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. 5130902/30002 М.Д. Полин

**Проверил**

Ст. преподаватель А.М. Журавская

Санкт-Петербург

2024 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.

7. Фирма – Отдел кадров

**Часть 2**

**(С++, С#,  Java, Python)**

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**

Предметная область: **Фирма грузоперевозок.**

Фирма имеет список тарифов по перевозке грузов. Клиент регистрируется в системе, после чего может заказать перевозку определенного объема груза.

Система должна позволять выполнять следующие задачи:

* ввод тарифов;
* регистрация клиента и заказ на перевозку грузов;
* вывод суммы заказа для определенного клиента;
* подсчет суммарной стоимости всех заказов.

## **Код программы**

#include <iostream>

#include <vector>

#include <string>

**using** **namespace** std;

**class** **Employee** {

**public:**

Employee(**const** string& name, **int** age, **const** string& position)

: name(name), age(age), position(position) {}

**void** displayInfo() **const** {

cout << "Name: " << name << ", Age: " << age << ", Position: " << position << endl;

}

**private:**

string name;

**int** age;

string position;

};

**class** **HRDepartment** {

**public:**

// singl а ниже запретььь

**static** HRDepartment& getInstance() {

**static** HRDepartment instance; // вот оно

**return** instance;

}

**void** addEmployee(**const** Employee& employee) {

employees.push\_back(employee);

}

**void** displayEmployees() **const** {

**for** (**const** **auto**& employee : employees) {

employee.displayInfo();

}

}

**private:**

HRDepartment() {}

HRDepartment(**const** HRDepartment&) = **delete**; // нельзя копировать

HRDepartment& **operator**=(**const** HRDepartment&) = **delete**; // нельзя переприсваивать

vector<Employee> employees;

};

**int** **main**() {

Employee emp1("John Doe", **30**, "Manager");

Employee emp2("Jane Smith", **25**, "Engineer");

Employee emp3("Alice Johnson", **28**, "Designer");

HRDepartment& hr = HRDepartment::getInstance();

hr.addEmployee(emp1);

hr.addEmployee(emp2);

hr.addEmployee(emp3);

cout << "List of Employees:**\n**";

hr.displayEmployees();

**return** **0**;

}

Часть 2

#include <iostream>

#include <string>

#include <vector>

#include <limits>

**using** **namespace** std;

**enum** **class** **TariffType** {

ECONOMY,

BUSINESS,

PREMIUM

};

**class** **Tariff** {

**public:**

Tariff(string name, **double** price, TariffType type)

: name(name), price(price), type(type) {}

string getName() **const** {

**return** name;

}

**double** getPrice() **const** {

**return** price;

}

TariffType getType() **const** {

**return** type;

}

**private:**

string name;

**double** price;

TariffType type;

};

**class** **Client** {

**public:**

Client(string name) : name(name) {}

string getName() **const** {

**return** name;

}

**private:**

string name;

};

**class** **Order** {

**public:**

Order(Client\* client, Tariff\* tariff, **double** volume)

: client(client), tariff(tariff), volume(volume) {}

**double** calculateTotal() **const** {

**return** volume \* tariff->getPrice();

}

Client\* getClient() **const** {

**return** client;

}

Tariff\* getTariff() **const** {

**return** tariff;

}

**double** getVolume() **const** {

**return** volume;

}

**private:**

Client\* client;

Tariff\* tariff;

**double** volume;

};

**class** **TransportCompany** {

**public:**

**void** addTariff(**const** string& name, **double** price, TariffType type) {

tariffs.push\_back(**new** Tariff(name, price, type));

}

**void** registerClient(**const** string& name) {

clients.push\_back(**new** Client(name));

}

**void** makeOrder(**const** string& clientName, **const** string& tariffName, **double** volume) {

Client\* client = findClientByName(clientName);

**if** (!client) {

**throw** runtime\_error("Клиент не найден!");

}

Tariff\* tariff = findTariffByName(tariffName);

**if** (!tariff) {

**throw** runtime\_error("Тариф не найден!");

}

orders.push\_back(**new** Order(client, tariff, volume));

}

**double** calculateTotalForClient(**const** string& clientName) **const** {

Client\* client = findClientByName(clientName);

**if** (!client) {

**throw** runtime\_error("Клиент не найден!");

}

**double** total = **0**;

**for** (**const** **auto**& order : orders) {

**if** (order->getClient() == client) {

total += order->calculateTotal();

}

}

**return** total;

}

**double** calculateTotalForAllOrders() **const** {

**double** total = **0**;

**for** (**const** **auto**& order : orders) {

total += order->calculateTotal();

}

**return** total;

}

**void** removeClient(**const** string& clientName) {

**for** (**auto** it = clients.begin(); it != clients.end(); ++it) {

**if** ((\*it)->getName() == clientName) {

**delete** \*it;

clients.erase(it);

cout << "Клиент " << clientName << " удален." << endl;

**return**;

}

}

cout << "Клиент " << clientName << " не найден." << endl;

