

}

std::string RentalMovies::show(){

std::string res = "";

info\* current = head;

**while**(current!=**nullptr**){

res += std::to\_string(current->OperId)+": "+" "+current->film\_name+", ";

current = current->next;

}

**return** res;

}

void RentalMovies::add(info\* input){

**if**(!input) **return**;

int number = get\_length();

int id = 1+number\*21;

input->OperId= id;

info\* sub = input;

**if**(head==**nullptr**){

head = sub;

count++;

**return**;

}

info\* current = head;

**while**(current->next!=**nullptr**){

current = current->next;

}

current->next = sub;

count++;

**return**;

}

#endif

rentalstrategy.h

#ifndef RENTALSTRATEGY\_H

#define RENTALSTRATEGY\_H

**class** **IRentalStrategy** {

**public**:

**virtual** double CalculateRentalCost(int rentalTime) = 0;

};

#endif *// RENTALSTRATEGY\_H*

shop.cpp

#include *"shop.h"*

#include *"movies.h"*

shop::shop()

{

**public**:

movies catalog;

}

shop.h

#ifndef SHOP\_H

#define SHOP\_H

#include *"string.h"*

**class** **shop**

{

**public**:

movies catalog;

std::string name;

void set\_name(std::string name){

**this**->name = name;

}

shop(std::string name){

set\_name(name);

}

std::string get\_name(){

**return** name;

}

};

#endif *// SHOP\_H*

subscriber.h

#ifndef SUBSCRIBER\_H

#define SUBSCRIBER\_H

#include *<string>*

**class** **subscriber**{

**public**:

int number;

int id;

std::string fio;

subscriber\* next = **nullptr**;

subscriber(std::string fio, int number, int id){

**this**->fio = fio;

**this**->id = id;

**this**->number = number;

}

};

**class** **sub\_table**{

**private**:

subscriber\* head = **nullptr**;

int count = 0;

**public**:

int get\_length();

std::string show();

void registers(std::string name);

subscriber\* find(std::string name);

subscriber\* get(int index);

};

subscriber\* sub\_table::get(int index){

subscriber\* current = head;

**for**(int i = 0; i<index; i++){

current=current->next;

}

**return** current;

}

std::string sub\_table::show(){

std::string res= "";

subscriber\* current = head;

**while**(current!=**nullptr**){

res = res + current->fio+", ";

current = current->next;

}

**return** res;

}

subscriber\* sub\_table::find(std::string name){

**if**(name == "")**return** **nullptr**;

subscriber\* current = head;

**while**(current!=**nullptr**){

**if** (current->fio == name){

**return** current;

}

current = current->next;

}

**return** **nullptr**;

}

void sub\_table::registers(std::string name){

int number = get\_length();

int id = 1+ number\*13;

subscriber\* sub = **new** subscriber(name, number, id);

**if**(head==**nullptr**){

head = sub;

count++;

**return**;

}

subscriber\* current = head;

**while**(current->next!=**nullptr**){

**if** (current->fio==name){

**return**;

}

current = current->next;

}

**if** (current->fio==name){

**return**;

}

current->next = sub;

count++;

**return**;

}

int sub\_table::get\_length(){

**return** count;

}

#endif *// SUBSCRIBER\_H*

weeklystrategy.h

#ifndef WEEKLYRENTALSTRATEGY\_H

#define WEEKLYRENTALSTRATEGY\_H

#include *"rentalstrategy.h"*

**class** **WeeklyRentalStrategy** : **public** IRentalStrategy {

**public**:

double CalculateRentalCost(int rentalTime) **override** {

**return** rentalTime \* 20.0;

}

};

#endif *// WEEKLYRENTALSTRATEGY\_H*

widget.cpp

#include *"widget.h"*

#include *"ui\_widget.h"*

#include *"subscriber.h"*

#include *"movies.h"*

#include *"rental.h"*

#include *"rentalmovies.h"*

#include *"shop.h"*

#include *"string.h"*

#include *<QMessageBox>*

sub\_table subs;

RentalMovies RM;

shop Dibenko("Дыбенко");

shop Nevskiy("Невский");

std::string cur\_user="";

void update\_films(Ui::Widget \*ui){

ui->comboBox\_film->clear();

shop \* current;

**if**(ui->comboBox\_\_shop\_select->currentIndex() == 0){

current =&Dibenko;

}

**else**{

current =&Nevskiy;

}

**for**(int i = 0; i< current->catalog.get\_length(); i++){

std::string name = current->catalog.get(i)->name + ", " + std::to\_string(current->catalog.get(i)->year);

ui->comboBox\_film->addItem(QString::fromStdString(name));

}

}

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Widget)

