Министерство науки и высшего образования РФ

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа «Компьютерных технологий и информационных систем»

ОТЧЕТ

по дисциплине «Практикум по программированию»

**Лабораторная работа № 2**

**Выполнил:**

Cтудент гр. 5130902/30002 С. С. Фибих

**Проверил**

Ст. преподаватель А.М. Журавская

Санкт-Петербург

2024 г.

## **Задание**

Вариант 4

Часть 1

Предметная область: **Аэропорт.**

Касса аэропорта имеет список тарифов на различные направления. При покупке

билета регистрируются паспортные данные.

Система должна:

* позволять вводить данные о тарифах;
* позволять вводить паспортные данные пассажира и регистрировать покупку билета;
* рассчитывать стоимость купленных пассажиром билетов;
* рассчитывать стоимость всех проданных билетов.

## **Код программы**

**Часть 1.**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

enum TariffType {

ECONOMY = 1,

BUSINESS = 2,

FIRST\_CLASS = 3

};

// Функция для проверки, что ввод состоит только из чисел и имеет определённую длину

bool isValidNumber(const string& input, int length) {

// Проверяем длину строки

if (input.length() != length) {

cout << "Ошибка: число должно содержать ровно " << length << " символов.\n";

return false;

}

// Проверяем, что каждая часть строки состоит из цифр

for (char c : input) {

if (!isdigit(c)) {

cout << "Ошибка: ввод должен состоять только из цифр.\n";

return false;

}

}

return true;

}

bool isNumber(string str) {

for (char c : str) {

if (!isdigit(c)) {

return false;

}

}

return true;

}

// Функция для преобразования типа тарифа в строку

string tariffTypeToString(TariffType type) {

switch (type) {

case ECONOMY: return "Эконом";

case BUSINESS: return "Бизнес";

case FIRST\_CLASS: return "Первый класс";

default: return "Неизвестный тип";

}

}

// Класс для хранения информации о тарифе

class Tariff {

private:

string destination; // Название направления

int price; // Цена за билет

TariffType type; // Тип билета

public:

Tariff(string dest, double pr, TariffType tp) : destination(dest), price(pr), type(tp) {}

string getDest() {

return destination;

}

int getPrice() {

return price;

}

string getType() {

return tariffTypeToString(type);

}

};

// Класс для хранения информации о пассажире

class Passenger {

private:

int series;

int number;

string Name;

string SecondName;

vector<Tariff> tickets; // Купленные билеты

public:

Passenger(int sr, int nm, string name, string secname) : series(sr), number(nm), Name(name), SecondName(secname) {}

// Метод для добавления билета

void addTicket(Tariff ticket) {

tickets.push\_back(ticket);

}

// Метод для расчета стоимости купленных билетов

double calculateTotal() {

double total = 0;

for (auto& ticket : tickets) {

total += ticket.getPrice();

}

return total;

}

int getSr() {

return series;

}

int getNm() {

return number;

}

string getName() {

return Name;

}

string getSecNam() {

return SecondName;

}

vector<Tariff> getTicket() {

return tickets;

}

~Passenger() { tickets.clear(); }

};

// Класс для управления продажей билетов

class Airport {

private:

vector<Tariff> tariffs; // Список тарифов

vector<Passenger> passengers; // Список пассажиров

double totalSales = 0; // Общая сумма продаж

static Airport\* instancePtsr;

Airport(){}

public:

Airport(const Airport\* obj) = delete;

static Airport\* getInstance() {

if (instancePtsr == nullptr) {

instancePtsr = new Airport;

return instancePtsr;

}

else return instancePtsr;

}

// Метод для добавления нового тарифа

void addTariff(string destination, double price, TariffType type) {

tariffs.push\_back(Tariff(destination, price, type));

}

// Метод для регистрации пассажира и покупки билета

void registerPurchase(int sr, int nm, string name, string secname, string destination) {

Passenger\* passenger = findPassenger(sr, nm);

if (passenger == nullptr) {

// Если пассажир не найден, создаем нового

passengers.push\_back(Passenger(sr, nm, name, secname));

passenger = &passengers.back();

}

// Поиск тарифа

Tariff\* tariff = findTariff(destination);

if (tariff != nullptr) {

passenger->addTicket(\*tariff);

totalSales += tariff->getPrice();

cout << "Покупка билета на " << destination << " для пассажира " << name << " успешно зарегистрирована.\n";

}

else {

cout << "Тариф на " << destination << " не найден.\n";

}

}

// Метод для расчета общей суммы всех проданных билетов

double calculateTotalSales() {

return totalSales;

}

// Метод для поиска пассажира по номеру паспорта

Passenger\* findPassenger(int sr, int nm) {

for (auto& passenger : passengers) {

if (passenger.getSr() == sr && passenger.getNm() == nm) {

return &passenger;

}

}

return nullptr;

