**Лабораторная работа 2**

**Вариант 16**

**Классы: инкапсуляция, наследование.**

**Задание** (2.1)

Реализовать иерархию классов в соответствии с вариантом. У всех вариантов в качестве **базового** класса выступает класс “**Сотрудник**”. Построить диаграмму классов.

В случае недопустимых значений полей выбрасываются исключения.

Примечание: если в задании требуется реализовать функционал для другого класса, которого нет в вашем варианте, делать этого не требуется

**Вариант задания:**

|  |  |
| --- | --- |
| 16: | 0 2 5 6 |

1. **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.

1. **Класс Бухгалтер: (Отсутствует по условию варианта)**
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.

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

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

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

1. **Класс Системный администратор: (Отсутствует по условию варианта)**
2. **Класс Охранник: (Отсутствует по условию варианта)**

1. **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.

1. **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.

+

Лабораторная работа №2.2. «Множественное наследование» Задание: Реализовать дочерний класс на основе двух классов, реализованных в предыдущей в лабораторной работе 2.1.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.

1. **Постановка задачи**

Разработать пользовательский интерфейс для работы с указанными в задании классами их полями и методами.

**Таблица 1. Входные данные**

|  |  |
| --- | --- |
| Входные данные | Назначение |
| firstName | Для хранения фамилии |
| name | Для хранения имени |
| lastName | Для хранения отчества |
| zarplata | Для хранения зарплаты |
| prava | Для хранения прав водителя |
| mashini | Для хранения машин водителя |
| technology | Для хранения технологий программиста |
| level | Для хранения уровня программиста |

**Таблица 2. Выходные данные**

|  |  |
| --- | --- |
| Выходные данные | Назначение |
| Таблица директора | Для вывода нанятых сотрудников |
| Биржа труда | Для вывода существующих сотрудников |

**Ограничение**

1) zarplata >= 0

2) Попытка найма несуществующего сотрудника

3) Попытка увольнения несуществующего сотрудника

4) Попытка удаление несуществующего сотрудника

5) Попытка изменения несуществующего сотрудника

6) Попытка выйти за пределы существующих сотрудников

**Среда разработки**

Необходимо разработать оконное приложение (WF) в среде разработки Visual Studio Community 2019 версии 16.3.0 C++ в соответствии с заданием. Продемонстрировать работу классов и выявить проблемы возникшие при разработки программы.

**2. Разработка программы**

**2.1 Разработка структуры программы**

В соответствии с заданием разработаем программу, я думаю разбить программу на несколько частей: Заголовочные файлы классов, которые мы будем создавать, реализация их методов в .cpp файлах, а также сама форма, в которой мы реализуем пользовательский интерфейс. Создадим эти файлы. Всего у нас будет 9 различных классов, со своими полями и методами. Все классы будут хранится в двумерном векторе типа указателя на класс Sotrudnik (vector<vector<Sotrudnik \*>> birja).

Пользователю будет доступна работа только с 5 классами: класс Director, класс Secretar, класс Voditel, класс Programmist. Распишем приблизительные поля и методы, которые должны содержать эти классы, а также вспомогательные классы для их создания.

**Таблица 3 Класс FIO**

|  |  |  |
| --- | --- | --- |
| Поля | Тип данных | Смысл поля |
| F (public) | string | Хранение фамилии |
| I (public) | string | Хранение имени |
| O (public) | string | Хранение отчества |

Класс FIO будет использоваться для хранение фамилии, имени и отчества сотрудника.

**Таблица 4 Класс Zarplata**

|  |  |  |
| --- | --- | --- |
| Поля | Тип данных | Смысл поля |
| Zarplata  (public) | double | Хранение текущей зарплаты |

Класс Zarplata будет использоваться для хранение текущей зарплаты сотрудника.

**Таблица 5 Класс FIOZarplata**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Родительские классы | Поля | Тип данных | Смысл поля | Методы | Смысл метода |
| FIO | Count (public) | int | Количество созданных экземпляров класса FIOZarplata в течении выполнения программы | getFIOZarplata | Принимает некоторые поля, которые меняет и возвращает в них фамилию, имя, отчество и текущую зарплату сотрудника |
| Zarplata | id  (public) | int | Уникальный номер сотрудника, для того чтобы отличать людей с разными фамилиями и работать с ним | getId | Возвращает id сотрудника |

Класс FIOZarplata будет использоваться для хранение текущей фамилии, имени, отчества и зарплаты сотрудника.

