**ФГБОУ ВО**

**Национальный исследовательский университет**

**«МЭИ»**

КУРСОВАЯ РАБОТА

по дисциплине

«Объектно-ориентированный анализ и программирование»

**Разработка объектно-ориентированной программы на языке С++**

**Вариант №16**

**Выполнил:**

студент: Сперанский Д.А

группа: ИЭ-61-21

**Москва, 2022**

# ВВЕДЕНИЕ

Ознакомиться с концепцией объектно-ориентированного программирования, согласно которой абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, а именно ознакомиться с наследованием.

## Задание

Реализовать иерархию классов в соответствии с вариантом. У всех вариантов в качестве базового класса выступает класс “Сотрудник”. Построить диаграмму классов.  
Примечание: если в задании требуется реализовать функционал для другого класса, которого нет в вашем варианте, делать этого не требуется.

Реализовать дочерний класс на основе двух классов, реализованных в предыдущем задании 1.1.  
Рассмотрите полученный результат и скажите, какие проблемы возникли при реализации данного множественного наследования (прим.: "алмаз смерти").  
Создать новые классы "Инициалы" и "Выплаты", на основе которых реализовать дочерний класс "Данные о работнике".   
Продемонстрировать работу класса.  
В случае некорректных данных выбрасывать исключения.

## Используемые классы согласно заданию

***0. Класс Директор:***

Должен содержать имя, фамилию, отчество и зарплату. Данные поля должны находиться в закрытой области класса.

Также класс должен содержать поле, содержащее ФИО сотрудников, которые находятся в подчинении у директора, и их заработную плату.

Реализовать методы для увольнения и принятия работников.

Реализовать методы, позволяющие читать/писать из/в полей класса.

*2.* ***Класс Секретарь***:

Должен содержать имя, фамилию, отчество и зарплату. Данные поля должны находиться в закрытой области класса.

Реализовать метод, который для данного директора выводит в виде таблицы список сотрудников.

Если в вашем варианте есть класс “Системный администратор”, то реализовать метод, который для данного системного администратора выводит в виде таблицы список оборудования.

Если в вашем варианте есть класс “Охранник”, то реализовать метод, который принимает массив охранников и выводит их в таблицу.

*3.* ***Класс Водитель****:*

Должен содержать имя, фамилию, отчество и зарплату. Данные поля должны находиться в закрытой области класса.

Класс должен содержать поле, которое хранит в себе массив из категорий прав.

Также должен содержать поле, содержащее массив транспортных средств, которыми управляет водитель.

Реализовать методы, позволяющие читать/писать из/в полей класса.

***4. Класс Программист:***

Должен содержать имя, фамилию, отчество и зарплату. Данные поля должны находиться в закрытой области класса.

Класс должен содержать поле, которое хранит уровень программиста (junior, middle, senior, lead).

Класс должен содержать поле, которое будет содержать список технологий, которыми владеет программист.

Реализовать методы, позволяющие читать/писать из/в полей класса.

# Постановка задачи

Консольное приложение, написанное в среде разработки <<Microsoft Visual Studio>>.

## Разработка программы

### Описание классов, полей и методов классов

#### Рассмотрение основных моментов

Для начала рассмотрения реализации, стоит уточнить, что код классов выполнялся по второму заданию изначально. То есть имеется два класса Initials и Payments на основе которых создан дочерний класс Employee. Этот код не сложно изменить для представления кода по 1-ому заданию, для этого было бы необходимо просто переопределить методы классов Initials и Payments в Employee.

Для дальнейшей успешной разработки стоит разобраться с таким явлением как «Алмаз смерти».

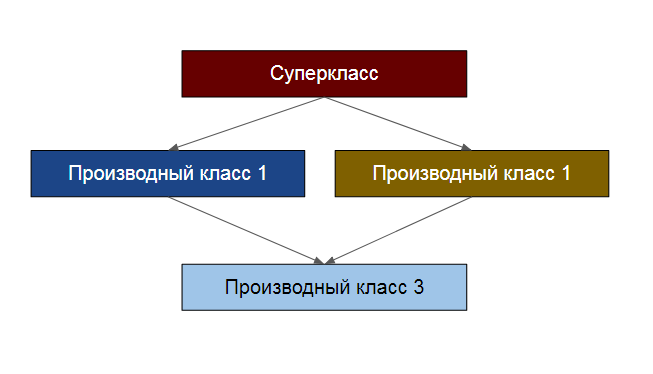


Рисунок -"Алмаз смерти"

Рассмотрим ситуацию: один производный класс имеет два базовых класса, при этом каждый из этих двух являются производным одного и того же суперкласса, такая ситуация демонстрируется на Рис.1. При этом в производных классах есть свойство с одинаковым именем.

