Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Билетная касса»**

**Выполнил**:

студент группы 381803-1

Стребкова К.О.

**Проверил**:

доцент кафедры МОСТ, к.т.н.,

Сысоев А.В.

Нижний Новгород

2020

Содержание

[Введение 3](#_Toc39668063)

[1. Постановка задачи 4](#_Toc39668064)

[2. Руководство пользователя 5](#_Toc39668065)

[3. Руководство программиста 6](#_Toc39668066)

[3.1. Описание структуры программы 6](#_Toc39668067)

[3.2. Описание алгоритмов 8](#_Toc39668068)

[Заключение 9](#_Toc39668069)

[Литература 10](#_Toc39668070)

[Приложение 11](#_Toc39668071)

# Введение

Одним из самых распространенных мест для развлечения являются кинотеатры. Каждый человек хоть раз бывал там и покупал билет на сеанс. В данной лабораторной работе представлен вариант реализации работы системы покупки билета.

# Постановка задачи

Требуется написать связанную систему классов, реализующую работу билетной кассы.

Обязательные требования:

Класс Билетная касса должен имитировать работу кассы по продаже билетов на киносеансы в многозальном кинотеатре. Считать, что продажа билетов проводится на сеансы в пределах трех дней от текущей даты. Каждый сеанс описывается датой, временем начала сеанса, названием фильма, номером зала, стоимостью билета в зависимости от зоны (VIP и обычная). Для упрощения считать, что покупатель указывает тип зоны и требуемое число билетов, а места выделяются кассой автоматически (при наличии свободных). Зрительные места в каждом зале описываются номером ряда и номером в ряду. Для упрощения считать, что число мест во всех рядах в одном зале одинаково. Продажа билетов на сеанс прекращается через 10 минут после начала сеанса.

Информация о всех сеансах на ближайшие 30 дней проката хранится в классе Кинотеатр. Для каждого зала установлена базовая стоимость билетов (на дневные сеансы – от 12.00 до 18.00). Стоимость билетов на утренние сеансы (до 12.00) составляет 75% от базовой, стоимость билетов на вечерние сеансы (после 18.00) – 150% от базовой. Информация о зрительных местах (свободно/занято) в каждом зале на каждом сеансе также хранится в классе Кинотеатр.

Класс Билетная касса должен предоставлять следующие операции: 1) принять данные покупателя: дату, время сеанса, название фильма, номер зала, тип зоны, число мест, 2) проверить наличие требуемого количества свободных мест в требуемой зоне, 3) зарезервировать требуемое количество мест, 4) рассчитать общую стоимость билетов, 5) отменить заказ билетов, 6) сформировать билеты (каждый билет включает: дату, время сеанса, название фильма, номер зала, номер ряда, номер места в ряду).

# Руководство пользователя

В данной программе пользовательский интерфейс не разрабатывался.

# Руководство программиста

## Описание структуры программы

В программе реализована одна функция Compare\_Time, которая сравнивает две временные структуры tm и возвращает true, если они равны, false – не равны.

Программа содержит в себе 6 классов: Place (описывает место в зале), Hall (описывает зал), Seance (описывает сеанс), Cinema (содержит информацию о залах и сеансах), Order (описывает заказ в билетной кассе), TicketOffice (реализует работу билетной кассы).

Разберем подробнее структуру каждого класса.

1. Place

Поля класса: переменная row типа int (хранит номер ряда), переменная col типа int (хранит номер места).

1. Hall

Поля класса: переменная number\_of\_hall типа int (хранит номер зала), переменная vip\_price типа int (хранит vip стоимость билета), переменная base\_price типа int (хранит обычную стоимость билета), переменная cols типа int (хранит количество места в ряду), переменная rows типа int (хранит количество рядов).

1. Seance

Поля класса: временная структура tm time (хранит дату и время сеанса), переменная name\_of\_film типа string (хранит название фильма), объект класса Hall hall (хранит информацию о зале), вектор объектов класса Place unavailable\_places (хранит занятые места).

Методы:

* Find\_Number\_Of\_Place

Результат: значение типа int, равное номеру места в списке недоступных к покупке

* Add\_Unavailible\_Place

Результат: добавляет место в недоступные к покупке

* Delete\_Unavailible\_Place

Результат: удаляет место из недоступных к покупке

* Show\_Seance()

Результат: выводит на экран информацию о сеансе

1. Cinema

Поля класса: вектор объектов класса Seance seances (хранит информацию о сеансах), вектор объектов класса Hall halls (хранит информацию о залах).