**Таблица 6 Класс Sotrudnik**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | Тип данных | Смысл поля | Методы | Смысл метода |
| fiozarplata (protected) | FIOZarplata | Хранение фамилии, имени, отчества и зарплаты | getFIOZarplata | Возвращает поле fiozarplata |
|  |  |  | setFIO | Принимает переменные, хранящие фамилию (string), имя (string), отчество (string), для их изменения у сотрудника |
|  |  |  | setZarplata | Принимает переменную хранящую новое значение зарплаты (double), для изменение зарплаты |
|  |  |  | setStrinng1Info | Виртуальный метод, специально для классов программиста и водителя |
|  |  |  | setStrinng2Info | Виртуальный метод, специально для классов программиста и водителя |
|  |  |  | getStrinng1Info | Виртуальный метод, специально для классов программиста и водителя |
|  |  |  | getStrinng2Info | Виртуальный метод, специально для классов программиста и водителя |
|  |  |  | stringAllInfo | Возвращает всю информацию в формате строки (string) |

Класс Sotrudnik, работает с полем fiozarplata. Также в нем есть дополнительные функции, которые нужны для работы с определенными классами через массив указателей на класс Sotrudnik.

**Таблица 7 Класс Director**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Родительские классы | Поля | Тип данных | Смысл поля | Методы | Смысл метода |
| Sotrudnik (virtual public) | table | vector<vector  <FIOZarplata\*>> | Таблица директора | nanyt | Найм сотрудников (добавление в таблицу) |
|  |  |  |  | uvolit | Увольнение сотрудников (удаление из таблицы) |

Класс Director, работает с полем table нанимая и увольняя сотрудников.

**Таблица 8 Класс Secretar**

|  |  |  |
| --- | --- | --- |
| Родительские классы | Методы | Смысл метода |
| Sotrudnik (public) | vivodInTable | Вывод таблицы о всех сотрудников для данного директора. В качестве параментров принимает объект класса диреткор и номер класса для вывода |
|  | vivodInTableProgramist | Вывод таблицы с программистами. В качестве параментров принимает объект класса диреткор и номер класса для вывода |
|  | vivodInTableSecretar | Вывод таблицы с секретарями. В качестве параментров принимает объект класса диреткор и номер класса для вывода |
|  | vivodInTableSuperProgrammist | Вывод таблицы с комбоклассом. В качестве параментров принимает объект класса диреткор и номер класса для вывода |
|  | vivodInTableVoditel | Вывод таблицы с водителями. В качестве параментров принимает объект класса диреткор и номер класса для вывода |

Класс Secretar, работает с полем table в классе директора.

**Таблица 9 Класс Voditel**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Родительские классы | Поля | Тип данных | Смысл поля | Методы | Смысл метода |
| Sotrudnik (public) | prava | string | Хранить права водителя | getString1Info | Показать права, которыми обладает водитель |
|  | mashini | string | Хранить машины, которые в распоряжении у водителя | getString2Info | Показать машины, которые в распоряжении у водителя. |
|  |  |  |  | setString1Info | Изменить права, которыми обладает водитель |
|  |  |  |  | setString2Info | Изменить машины, которые в распоряжении у водителя. |
|  |  |  |  | stringAllInfo | Возвращает всю информацию в формате строки (string) |

Класс Voditel, работает с полями mashini и prava, показывая их и меняя их.

**Таблица 10 Класс Programmist**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Родительские классы | Поля | Тип данных | Смысл поля | Методы | Смысл метода |
| Sotrudnik (virtual public) | level | string | Хранить уровень программиста | getString1Info | Показать уровень программиста |
|  | technology | string | Хранить технологии, которыми обладает программист | getString2Info | Показать технологии, которыми обладает программист |
|  |  |  |  | setString1Info | Изменить уровень программиста |
|  |  |  |  | setString2Info | Изменить технологии, которыми обладает программист. |
|  |  |  |  | stringAllInfo | Возвращает всю информацию в формате строки (string) |

Класс Programmist, работает с полями level и technology, показывая их и меняя их.

**Таблица 11 Класс Class**

|  |  |  |
| --- | --- | --- |
| Родительские классы | Метод | Смысл метода |
| Programmist (public) | stringAllInfo | Возвращает всю информацию в формате строки (string) |
| Director (public) |  |  |

Класс Class, работает как классы Programmist и Director вместе.

**Диаграмма классов**

Ниже представлена диаграмма классов, для проекта **ProjectCompany.sln,** на которой указаны класссы, их поля, методы и способ взаимодействия с другими классами.