Как мы знаем, если в базовых классах элементы называются одинаково, то в производном классе появляется ошибка неоднозначности. Вот чем сулит «алмаз смерти», а [название он свое получил](https://topuch.com/kursovaya-rabota-po-teme-svojstva-adamantana-vipolnili-st-gr-h/index.html), так как на диаграмме классов такая ситуация выглядит как алмаз.

### Описание разработанных классов

Класс **Initials**:

*Поля*

Char c1 – переменная для хранения 1-й буквы фамилии

Char c2 – переменная для хранения 1-й буквы имени

*Взаимодействие*

Является родительским классом для Employee

Класс **payments**:  
Поля

int amount– поле со значением выплаты

*Взаимодействие*

Является родительским классом для Employee

Класс **Employee**:  
Поля

string Name – хранит имя работника

string Surname – хранит фамилию работника

string MidName – хранит отчество работника

double salary – содержит сведения о зарплате

int position – хранит должность работника

*Методы*

Задаются базовые для всех классов данные

void read\_basic\_info() {

Surname = input\_name(0);

Name = input\_name(1);

MidName = input\_name(2);

Salary = input\_salary();

}

string GET\_Name() { return Name; } – возвращает имя

string GET\_Surname() { return Surname; } – возвращает фамилию

string GET\_MidName() { return MidName; } – возвращает отчество

double GET\_Salary() { return Salary; } – возвращает зарплату

int GET\_Position() { return Position; } – возвращает должность

*Взаимодействие*

Является дочерним классом для Initials и Payments, а также в свою очередь является родительским для классов Director, Driver, Secretary и Programmer

Класс **Director**:  
*Поля*

vector < Employee> employees – вектор каждого подчиненного директора

vector<Driver> drivers – вектор всех водителей

vector<Programmist> programmists– вектор всех системных программистов

vector< Secretary> secretaries– вектор всех секретарей

Методы

void hire(void\* data, int pos) – создает нового работника у конкретного директора

void fire(Employee c) – увольнение сотрудника

*Взаимодействие*

Является дочерним классом для Employee

Класс **Secretary**:

*Поля*

*— нет*

Методы

void print\_programmers(vector <Programmist>& p) – выводит всех программистов

Взаимодействие

Является дочерним классом для Employee

Класс **Programmist**:  
Поля

vector<string> languages – массив для записи языков программирования

string level – поле для записи уровня программиста

Методы

void set\_level(string level) – позволяет установить уровень программиста

void set\_languages(vector <string> l) – позволяет установить языки программирования

void add\_languages(string c) – позволяет добавить языки программирования

Взаимодействие

Является дочерним классом для Employee

Класс **Driver**:

Поля

vector<string> category – вектор хранит в себе данные о текущих категориях прав

vector<string> vehicle – вектор хранит в себе данные о транспортных средствах водителя

Методы

void add\_category(string c) – добавляет категорию прав

void add\_vehicle(string c) – добавляет транспортные средства

Взаимодействие

Является дочерним классом для Employee

## Описание пользовательского интерфейса

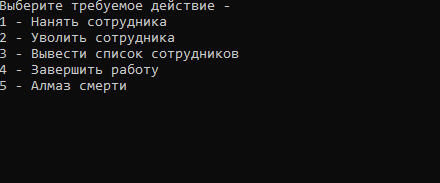
Взаимодействие пользователя с программой осуществляется с помощью диалога. На Рис.4 показано как как выглядит главное меню:

Рисунок – главное меню

На Рис.5 показан пример взаимодействия

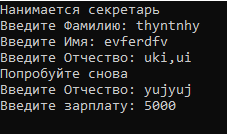


Рисунок 5– пример взаимодействия с диалоговым окном

**Реализация и тестирование программы**

**Описание разработанной программы**

С листингом программы можно ознакомиться в ПРИЛОЖЕНИЕ 1. Код представляет собой CLI приложение, в котором представлен некоторый менеджер сотрудников компании.

