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. 5130902/40002 Г.Ю. Рюмин

**Проверил**

Ст. преподаватель А.М. Журавская

Санкт-Петербург

2025 г.

## **Задание**

**Часть 1**

**(С++, С#,  Java, Python)**

1. Спроектировать классы  для выбранной предметной области.

2. Нарисовать диаграмму классов.

3. Применить к одному из классов шаблон проектирования Singleton.

**Индивидуальные задания:**

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

Описание предметной области:

5. Аэропорт – Взлетная полоса.

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.

**Часть 2**

**(С++, С#,  Java, Python)**

**Задание**

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.

**Индивидуальные задания**

4. 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 <string>

#include <vector>

#include <limits>

#include <locale>

#include <codecvt>

using namespace std;

class Runway {

public:

int length;

int width;

string name;

bool isOccupied;

};

class Airport {

private:

vector<Runway> runways;

public:

void addRunway(const Runway& runway) {

runways.push\_back(runway);

}

void displayRunways() const {

for (const auto& runway : runways) {

cout << "Имя полосы: " << runway.name << ", Длина: " << runway.length

<< ", Ширина: " << runway.width << ", Занята: "

<< (runway.isOccupied ? "Да" : "Нет") << endl;

}

}

};

int getIntInput(string prompt, string errorMessage,int minValue = numeric\_limits<int>::min() , int maxValue = numeric\_limits<int>::max()) {

int value;

cout << prompt;

cin >> value;

while(cin.fail() || maxValue < value || value < minValue) {

cin.clear();

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

cout << errorMessage << endl;

cin >> value;

}

return value;

}

string getStringInput(string prompt, string errorMessage) {

string value;

cout << prompt;

cin >> value;

while(value.empty() || cin.fail()) {

cin.clear();

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

cout << errorMessage << endl;

cin >> value;

}

return value;

}

std::wstring toLower(const std::wstring& input) {

std::locale loc("ru\_RU.UTF-8");

std::wstring result;

for (wchar\_t c : input) {

result += std::use\_facet<std::ctype<wchar\_t>>(loc).tolower(c);

}

return result;

}

std::string toLowerUtf8(const std::string& input) {

std::wstring\_convert<std::codecvt\_utf8<wchar\_t>> conv;

std::wstring wide = conv.from\_bytes(input);

wide = toLower(wide);

return conv.to\_bytes(wide);

}

bool getBoolInput(string prompt, string errorMessage) {

string value;

cout << prompt;

cin >> value;

value = toLowerUtf8(value);

while(value != "yes" && value != "no" && value != "y" && value != "n" && value != "да" && value != "д" && value != "нет" && value != "н" || cin.fail()) {

cin.clear();

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

cout << errorMessage << endl;

cin >> value;

value = toLowerUtf8(value);

}

return (value == "yes" || value == "y" || value == "да" || value == "д");

}

int main() {

Airport airport;

Runway runway1 = {getIntInput("Введите длину взлётной полосы 1: ", "Ошибка! Попробуйте ещё раз:", 1), getIntInput("Введите ширину взлётной полосы 1: ", "Ошибка! Попробуйте ещё раз:", 1), getStringInput("Введите название полосы 1:", "Ошибка! Попробуйте ещё раз:"), getBoolInput("Полоса 1 занята? (yes/no;да/нет):", "Ошибка! Попробуйте ещё раз:")};

cout << endl;

Runway runway2 = {getIntInput("Введите длину взлётной полосы 2: ", "Ошибка! Попробуйте ещё раз:", 1), getIntInput("Введите ширину взлётной полосы 2: ", "Ошибка! Попробуйте ещё раз:", 1), getStringInput("Введите название полосы 2:", "Ошибка! Попробуйте ещё раз:"), getBoolInput("Полоса 2 занята? (yes/no;да/нет):", "Ошибка! Попробуйте ещё раз:")};

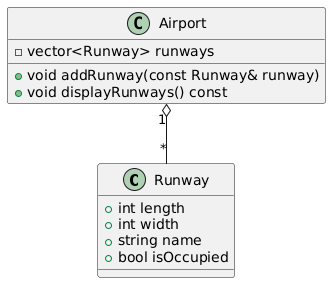
airport.addRunway(runway1);

airport.addRunway(runway2);

airport.displayRunways();

return 0;