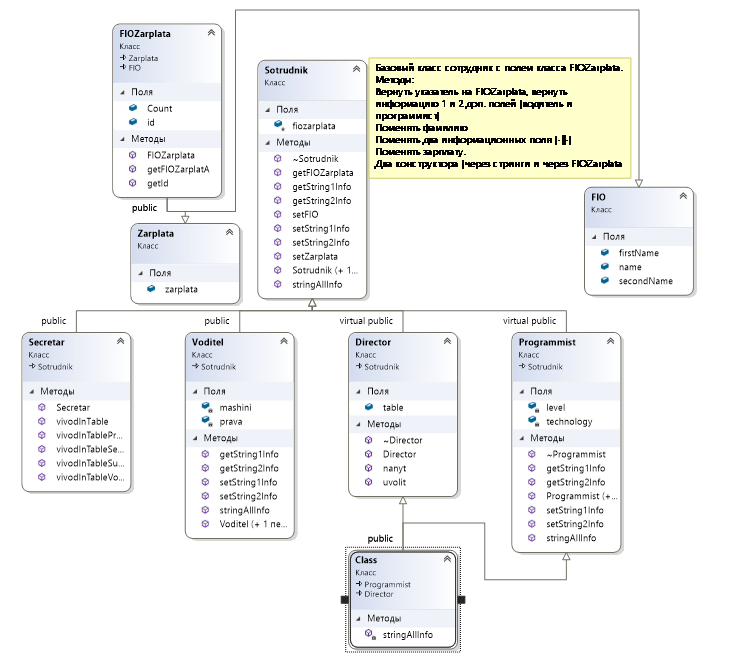


Рис. 1 Диаграмма классов

**2.2 Разработка алгоритмов**

Сложные алгоритмы и формулы не будут использоваться при разработке программы.

**2.3 Разработка пользовательского интерфейса**

Пользователями программы могут быть люди, которые работают директорами или секретарями в компаниях, чтобы хранить данные о сотрудниках и изменять эти данные.

Программа представленна в визуальном виде и содержит в себе:

**textBox**’ы, (6 штук) для записи фамилии, имени, отчества, уровня и технологий программиста, прав и машин водителя.

**Label’**ы (14 штук) для разъяснения.

**MessageBox’**ы для вывода таблиц и ошибок,

**groupBox’**ы для сокрытия частей программы.

**numericUpDown1’**ы (3 штуки) для введения индексов сотрудников и их зарплаты.

**button’**ы (7 штук) для взаимодействия с пользователем.

**comboBox’**ы для взаимодействия с пользователем.

**Интерфейс программы:**

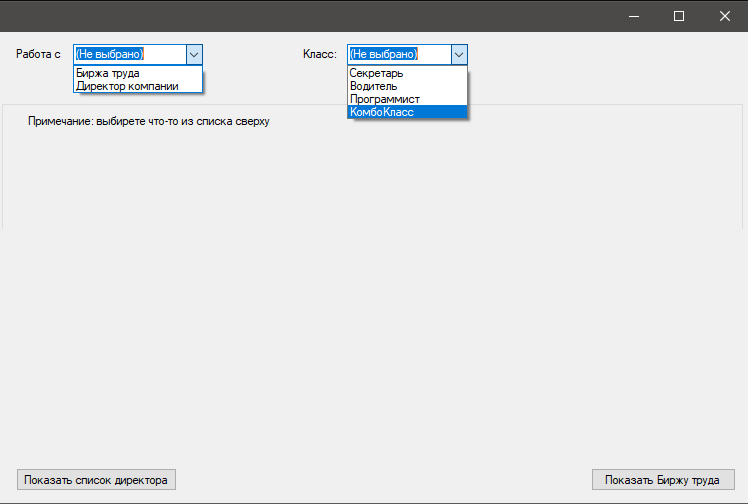
1)Программа при запуске. Необходимо выбрать работу с биржей, чтобы создать нужные классы и только после их можно нанять директором (рис. 2.1 – рис 2.2).

Рис. 2.1

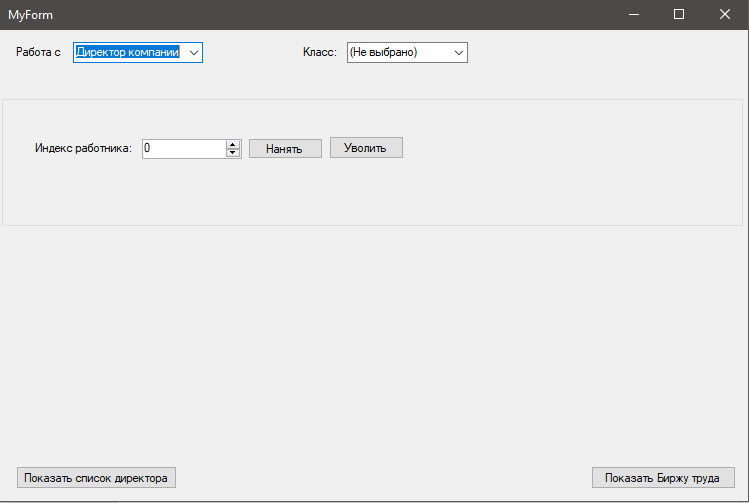
При работе с биржей труда необходимо выбрать какой объект класса будем создавать/менять/удалять. 

Рис. 2.2

При попытке удалить, изменить, несуществующий объект выскакивает ошибка (Рис 2.5)

При изменении объектов, которые записаны в списке директора, информация о них автоматически обновится и удаление с биржи. При работе с Директором компании мы можем нанимать и удалять сотрудников в компанию директора (Рис. 2.2). При попытке нанять/удалить несуществующий объект класса, выскакивает ошибка (Рис 2.5).

