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 № 3**

**Выполнил:**

Cтудент гр. 5130902/30003 Блинков П.В.

**Проверил**

Ст. преподаватель Журавская А.М.

Санкт-Петербург

2024 г.

## **Задание**

Вариант 4

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.

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 <algorithm>

#include <stdexcept>

#include <memory>

#include <string>

using namespace std;

class Tariff

{

private:

int price;

string destination;

int discount;

public:

Tariff(int prc, string dest, int disc = 0) : price(prc), destination(dest), discount(disc) {};

string getDestination() const {

return destination;

}

int getprice() const {

return price;

}

int getdiscount() const {

return discount;

}

void setprice(int new\_price) {

price = new\_price;

}

};

class IStrategy

{

public:

virtual ~IStrategy() = default;

virtual Tariff doAlgorithm(int price, const string& destination) const = 0;

};

class OriginalPrice : public IStrategy

{

Tariff doAlgorithm(int price, const string& destination) const override

{

return Tariff(price, destination);

}

};

class DiscountPrice : public IStrategy

{

private:

int discount;

public:

DiscountPrice(int discount) : discount(discount) {}

Tariff doAlgorithm(int price, const string& destination) const override

{

int discountedPrice = price - discount;

if (discountedPrice < 0) throw invalid\_argument("Discount is greater than price.\n");

return Tariff(discountedPrice, destination, discount);

}

};

class Realisation

{

unique\_ptr<IStrategy> strategy\_;

public:

explicit Realisation(unique\_ptr<IStrategy>&& strategy = {}) : strategy\_(move(strategy)) {}

void set\_strategy(unique\_ptr<IStrategy>&& strategy)

{

strategy\_ = move(strategy);

}

void doChoosenStrategy(int price, const string& destination) const

{

if (strategy\_) {

try {

Tariff result = strategy\_->doAlgorithm(price, destination);

cout << "Price - " << result.getprice() << ", Destination - " << result.getDestination() << endl;

}

catch (const exception& e) {

cout << "Error: " << e.what() << endl;

}

}

else {

cout << "Strategy isn't set \n";

}

}

};

class Airport

{

private:

vector<Tariff> tariffs;

public:

void addTariff(int price, const string& destination, const IStrategy& strategy) {

Tariff newTariff = strategy.doAlgorithm(price, destination);

tariffs.push\_back(newTariff);

}

void displayTariffs() const {

if (tariffs.empty()) {

cout << "No tariffs available.\n";

return;

}

cout << "\nCurrent list of tariffs:\n";

for (const auto& tariff : tariffs) {

cout << "Destination: " << tariff.getDestination()

<< ", Price: " << tariff.getprice();

if (tariff.getdiscount() > 0) {

cout << " (Discount: " << tariff.getdiscount() << ")";

}

cout << endl;

}

}

string Maxcost() const

{

if (tariffs.empty()) return "There are no tariffs.\n";

const Tariff\* maxTariff = &tariffs[0];

for (const auto& tariff : tariffs) {

if (tariff.getprice() > maxTariff->getprice()) {

maxTariff = &tariff;

}

}

return "Price: " + to\_string(maxTariff->getprice()) + ", Destination: " + maxTariff->getDestination();

}

};

int main()

{

Airport airport;

int price;

string destination;

char discountChoice;

OriginalPrice originalStrategy;

DiscountPrice discountStrategy(500);

airport.addTariff(5000, "London", originalStrategy);

airport.addTariff(10000, "Kazakhstan", discountStrategy);

cout << "Welcome to the Airport Tariff System\n";

airport.displayTariffs();

cout << "But you can add more.\n";

while (true) {

cout << "Do you want to add another tariff? (y/n): ";

char choice;

cin >> choice;

if (choice != 'y' && choice != 'Y') break;

cout << "\nEnter the price of the ticket: ";

cin >> price;

if (cin.fail() || price <= 0) {

cout << "Invalid price. Please enter a positive integer.\n";

cin.clear();

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

continue;

}

cout << "Enter the destination: ";

cin >> destination;

cout << "Apply discount? (y/n): ";

cin >> discountChoice;

unique\_ptr<IStrategy> strategy;

if (discountChoice == 'y' || discountChoice == 'Y') {

int discount;

cout << "Enter discount amount: ";

cin >> discount;

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

cout << "Invalid discount. Please enter a non-negative integer.\n";

cin.clear();

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

continue;

}

strategy = make\_unique<DiscountPrice>(discount);

}

else {

strategy = make\_unique<OriginalPrice>();

}

try {

airport.addTariff(price, destination, \*strategy);

cout << "Tariff added successfully!\n";

}

catch (const exception& e) {

cout << "Error adding tariff: " << e.what() << endl;

}

airport.displayTariffs();

}

cout << "\nThe most expensive ticket: " << airport.Maxcost() << endl;

return 0;