}

// Метод для поиска тарифа по направлению

Tariff\* findTariff(string destination) {

for (auto& tariff : tariffs) {

if (tariff.getDest() == destination) {

return &tariff;

}

}

return nullptr;

}

// Метод для вывода списка всех тарифов

void printTariffs() {

cout << "Список тарифов:\n";

for (auto& tariff : tariffs) {

cout << "Направление: " << tariff.getDest() << ", Цена: " << tariff.getPrice() << " руб. " << ", Класс: " << tariff.getType() <<"\n";

}

}

// Метод для вывода информации о купленных билетах пассажира

void printPassengerTickets(int sr, int nm) {

Passenger\* passenger = findPassenger(sr, nm);

if (passenger != nullptr) {

cout << "Билеты пассажира с паспортом " << sr << " " << nm << ":\n";

for (auto& ticket : passenger->getTicket()) {

cout << "Направление: " << ticket.getDest() << ", Цена: " << ticket.getPrice() << " руб.\n";

}

cout << "Общая стоимость: " << passenger->calculateTotal() << " руб.\n";

}

else {

cout << "Пассажир с паспортом " << sr << " " << nm << " не найден.\n";

}

}

vector<Tariff> getTarif() {

return tariffs;

}

~Airport() {

tariffs.clear(); passengers.clear();

}

};

Airport\* Airport::instancePtsr = nullptr;

int main() {

setlocale(LC\_ALL, "RU");

string wntd;

do {

cout << "Список комманд:\n1 - Добавить тариф\n2 - Вывести список тарифов\n3 - Купить билет пассажиру (Даже если пассажира нет в базе)\n4 - Вывести информацию о пассажире (По серии и номеру паспорта)\n5 - Расчитать стоимость всех купленных билетов\n6 - Завершить работу\n";

cout << "Введите комманду: ";

cin >> wntd;

cout << "\n";

if (!(wntd == "1" || wntd == "2" || wntd == "3" || wntd == "4" || wntd == "5" || wntd == "6")) wntd = "0";

if (wntd == "0") {

cout << "Неправильная команда\n";

}

else if (wntd == "1") {

Airport\* office = Airport::getInstance();

string des, prc; string tup; TariffType boom;

cout << "Пункт назначения" << endl;

cin >> des;

do {

cout << "Введите цену билета" << endl;

cin >> prc;

} while (!isNumber(prc));

do {

cout << "Выберите класс (1 - эконом, 2 - бизнес, 3 - первый)" << endl;

cin >> tup;

} while (!(tup == "1" || tup == "2" || tup == "3"));

if (tup == "1") boom = ECONOMY;

else if (tup == "2") boom = BUSINESS;

else boom = FIRST\_CLASS;

office->addTariff(des, stoi(prc), boom);

}

else if (wntd == "2") {

Airport\* office = Airport::getInstance();

office->printTariffs();

}

else if (wntd == "3") {

Airport\* office = Airport::getInstance();

if (office->getTarif().size() == 0) continue;

string nm, snm, destin; string ser, nomer;

do {

cout << "Пункт назначения" << endl;

cin >> destin;

} while (office->findTariff(destin) == nullptr);

do {

cout << "Серию паспорта" << endl;

//cin.get();

cin >> ser;

} while (!isValidNumber(ser, 4));

do {

cout << "Номер паспорта" << endl;

//cin.get();

cin >> nomer;

} while (!isValidNumber(nomer, 6));

cout << "Имя" << endl;

cin >> nm;

cout << "Фамилия" << endl;

cin >> snm;

office->registerPurchase(stoi(ser), stoi(nomer), nm, snm, destin);

}

else if (wntd == "4") {

Airport\* office = Airport::getInstance();

string ser, nomer;

do {

cout << "Серию паспорта" << endl;

//cin.get();

cin >> ser;

} while (!isValidNumber(ser, 4));

do {

cout << "Номер паспорта" << endl;

//cin.get();

cin >> nomer;

} while (!isValidNumber(nomer, 6));

office->printPassengerTickets(stoi(ser), stoi(nomer));

}

else if (wntd == "5") {

Airport\* office = Airport::getInstance();

cout << "Общая сумма проданных билетов: " << office->calculateTotalSales() << " руб.\n";

}

} while (wntd != "6");

printf("\n\n\n\tСПАСИБО ЗА ВНИМАНИЕ\n\n\n");

return 0;