}

  
Рисунок 1 — Диаграмма классов для части 1

Часть 2:  
#include <iostream>

#include <vector>

#include <string>

#include <mutex>

#include <limits>

using namespace std;

class Tariff {

public:

string name;

int cost;

};

class Client {

public:

string name;

string passport;

};

class Ticket {

public:

Client client;

Tariff tariff;

};

class Till {

private:

vector<Ticket> tickets;

vector<Tariff> tariffs;

vector<Client> clients;

static Till\* instancePtr;

static mutex mtx;

Till() {}

public:

Till(const Till& obj) = delete;

static Till\* getInstance() {

if (instancePtr == nullptr) {

lock\_guard<mutex> lock(mtx);

if (instancePtr == nullptr) {

instancePtr = new Till();

}

}

return instancePtr;

}

void buyTicket(const Client& client, const Tariff& tariff) {

Ticket ticket;

ticket.client = client;

ticket.tariff = tariff;

tickets.push\_back(ticket);

}

void addClient(const Client& client) {

clients.push\_back(client);

}

void addTariff(const Tariff& tariff) {

tariffs.push\_back(tariff);

}

int summaryOfSoldTickets() const {

int sum = 0;

for (const auto& ticket : tickets) {

sum += ticket.tariff.cost;

}

return sum;

}

int summaryOfSoldTicketsByName(const string& name) {

int sum = 0;

for (const auto& ticket : tickets) {

if (ticket.client.name == name) {

sum += ticket.tariff.cost;

}

}

return sum;

}

const vector<Client>& getClients() const {

return clients;

}

const vector<Tariff>& getTariffs() const {

return tariffs;

}

const vector<Ticket>& getTickets() const {

return tickets;

}

};

int getIntInput(string prompt, string errorMessage, int minValue = numeric\_limits<int>::min(), int maxValue = numeric\_limits<int>::max()) {

int value;

string input;

cout << prompt << endl;

while (true) {

cin >> input;

bool isValid = !input.empty() && (input[0] == '-' ? input.size() > 1 : true);

for (size\_t i = (input[0] == '-' ? 1 : 0); i < input.size(); ++i) {

if (!isdigit(input[i])) {

isValid = false;

break;

}

}

if (isValid) {

try {

size\_t pos;

value = stoi(input, &pos);

if (pos == input.size() && value >= minValue && value <= maxValue) {

break;

}

} catch (...) {}

}

cin.clear();

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

cout << errorMessage << endl;

}

return value;

}

string getStringInput(string prompt, string errorMessage) {

string value;

cout << prompt;

cin >> value;

while(value.empty() || cin.fail()) {

cin.clear();

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

cout << errorMessage << endl;

cin >> value;

}

return value;

}

enum MenuAction {

ADD\_TARIFF = 1,

ADD\_CLIENT,

BUY\_TICKET,

CLIENT\_SUM,

TOTAL\_SUM,

SHOW\_TARIFFS,

SHOW\_CLIENTS,

SHOW\_TICKETS,

EXIT = 0

};

Till\* Till::instancePtr = nullptr;

mutex Till::mtx;

int main() {

Till\* till = Till::getInstance();

while (true) {

cout << endl << "Меню:" << endl;

cout << "1) Добавить тариф" << endl;

cout << "2) Ввести данные пассажира" << endl;

cout << "3) Зарегистрировать покупку билета" << endl;

cout << "4) Рассчитать стоимость купленных билетов конкретным пассажиром" << endl;

cout << "5) Рассчитать стоимость всех проданных билетов" << endl;

cout << "6) Показать все тарифы" << endl;

cout << "7) Показать всех пассажиров" << endl;

cout << "8) Показать все билеты" << endl;

cout << "0) Выход" << endl;

int choice = getIntInput("Выберите пункт меню: ", "Ошибка ввода!", 0, 8);

cout << endl;

if (choice == EXIT) break;

if (choice == ADD\_TARIFF) {

Tariff tariff;

tariff.name = getStringInput("Введите название тарифа: ", "Ошибка ввода!");

tariff.cost = getIntInput("Введите стоимость тарифа: ", "Ошибка ввода!", 1);

till->addTariff(tariff);

cout << "Тариф добавлен." << endl;

