Выполнил

студент КТбо1-4 \_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Воронов

Принял

ст. пр. каф. САПР, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Данильченко

Таганрог 2024

ИНДИВИДУАЛЬНАЯ РАБОТА №2

по дисциплине «Алгоритмизация и программирование»

**«Объектно-ориентированное программирование на языке С++»**

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

(ФГАОУ ВО «ЮФУ»)

1. **Цель индивидуальной работы:**

Для заданной предметной области разработать:

1) UML-диаграмму классов;

2) программный код заданных классов на С++ в среде MS Visual Studio;

3) программный код методов модификации объектов с контролем всех входных параметров;

4) программный код метода соответственно таблице индивидуальных заданий;

5) программу тестирования разработанных классов.

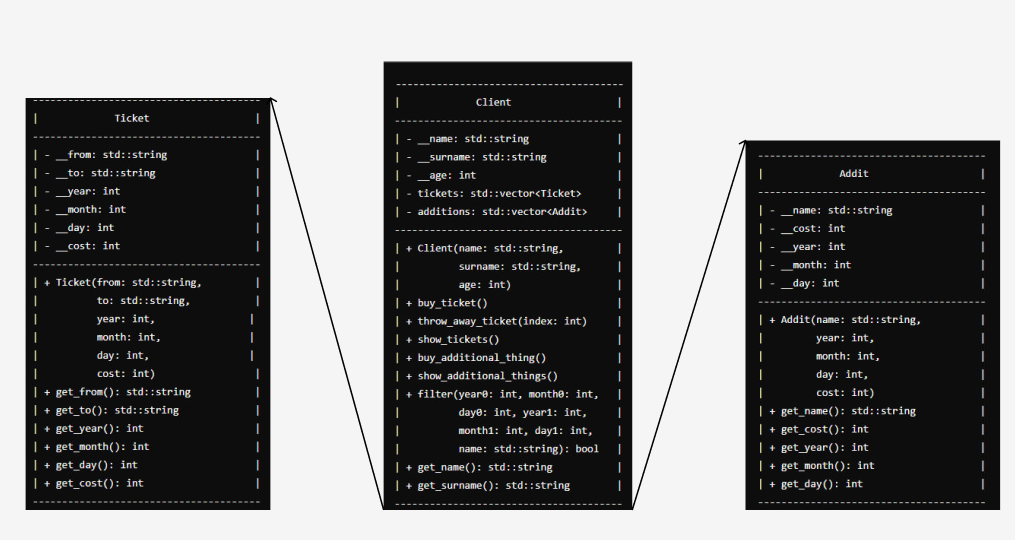
1. **Задача:** написать программу, в предметной области торговли авиабилетами, в которой созданы сущности клиента, заказа билета и дополнительных услуг. Должны быть реализованы методы контроля данных (вывод, добавление, удаление). Также нужно реализовать метод, который будет выводить список клиентов, по заданным услуге и интервалу времени. Создать код для тестирования программы.
2. **Ход работы:** я создал проект консольного приложения в Microsoft Visual Studio 2022. В нем реализовал несколько файлов: основной, с точкой входа main() — S2\_IndTask.cpp, и файлы с прототипами и реализацией трех классов: Client.cpp, Client.h — класс, описывающий клиента, Ticket.cpp, Ticket.h — класс, описывающий билет, Addit.cpp, Addit.h — класс, описывающий дополнительные услуги.

Класс Client: имеет методы buy\_ticket() — покупка билета, создание объекта класса Ticket и добавление в соответствующий контейнер vector, throw\_away\_ticket() — удаление билет по индексу, show\_tickets() — вывод информации о всех билетах, buy\_additional\_thing() — покупка дополнительной услуги, создание объекта класса Addit и добавление в соответствующий контейнер vector, show\_additional\_things() — вывод информации о всех дополнительных услугах, filter() — фильтр по интервалу времени и названию услуги, get\_name() и get\_surname() — геттеры приватных атрибутов \_\_name и \_\_surname.

Класс Ticket: имеет методы get\_from(), get\_to(), get\_year(), get\_month(), get\_day(), get\_cost() — геттеры всех одноименных приватных атрибутов класса.

