Министерство образования и науки 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. 5130902/30003 Садчев Р.В

**Проверил**

Ст. преподаватель А.М. Журавская

Санкт-Петербург

2024 г.

## **Задание**

Часть 1:

1. Спроектировать классы для выбранной предметной области.

2. Нарисовать диаграмму классов.

3. Применить к одному из классов шаблон проектирования Singleton.

Разработать два класса: класс-контейнер, управляющий контейнеризируемым классом, и контейнеризируемый класс. Для класса контейнера применить шаблон проектирования Singleton.

Если вы можете применить Singleton ко второй части, то можете оформить его там, тогда можно не делать первую часть, если требуемые в ней условия удалось воплотить во второй части.

Предметная область. Аэропорт – Взлетная полоса.

Часть 2:  
1. Для заданной предметной области спроектировать программную структуру, состоящую из 3–5 классов.

2. В соответствии с разработанной диаграммой классов выполнить программную реализацию.

3. Предусмотреть использование типа данных – перечисление (enum).

4. Ввод/вывод и проверки на ввод должен быть реализован вне проектируемого класса. Проверка полей на правильность ввода обязательна.

5. Реализовать деструктор.

6. Реализовать меню.

Предметная область: Аэропорт.

Касса аэропорта имеет список тарифов на различные направления. При покупке билета регистрируются паспортные данные. Система должна:

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

## **Код программы**

Часть 1:

#include <iostream>

#include <vector>

#include <limits>

#include <locale>

#include <string>

using namespace std;

enum RunwayStatus {

FREE = 0,

OCCUPIED = 1

};

struct TicketSale {

string passportNumber;

double price;

TicketSale(const string& passport, double p) : passportNumber(passport), price(p) {}

};

struct Fare {

string name;

double price;

Fare(const string& fareName, double farePrice) : name(fareName), price(farePrice) {}

};

class Runway {

private:

int id;

RunwayStatus status;

public:

Runway(int id, RunwayStatus status) : id(id), status(status) {}

int getId() const { return id; }

RunwayStatus getStatus() const { return status; }

void setStatus(RunwayStatus newStatus) { status = newStatus; }

void printInfo() const {

cout << "Идентификатор взлетной полосы: " << id << " | "

<< (status == OCCUPIED ? "Занята" : "Свободна") << endl;

}

};

class AirportCashier {

private:

vector<Fare> fares;

double totalSales;

vector<TicketSale> sales;

public:

AirportCashier() : totalSales(0.0) {}

void addFare(const string& fareName, double farePrice) {

fares.emplace\_back(fareName, farePrice);

cout << "Тариф добавлен успешно: " << fareName << " по цене " << farePrice << endl;

}

void printFares() const {

cout << "Список тарифов:\n";

for (size\_t i = 0; i < fares.size(); ++i) {

cout << i + 1 << ". " << fares[i].name << " - " << fares[i].price << endl;

}

}

void registerSale(const string& passportNumber, double price) {

totalSales += price;

sales.emplace\_back(passportNumber, price);

cout << "Билет куплен на сумму: " << price << " (номер паспорта: " << passportNumber << ")\n";

}

double getTotalSales() const { return totalSales; }

double getFarePrice(int index) const {

if (index >= 0 && index < fares.size()) {

return fares[index].price;

}

return 0.0;

}

};

class Airport {

private:

vector<Runway> runways;

AirportCashier cashier;

Airport() {}

Airport(const Airport&) = delete;

Airport& operator=(const Airport&) = delete;

public:

static Airport& getInstance() {

static Airport instance;

return instance;

}

bool runwayExists(int id) const {

for (const auto& runway : runways) {

if (runway.getId() == id) {

return true;

}

}

return false;

}

void addRunway(int id, RunwayStatus status) {

if (runwayExists(id)) {

cout << "Ошибка: Взлетная полоса с идентификатором " << id << " уже существует!\n";

}

else {

runways.emplace\_back(id, status);

cout << "Взлетная полоса добавлена успешно.\n";

}

}

void printRunways() const {

for (const auto& runway : runways) {

runway.printInfo();