**Тестирование программы**

Проверим программу при нормальных условиях результатыпредставленынаРис.с 6поРис.9

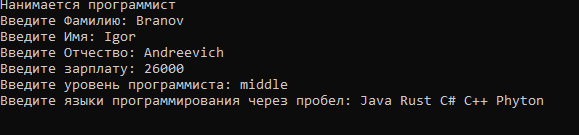


Рисунок 6

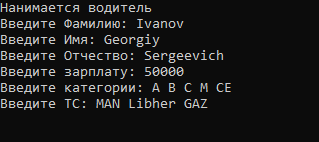


Рисунок 7

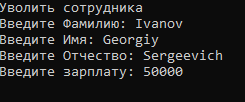


Рисунок 8

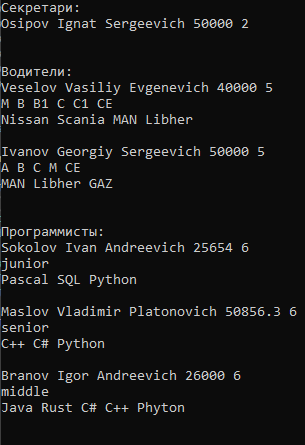


Рисунок 9

Тест если в имя вводим цифры результат представлен на Рис.10

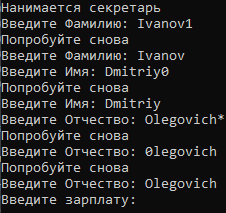


Рисунок 10 - тест на правильность ввода

Тест если ввести не верное значение зарплаты результат представлен на Рис.11

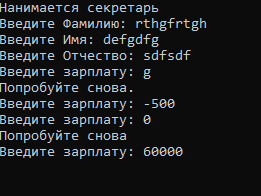


Рисунок 11 – не верное значение зарплаты

### «Алмаз смерти» демонстрация проблемы

Для демонстрирования проблемы множественного наследования добавим новый класс class UberProgrammer : public Programmist, public Driver {};

Полный листинг программы находится в ПРИЛОЖЕНИЕ 1, однако данные строки закомментированы для нормальной реализации кода, здесь же это показано для наглядности проблемы «алмаз смерти».

При компилировании программы, как и ожидалось, мы видим следующую ошибку Рис.12:



Рисунок 12 – Алмаз смерти

Что мы и ожидали исходя из теоретической части, что при создании объектов класса UberProgrammer создаются в начале объекты классов Programmist, а также Driver, в каждой из них присутствует метод read\_basic\_info(), что и порождает данную неопределенность.

# ВЫВОД

В данной работе произвелось знакомство с разработкой приложений с использованием ООП. Познакомились с механизмом инкапсуляции и наследования. Помимо простой работы с классами наследования были протестированы частые ошибки программистов такие как множественное наследование или «алмаз смерти». Рассмотрение данной ошибки поможет при дальнейшей работе, ведь работая над проектом в одиночку если человек и столкнется с данной проблемой, то без особых трудностей исправит ее, но работая в команде эта ошибка сулит очень большие проблемы.

В ходе тестирования разработанной программы ошибок не было выявлено, все работает должным образом.

## ПРИЛОЖЕНИЕ 1

### Меню программы

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "Director.h"

#include "Employee.h"

#include "Driver.h"

#include "Secretary.h"

#include "Programmist.h"

#include "Functions.h"

#include "UberProgrammer.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

int x = 0;

string name, surname, midname;

double salary;

int position;

Director n = load\_from\_file("text");

Employee h;

Employee f;

while (1) {

cout << "Выберите требуемое действие - " << endl;

cout << "1 - Нанять сотрудника" << endl;

cout << "2 - Уволить сотрудника" << endl;

cout << "3 - Вывести список сотрудников" << endl;

cout << "4 - Завершить работу" << endl;

cout << "5 - Алмаз смерти" << endl;

cout << endl;

cin >> x;

system("cls");

if (x == 1) {

while (1) {

int z;

cout << "Нанимается - " << endl;

cout << "1 - Секретарь" << endl;

cout << "2 - Программист" << endl;

cout << "3 - Водитель" << endl;

cout << "4 - Выход" << endl;

cout << endl;

cin >> z;

if (z == 1) {

system("cls");

cout << "Нанимается секретарь" << endl;

h.read\_basic\_info();

h.set\_position(2);

n.hire(&h, h.GET\_Position());

system("cls");