Класс Addit: имеет методы get\_name(), get\_year(), get\_month(), get\_day(), get\_cost() — геттеры всех одноименных приватных атрибутов класса.

1. **UML-диаграмма классов:**

****

1. **Программный код всех классов:**
   1. **Client**

**Client.h**

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include "Ticket.h"

#include "Addit.h"

class Client

{

public:

Client(std::string name, std::string surname, int age);

void buy\_ticket(); //done

void throw\_away\_ticket(int index); //done

void show\_tickets(); //done

void buy\_additional\_thing();

void show\_additional\_things();

bool filter(int year0, int month0, int day0, int year1, int month1, int day1, std::string name);

std::string get\_name() { return \_\_name; }

std::string get\_surname() { return \_\_surname; }

private:

std::string \_\_name, \_\_surname;

int \_\_age;

std::vector<Ticket> tickets;

std::vector<Addit> additions;

};

**Client.cpp**

#include "Client.h"

Client::Client(std::string name, std::string surname, int age)

{

\_\_name = name;

\_\_surname = surname;

\_\_age = age;

}

void Client::buy\_ticket()

{

std::string from, to;

int year, month, day, cost;

std::cout << "Enter ticket info: from, to, year, month, day, cost: ";

std::cin >> from >> to >> year >> month >> day >> cost;

Ticket tckt(from, to, year, month, day, cost);

tickets.push\_back(tckt);

}

void Client::throw\_away\_ticket(int index)

{

tickets.erase(tickets.begin() + index);;

}

void Client::show\_tickets()

{

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

{

std::cout

<< "TICKET " << i << "\n"

<< "From: " << tickets[i].get\_from() << "\n"

<< "To: " << tickets[i].get\_to() << "\n"

<< "Year: " << tickets[i].get\_year() << "\n"

<< "Month: " << tickets[i].get\_month() << "\n"

<< "Day: " << tickets[i].get\_day() << "\n"

<< "Cost: " << tickets[i].get\_cost() << "\n\n";

}

}

void Client::buy\_additional\_thing()

{

std::string name;

int year, month, day, cost;

std::cout << "Enter thing info: name, year, month, day, cost: ";

std::cin >> name >> year >> month >> day >> cost;

Addit add(name, year, month, day, cost);

additions.push\_back(add);

}

void Client::show\_additional\_things()

{

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

{

std::cout

<< "ADDITIONAL THING " << i << "\n"

<< "Name: " << additions[i].get\_name() << "\n"

<< "Year: " << additions[i].get\_year() << "\n"

<< "Month: " << additions[i].get\_month() << "\n"

<< "Day: " << additions[i].get\_day() << "\n"

<< "Cost: " << additions[i].get\_cost() << "\n\n";

}

}

bool Client::filter(int year0, int month0, int day0, int year1, int month1, int day1, std::string name)

{

int time0 = year0 \* 365 + month0 \* 31 + day0;

int time1 = year1 \* 365 + month1 \* 31 + day1;

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

{

int timex = additions[i].get\_year() \* 360 + additions[i].get\_month() \* 31 + additions[i].get\_day();

if ((time0 < timex < time1) and name == additions[i].get\_name()) return true;

}

}

* 1. **Ticket**

**Ticket.h**

#pragma once

#include<iostream>

#include <string>

class Ticket

{

public:

Ticket(std::string from, std::string to, int year, int month, int day, int cost);

std::string get\_from() { return \_\_from; }

std::string get\_to() { return \_\_to; }

int get\_year() { return \_\_year; }

int get\_month() { return \_\_month; }

int get\_day() { return \_\_day; }

int get\_cost() { return \_\_cost; }

private:

std::string \_\_from, \_\_to;

int \_\_year, \_\_month, \_\_day, \_\_cost;

};

**Ticket.cpp**

#include "Ticket.h"

Ticket::Ticket(std::string from, std::string to, int year, int month, int day, int cost)

{

\_\_from = from;

\_\_to = to;

\_\_year = year;

\_\_month = month;

\_\_day = day;

\_\_cost = cost;

}

* 1. **Addit**

**Addit.h**

#pragma once

#include <string>

class Addit

{

public:

