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 № 3**

**Выполнил:**

Cтудент гр. 5130902/40002 Г.Ю. Рюмин

**Проверил**

Ст. преподаватель А.М. Журавская

Санкт-Петербург

2025 г.

## **Задание**

**Общая часть**

В данной работе необходимо использовать один из шаблонов проектирования, например описанный выше **Strategy,** реализовать наследование, полиморфизм и парадигму интерфейсных классов. В работе для обработки ошибок должны быть реализованы исключения (часть проверок можно сделать классическим способом).

Сначала необходимо создать диаграмму классов, а потом уже начинать реализацию.

**Варианты:**

4. Предметная область: **Аэропорт.** Касса аэропорта имеет список тарифов на различные направления. Тариф содержит название направления и стоимость перевозки. На некоторые направления предоставляется фиксированная скидка. В классе «аэропорт» реализовать метод добавления нового тарифа и метод поиска направления с максимальной стоимостью.

## **Код программы**

#include <iostream>

#include <vector>

#include <string>

#include <stdexcept>

#include <algorithm>

#include <limits>

class DiscountStrategy {

public:

virtual ~DiscountStrategy() = default;

virtual double calculate(double baseCost) const = 0;

virtual std::string getDescription() const = 0;

};

class NoDiscount : public DiscountStrategy {

public:

double calculate(double baseCost) const override {

return baseCost;

}

std::string getDescription() const override {

return "Без скидки";

}

};

class FixedDiscount : public DiscountStrategy {

private:

double discountPercent;

public:

explicit FixedDiscount(double percent) : discountPercent(percent) {

if (percent < 1 || percent > 100) {

throw std::invalid\_argument("Скидка должна быть в диапазоне от 1 до 100 процентов.");

}

}

double calculate(double baseCost) const override {

double finalCost = baseCost \* (1.0 - discountPercent / 100.0);

return (finalCost > 0) ? finalCost : 0;

}

std::string getDescription() const override {

return "Скидка " + std::to\_string(static\_cast<int>(discountPercent)) + "%";

}

};

class Tariff {

protected:

std::string destination;

double baseCost;

DiscountStrategy\* discountStrategy;

public:

Tariff(std::string dest, double cost, DiscountStrategy\* strategy)

: destination(std::move(dest)), baseCost(cost), discountStrategy(strategy) {

if (cost <= 0) {

throw std::invalid\_argument("Базовая стоимость должна быть положительной.");

}

if (!discountStrategy) {

throw std::invalid\_argument("Стратегия скидок не может быть пустой (null).");

}

}

virtual ~Tariff() {

delete discountStrategy;

}

std::string getDestination() const { return destination; }

double getBaseCost() const { return baseCost; }

double getCost() const {

return discountStrategy->calculate(baseCost);

}

std::string getDiscountInfo() const {

return discountStrategy->getDescription();

}

};

class Airport {

private:

std::string name;

std::vector<Tariff\*> tariffs;

public:

explicit Airport(std::string airportName) : name(std::move(airportName)) {}

~Airport() {

for (Tariff\* tariff : tariffs) {

delete tariff;

}

}

Airport(const Airport& other) = delete;

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

void addTariff(Tariff\* tariff) {

for (const auto\* t : tariffs) {

if (t->getDestination() == tariff->getDestination()) {

throw std::runtime\_error("Тариф для направления '" + tariff->getDestination() + "' уже существует.");

}

}

tariffs.push\_back(tariff);

}

Tariff\* findMaxCostTariff() const {

if (tariffs.empty()) {

throw std::runtime\_error("Невозможно найти тариф с максимальной стоимостью: список пуст.");

}

auto maxElementIt = std::max\_element(tariffs.cbegin(), tariffs.cend(),

[](const Tariff\* a, const Tariff\* b) {

return a->getCost() < b->getCost();

});

return \*maxElementIt;

}

const std::vector<Tariff\*>& getTariffs() const {

return tariffs;

}

};

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

int value;

std::string input;

std::cout << prompt << std::endl;

while (true) {

std::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 (...) {}

}

std::cin.clear();

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

std::cout << errorMessage << std::endl;