};

if (z == 2) {

system("cls");

cout << "Нанимается программист" << endl;

Programmist p;

string lv;

vector <string> languages;

string l;

p.read\_basic\_info();

p.set\_position(6);

lv = p.input\_level();

cout << "Введите языки программирования через пробел: ";

clear();

getline(cin, l);

cout << endl;

p.set\_level(lv);

p.set\_languages(split(l, " "));

n.hire(&p, p.GET\_Position());

system("cls");

}

if (z == 3) {

system("cls");

cout << "Нанимается водитель" << endl;

Driver d;

vector <string> cat;

string s;

vector <string> vehicle;

string v;

d.read\_basic\_info();

d.set\_position(5);

cout << "Введите категории: ";

clear();

getline(cin, s);

cout << "Введите ТС: ";

getline(cin, v);

d.set\_categories(split(s, " "));

d.set\_vehicle(split(v, " "));

n.hire(&d, d.GET\_Position());

system("cls");

}

if (z == 4) {

system("cls");

break;

}

}

}

if (x == 2) {

int x;

cout << "Уволить сотрудника" << endl;

f.read\_basic\_info();

cout << "Введите должность: ";

cin >> x;

f.set\_position(x);

n.fire(f);

system("cls");

}

if (x == 3) {

cout << "Секретари: ";

cout << endl;

for (Employee e : n.employees) {

e.print();

cout << endl;

};

cout << endl;

cout << "Водители: ";

cout << endl;

for (Driver d : n.drivers) {

d.print();

d.print\_categories();

d.print\_vehicle();

cout << endl;

};

cout << endl;

cout << "Программисты: ";

cout << endl;

for (Programmist p : n.programmists) {

p.print();

cout << p.GET\_level();

cout << endl;

p.print\_languages();

cout << endl;

}

cout << endl;

};

if (x == 4) {

break;

system("cls");

};

/\*if (x == 5) {

UberProgrammer u;

system("cls");

cout << "Нанимается Uber" << endl;

u.read\_basic\_info();

u.set\_data('B', 'G', 2800);

n.hire(&h, h.GET\_Position());

system("cls");

//Так как класс Программист и класс Водитель имеют одинаковый метод с одним название, программа не может понять какой именно метод должен использоваться

};\*/

};

save\_to\_file(n, "text");

}

### Класс директор

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "io.h"

#include "Employee.h"

#include "Driver.h"

#include "Secretary.h"

#include "Programmist.h"

using namespace std;

class Director : public Employee {

public:

vector<Employee> employees;

vector<Driver> drivers;

vector<Programmist> programmists;

vector<Secretary> Secretaries;

void hire(void\* data, int pos) {

if (pos == 2) {

Employee\* e = (Employee\*)data;

employees.push\_back(\*e);

}

if (pos == 5) {

Driver\* d = (Driver\*)data;

drivers.push\_back(\*d);

}

if (pos == 6) {

Programmist\* p = (Programmist\*)data;

programmists.push\_back(\*p);

}

};

void fire(Employee c) {

int num = -1;

if (c.GET\_Position() == 2) {

for (int i = 0; i < employees.size(); i++) {

if (c == employees[i]) {

num = i;

};

};

if (num != -1) {

employees.erase(employees.begin() + num);

}

else {

cout << "ERROR";

return;

};

}

if (c.GET\_Position() == 5) {

for (int i = 0; i < drivers.size(); i++) {

if (c == drivers[i]) {

num = i;

};

};

if (num != -1) {

drivers.erase(drivers.begin() + num);

}

else {

cout << "ERROR";

return;

};

}

if (c.GET\_Position() == 6) {

for (int i = 0; i < programmists.size(); i++) {

if (c == programmists[i]) {

num = i;

};

};

if (num != -1) {

programmists.erase(programmists.begin() + num);

}

else {

cout << "ERROR";

return;

};

}

};

};

### Класс секретарь

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "io.h"

#include "Employee.h"

using namespace std;

class Programmist : public Employee {

vector<string> languages;

string level;

public:

string GET\_level() { return level; };

vector<string> GET\_languages() { return languages; };

void write\_to\_file(ofstream& f) {

Employee::write\_to\_file(f);

f << this->level << endl;

f << languages.size() << endl;

for (string l : languages)

f << l << endl;