}

}

void setRunwayStatus(int id, RunwayStatus newStatus) {

for (auto& runway : runways) {

if (runway.getId() == id) {

runway.setStatus(newStatus);

cout << "Статус взлетной полосы " << id << " обновлён.\n";

return;

}

}

cout << "Взлетная полоса " << id << " не найдена.\n";

}

static int inputInteger(const string& prompt) {

int num;

while (true) {

cout << prompt;

cin >> num;

if (cin.fail() || num < 0) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Неверный ввод. Пожалуйста, введите неотрицательное число.\n";

}

else {

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return num;

}

}

}

static double inputDouble(const string& prompt) {

double num;

while (true) {

cout << prompt;

cin >> num;

if (cin.fail() || num < 0.0) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Неверный ввод. Пожалуйста, введите неотрицательное число.\n";

}

else {

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return num;

}

}

}

static string inputString(const string& prompt) {

string input;

cout << prompt;

getline(cin, input);

return input;

}

AirportCashier& getCashier() {

return cashier;

}

};

void clearConsole() {

#ifdef \_WIN32

system("cls");#else

system("clear");

#endif

}

// Главное меню

static void menu() {

Airport& airport = Airport::getInstance();

bool OnDisplay = true;

while (OnDisplay) {

clearConsole();

cout << "===== Система управления аэропортом =====\n";

cout << "1. Добавить новую взлетную полосу\n";

cout << "2. Обновить статус взлетной полосы\n";

cout << "3. Просмотреть все взлетные полосы\n";

cout << "4. Добавить новый тариф\n";

cout << "5. Просмотреть все тарифы\n";

cout << "6. Купить билет\n";

cout << "7. Просмотреть общую сумму продаж\n";

cout << "0. Выход\n";

cout << "=========================================\n";

int choice = Airport::inputInteger("Выберите опцию: ");

switch (choice) {

case 1: {

int id = Airport::inputInteger("Введите идентификатор взлетной полосы: ");

int status = Airport::inputInteger("Введите статус (0 для Свободна, 1 для Занята): ");

airport.addRunway(id, static\_cast<RunwayStatus>(status));

break;

}

case 2: {

int id = Airport::inputInteger("Введите идентификатор взлетной полосы для обновления: ");

int status = Airport::inputInteger("Введите новый статус (0 для Свободна, 1 для Занята): ");

airport.setRunwayStatus(id, static\_cast<RunwayStatus>(status));

break;

}

case 3:

airport.printRunways();

break;

case 4: {

string fareName = Airport::inputString("Введите название тарифа: ");

double farePrice = Airport::inputDouble("Введите цену тарифа: ");

airport.getCashier().addFare(fareName, farePrice);

break;

}

case 5:

airport.getCashier().printFares();

break;

case 6: {

string passport = Airport::inputString("Введите номер паспорта (только цифры): ");

airport.getCashier().printFares();

int fareIndex = Airport::inputInteger("Выберите тариф (введите номер): ") - 1;

double farePrice = airport.getCashier().getFarePrice(fareIndex);

airport.getCashier().registerSale(passport, farePrice);

break;

}

case 7:

cout << "Общая сумма продаж: " << airport.getCashier().getTotalSales() << endl;

break;

case 0:

OnDisplay = false;

break;

default:

cout << "Неверный выбор! Пожалуйста, попробуйте снова.\n";

}

cout << "\nНажмите Enter, чтобы продолжить...";

cin.get();

}

}

int main() {

setlocale(LC\_ALL, "Russian");

menu();

return 0;

}

Часть 2:

#include <iostream>

#include <vector>

#include <limits>

#include <locale>

#include <string>

using namespace std;

enum RunwayStatus {

FREE = 0,

OCCUPIED = 1

};

struct TicketSale {

string passportNumber;

double price;

TicketSale(const string& passport, double p) : passportNumber(passport), price(p) {}

~TicketSale() {

cout << "Деструктор для продажи билета: " << passportNumber << endl;

}

};

struct Fare {

string name;

double price;