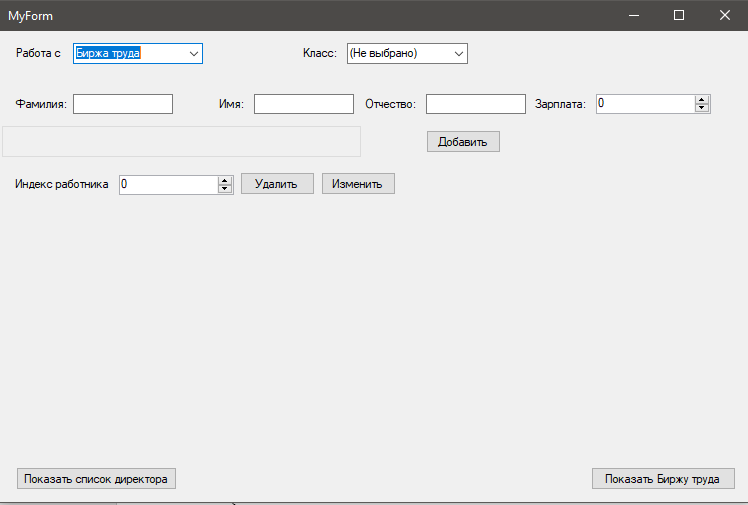


Рис. 2.3

Также можно посмотреть список Директора и биржу нажатием кнопки (Биржа также выводит дополнительную информацию о сотрудниках (если она имеется) (Рис. 2.4)

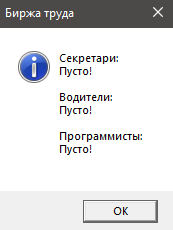
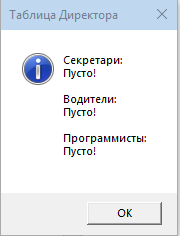


Рис. 2.4 а Рис. 2.4 б

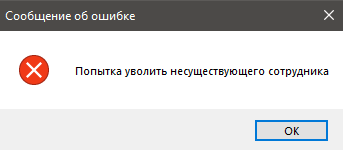
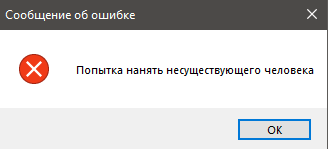


Рис. 2.5 а Рис. 2.5 б

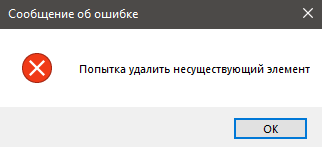
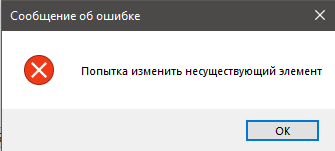
Попробуем удалить и изменить сотрудников на бирже для проверки исключений. 

Рис. 2.5 в Рис. 2.5 г

**3.Реализация и тестирование приложения**

**Объект испытания:** Программа ProjectCompany.exe

**Цель испытания:** Проверка общей работоспособности программы.

**Описание компьютера:**

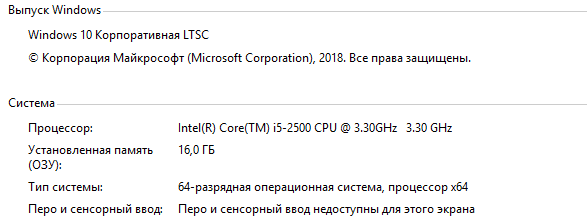




Рис. 3.1 Характеристики компьютера

**Метод испытаний:** функциональное тестирование и проход отладчиком для каждого класса в целом и в общем всей программы.

**Тестирование:**

Создадим объект класса, наймем его, изменим его поля, и удалим с биржи, при этом наблюдая за списками.