}

**void** removeTariff(**const** string& tariffName) {

**for** (**auto** it = tariffs.begin(); it != tariffs.end(); ++it) {

**if** ((\*it)->getName() == tariffName) {

**delete** \*it;

tariffs.erase(it);

cout << "Тариф " << tariffName << " удален." << endl;

**return**;

}

}

cout << "Тариф " << tariffName << " не найден." << endl;

}

**void** listClients() **const** {

**if** (clients.empty()) {

cout << "Список клиентов пуст." << endl;

**return**;

}

cout << "Список клиентов:" << endl;

**for** (**const** **auto**& client : clients) {

cout << client->getName() << endl;

}

}

**void** listTariffs() **const** {

**if** (tariffs.empty()) {

cout << "Список тарифов пуст." << endl;

**return**;

}

cout << "Список тарифов:" << endl;

**for** (**const** **auto**& tariff : tariffs) {

cout << "Тариф: " << tariff->getName() << ", Цена: " << tariff->getPrice() << endl;

}

}

~TransportCompany() {

**for** (**auto** t : tariffs) **delete** t;

**for** (**auto** c : clients) **delete** c;

**for** (**auto** o : orders) **delete** o;

}

**private:**

vector<Tariff\*> tariffs;

vector<Client\*> clients;

vector<Order\*> orders;

Client\* **findClientByName**(**const** string& name) **const** {

**for** (**const** **auto**& client : clients) {

**if** (client->getName() == name) {

**return** client;

}

}

**return** nullptr;

}

Tariff\* findTariffByName(**const** string& name) **const** {

**for** (**const** **auto**& tariff : tariffs) {

**if** (tariff->getName() == name) {

**return** tariff;

}

}

**return** nullptr;

}

};

**void** **displayTariff**(**const** Tariff\* tariff) {

cout << "Тариф: " << tariff->getName() << ", Цена за единицу: "

<< tariff->getPrice() << endl;

}

**void** **displayOrder**(**const** Order\* order) {

cout << "Клиент: " << order->getClient()->getName() << " заказал "

<< order->getVolume() << " единиц груза. Общая стоимость: "

<< order->calculateTotal() << endl;

}

**void** **displayMenu**() {

cout << "Меню:" << endl;

cout << "1. Добавить тариф" << endl;

cout << "2. Зарегистрировать клиента" << endl;

cout << "3. Оформить заказ" << endl;

cout << "4. Вывести общую сумму для клиента" << endl;

cout << "5. Вывести общую сумму всех заказов" << endl;

cout << "6. Удалить тариф" << endl;

cout << "7. Удалить клиента" << endl;

cout << "8. Просмотреть список клиентов" << endl;

cout << "9. Просмотреть список тарифов" << endl;

cout << "10. Выход" << endl;

}

**int** **main**() {

TransportCompany company;

**int** choice;

**do** {

displayMenu();

cout << "Введите ваш выбор: ";

**while** (!(cin >> choice) || choice < **1** || choice > **10**) {

cout << "Ошибка! Введите корректный выбор (от 1 до 10): ";

cin.clear();

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

}

**switch** (choice) {

**case** **1**: {

string name;

**double** price;

**int** typeInt;

TariffType type;

cout << "Введите название тарифа: ";

cin >> name;

cout << "Введите цену за единицу: ";

**while** (!(cin >> price) || price <= **0**) {

cout << "Ошибка! Введите корректную цену (число больше нуля): ";

cin.clear();

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

}

cout << "Выберите тип тарифа (0 - Economy, 1 - Business, 2 - Premium): ";

**while** (!(cin >> typeInt) || typeInt < **0** || typeInt > **2**) {

cout << "Ошибка! Введите корректное значение (0, 1 или 2): ";

cin.clear();

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

}

type = **static\_cast**<TariffType>(typeInt);

company.addTariff(name, price, type);

**break**;

}

**case** **2**: {

string name;

cout << "Введите имя клиента: ";

cin >> name;

company.registerClient(name);

**break**;

}

**case** **3**: {

string clientName, tariffName;

**double** volume;

cout << "Введите имя клиента: ";

cin >> clientName;

cout << "Введите название тарифа: ";

cin >> tariffName;

cout << "Введите объем груза: ";

**while** (!(cin >> volume) || volume <= **0**) {

cout << "Ошибка! Введите корректный объем (число больше нуля): ";

cin.clear();

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

}

try {

company.makeOrder(clientName, tariffName, volume);

} **catch** (**const** exception& e) {

cout << e.what() << endl;

}

**break**;

}

**case** **4**: {

string clientName;

cout << "Введите имя клиента: ";

cin >> clientName;

try {

**double** total = company.calculateTotalForClient(clientName);

cout << "Общая стоимость для клиента " << clientName << ": " << total << endl;

} **catch** (**const** exception& e) {

cout << e.what() << endl;

}

**break**;

}

**case** **5**: {

**double** total = company.calculateTotalForAllOrders();

cout << "Общая стоимость всех заказов: " << total << endl;

**break**;

}

**case** **6**: {

string tariffName;

cout << "Введите название тарифа для удаления: ";

cin >> tariffName;

company.removeTariff(tariffName);

**break**;

}

**case** **7**: {

string clientName;

cout << "Введите имя клиента для удаления: ";

cin >> clientName;

company.removeClient(clientName);