}

return value;

}

std::string getStringInput(std::string prompt, std::string errorMessage) {

std::string value;

std::cout << prompt;

std::cin >> value;

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

std::cin.clear();

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

std::cout << errorMessage << std::endl;

std::cin >> value;

}

return value;

}

int main() {

setlocale(LC\_ALL, "Russian");

Airport airport("Международный Аэропорт");

while (true) {

std::cout << "\n--- Меню кассы аэропорта ---" << std::endl;

std::cout << "1. Добавить новый тариф" << std::endl;

std::cout << "2. Найти направление с максимальной стоимостью" << std::endl;

std::cout << "3. Показать все тарифы" << std::endl;

std::cout << "0. Выход" << std::endl;

int choice = getIntInput("Выберите действие: ", "Ошибка ввода!", 0, 3);

if (choice == 1) {

try {

std::string dest = getStringInput("Введите название направления: ", "Ошибка ввода!");

double cost = getIntInput("Введите базовую стоимость: ", "Ошибка ввода!", 1);

DiscountStrategy\* strategy = nullptr;

int discountChoice = getIntInput("Предоставляется ли скидка? (1-Да, фиксированная / 2-Нет): ", "Ошибка ввода!", 1, 2);

if (discountChoice == 1) {

double discount = getIntInput("Введите размер скидки в процентах: ", "Ошибка ввода!", 1, 100);

strategy = new FixedDiscount(discount);

} else {

strategy = new NoDiscount();

}

Tariff\* newTariff = new Tariff(dest, cost, strategy);

airport.addTariff(newTariff);

std::cout << "Тариф успешно добавлен!" << std::endl;

} catch (const std::exception& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

}

}

else if (choice == 2) {

try {

Tariff\* maxTariff = airport.findMaxCostTariff();

std::cout << "\n--- Направление с максимальной стоимостью ---" << std::endl;

std::cout << "Направление: " << maxTariff->getDestination() << std::endl;

std::cout << "Итоговая стоимость: " << maxTariff->getCost() << std::endl;

std::cout << " (Базовая: " << maxTariff->getBaseCost() << ", Информация о скидке: " << maxTariff->getDiscountInfo() << ")" << std::endl;

} catch (const std::exception& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

}

}

else if (choice == 3) {

const auto& tariffs = airport.getTariffs();

if (tariffs.empty()) {

std::cout << "\nСписок тарифов пуст." << std::endl;

} else {

std::cout << "\n--- Все доступные тарифы ---" << std::endl;

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

std::cout << i + 1 << ". Направление: " << tariffs[i]->getDestination()

<< ", Итоговая стоимость: " << tariffs[i]->getCost()

<< " (" << tariffs[i]->getDiscountInfo() << ")" << std::endl;

}

}

}

else if (choice == 0) {

break;

}

}

return 0;