Addit(std::string name, int year, int month, int day, int cost);

std::string get\_name() { return \_\_name; }

int get\_cost() { return \_\_cost; }

int get\_year() { return \_\_year; }

int get\_month() { return \_\_month; }

int get\_day() { return \_\_day; }

private:

std::string \_\_name;

int \_\_cost, \_\_year, \_\_month, \_\_day;

};

**Addit.cpp**

#include "Addit.h"

Addit::Addit(std::string name, int year, int month, int day, int cost)

{

\_\_name = name;

\_\_cost = cost;

}

1. **Программный код методов модификации:**

#include <iostream>

#include "Client.h"

std::vector <Client> DB;

void add\_client()

{

std::cout << "Inter Client Info: Name, Surname, Age: ";

std::string name, surname;

int age;

std::cin >> name >> surname >> age;

Client clnt(name, surname, age);

DB.push\_back(clnt);

}

int main()

{

while(true)

{

std::cout

<< "0. Exit\n"

<< "1. Add Client\n"

<< "2. Buy Ticket\n"

<< "3. Throw Away Ticket\n"

<< "4. Show Tickets\n"

<< "5. Buy Additional Thing\n"

<< "6. Show Additional Things\n"

<< "7. Filter By Time And Thing Name\n"

<< "Enter: ";

int idx;

std::cin >> idx;

switch (idx)

{

case(0):

break;

case(1):

add\_client();

std::cout << "Success!\n\n";

break;

case(2):

int id; //index in vector

std::cout << "Enter Client ID: ";

std::cin >> id;

DB[id].buy\_ticket();

std::cout << "Success!\n\n";

break;

case(3):

id = 0;

int indx; //index in vector

std::cout << "Enter Client ID: ";

std::cin >> id;

std::cout << "Enter Ticket Index: ";

std::cin >> indx;

DB[id].throw\_away\_ticket(indx);

std::cout << "Success!\n\n";

break;

case(4):

id = 0;

std::cout << "Enter Client ID: ";

std::cin >> id;

std::cout << "------\n";

DB[id].show\_tickets();

std::cout << "------\n";

case(5):

id = 0; //index in vector

std::cout << "Enter Client ID: ";

std::cin >> id;

DB[id].buy\_additional\_thing();

std::cout << "Success!\n\n";

break;

case(6):

id = 0;

std::cout << "Enter Client ID: ";

std::cin >> id;

std::cout << "------\n";

DB[id].show\_additional\_things();

std::cout << "------\n";

break;

case(7):

int y0, m0, d0, y1, m1, d1;

std::string name;

std::cout << "Enter Name of a Thing: ";

std::cin >> name;

std::cout << "Enter start Y, M, D: ";

std::cin >> y0 >> m0 >> d0;

std::cout << "Enter finish Y, M, D: ";

std::cin >> y1 >> m1 >> d1;

std::cout << "\n------\n";

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

{

if (DB[i].filter(y0, m0, d0, y1, m1, d1, name) == true)

{

std::cout

<< "CLIENT " << i << "\n"

<< "Name: " << DB[i].get\_name() << "\n"

<< "Surname: " << DB[i].get\_surname() << "\n\n";

}

}

std::cout << "------\n\n";

}

}

}

1. **Программа тестирования:**

#include <gtest/gtest.h>

#include <iostream>

#include <sstream>

#include "Client.h"

class MockInput {

public:

MockInput(const std::string& input) : mock\_input(input), original\_input(nullptr) {

original\_input = &std::cin;

std::cin.rdbuf(mock\_input.rdbuf());

}

~MockInput() {

std::cin.rdbuf(original\_input);

}

private:

std::istringstream mock\_input;

std::istream\* original\_input;

};

class ClientTest : public ::testing::Test {

protected:

void SetUp() override {

DB.clear();

Client clnt("Ivan", "Petrov", 30);

DB.push\_back(clnt);

}

void TearDown() override {

DB.clear();

}

};

TEST\_F(ClientTest, BuyTicketTest) {

{

MockInput mock\_input("CityA CityB 2024 6 13 50\n");

std::cin.ignore();

DB[0].buy\_ticket();