Fare(const string& fareName, double farePrice) : name(fareName), price(farePrice) {}

~Fare() {

cout << "Деструктор для тарифа: " << name << endl;

}

};

static bool isValidNumber(const string& str) {

if (str.empty()) return false;

for (char c : str) {

if (!isdigit(c)) return false;

}

return true;

}

template <typename T>

T getValidInput(const string& prompt) {

string input;

while (true) {

cout << prompt;

cin >> input;

if (isValidNumber(input)) {

try {

if constexpr (is\_same<T, int>::value) {

return stoi(input);

}

else if constexpr (is\_same<T, double>::value) {

return stod(input);

}

}

catch (const invalid\_argument& e) {

cout << "Неверный ввод. Пожалуйста, введите только число." << endl;

}

}

else {

cout << "Неверный ввод. Пожалуйста, введите только число." << endl;

}

}

}

class Runway {

private:

int id;

RunwayStatus status;

public:

Runway(int id, RunwayStatus status) : id(id), status(status) {}

~Runway() {

cout << "Деструктор для взлетной полосы с ID: " << id << endl;

}

int getId() const { return id; }

RunwayStatus getStatus() const { return status; }

void setStatus(RunwayStatus newStatus) { status = newStatus; }

void printInfo() const {

cout << "Идентификатор взлетной полосы: " << id << " | "

<< (status == OCCUPIED ? "Занята" : "Свободна") << endl;

}

};

class AirportCashier {

private:

vector<Fare> fares;

double totalSales;

vector<TicketSale> sales;

public:

AirportCashier() : totalSales(0.0) {}

~AirportCashier() {

cout << "Деструктор для кассира аэропорта\n";

}

void addFare(const string& fareName, double farePrice) {

fares.emplace\_back(fareName, farePrice);

cout << "Тариф добавлен успешно: " << fareName << " по цене " << farePrice << endl;

}

void printFares() const {

cout << "Список тарифов:\n";

for (size\_t i = 0; i < fares.size(); ++i) {

cout << i + 1 << ". " << fares[i].name << " - " << fares[i].price << endl;

}

}

void registerSale(const string& passportNumber, double price) {

totalSales += price;

sales.emplace\_back(passportNumber, price);

cout << "Билет куплен на сумму: " << price << " (номер паспорта: " << passportNumber << ")\n";

}

double getTotalSales() const { return totalSales; }

double getFarePrice(int index) const {

if (index >= 0 && index < fares.size()) {

return fares[index].price;

}

return 0.0;

}

};

class Airport {

private:

vector<Runway> runways;

AirportCashier cashier;

Airport() {}

Airport(const Airport&) = delete;

Airport& operator=(const Airport&) = delete;

public:

static Airport& getInstance() {

static Airport instance;

return instance;

}

~Airport() {

cout << "Деструктор для аэропорта\n";

}

bool runwayExists(int id) const {

for (const auto& runway : runways) {

if (runway.getId() == id) {

return true;

}

}

return false;

}

void addRunway(int id, RunwayStatus status) {

if (runwayExists(id)) {

cout << "Ошибка: Взлетная полоса с идентификатором " << id << " уже существует!\n";

}

else {

runways.emplace\_back(id, status);

cout << "Взлетная полоса добавлена успешно.\n";

}

}

void printRunways() const {

for (const auto& runway : runways) {

runway.printInfo();

}

}

void setRunwayStatus(int id, RunwayStatus newStatus) {

for (auto& runway : runways) {

if (runway.getId() == id) {

runway.setStatus(newStatus);

cout << "Статус взлетной полосы " << id << " обновлён.\n";

return;

}

}

cout << "Взлетная полоса " << id << " не найдена.\n";

}

AirportCashier& getCashier() {

return cashier;

}

};

static void clearConsole() {

#ifdef \_WIN32

system("cls");

#else

system("clear");