};

void set\_level(string level) {

this->level = level;

};

void add\_language(string c) {

languages.push\_back(c);

}

void set\_languages(vector <string> l) {

languages = l;

}

void print\_languages() {

for (string c : languages)

cout << c << " ";

cout << endl;

}

string input\_level() {

string y;

while (1) {

cout << "Введите уровень программиста: ";

cin >> y;

if (y == "junior" || y == "middle" || y == "senior") {

return y;

}

else {

cout << "Попробуйте снова" << endl;

}

}

}

};

### Класс водитель

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "io.h"

#include "Employee.h"

using namespace std;

class Driver : public Employee {

vector<string> category;

vector<string> vehicle;

public:

void write\_to\_file(ofstream& f) {

Employee::write\_to\_file(f);

f << category.size() << endl;

for (string c : category)

f << c << endl;

f << vehicle.size() << endl;

for (string v : vehicle)

f << v << endl;

};

void add\_category(string c) {

category.push\_back(c);

}

void set\_categories(vector <string> v) {

category = v;

}

void print\_categories() {

for (string c : category)

cout << c << " ";

cout << endl;

}

void add\_vehicle(string c) {

vehicle.push\_back(c);

}

void set\_vehicle(vector <string> v) {

vehicle = v;

}

void print\_vehicle() {

for (string v : vehicle)

cout << v << " ";

cout << endl;

}

};

### Класс программист

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "io.h"

#include "Employee.h"

using namespace std;

class Programmist : public Employee {

vector<string> languages;

string level;

public:

string GET\_level() { return level; };

vector<string> GET\_languages() { return languages; };

void write\_to\_file(ofstream& f) {

Employee::write\_to\_file(f);

f << this->level << endl;

f << languages.size() << endl;

for (string l : languages)

f << l << endl;

};

void set\_level(string level) {

this->level = level;

};

void add\_language(string c) {

languages.push\_back(c);

}

void set\_languages(vector <string> l) {

languages = l;

}

void print\_languages() {

for (string c : languages)

cout << c << " ";

cout << endl;

}

string input\_level() {

string y;

while (1) {

cout << "Введите уровень программиста: ";

cin >> y;

if (y == "junior" || y == "middle" || y == "senior") {

return y;

}

else {

cout << "Попробуйте снова" << endl;

}

}

}

};

### Функции проверки ввода

#include <io.h>

#include <iostream>

#include <string>

using namespace std;

void clear()

{

// очищаем входной поток, чтобы заново не читать этот мусор

char ch;

int i = 0;

while ((ch = getchar()) != '\n' && ch != EOF)

i++;

}

string input\_name(int t) {

string x;

string s = "Фамилию";

if (t == 1)

s = "Имя";

if (t == 2)

s = "Отчество";

while (1) {

cout << "Введите " << s << ": ";

cin >> x;

//abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

if (x.find\_first\_of("АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзийклмнопрстуфхцчшщъыьэюя!\"'@#$%^&\*()\_+-№;:.,{}[]|/<>?0123456789") == string::npos) {

return x;

}

else {

cout << "Попробуйте снова" << endl;

}

}

}

double input\_salary() {

double y;

while (cout << "Введите зарплату: " && !(cin >> y)) {

cin.clear();

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

cout << "Попробуйте снова.\n";

}

if (y <= 0) {

while (1) {

cout << "Введите зарплату: ";

cin >> y;

if (y > 0) {

return y;

}

else {

cout << "Попробуйте снова" << endl;

}

}

} else

return y;

}

### Основной класс Employee

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "io.h"

using namespace std;

class Initials {

public:

char c1;

char c2;

};

class Payments {

public:

int amount;

};

class EmployeeData : public Initials, public Payments {

};

class Employee : public EmployeeData {

protected:

string Name;

string Surname;

string MidName;

double Salary;

int Position;

public:

void set\_data(char c1, char c2, int amount) {

this->c1 = c1;

this->c2 = c2;

this->amount = amount;

};

void print\_data() {

cout << get\_c1() << ' ';

cout << get\_c2() << ' ';

cout << get\_payments() << ' ';

};

void SET(string Name, string Surname, string MidName, double Salary, int Position) {

this->Name = Name;

this->Surname = Surname;

this->MidName = MidName;

this->Salary = Salary;

this->Position = Position;