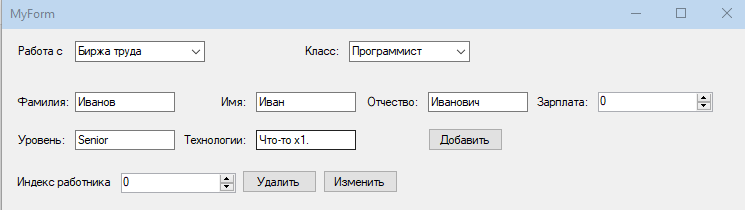


Рис. 3.2

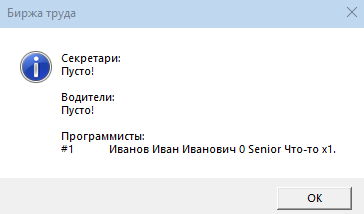


Рис. 3.3

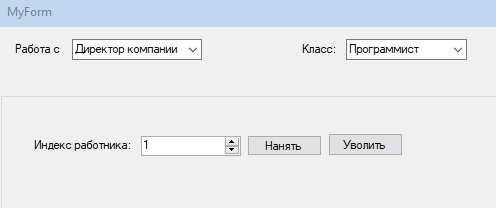


Рис. 3.4

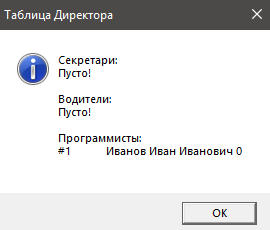


Рис. 3.5

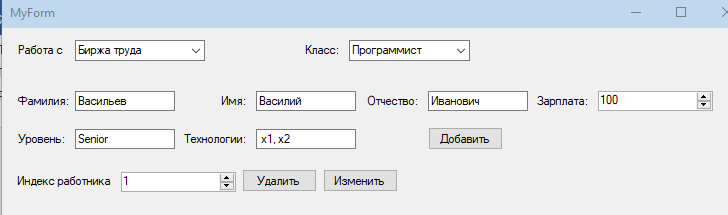


Рис. 3.6

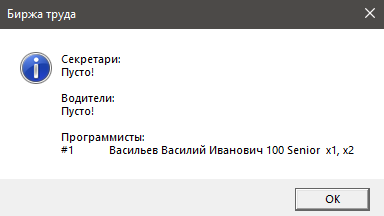


Рис. 3.7

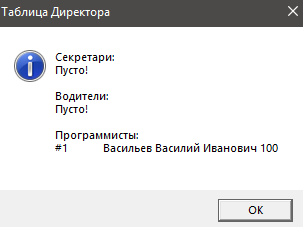


Рис. 3.8

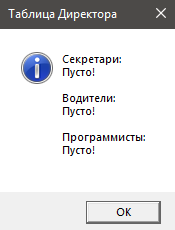
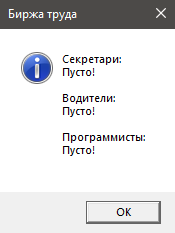


Рис. 3.9 a Рис. 3.9 б

Как видим, программа работает корректно.

**Вывод:** Во время выполнения данной работы мы научились использовать одиночное и множественное наследование и работать с ними. Была создана программа для компании из 4 (5 вместе с директором) классов сотрудников. Программа работает корректно. Исключения выбрасываются корректно. Можно доработать обработку полей для Программиста и Водителя, а также проверку имен для всех классов.

**Приложение. Листинг программы**

**Модуль FIO.h**

#pragma once

#include <string>

#include <vector>

#include <iostream>

#include <windows.h>

#include <msclr/marshal\_cppstd.h>

using namespace std;

class FIO

{

public:

string firstName, name, secondName;

};

**Модуль Zarplata.h**

#pragma once

class Zarplata

{

public:

double zarplata;

};

**Модуль FIOZarplata.h**

#pragma once

#include "FIO.h"

#include "Zarplata.h"

class FIOZarplata :public Zarplata, public FIO

{

public:

int id;

static int Count;

FIOZarplata()

{

Count++;

id = Count;

}

void getFIOZarplatA(string &firstName, string &name, string &secondName, double &zarplata);

int getId();

};

**Модуль FIOZarplata.cpp**

#include "FIOZarplata.h"

int FIOZarplata::Count = 0;

void FIOZarplata::getFIOZarplatA(string& firstName, string& name, string& secondName, double &zarplata)//ИНициализация полей

{

firstName = this->firstName;

name = this->name;

secondName = this->secondName;

zarplata = this->zarplata;

}

int FIOZarplata::getId() { return id; }//Вернуть id

**Модуль Sotrudnik.h**

#pragma once

#include "FIOZarplata.h"

class Sotrudnik

{

protected:

FIOZarplata fiozarplata;

public:

Sotrudnik(string FirstName, string Name, string SecondName, double zarplata = 0);

Sotrudnik(FIOZarplata fiozarplata);

void setFIO(string FirstName, string Name, string SecondName);

FIOZarplata\* getFIOZarplata();

void setZarplata(double zarplata);

virtual string stringAllInfo();

virtual string getString1Info();

virtual void setString1Info(string);

virtual void setString2Info(string);

virtual string getString2Info();

virtual ~Sotrudnik();

//int getID();

};

**Модуль Sotrudnik.cpp**

#include "Sotrudnik.h"

Sotrudnik::Sotrudnik(string firstName, string name, string secondName, double zarplata)//Инициализация полей

{

this->fiozarplata.firstName = firstName;

this->fiozarplata.name = name;

this->fiozarplata.secondName = secondName;

this->fiozarplata.zarplata = zarplata;

}

Sotrudnik::Sotrudnik(FIOZarplata fiozarplata) { this->fiozarplata = fiozarplata; }//Инициализация полей

void Sotrudnik::setFIO(string firstName, string name, string secondName)//Изменение Фамилии, имени, отчества