}

ASSERT\_EQ(DB[0].show\_tickets().size(), 1);

}

TEST\_F(ClientTest, ThrowAwayTicketTest) {

DB[0].buy\_ticket();

{

MockInput mock\_input("0\n");

std::cin.ignore();

DB[0].throw\_away\_ticket(0);

}

ASSERT\_EQ(DB[0].show\_tickets().size(), 0);

}

TEST\_F(ClientTest, BuyAdditionalThingTest) {

{

MockInput mock\_input("TestThing 2024 6 13 100\n");

std::cin.ignore();

DB[0].buy\_additional\_thing();

}

ASSERT\_EQ(DB[0].show\_additional\_things().size(), 1);

}

TEST\_F(ClientTest, FilterTest) {

DB[0].buy\_additional\_thing();

{

MockInput mock\_input("TestThing 2024 6 13 2024 12 31\n");

std::cin.ignore();

ASSERT\_TRUE(DB[0].filter(2024, 6, 13, 2024, 12, 31, "TestThing"));

}

}

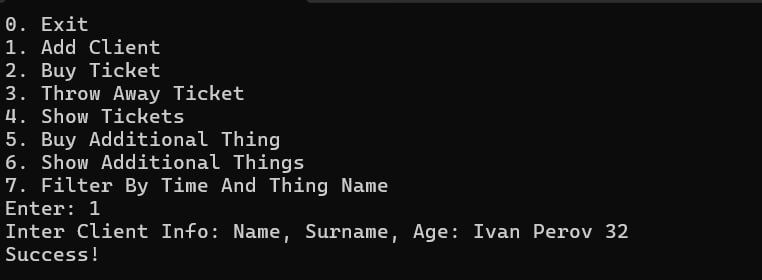
int main(int argc, char\*\* argv) {

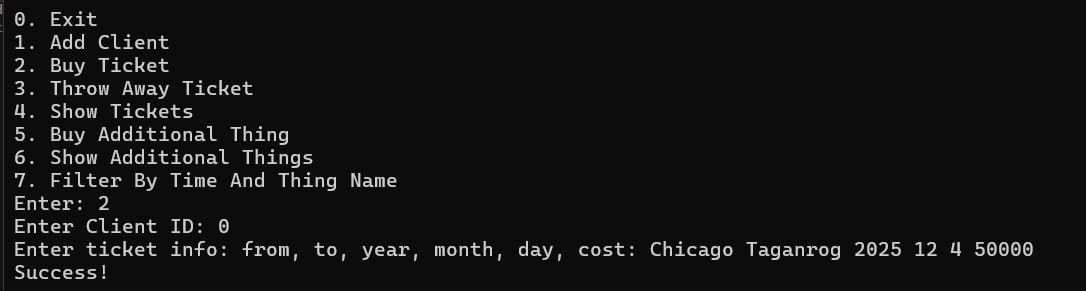
::testing::InitGoogleTest(&argc, argv);

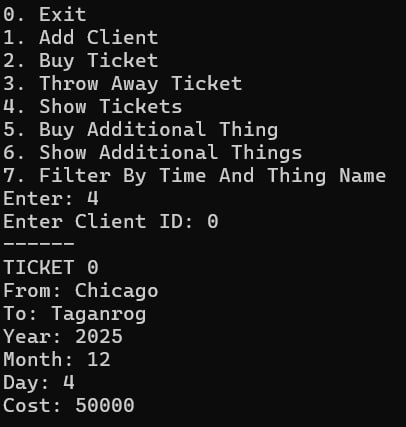
return RUN\_ALL\_TESTS();

}

1. **Вывод основной программы:**







1. **Вывод:** были выполнены все цели и задачи. Реализованы 3 сущности в предметной области торговли авиабилетами, описывающие клиента, билет, дополнительную услугу. Реализован фильтр по заданной услуге и периоду времени. Программа протестирована при помощи google test в visual studio.
2. **Источники:**

<http://cppstudio.com/post/439/>

<https://metanit.com/cpp/tutorial/3.8.php>

<https://metanit.com/cpp/tutorial/7.2.php>

<https://metanit.com/cpp/tutorial/2.17.php>

<https://habr.com/ru/articles/667880/>