Методы:

* Add\_Seance

Результат: добавляет сеанс в кинотеатр

* Add\_Hall

Результат: добавляет зал в кинотеатр

* Find\_Number\_Of\_Seance

Результат: возвращает номер сеанса в списке сеансов кинотеатра

* Add\_Unavailible\_Place

Результат: добавляет место к забронированным на определенный сеанс

* Delete\_Unavailible\_Place

Результат: удаляет место из забронированных на определенный сеанс

1. Order

Поля класса: объект класса Seance seance (хранит информацию о сеансе, на который осуществляется заказ), вектор объектов класса Place places (хранит бронируемые места), переменная type типа bool (хранит тип зоны: true – vip, false – обычная).

Методы:

* Get\_Count\_Of\_Places

Результат: возвращает значение типа int, равное количеству требуемых покупателем мест

* Show\_Order

Результат: выводит на экран информацию о заказе

1. TicketOffice

Основной класс программы.

Поля класса: объект класса Cinema cinema (хранит информацию кинотеатра).

Методы:

* Accept\_Data

Результат: принимает данные покупателя, возвращает объект класса Order с информацией о заказе

* Check\_Free\_Place

Результат: возвращает значение типа bool, true – места свободны, false – места заняты

* Reserve\_Places

Результат: резервирует места, указанные в заказе, если это возможно

* Calculate\_Sum

Результат: возвращает значение типа double, равное сумме заказа

* Cancel\_Place

Результат: отменяет бронирование мест, указанных в заказе

* Show\_Ticket

Результат: выводит билеты на экран

Также в каждом классе реализованы конструктор, конструктор копирования, оператор присваивания, деструктор.

В классах Place, Hall и Seance перегружены операторы сравнения.

## Описание алгоритмов

В данной программе реализовано взаимодействие шести классов с помощью агрегаций и публичных методов.

Cinema

Seance

Hall

Order

Place

TicketOffice

Схема 1. Система агрегаций

# Заключение

Поставленная задача была выполнена. Реализована программа, позволяющая имитировать работу билетной кассы, минимальные требования к классам также были соблюдены.

# Литература

1. Бьерн Страуструп. Язык программирования С++

# Приложение

Код программы:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <vector>

#include <ctime>

using std::cout;

using std::cin;

using std::endl;

using std::string;

using std::vector;

bool Compare\_Time(tm time, tm \_time)

{

if ((time.tm\_year == \_time.tm\_year) && (time.tm\_mon == \_time.tm\_mon) && (time.tm\_mday == \_time.tm\_mday) && (time.tm\_hour == \_time.tm\_hour) && (time.tm\_min == \_time.tm\_min))

return true;

else

return false;

}

class Place

{

private:

int row;

int col;

public:

Place(int \_row, int \_col):row(\_row), col(\_col) {}

~Place() {}

int Get\_Col()

{

return col;

}

int Get\_Row()

{

return row;

}

Place(const Place& place)

{

col = place.col;

row = place.row;

}

Place& operator=(const Place& place)

{

col = place.col;

row = place.row;

return \*this;

}

bool operator==(const Place &place)

{

if ((row == place.row) && (col == place.col))

return true;

else

return false;

}

};

class Hall

{

private:

int number\_of\_hall;

int vip\_price;

int base\_price;

int cols;

int rows;

public:

Hall(int \_number\_of\_hall = 0, int \_vip\_price = 0, int \_base\_price = 0, int \_cols = 0, int \_rows = 0):number\_of\_hall(\_number\_of\_hall), vip\_price(\_vip\_price), base\_price(\_base\_price),

cols(\_cols), rows(rows)

{}

~Hall() {}

Hall (const Hall& hall)

{

number\_of\_hall = hall.number\_of\_hall;

vip\_price = hall.vip\_price;

base\_price = hall.base\_price;

cols = hall.cols;

rows = hall.rows;

}

Hall& operator=(const Hall &hall)

{

number\_of\_hall = hall.number\_of\_hall;

vip\_price = hall.vip\_price;

base\_price = hall.base\_price;

cols = hall.cols;

rows = hall.rows;

return \*this;

}

bool operator== (const Hall &hall)

{

if (number\_of\_hall == hall.number\_of\_hall)

return true;

else

return false;

}

int Get\_Number\_Of\_Hall()

{

return number\_of\_hall;

}

int Get\_Vip\_Price()

{

return vip\_price;

}

int Get\_Base\_Price()

{

return base\_price;

}

int Get\_Cols()

{

return cols;

}

int Get\_Rows()

{

return rows;

}

};

class Seance

{

private:

tm time;

string name\_of\_film;

