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;

}

void displayTariff() const {

cout << "Тариф: " << name << ", Цена за единицу: " << price << endl;

}

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();

}

void displayOrder() const {

cout << "Клиент: " << client->getName() << " заказал "

<< volume << " единиц груза. Общая стоимость: "

<< calculateTotal() << endl;

}

Client\* getClient() const {

return client;

}

private:

Client\* client;

Tariff\* tariff;

double volume;

};

class TransportCompany {

public:

void addTariff() {

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);

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

}

void registerClient() {

string name;

cout << "Введите имя клиента: ";

cin >> name;

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

}

void makeOrder() {

string clientName, tariffName;

double volume;

cout << "Введите имя клиента: ";

cin >> clientName;

Client\* client = findClientByName(clientName);

if (!client) {

cout << "Клиент не найден!" << endl;

return;

}

cout << "Введите название тарифа: ";

cin >> tariffName;

Tariff\* tariff = findTariffByName(tariffName);

if (!tariff) {

cout << "Тариф не найден!" << endl;

return;

}

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');

}

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

}

void displayTotalForClient(const string& clientName) const {

Client\* client = findClientByName(clientName);

if (!client) {

cout << "Клиент не найден!" << endl;

return;

}

double total = 0;

for (const auto& order : orders) {

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

total += order->calculateTotal();

}

}

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

}

void displayTotalForAllOrders() const {

double total = 0;

for (const auto& order : orders) {

total += order->calculateTotal();

}

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

}

int main() {

TransportCompany company;

int choice;

do {

displayMenu();

cout << "Введите ваш выбор: ";

while (!(cin >> choice) || choice < 1 || choice > 6) {

cout << "Ошибка! Введите корректный выбор (от 1 до 6): ";

cin.clear();

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

}

switch (choice) {

case 1:

company.addTariff();

break;

case 2:

company.registerClient();

break;

case 3:

company.makeOrder();

break;

case 4: {

string clientName;

cout << "Введите имя клиента: ";

cin >> clientName;

company.displayTotalForClient(clientName);

break;

}

case 5:

company.displayTotalForAllOrders();

break;

case 6:

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

break;

}

} while (choice != 6);

return 0;

}

**Пример работы программы**

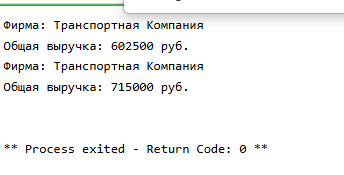
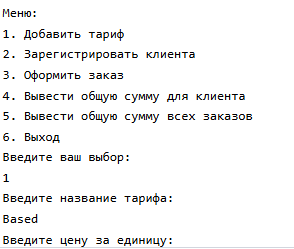


Рисунок 1 – Правильная работа программы

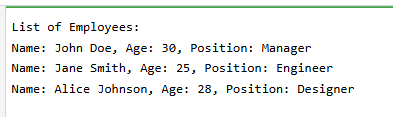


Рисунок 2 – Взаимодействие с единственным экземпляром отдела кадров

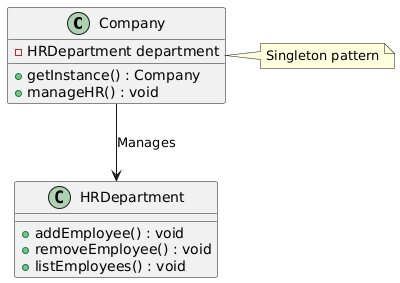


Рисунок 3 – Диаграмма первой части

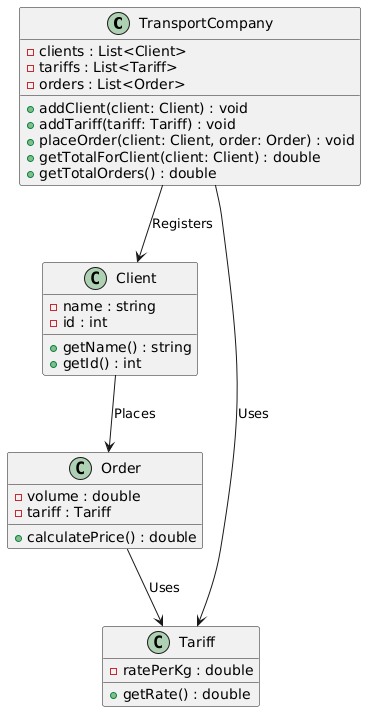


Рисунок 4 – Диаграмма второй части

## **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.