#endif

}

static void menu() {

Airport& airport = Airport::getInstance();

bool OnDisplay = true;

while (OnDisplay) {

clearConsole();

cout << "===== Система управления аэропортом =====\n";

cout << "1. Добавить новую взлетную полосу\n";

cout << "2. Обновить статус взлетной полосы\n";

cout << "3. Просмотреть все взлетные полосы\n";

cout << "4. Добавить новый тариф\n";

cout << "5. Просмотреть все тарифы\n";

cout << "6. Купить билет\n";

cout << "7. Просмотреть общую сумму продаж\n";

cout << "0. Выход\n";

cout << "=========================================\n";

int choice = getValidInput<int>("Выберите опцию: ");

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

switch (choice) {

case 1: {

int id = getValidInput<int>("Введите идентификатор взлетной полосы: ");

int status = getValidInput<int>("Введите статус (0 для Свободна, 1 для Занята): ");

airport.addRunway(id, static\_cast<RunwayStatus>(status));

break;

}

case 2: {

int id = getValidInput<int>("Введите идентификатор взлетной полосы для обновления: ");

int status = getValidInput<int>("Введите новый статус (0 для Свободна, 1 для Занята): ");

airport.setRunwayStatus(id, static\_cast<RunwayStatus>(status));

break;

}

case 3:

airport.printRunways();

break;

case 4: {

string fareName;

cout << "Введите название тарифа: ";

getline(cin, fareName);

double farePrice = getValidInput<double>("Введите цену тарифа: ");

airport.getCashier().addFare(fareName, farePrice);

break;

}

case 5:

airport.getCashier().printFares();

break;

case 6: {

string passport;

cout << "Введите номер паспорта (только цифры): ";

getline(cin, passport);

airport.getCashier().printFares();

int fareIndex = getValidInput<int>("Выберите тариф (введите номер): ") - 1;

double farePrice = airport.getCashier().getFarePrice(fareIndex);

airport.getCashier().registerSale(passport, farePrice);

break;

}

case 7:

cout << "Общая сумма продаж: " << airport.getCashier().getTotalSales() << endl;

break;

case 0:

OnDisplay = false;

break;

default:

cout << "Неверный выбор! Пожалуйста, попробуйте снова.\n";

}

cout << "\nНажмите Enter, чтобы продолжить...";

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

}

int main() {

setlocale(LC\_ALL, "");

menu();

return 0;

}

**Пример работы программы**

Часть 1:

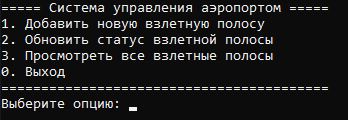


Рисунок 1 – Правильная работа программы

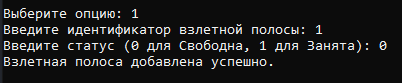


Рисунок 2 –Правильная работа программы



Рисунок 3 –Правильная работа программы

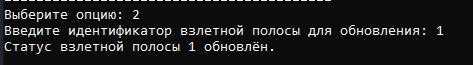


Рисунок 4 – Правильная работа программы



Рисунок 5 – Правильная работа программы

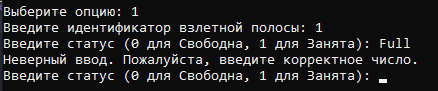


Рисунок 6 – Правильная работа программы при вводе неверных данных

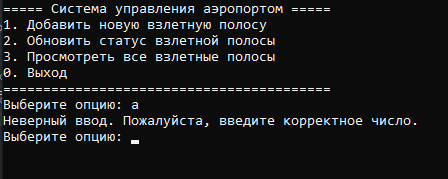
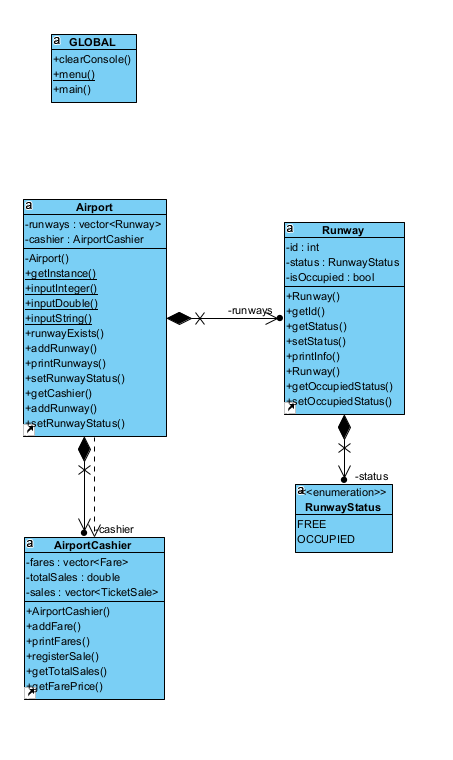