} else if (choice == ADD\_CLIENT) {

Client client;

client.name = getStringInput("Введите имя клиента: ", "Ошибка ввода!");

client.passport = getStringInput("Введите паспорт клиента: ", "Ошибка ввода!");

till->addClient(client);

cout << "Клиент добавлен." << endl;

} else if (choice == BUY\_TICKET) {

if (till->getClients().empty() || till->getTariffs().empty()) {

cout << "Нет клиентов или тарифов для покупки билета." << endl;

continue;

}

cout << "Выберите клиента:" << endl;

const auto& clients = till->getClients();

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

cout << i + 1 << ") " << clients[i].name << " (" << clients[i].passport << ")" << endl;

}

int clientIdx = getIntInput("Введите номер клиента: ", "Ошибка ввода!", 1, clients.size()) - 1;

cout << "Выберите тариф:" << endl;

const auto& tariffs = till->getTariffs();

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

cout << i + 1 << ") " << tariffs[i].name << " (" << tariffs[i].cost << ")" << endl;

}

int tariffIdx = getIntInput("Введите номер тарифа: ", "Ошибка ввода!", 1, tariffs.size()) - 1;

till->buyTicket(clients[clientIdx], tariffs[tariffIdx]);

cout << "Билет куплен." << endl;

} else if (choice == CLIENT\_SUM) {

if (till->getClients().empty()) {

cout << "Нет клиентов." << endl;

continue;

}

cout << "Выберите клиента:" << endl;

const auto& clients = till->getClients();

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

cout << i + 1 << ") " << clients[i].name << " (" << clients[i].passport << ")" << endl;

}

int clientIdx = getIntInput("Введите номер клиента: ", "Ошибка ввода!", 1, clients.size()) - 1;

int sum = till->summaryOfSoldTicketsByName(clients[clientIdx].name);

cout << "Сумма купленных билетов для " << clients[clientIdx].name << ": " << sum << endl;

} else if (choice == TOTAL\_SUM) {

int sum = till->summaryOfSoldTickets();

cout << "Общая сумма проданных билетов: " << sum << endl;

} else if (choice == SHOW\_TARIFFS) {

const auto& tariffs = till->getTariffs();

if (tariffs.empty()) {

cout << "Нет тарифов." << endl;

} else {

cout << "Тарифы:" << endl;

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

cout << i + 1 << ") " << tariffs[i].name << " (Стоимость: " << tariffs[i].cost << ")" << endl;

}

}

} else if (choice == SHOW\_CLIENTS) {

const auto& clients = till->getClients();

if (clients.empty()) {

cout << "Нет пассажиров." << endl;

} else {

cout << "Пассажиры:" << endl;

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

cout << i + 1 << ") " << clients[i].name << " (Паспорт: " << clients[i].passport << ")" << endl;

}

}

} else if (choice == SHOW\_TICKETS) {

const auto& tickets = till->getTickets();

if (tickets.empty()) {

cout << "Нет билетов." << endl;

} else {

cout << "Билеты:" << endl;

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

cout << i + 1 << ") "

<< "Пассажир: " << tickets[i].client.name << " (Паспорт: " << tickets[i].client.passport << "), "

<< "Тариф: " << tickets[i].tariff.name << " (Стоимость: " << tickets[i].tariff.cost << ")" << endl;

}

}

}

}

return 0;