{

ui->setupUi(**this**);

ui->comboBox\_\_shop\_select->addItem("Дыбенко");

ui->comboBox\_\_shop\_select->addItem("Невский");

ui->comboBox\_period->addItem("1 день");

ui->comboBox\_period->addItem("2 дня");

ui->comboBox\_period->addItem("1 неделя");

Dibenko.catalog.add(1993, "Питер Джексон", "Властелин колец: Братство кольца" );

Dibenko.catalog.add(1993, "Квентин Тарантино", "Криминальное чтиво" );

Dibenko.catalog.add(1995, "Мартин Скорсезе", "Таксист" );

Dibenko.catalog.add(2006, "Квентин Тарантино", "Убить Билла");

Dibenko.catalog.add(2013, "Алексей Балабанов", "Морфий");

Dibenko.catalog.add(2018, "Денис Вильнев", "Бегущий в лабиринте");

Dibenko.catalog.add(2000, "Стивен Спилберг", "Список Шиндлера");

Dibenko.catalog.add(2014, "Алексей Учитель", "Кислород");

Dibenko.catalog.add(2016, "Дэвид Финчер", "Социальная сеть");

Dibenko.catalog.add(2019, "Дэвид Финчер", "Манк");

Dibenko.catalog.add(2005, "Фрэнк Маршалл", "Последний самурай");

Nevskiy.catalog.add(1997, "Фрэнк Дарабонт", "Темная башня" );

Nevskiy.catalog.add(1990, "Дэвид Кроненберг", "История насилия");

Nevskiy.catalog.add(1999,"Найт Шьямалан", "Шестое чувство" );

Nevskiy.catalog.add(2006, "Квентин Тарантино", "Убить Билла");

Nevskiy.catalog.add(2013, "Алексей Балабанов", "Морфий");

Nevskiy.catalog.add(2018, "Денис Вильнев", "Бегущий в лабиринте");

Nevskiy.catalog.add(2000, "Стивен Спилберг", "Список Шиндлера");

Nevskiy.catalog.add(2014, "Алексей Учитель", "Кислород");

Nevskiy.catalog.add(2016, "Дэвид Финчер", "Социальная сеть");

Nevskiy.catalog.add(2019, "Дэвид Финчер", "Манк");

ui->label\_welcome->setHidden(1);

ui->groupBox\_logs->setHidden(1);

ui->groupBox\_registration->setHidden(1);

ui->pushButton\_quit->setHidden(1);

update\_films(ui);

}

Widget::~Widget()

{

**delete** ui;

}

void Widget::on\_pushButton\_check\_cost\_clicked()

{

double cost;

**auto** rent = **new** Rental(**new** DailyRentalStrategy());

int period = ui->comboBox\_period->currentIndex();

**switch** (period) {

**case** 0:

rent = **new** Rental(**new** DailyRentalStrategy());

cost = rent->CalculateRentalCost(1);

**break**;

**case** 1:

rent = **new** Rental(**new** DailyRentalStrategy());

cost = rent->CalculateRentalCost(2);

**break**;

**case** 2:

rent = **new** Rental(**new** WeeklyRentalStrategy());

cost = rent->CalculateRentalCost(1);

**break**;

**default**:

**break**;

}

ui->label\_cost->setText(QString::number(cost));

}

void Widget::on\_pushButton\_take\_clicked()

{

shop \* current;

**if**(ui->comboBox\_\_shop\_select->currentIndex() == 0){

current =&Dibenko;

}

**else**{

current =&Nevskiy;

}

std::string name =cur\_user;

**if**(name=="") {

QMessageBox msgBox;

msgBox.setText("Вы не прошли регистрацию");

msgBox.exec();

**return**;

}

subscriber\* sub = subs.find(name);

movie\* mov = current->catalog.get(ui->comboBox\_film->currentIndex());

info \* input = **new** info(mov, sub);

RM.add(input);

ui->comboBox\_film\_return->clear();

sub = subs.find(name);

**if** (sub==**nullptr**) **return**;

info\* cur = RM.head;

std::string res ;

**while** (cur){

**if**(cur->renterId == sub->id)

{res= cur->film\_name;

ui->comboBox\_film\_return->addItem(QString::fromStdString(res));}

cur = cur->next;

}

*// std::string addition = mov->name;*

*// ui->comboBox\_film\_return->addItem(QString::fromStdString(addition));*

}

void Widget::on\_pushButton\_return\_clicked()

{

std::string name = cur\_user;

**if**(name=="") {

QMessageBox msgBox;

msgBox.setText("Вы не прошли регистрацию");

msgBox.exec();

**return**;

}

subscriber\* sub = subs.find(name);

std::string mov = ui->comboBox\_film\_return->currentText().toStdString();

RM.return\_film(mov, sub);

info\* cur = RM.head;

std::string res ;

ui->comboBox\_film\_return->clear();