}

**Пример работы программы**

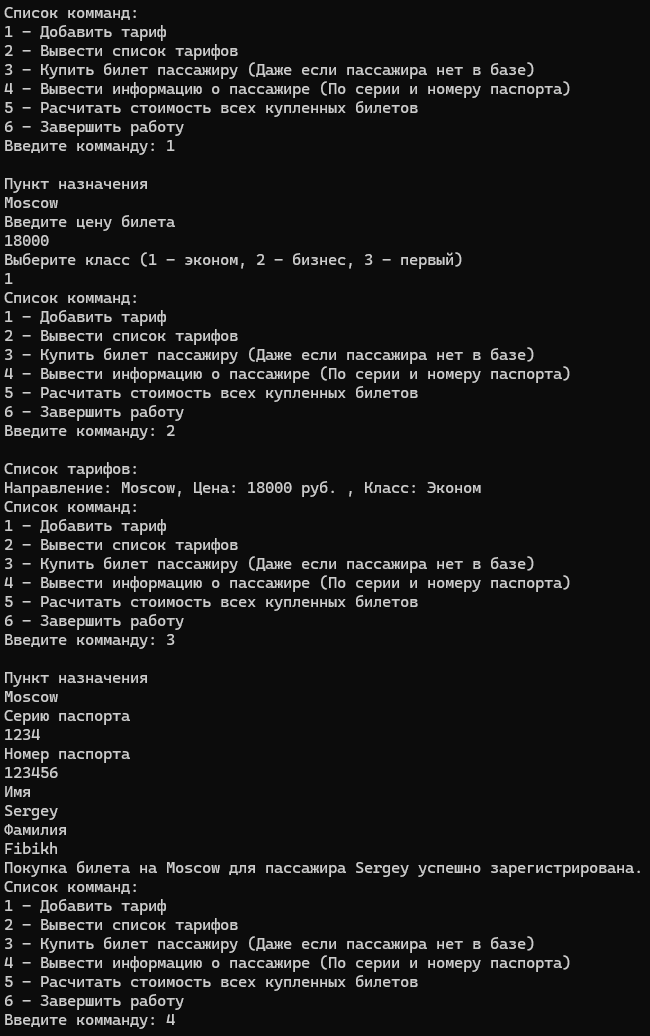


Рис. 1: демонстрация работы части 1

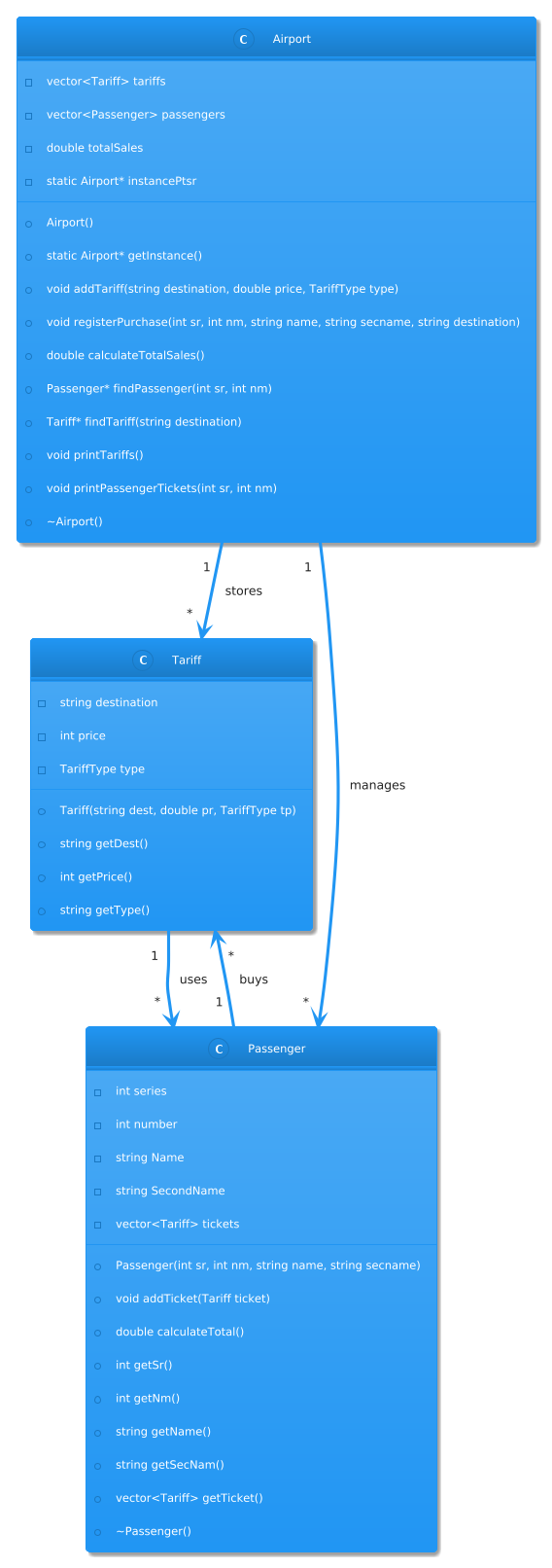
****

Рис. 2: Диаграмма классов

## **Вывод**

В ходе выполнения этой работы, я научился создавать класс-контейнер и контейнизируемый класс, а также сохранять в объекте одного класса объекты других классов.