Hall hall;

vector <Place> unavailible\_place;

public:

Seance(tm \_time = tm(), string \_name\_of\_film = "", Hall \_hall = Hall(), vector <Place> \_unavailible\_place = {}) :

time(\_time), name\_of\_film(\_name\_of\_film), hall(\_hall), unavailible\_place(\_unavailible\_place) {}

~Seance() {}

Seance(const Seance &seance)

{

time = seance.time;

name\_of\_film = seance.name\_of\_film;

hall = seance.hall;

unavailible\_place = seance.unavailible\_place;

}

Seance& operator= (const Seance &seance)

{

time = seance.time;

name\_of\_film = seance.name\_of\_film;

hall = seance.hall;

unavailible\_place = seance.unavailible\_place;

return \*this;

}

bool operator== (const Seance &seance)

{

if ((Compare\_Time(time, seance.time) == true) && (name\_of\_film == seance.name\_of\_film) && (hall == seance.hall))

return true;

else

return false;

}

tm Get\_Time()

{

return time;

}

string Get\_Name\_Of\_Film()

{

return name\_of\_film;

}

Hall Get\_Hall()

{

return hall;

}

vector <Place> Get\_Unavailible\_Place()

{

return unavailible\_place;

}

int Find\_Number\_Of\_Place(const Place &place)

{

for (int i = 0; i < unavailible\_place.size(); i++)

{

if (unavailible\_place[i] == place)

return i;

}

}

void Add\_Unavailible\_Place(const Place &place)

{

unavailible\_place.push\_back(place);

}

void Delete\_Unavailible\_Place(const Place &place)

{

int i = Find\_Number\_Of\_Place(place);

unavailible\_place.erase(unavailible\_place.begin() + i);

}

void Show\_Seance()

{

cout << "Дата: " << time.tm\_mday << "." << time.tm\_mon << "." << time.tm\_year << endl << "Время: " << time.tm\_hour << ":" << time.tm\_min << endl << "Название фильма:" << name\_of\_film << endl;

cout << "Номер зала: " << hall.Get\_Number\_Of\_Hall() << endl;

}

};

class Cinema

{

private:

vector <Seance> seances;

vector <Hall> halls;

public:

Cinema() {}

~Cinema() {}

void Add\_Seance(Seance& seance)

{

seances.push\_back(seance);

}

void Add\_Hall(Hall& hall)

{

halls.push\_back(hall);

}

vector<Seance> Get\_Seances()

{

return seances;

}

int Find\_Number\_Of\_Seance(Seance &seance)

{

for (int i = 0; i < seances.size(); i++)

if (seance == seances[i])

return i;

}

void Add\_Unavailible\_Place(const Place &place, int j)

{

seances[j].Add\_Unavailible\_Place(place);

}

void Delete\_Unavailible\_Place(const Place &place, int j)

{

seances[j].Delete\_Unavailible\_Place(place);

}

};

class Order

{

private:

Seance seance;

vector <Place> places;

bool type;

public:

Order(Seance \_seance = Seance(), vector <Place> \_places = {}, bool \_type = false) : seance(\_seance), places(\_places), type(\_type) {}

~Order() {}

Order(Order &order)

{

seance = order.seance;

places = order.places;

type = order.type;

}

Order& operator=(const Order &order)

{

seance = order.seance;

places = order.places;

type = order.type;

return \*this;

}

Seance Get\_Seance()

{

return seance;

}

bool Get\_Type()

{

return type;

}

vector <Place> Get\_Places()

{

return places;

}

int Get\_Count\_Of\_Places()

{

return places.size();

}

void Show\_Order()

{

for (int i = 0; i < places.size(); i++)

{

cout << "Билет" << endl;

seance.Show\_Seance();

cout << "Ряд: " << places[i].Get\_Row() << endl << "Место: " << places[i].Get\_Col() << endl;

}

}

};

class TicketOffice

{

private:

Cinema cinema;

public:

TicketOffice(Cinema \_cinema = Cinema()): cinema(\_cinema) {}

~TicketOffice() {}

TicketOffice(const TicketOffice &ticket\_office)

{

cinema = ticket\_office.cinema;

}

TicketOffice& operator=(const TicketOffice &ticket\_office)

{

cinema = ticket\_office.cinema;

return \*this;

}

Cinema Get\_Cinema()

{

return cinema;

}

Order Accept\_Data(int day, int month, int year, int hour, int min, string name\_of\_film, int number\_of\_hall, bool type, vector <Place> places)

{

vector <Seance> seances = cinema.Get\_Seances();