**while** (cur){

**if**(cur->renterId == sub->id)

{res= cur->film\_name;

ui->comboBox\_film\_return->addItem(QString::fromStdString(res));}

cur = cur->next;

}

*// ui->comboBox\_film\_return->removeItem(0);*

}

void Widget::on\_pushButton\_check\_clicked()

{

}

void Widget::on\_pushButton\_show\_films\_clicked()

{

ui->label\_logs->clear();

std::string outp = "Класс Фильмов:**\n**id**\t**Название**\t**Автор**\t\t**год";

shop \* current;

**if**(ui->comboBox\_\_shop\_select->currentIndex() == 0){

current =&Dibenko;

}

**else**{

current =&Nevskiy;

}

**for**(int i = 0; i<current->catalog.get\_length();i++){

outp+="**\n** "+std::to\_string(current->catalog.get(i)->id)+"**\t**"+current->catalog.get(i)->name+"**\t**"+current->catalog.get(i)->author+"**\t**"+std::to\_string(current->catalog.get(i)->year);

}

ui->label\_logs->setText(QString::fromStdString(outp));

}

void Widget::on\_pushButton\_show\_films\_2\_clicked()

{

ui->label\_logs->clear();

std::string outp = "Класс юзеров:**\n**id**\t**ФИО";

**for**(int i = 0; i<subs.get\_length();i++){

outp+="**\n** "+std::to\_string(subs.get(i)->id)+"**\t**"+subs.get(i)->fio;

}

ui->label\_logs->setText(QString::fromStdString(outp));

}

void Widget::on\_pushButton\_show\_films\_3\_clicked()

{

ui->label\_logs->clear();

std::string outp = "Класс Фильмов:**\n**idOP**\t**Renter id**\t**Назв фильма**\t**Дата";

**for**(int i = 0; i<RM.get\_length();i++){

outp+="**\n** "+std::to\_string(RM.get(i)->OperId)+"**\t**"+std::to\_string(RM.get(i)->renterId)+"**\t**"+RM.get(i)->film\_name+"**\t**"+RM.get(i)->date\_return;

}

ui->label\_logs->setText(QString::fromStdString(outp));

}

void Widget::on\_pushButton\_clicked()

{

ui->groupBox\_registration->setHidden(0);

}

void Widget::on\_pushButton\_registrate\_clicked()

{

std::string name = ui->lineEdit\_registration->text().toStdString();

subs.registers(name);

ui->groupBox\_registration->setHidden(1);

ui->label\_welcome->setText(QString::fromStdString("Добро пожаловать, "+name));

ui->label\_welcome->setHidden(0);

ui->pushButton->setHidden(1);

cur\_user = name;

ui->pushButton\_quit->setHidden(0);

}

void Widget::on\_pushButton\_2\_clicked()

{

**if** (ui->groupBox\_logs->isHidden()){

ui->groupBox\_logs->setHidden(0);

ui->groupBox\_ArendaTake->setHidden(1);

ui->groupBox\_ArendaReturn->setHidden(1);

ui->pushButton\_2->setText(QString::fromStdString("Скрыть логи"));

}

**else**{

ui->groupBox\_logs->setHidden(1);

ui->pushButton\_2->setText(QString::fromStdString("Показать логи"));

ui->groupBox\_ArendaTake->setHidden(0);

ui->groupBox\_ArendaReturn->setHidden(0);

}

}

void Widget::on\_comboBox\_\_shop\_select\_currentIndexChanged(int index)

{

update\_films(ui);

}

void Widget::on\_pushButton\_quit\_clicked()

{

cur\_user="";

ui->pushButton\_quit->setHidden(1);

ui->pushButton->setHidden(0);

ui->label\_welcome->setHidden(1);

}

widget.h

#ifndef WIDGET\_H

#define WIDGET\_H

#include *<QWidget>*

#include *<string>*

**namespace** **Ui** {

**class** **Widget**;

}

**class** **Widget** : **public** QWidget

{

Q\_OBJECT

**public**:

**explicit** Widget(QWidget \*parent = 0);

~Widget();

**private** slots:

void on\_pushButton\_registrate\_clicked();

void on\_pushButton\_check\_cost\_clicked();

void on\_pushButton\_take\_clicked();

void on\_pushButton\_return\_clicked();

void on\_pushButton\_check\_clicked();

void on\_pushButton\_show\_films\_clicked();

void on\_pushButton\_show\_films\_2\_clicked();

void on\_pushButton\_show\_films\_3\_clicked();

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_comboBox\_\_shop\_select\_currentIndexChanged(int index);

void on\_pushButton\_quit\_clicked();

**private**:

Ui::Widget \*ui;

};

#endif *// WIDGET\_H*