}

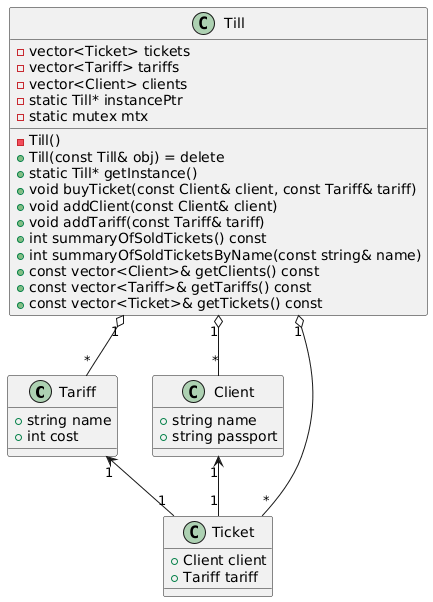


Рисунок 2 — Диаграммы классов для 2 части

**Пример работы программы**

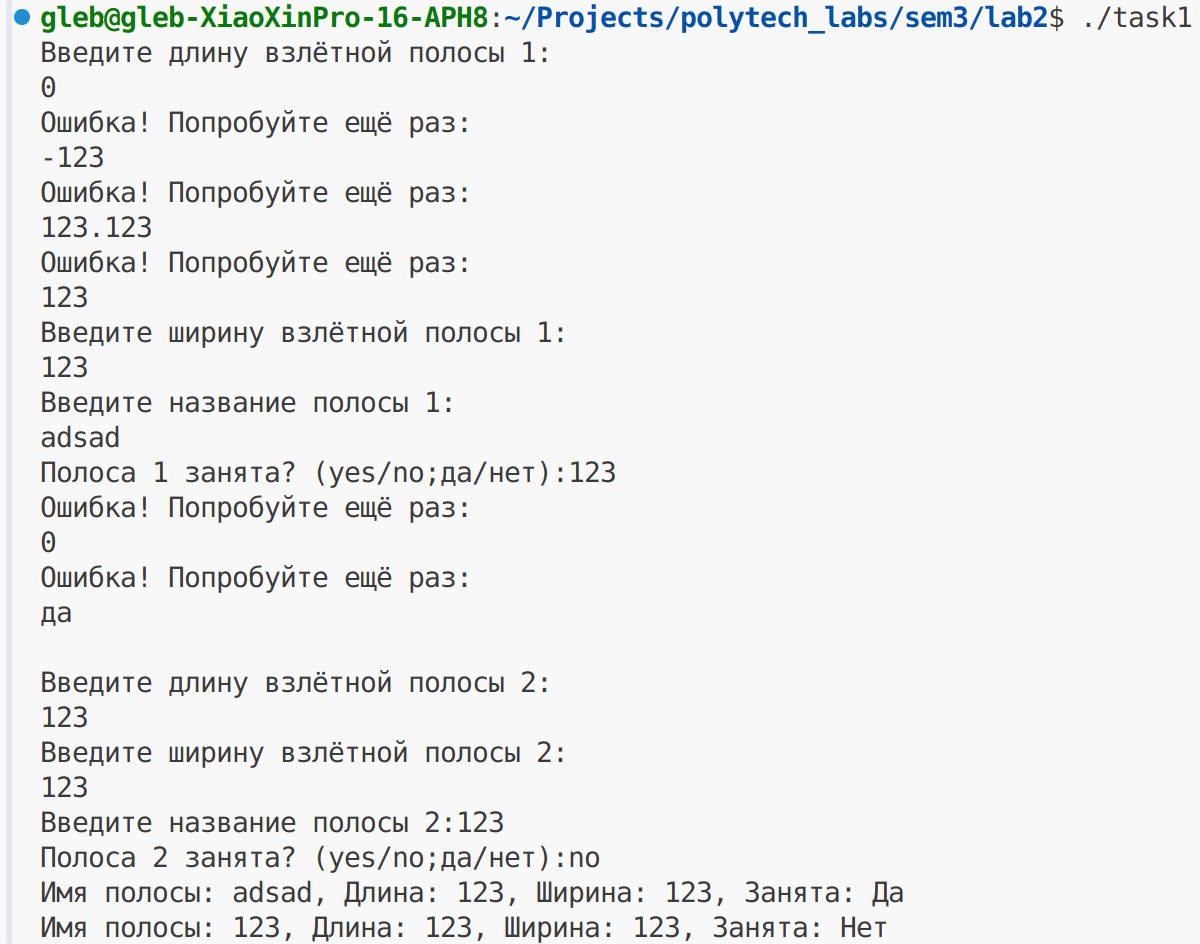