}

**Пример работы программы**

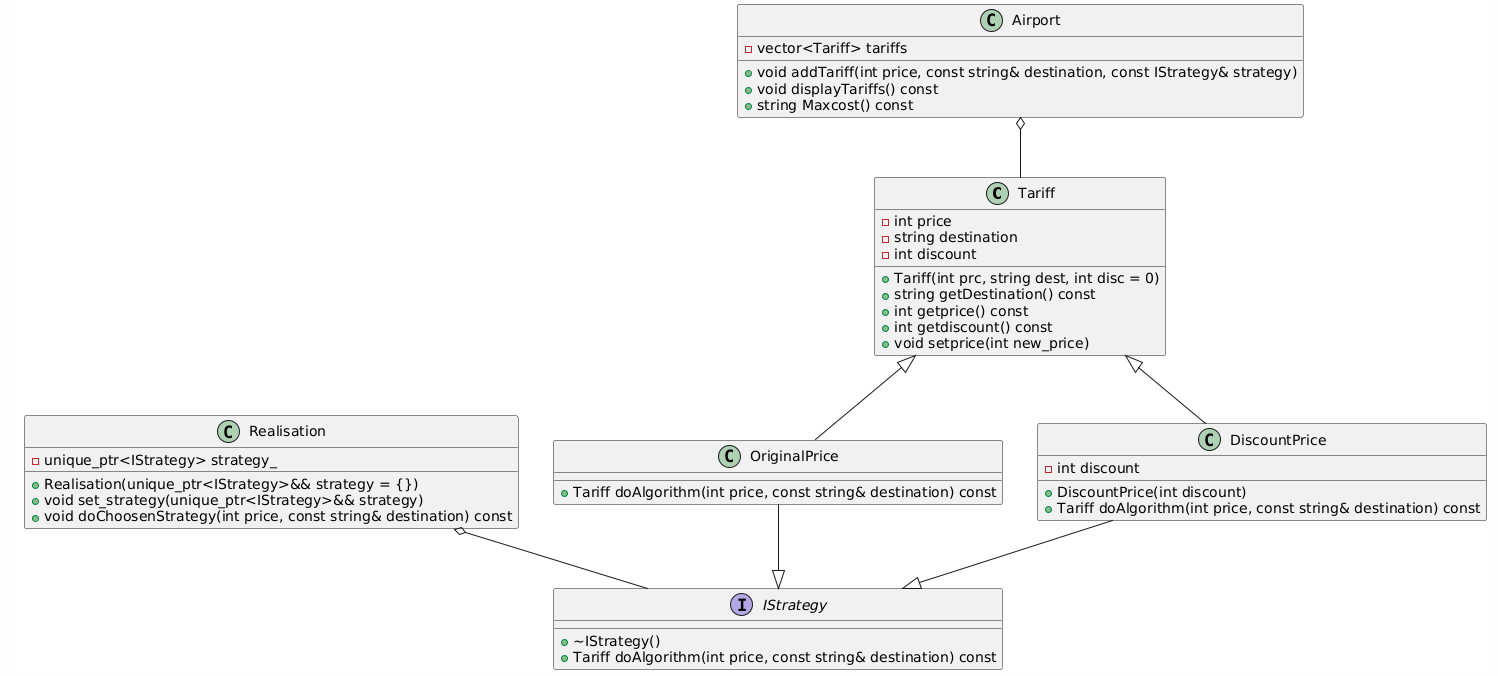


Рисунок 1 – Диаграмма классов

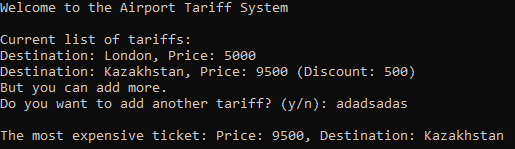


Рисунок 2 – Работа программы при вводе некорректных данных (аналогична вводу n)

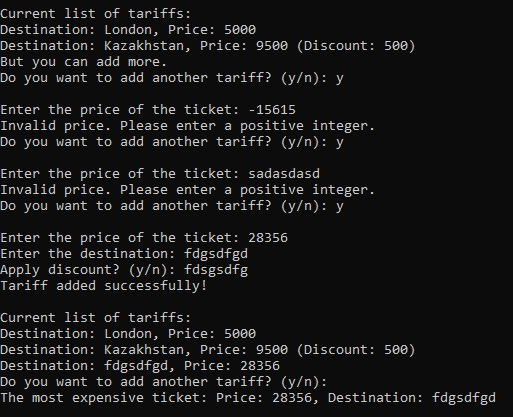


Рисунок 3 – Работа программы при вводе некорректных данных

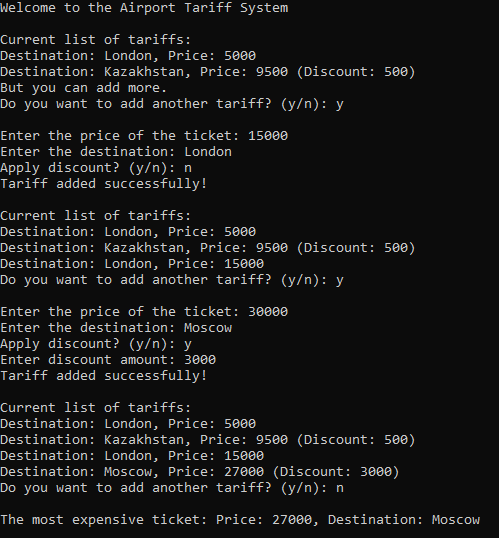


Рисунок 4 – Демонстрация работы программы при вводе корректных данных

## **Вывод**

В ходе выполнения данной работы были приобретены навыки работы с наследованием, полиморфизмом и парадигмой интерфейсных классов, а также получен опыт реализации шаблона проектирования классов Strategy и обработки ошибок с помощью исключения.