Hall hall;

tm time;

time.tm\_mday = day;

time.tm\_mon = month;

time.tm\_year = year;

time.tm\_min = min;

time.tm\_hour = hour;

int i;

for (i = 0; i < seances.size(); i++)

{

hall = seances[i].Get\_Hall();

if ((Compare\_Time(seances[i].Get\_Time(), time) == true) && (seances[i].Get\_Name\_Of\_Film() == name\_of\_film) && (hall.Get\_Number\_Of\_Hall() == number\_of\_hall))

break;

}

Order order = Order(seances[i], places, type);

return order;

}

bool Check\_Free\_Place(Order &order)

{

bool flag = true;

Seance seance = order.Get\_Seance();

vector <Place> places = order.Get\_Places();

Hall hall = seance.Get\_Hall();

for (int i = 0; i < places.size(); i++)

{

vector <Place> unavailible\_place = seance.Get\_Unavailible\_Place();

for (int j = 0; j < unavailible\_place.size(); j++)

{

if (places[i] == unavailible\_place[j])

{

flag = false;

break;

}

}

}

return flag;

}

void Reserve\_Places(Order &order)

{

vector <Place> places = order.Get\_Places();

Seance seance = order.Get\_Seance();

int j = cinema.Find\_Number\_Of\_Seance(seance);

time\_t now;

time(&now);

tm \_time = seance.Get\_Time();

\_time.tm\_year -= 1900;

\_time.tm\_mon -= 1;

\_time.tm\_sec = 0;

if (Check\_Free\_Place(order) == true)

{

if (difftime(now, mktime(&\_time)) < 600)

{

tm \*\_now = localtime(&now);

\_now->tm\_hour = 0;

\_now->tm\_min = 0;

\_now->tm\_sec = 0;

now = mktime(\_now);

\_time.tm\_hour = 0;

\_time.tm\_min = 0;

if (difftime(mktime(&\_time), now) < 345600)

for (int i = 0; i < places.size(); i++)

cinema.Add\_Unavailible\_Place(places[i], j);

else

cout << "Продажа билетов еще не началась!" << endl;

}

else

cout << "Продажа билетов закончилась!" << endl;

}

else

cout << "Невозможно зарезервировать билеты" << endl;

}

double Calculate\_Sum(Order &order)

{

Seance seance = order.Get\_Seance();

Hall hall = seance.Get\_Hall();

double price;

tm time = seance.Get\_Time();

bool type = order.Get\_Type();

if (type == true)

price = hall.Get\_Vip\_Price();

else

price = hall.Get\_Base\_Price();

float koef = 1;

if ( time.tm\_hour < 12)

koef = 0.75;

if (time.tm\_hour > 18)

koef = 1.5;

return price \* koef \* order.Get\_Count\_Of\_Places();

}

void Cancel\_Place(Order &order)

{

Seance seance = order.Get\_Seance();

vector <Place> places = order.Get\_Places();

int j = cinema.Find\_Number\_Of\_Seance(seance);

for (int i = 0; i < places.size(); i++)

cinema.Delete\_Unavailible\_Place(places[i], j);

}

void Show\_Ticket (Order &order)

{

order.Show\_Order();

}

};

void main()

{

setlocale(LC\_ALL, "Rus");

Cinema cinema;

Hall hall = Hall(1, 300, 150, 15, 20);

cinema.Add\_Hall(hall);

tm time;

time.tm\_year = 2020;

time.tm\_mon = 4;

time.tm\_mday = 30;

time.tm\_hour = 13;

time.tm\_min = 30;

Seance seance = Seance(time, "The Lion King", hall);

cinema.Add\_Seance(seance);

TicketOffice ticket\_office = TicketOffice(cinema);

Order order1 = ticket\_office.Accept\_Data(30, 4, 2020, 13, 30, "The Lion King", hall.Get\_Number\_Of\_Hall(), 0, { Place(15, 10) });

ticket\_office.Reserve\_Places(order1);

ticket\_office.Cancel\_Place(order1);

Order order2 = ticket\_office.Accept\_Data(30, 4, 2020, 13, 30, "The Lion King", hall.Get\_Number\_Of\_Hall(), 0, { Place(15, 10), Place(15, 9) });

if (ticket\_office.Check\_Free\_Place(order2))

cout << "Места свободны" << endl;

else

cout << "Места заняты" << endl;

cout << "Общая сумма: " << ticket\_office.Calculate\_Sum(order2) << endl;

ticket\_office.Reserve\_Places(order2);

ticket\_office.Show\_Ticket(order2);

system("pause");

}