};

void set\_position(int Position) {

this->Position = Position;

}

void print() {

cout << GET\_Surname() << ' ';

cout << GET\_Name() << ' ';

cout << GET\_MidName() << ' ';

cout << GET\_Salary() << ' ';

cout << GET\_Position() << endl;

}

bool operator==(const Employee& right) {

if (Name == right.Name && Surname == right.Surname && MidName == right.MidName && Salary == right.Salary && Position == right.Position) {

return true;

}

else {

return false;

};

};

virtual void write\_to\_file(ofstream& f) {

f << this->GET\_Name() << endl;

f << this->GET\_Surname() << endl;

f << this->GET\_MidName() << endl;

f << this->GET\_Salary() << endl;

f << this->GET\_Position() << endl;

}

void read\_basic\_info() {

Surname = input\_name(0);

Name = input\_name(1);

MidName = input\_name(2);

Salary = input\_salary();

}

string GET\_Name() { return Name; };

string GET\_Surname() { return Surname; };

string GET\_MidName() { return MidName; };

double GET\_Salary() { return Salary; };

int GET\_Position() { return Position; };

char get\_c1() { return c1; };

char get\_c2() { return c2; };

int get\_payments() { return amount; };

};

### Дополнительные функции

#include "Functions.h"

#include "Director.h"

#include "Employee.h"

#include "Driver.h"

#include "Secretary.h"

#include "Programmist.h"

Director load\_from\_file(string file\_name) {

ifstream f;

Director d;

string name, surname, midname, level;

double salary;

int position;

f.open(file\_name);

if (!f) {

d.SET("Dmitriy", "Speranskiy", "Andreevich", 20500.1, 0);

return d;

}

f >> name >> surname >> midname >> salary >> position;

/\*catch (const ifstream::failure& e) {

cout << "ERROR" << endl;

return d;

}\*/

d.SET(name, surname, midname, salary, position);

while (f >> name >> surname >> midname >> salary >> position) {

if (position == 5) {

Driver c;

c.SET(name, surname, midname, salary, position);

int x;

f >> x;

for (int i = 0; i < x; ++i) {

string cat;

f >> cat;

c.add\_category(cat);

}

f >> x;

for (int i = 0; i < x; ++i) {

string v;

f >> v;

c.add\_vehicle(v);

}

d.hire(&c, position);

}

else if (position == 6) {

Programmist p;

int y;

p.SET(name, surname, midname, salary, position);

f >> level;

p.set\_level(level);

f >> y;

for (int i = 0; i < y; ++i) {

string l;

f >> l;

p.add\_language(l);

}

d.hire(&p, position);

}

else {

Employee n;

n.SET(name, surname, midname, salary, position);

d.hire(&n, position);

}

};

return d;

};

void write\_employee(Employee e, ofstream& f) {

f << e.GET\_Name() << endl;

f << e.GET\_Surname() << endl;

f << e.GET\_MidName() << endl;

f << e.GET\_Salary() << endl;

f << e.GET\_Position() << endl;

};

void save\_to\_file(Director& d, string file\_name) {

ofstream f(file\_name);

write\_employee(d, f);

for (Employee e : d.employees) {

e.write\_to\_file(f);

};

for (Driver d : d.drivers) {

d.write\_to\_file(f);

};

for (Programmist p : d.programmists) {

p.write\_to\_file(f);

};

};

vector<string> split(string s, string delimiter) {

size\_t pos\_start = 0, pos\_end, delim\_len = delimiter.length();

string token;

vector<string> res;

while ((pos\_end = s.find(delimiter, pos\_start)) != string::npos) {

token = s.substr(pos\_start, pos\_end - pos\_start);

pos\_start = pos\_end + delim\_len;

res.push\_back(token);

}

res.push\_back(s.substr(pos\_start));

return res;

};

### Заголовочные файлы

#pragma once

#include <string>

using namespace std;

void clear();

string input\_name(int t);

double input\_salary();

string input\_level();

#pragma once

#include "Employee.h"

#include "Director.h"

Director load\_from\_file(string file\_name);

void write\_employee(Employee e, ofstream& f);

void save\_to\_file(Director& d, string file\_name);

vector<string> split(string s, string delimiter);

### Класс UberProgrammer

#pragma once

#include "Programmist.h"

#include "Driver.h"

class UberProgrammer : public Programmist, public Driver {};