**break**;

}

**case** **8**: {

company.listClients();

**break**;

}

**case** **9**: {

company.listTariffs();

**break**;

}

**case** **10**:

cout << "Выход из программы..." << endl;

**break**;

}

} **while** (choice != **10**);

**return** **0**;

}

**Пример работы программы**

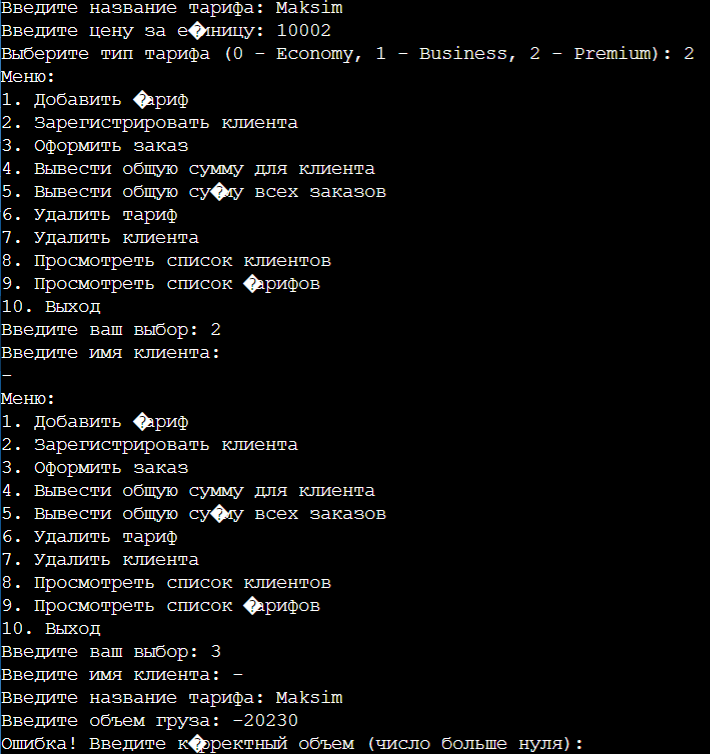


Рисунок 1 – Правильная работа программы

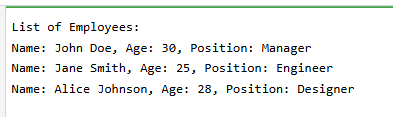


Рисунок 2 – Взаимодействие с единственным экземпляром отдела кадров

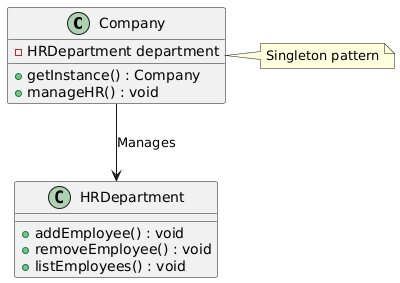


Рисунок 3 – Диаграмма первой части

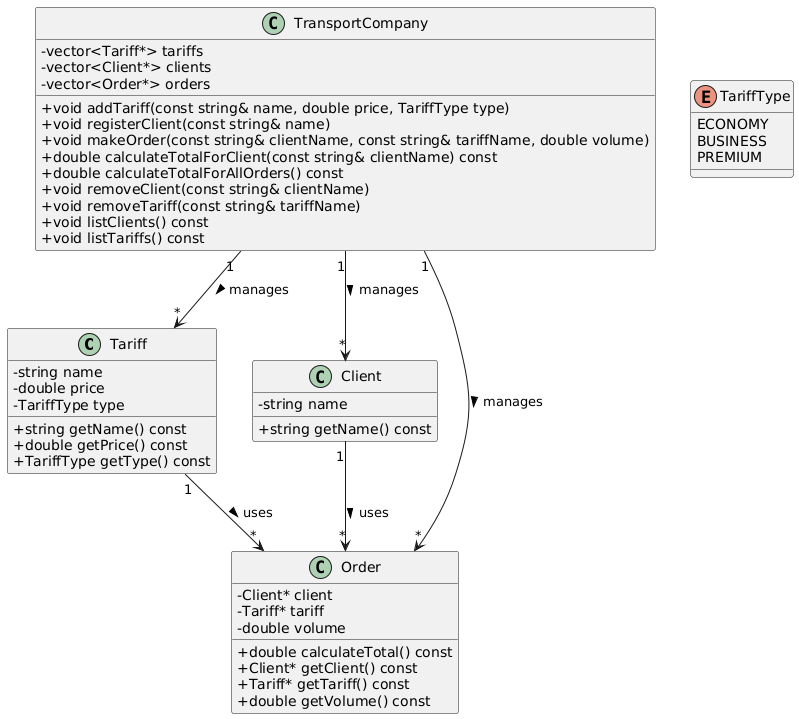


Рисунок 4 – Диаграмма второй части

## **Вывод**

В ходе лабораторной работы был реализованы два мини-проекта, содержащие в себе от 2 до 4 классов для их непосредственного взаимодействия. Применен шаблон проектирования Singleton, который необходим для однозначного объявления экземпляра класса в единственном варианте.