Рисунок 7 – Правильная работа программы при вводе неверных данных

Рисунок 8 – Диаграмма (Построена в Visual Paradigm)

Часть 2:

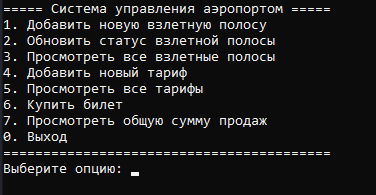


Рисунок 9 – Правильная работа программы

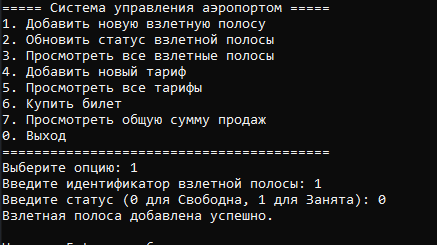


Рисунок 10 –Правильная работа программы

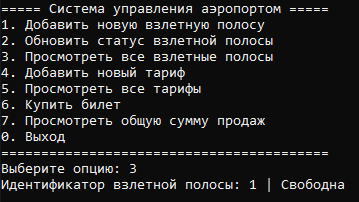
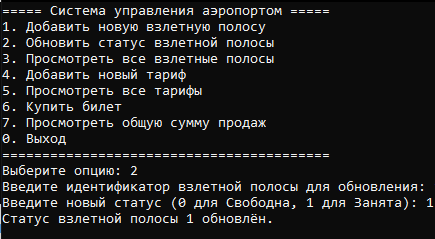


Рисунок 11 –Правильная работа программы



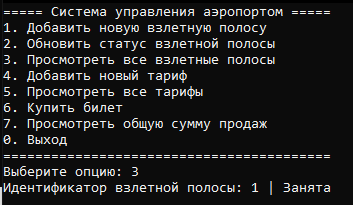
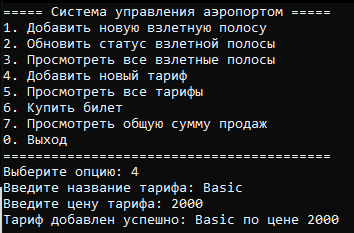
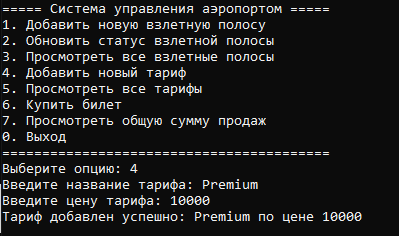