Рисунок 3 – Правильная работы программы части 1

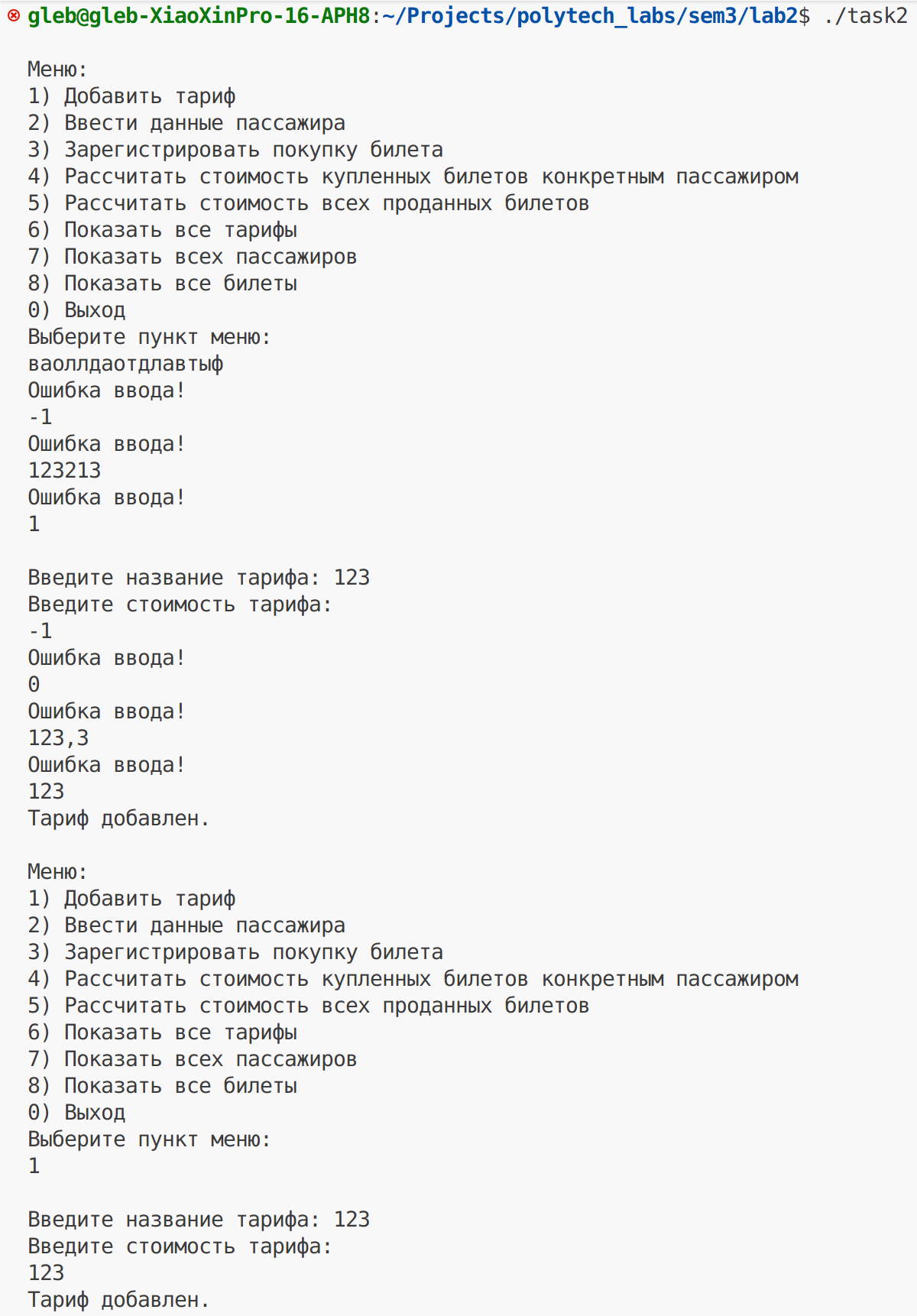


Рисунок 4 — Правильная работа программы 2 части (часть 1)