}

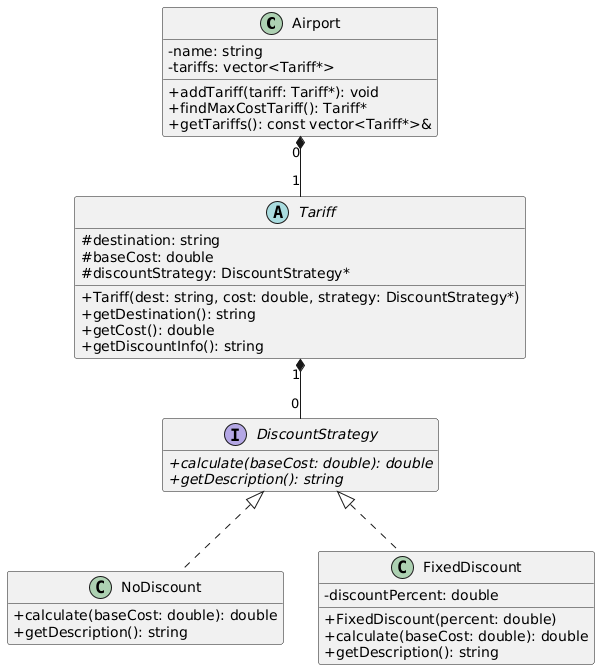
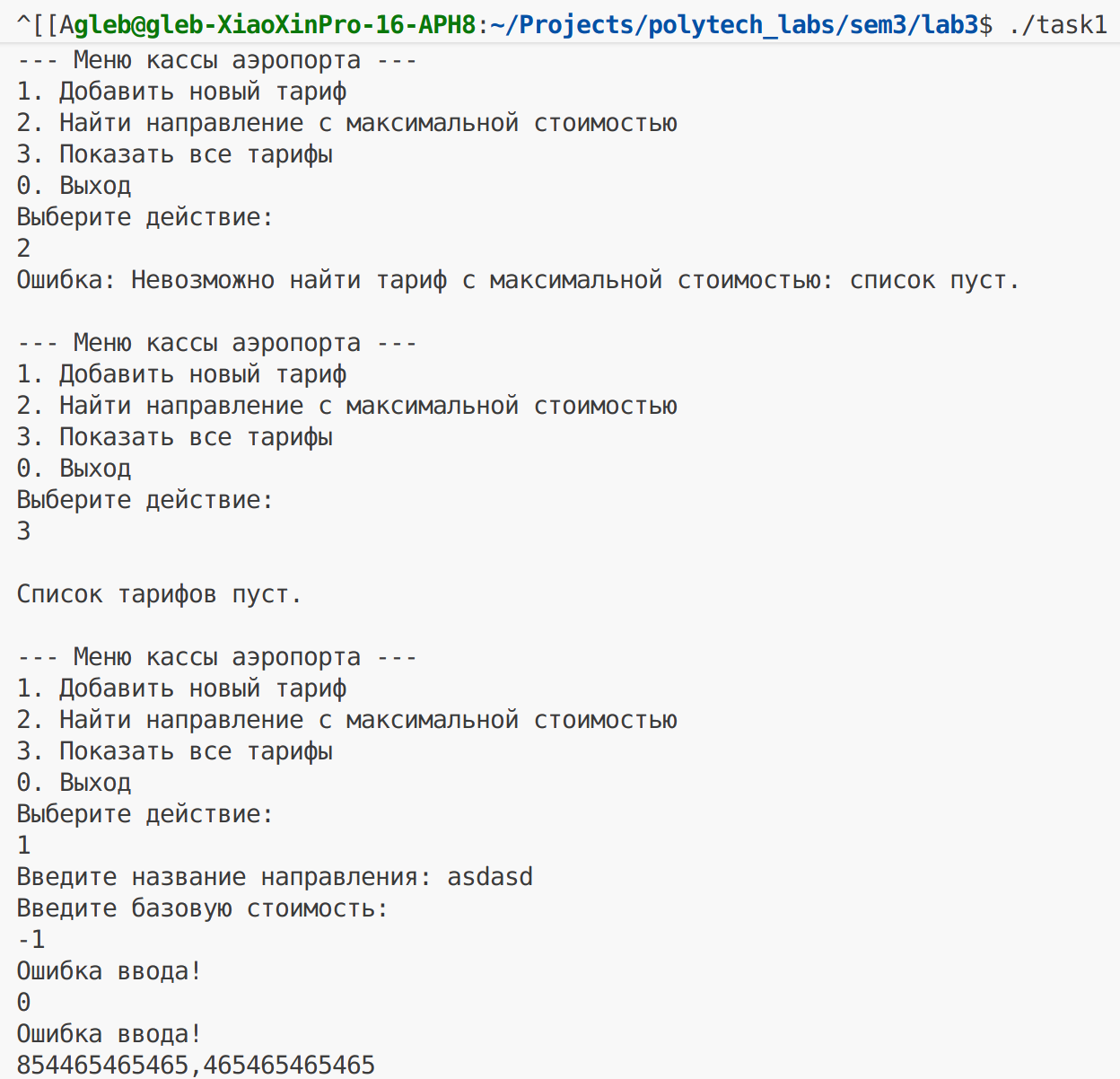