{

this->fiozarplata.firstName = firstName;

this->fiozarplata.name = name;

this->fiozarplata.secondName = secondName;

}

FIOZarplata\* Sotrudnik::getFIOZarplata() { return &fiozarplata; }//Возвращении информации

void Sotrudnik::setZarplata(double zarplata) { this->fiozarplata.zarplata = zarplata; }//Изменение зарплаты

string Sotrudnik::stringAllInfo() { //Вывод всей информации в строчку

msclr::interop::marshal\_context con;

string s1 = con.marshal\_as<string>(fiozarplata.zarplata.ToString());

return fiozarplata.firstName + " " + fiozarplata.name + " " + fiozarplata.secondName + " " + s1;

}

string Sotrudnik::getString1Info() { return string(); }//Показать 1-ое спецполе

void Sotrudnik::setString1Info(string) {}//Изменить 1-ое спецполе

void Sotrudnik::setString2Info(string) {}//Изменить 2-ое спецполе

string Sotrudnik::getString2Info() { return string(); }//Показать 2-ое спецполе

Sotrudnik::~Sotrudnik(){};

**Модуль Director.h**

#pragma once

#include "Sotrudnik.h"

class Director :public virtual Sotrudnik {

public:

vector<vector<FIOZarplata\*>> table;

Director(string firstName, string name, string secondName, double zarplata = 0);

void nanyt(Sotrudnik\* i, unsigned int id);

void uvolit(unsigned int id, unsigned int i);

virtual ~Director() override;

};

**Модуль Director.cpp**

#include "Director.h"

Director::Director(string firstName, string name, string secondName, double zarplata) :Sotrudnik(firstName, name, secondName, zarplata)

{//Заполнение таблицы директора

vector<FIOZarplata\*> a1;

table.push\_back(a1);

table.push\_back(a1);

table.push\_back(a1);

table.push\_back(a1);

}

void Director::nanyt(Sotrudnik\* i, unsigned int id) { table[id].push\_back(i->getFIOZarplata()); }; //Нанятие сотрудника

void Director::uvolit(unsigned int id, unsigned int i) { table[id].erase(table[id].begin() + i - 1); } //Увольнение сотрудника

Director::~Director()

{

table[0].clear();

table[1].clear();

table[2].clear();

table[3].clear();

}

**Модуль Secretar.h**

#pragma once

#include "Director.h"

class Secretar :public Sotrudnik

{

public:

Secretar(string firstName, string name, string secondName, double zarplata = 0);

string vivodInTableSecretar(Director boss);

string vivodInTableVoditel(Director boss);

string vivodInTableProgrammist(Director boss);

string Secretar::vivodInTableSuperProgrammist(Director boss);

string vivodInTable(Director boss);

};

**Модуль Secretar.cpp**

#include "Secretar.h"

Secretar::Secretar(string firstName, string name, string secondName, double zarplata) :Sotrudnik(firstName, name, secondName, zarplata) {}

string Secretar::vivodInTableSecretar(Director boss) {//Вывод секреторей в таблицу

msclr::interop::marshal\_context con;

string s1 = "Секретари:\n";

if (boss.table[0].size() == 0) { s1 += "Пусто!\n\n"; }//Если есть секретари

for (unsigned int j = 0; j < boss.table[0].size(); j++)//Цикл по их выводу

{

s1 += "#" + con.marshal\_as<string>((j + 1).ToString()) + " \t" + boss.table[0][j]->firstName + " " + boss.table[0][j]->name + " " + boss.table[0][j]->secondName + " " + con.marshal\_as<string>(boss.table[0][j]->zarplata.ToString()) + "\n\n";

}

return s1;

}

string Secretar::vivodInTableVoditel(Director boss) {//Вывод водителей в таблицу

msclr::interop::marshal\_context con;

string s1 = "Водители:\n";

if (boss.table[1].size() == 0) { s1 += "Пусто!\n\n"; }//Если есть водители

for (unsigned int j = 0; j < boss.table[1].size(); j++)//Цикл по их выводу

{

s1 += "#" + con.marshal\_as<string>((j + 1).ToString()) + " \t" + boss.table[1][j]->firstName + " " + boss.table[1][j]->name + " " + boss.table[1][j]->secondName + " " + con.marshal\_as<string>(boss.table[1][j]->zarplata.ToString()) + "\n\n";

}

return s1;

}

string Secretar::vivodInTableProgrammist(Director boss) {//Вывод программистов в таблицу

msclr::interop::marshal\_context con;

string s1 = "Программисты:\n";

if (boss.table[2].size() == 0) { s1 += "Пусто!\n\n"; }//Если есть программисты

for (unsigned int j = 0; j < boss.table[2].size(); j++)//Цикл по их выводу

{

s1 += "#" + con.marshal\_as<string>((j + 1).ToString()) + " \t" + boss.table[2][j]->firstName + " " + boss.table[2][j]->name + " " + boss.table[2][j]->secondName + " " + con.marshal\_as<string>(boss.table[2][j]->zarplata.ToString()) + "\n\n";