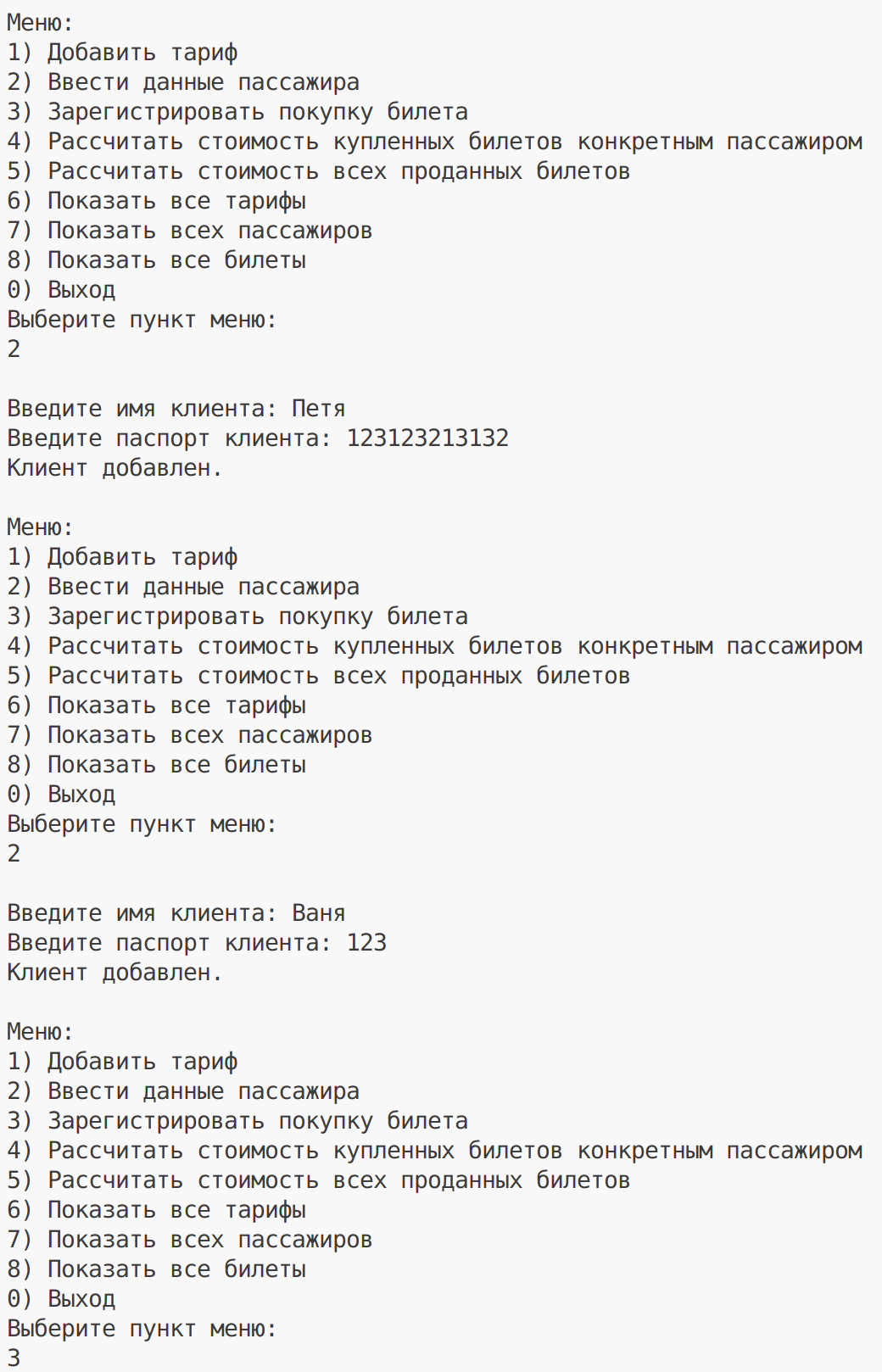


Рисунок 5 — Правильная работа программа 2 части (часть 2)

## 

Рисунок 6 — Правильная работа программы 2 части (часть 3)

## 

Рисунок 7 — Правильная работа программы 2 части (часть 4)