Рисунок 1 — Диаграмма классов в программе

**Пример работы программы**

Рисунок 2 — Пример работы программы, часть 1

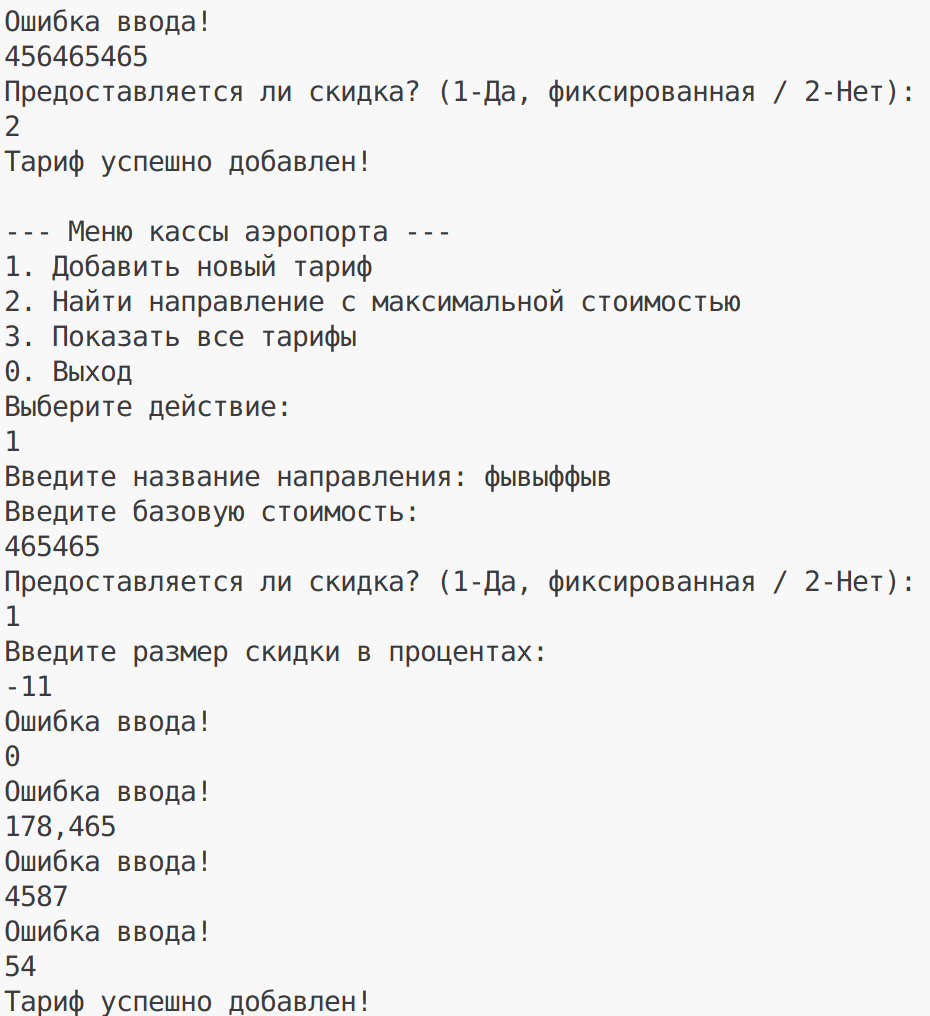


Рисунок 3 — Пример работы программы, часть 2

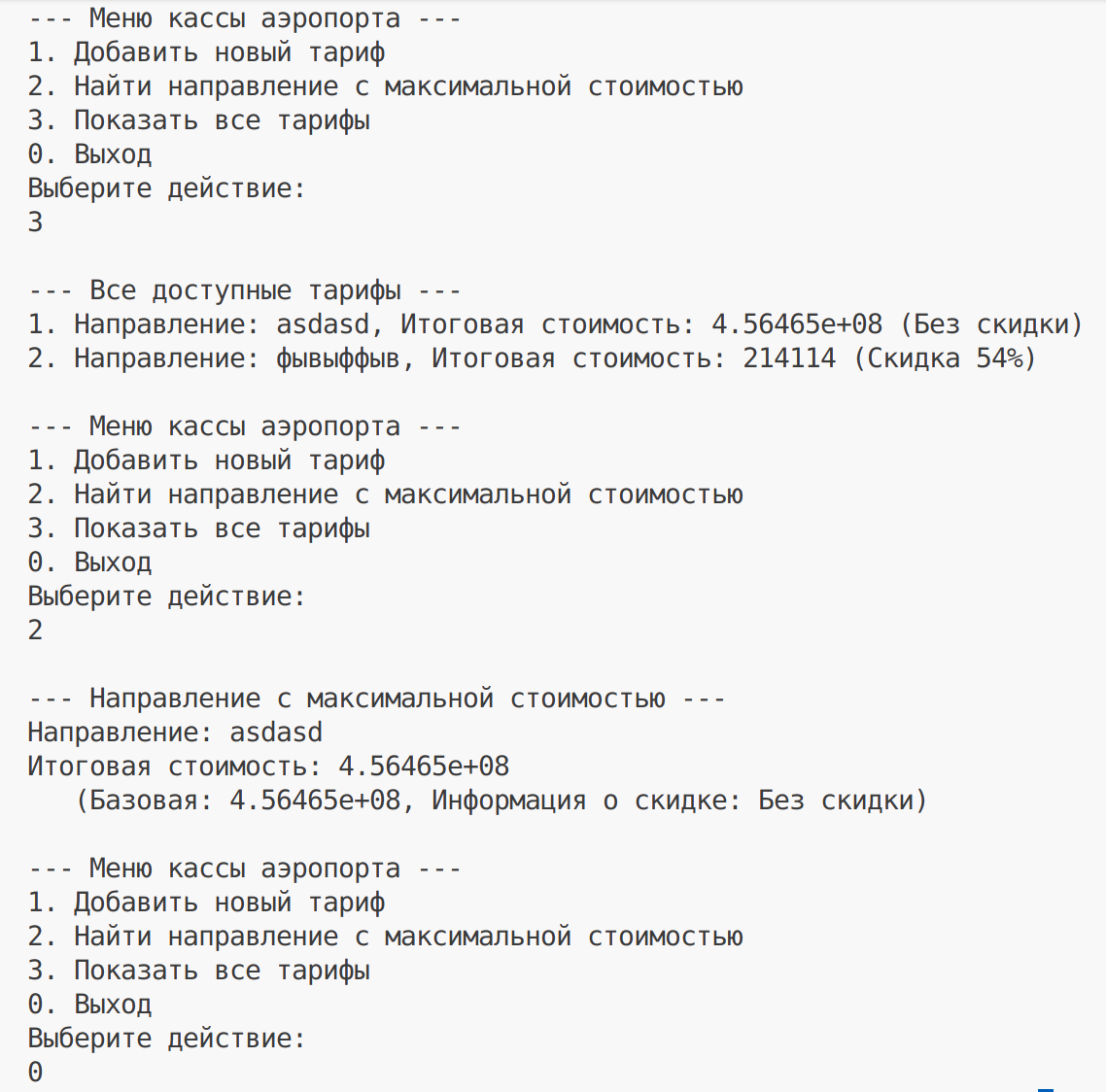


Рисунок 4 — Пример работы программы, часть 3

## **Вывод**

В ходе выполнения лабораторной работы была успешно спроектирована и реализована программа на языке C++, моделирующая кассу аэропорта для управления тарифами. В процессе разработки были применены ключевые принципы объектно-ориентированного программирования, включая наследование, полиморфизм и инкапсуляцию. Центральным архитектурным решением стало использование шаблона проектирования Strategy для обеспечения гибкого управления скидками, что позволило создать расширяемую и модульную структуру кода.

Основу системы составляет класс Airport, управляющий коллекцией тарифов, представленных классом Tariff. Логика расчета итоговой стоимости была вынесена в отдельную иерархию классов, реализующих интерфейс DiscountStrategy. Такой подход позволил отделить алгоритмы расчета скидок от данных о тарифе, благодаря чему система может быть легко дополнена новыми видами скидок без изменения существующего кода. Полиморфизм был реализован через использование виртуальных функций и указателей на базовые классы, что обеспечило единообразную обработку всех тарифов независимо от их типа скидки.