}

return s1;

}

string Secretar::vivodInTableSuperProgrammist(Director boss) {//Вывод класса в таблицу

msclr::interop::marshal\_context con;

string s1 = "СуперПрограммисты:\n";

if (boss.table[3].size() == 0) { s1 += "Пусто!\n\n"; }//Если есть классы

for (unsigned int j = 0; j < boss.table[3].size(); j++)//Цикл по их выводу

{

s1 += "#" + con.marshal\_as<string>((j + 1).ToString()) + " \t" + boss.table[3][j]->firstName + " " + boss.table[3][j]->name + " " + boss.table[3][j]->secondName + " " + con.marshal\_as<string>(boss.table[3][j]->zarplata.ToString()) + "\n\n";

}

return s1;

}

string Secretar::vivodInTable(Director boss) {//Вывод всех кроме директора, для директора

string s1;

s1 = vivodInTableSecretar(boss);

s1 += vivodInTableVoditel(boss);

s1 += vivodInTableProgrammist(boss);

return s1;

}

**Модуль Voditel.h**

#pragma once

#include "Sotrudnik.h"

class Voditel :public Sotrudnik

{

private:

string prava, mashini;

public:

Voditel(string firstName, string name, string secondName, double zarplata, string prava, string mashini);

Voditel(FIOZarplata fiozarplata, string prava, string mashini);

string stringAllInfo() override;

string getString1Info() override;

string getString2Info() override;

void setString1Info(string prava) override;

void setString2Info(string mashini) override;

};

**Модуль Voditel.cpp**

#include "Voditel.h"

Voditel::Voditel(string firstName, string name, string secondName, double zarplata, string prava, string mashini) :Sotrudnik(firstName, name, secondName, zarplata)//Инициализация полей

{

this->prava = prava;

this->mashini = mashini;

}

Voditel::Voditel(FIOZarplata fiozarplata, string prava, string mashini) :Sotrudnik(fiozarplata)//Инициализация полей

{

this->prava = prava;

this->mashini = mashini;

}

string Voditel::stringAllInfo()//Показать всю информацию в 1 строчку

{

msclr::interop::marshal\_context con;

string s1 = con.marshal\_as<string>(fiozarplata.zarplata.ToString());

return fiozarplata.firstName + " " + fiozarplata.name + " " + fiozarplata.secondName + " " + s1 + " " + prava + " " + mashini;

}

string Voditel::getString1Info() { return prava; }//Показать права

string Voditel::getString2Info() { return mashini; }//Показать машины

void Voditel::setString1Info(string prava) { this->prava = prava; }//Поменять права

void Voditel::setString2Info(string mashini) { this->mashini = mashini; }//Поменять машины

**Модуль Programmist.h**

#pragma once

#include "Sotrudnik.h"

class Programmist :

public virtual Sotrudnik

{

private:

string technology, level;

public:

Programmist(string firstName, string name, string secondName, double zarplata, string level, string technology);

Programmist(FIOZarplata fiozarplata, string level, string technology);

string stringAllInfo() override;

string getString1Info() override;

string getString2Info() override;

void setString1Info(string level) override;

void setString2Info(string technology) override;

virtual ~Programmist() override;

};

**Модуль Programmist.cpp**

#include "Programmist.h"

Programmist::Programmist(string firstName, string name, string secondName, double zarplata, string level, string technology) :Sotrudnik(firstName, name, secondName, zarplata)//Инициализация полей

{

this->level = level;

this->technology = technology;

}

Programmist::Programmist(FIOZarplata fiozarplata, string level, string technology) : Sotrudnik(fiozarplata)//Инициализация полей

{

this->level = level;

this->technology = technology;

}

string Programmist::stringAllInfo() // Показать всю информацию в 1 строчку

{

msclr::interop::marshal\_context con;

string s1 = con.marshal\_as<string>(fiozarplata.zarplata.ToString());

return fiozarplata.firstName + " " + fiozarplata.name + " " + fiozarplata.secondName + " " + s1 + " " + level + " " + technology;

}

string Programmist::getString1Info() { return level; }//Показать уровень

string Programmist::getString2Info() { return technology; }//Показать технологии

void Programmist::setString1Info(string level) { this->level = level; }//Поменять уровень

void Programmist::setString2Info(string technology) { this->technology = technology; }//Поменять технологии

Programmist::~Programmist()

{

}

**Модуль Class.h**

#pragma once

#include "Director.h"

#include "Programmist.h"

class Class :

public Director, public Programmist

{

string stringAllInfo() override;

};

**Модуль Class.cpp**

#include "Class.h"

string Class::stringAllInfo()

{

return Programmist::stringAllInfo();