Рисунок 12 – Правильная работа программы





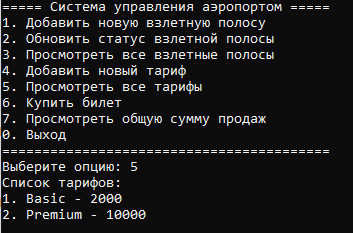


Рисунок 13 – Правильная работа программы

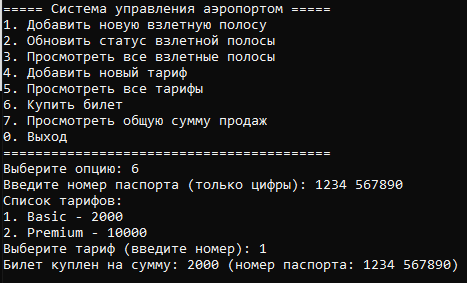


Рисунок 14 – Правильная работа программы

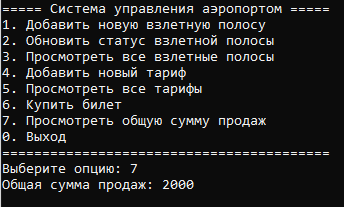


Рисунок 15 – Правильная работа программы

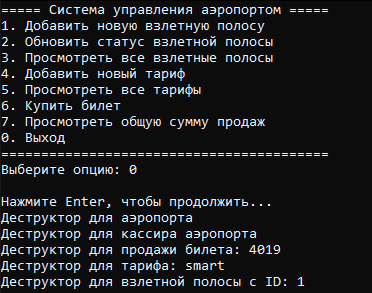


Рисунок 16 – Правильная работа программы

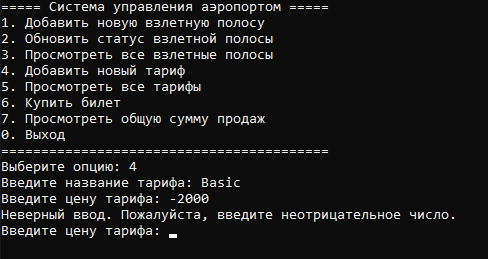


Рисунок 17 – Правильная работа программы при вводе неверных данных

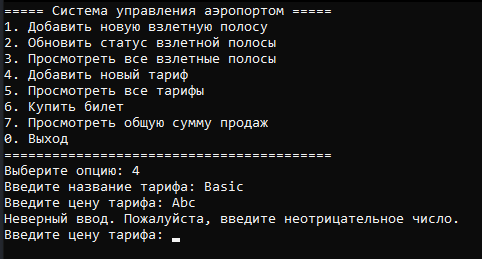


Рисунок 18 – Правильная работа программы при вводе неверных данных

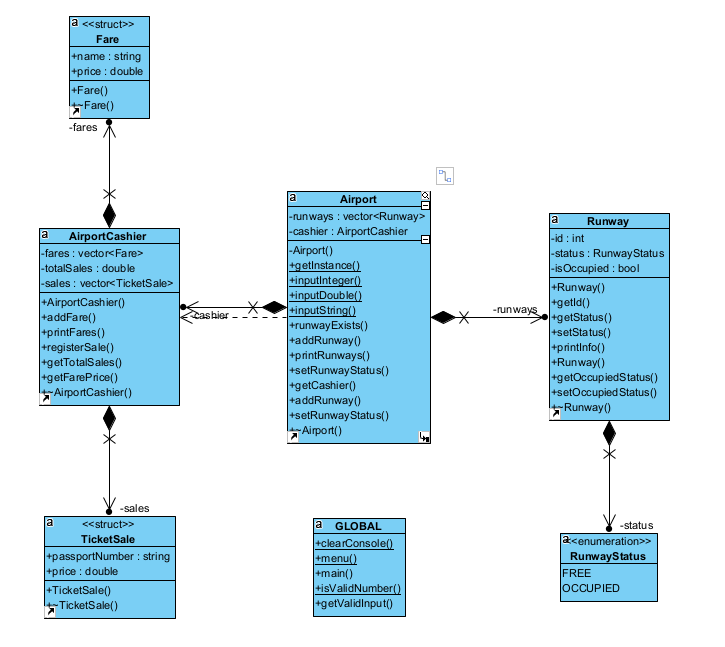


Рисунок 18 – Диаграмма (Построена в Visual Paradigm)

Github: [1 часть](https://github.com/walterwar92/PolyLabs/blob/main/Lab_PPP_2_1.cpp) | [2 часть](https://github.com/walterwar92/PolyLabs/blob/main/Lab_PPP_2_2.cpp)

## **Вывод**

В ходе выполнения лабораторной работы были достигнуты все цели и задачи. Были приобретены навыки работы с функциями, циклами, условными операторами, классами C++, а также с классами типа Singleton. Кроме того, научились одновременно работать с несколькими классами и составлять диаграммы. В результате выполнения программы была разработана система, которая позволяет работать с информацией о занятости определённых полос аэропорта, тарифами на билеты, а также осуществлять покупку билетов, вводя номер паспорта и выбирая тариф.