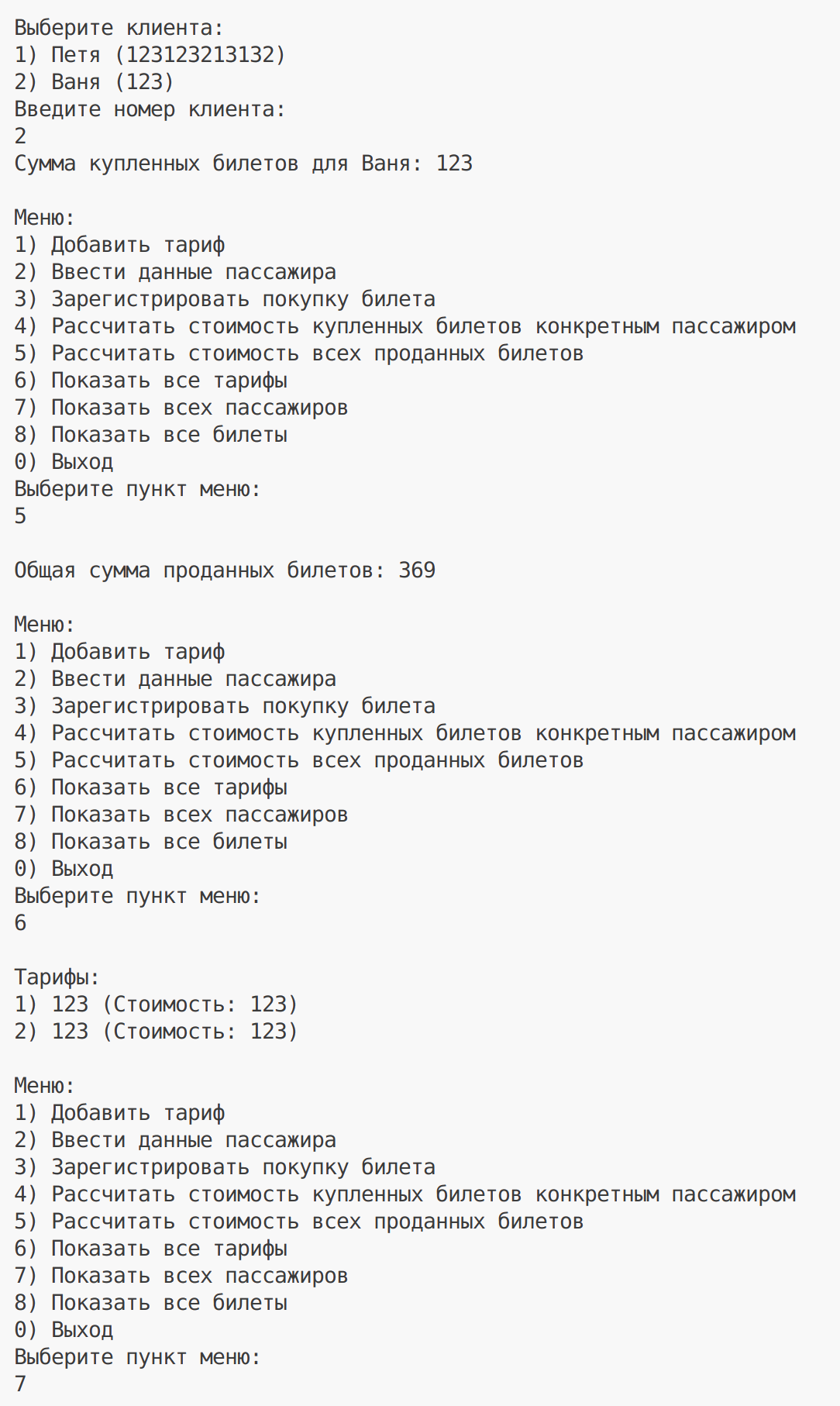


Рисунок 8 — Правильная работа программы 2 части (часть 5)

## 

Рисунок 9 — Правильная работы программы 2 части (часть 6)

## **Вывод**

В лабораторной работе были реализованы две программные структуры для разных предметных областей. В первой части был выполнен вариант 5 — «Аэропорт – Взлетная полоса», где система хранит информацию о размерах взлётных полос, их наименованиях и статусе занятости. Класс-контейнер для взлётных полос реализован с применением шаблона проектирования Singleton, что обеспечивает уникальность экземпляра и централизованное управление всеми полосами аэропорта. Во второй части реализован вариант 4 — «Аэропорт», где касса аэропорта управляет списком тарифов на направления, данными пассажиров и купленными билетами. В системе реализованы классы для тарифов, клиентов, билетов и кассы, что позволяет гибко управлять данными и выполнять все необходимые операции: добавление тарифов и клиентов, регистрацию покупки билета, расчет стоимости билетов для конкретного пассажира и общей суммы проданных билетов, а также вывод списков всех сущностей. В обеих частях реализованы необходимые классы, предусмотрены проверки ввода вне классов, используется перечисление (enum) для организации меню, реализован деструктор, а также построена диаграмма классов UML, отражающая структуру и связи между объектами. Система получилась гибкой, расширяемой, соответствует требованиям задания и полностью охватывает описанные предметные области.