}

**Модуль MyForm.h**

#pragma endregion

private: System::Void MyForm\_Load(System::Object ^ sender, System::EventArgs ^ e) { //Заполнение вектора birja при загрузке формы

vector<Sotrudnik\*> a1;

birja.push\_back(a1);

birja.push\_back(a1);

birja.push\_back(a1);

birja.push\_back(a1);

}

private: System::Void ComboBox1\_SelectedIndexChanged(System::Object ^ sender, System::EventArgs ^ e) {

//Скрытие и открытие groupBox'ов при выборе в comboBox1 (Работа с)

if (comboBox1->SelectedIndex == 0) { //Биржа

groupBox6->Visible = false; //Поля дополнительной информации, для классов Programmist, Voditel, Class

groupBox4->Visible = false; //Поля дополнительной информации, для классов Programmist, Voditel, Class

}

else

{

groupBox4->Visible = false; //Директор

groupBox6->Visible = true;

}

}

private: System::Void ComboBox2\_SelectedIndexChanged(System::Object ^ sender, System::EventArgs ^ e) { //Скрытие и открытие groupBox'ов при выборе в comboBox2 (Класс:)

if (comboBox1->SelectedIndex == 0) {

switch (comboBox2->SelectedIndex)

{

case 1: //Voditel

groupBox3->Visible = false;

groupBox1->Visible = true;

groupBox2->Visible = true;

break;

case 2: //Programmist

case 3: //Class

groupBox3->Visible = false;

groupBox1->Visible = false;

groupBox2->Visible = false;

break;

default: //another

groupBox3->Visible = true;

break;

}

}

else

{

groupBox6->Visible = true;

}

}

private: System::Void Button2\_Click(System::Object ^ sender, System::EventArgs ^ e) {

try { //Нанятие сотрудников с биржы (Кнопка «Нанять» у директора)

if (comboBox2->SelectedIndex < 0 || numericUpDown2->Value > birja[comboBox2->SelectedIndex].size() || numericUpDown2->Value == 0) { throw 1; } //Проверка на существование человека

boss.nanyt(birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown2->Value) - 1], comboBox2->SelectedIndex);//Нанятие

}

catch (...) {

MessageBox::Show(

"Попытка нанять несуществующего человека",

"Сообщение об ошибке",

MessageBoxButtons::OK,

MessageBoxIcon::Error,

MessageBoxDefaultButton::Button1,

MessageBoxOptions::RightAlign);

}

numericUpDown2->Value = 0;

}

private: System::Void Button6\_Click(System::Object ^ sender, System::EventArgs ^ e) {

try { //Увольнение сотрудников с биржы (Кнопка «Уволить» у директора)

if (comboBox2->SelectedIndex < 0 || numericUpDown2->Value > boss.table[comboBox2->SelectedIndex].size() || numericUpDown2->Value == 0) { throw 1; } //Проверка на существование человека

boss.uvolit(comboBox2->SelectedIndex, Convert::ToInt32(numericUpDown2->Value));//Увольнение

}

catch (...) {

MessageBox::Show(

"Попытка уволить несуществующего сотрудника",

"Сообщение об ошибке",

MessageBoxButtons::OK,

MessageBoxIcon::Error,

MessageBoxDefaultButton::Button1,

MessageBoxOptions::RightAlign);

}

numericUpDown2->Value = 0;

}

private: System::Void Button1\_Click(System::Object ^ sender, System::EventArgs ^ e) {//(Кнопка «Добавить» в бирже)

string s1, s2, s3, s5, s6; //Добавление сотрудников в биржу

double s4;

msclr::interop::marshal\_context con;

s5 = con.marshal\_as<string>(textBox5->Text);//Поле для водители и программиста

s6 = con.marshal\_as<string>(textBox6->Text);//Поле для водители и программиста

s1 = con.marshal\_as<string>(textBox1->Text);//Ф

s2 = con.marshal\_as<string>(textBox2->Text);//И

s3 = con.marshal\_as<string>(textBox3->Text);//О

s4 = Convert::ToDouble(numericUpDown3->Value);//Зарплата

textBox5->Text = "";

textBox6->Text = "";

textBox1->Text = "";

textBox2->Text = "";

textBox3->Text = "";

numericUpDown3->Value = 0;

switch (comboBox2->SelectedIndex)

{

case 0://Секретарь

birja[0].push\_back(new Secretar(s1, s2, s3, s4));

break;

case 1://Водитель

birja[1].push\_back(new Voditel(s1, s2, s3, s4, s5, s6));

break;

case 2://Программист

birja[2].push\_back(new Programmist(s1, s2, s3, s4, s5, s6));

break;

case 3://Директор-Программист

birja[3].push\_back(new Programmist(s1, s2, s3, s4, s5, s6));

break;

default://Ничего не выбрали

break;

}

}

private: System::Void Button5\_Click(System::Object ^ sender, System::EventArgs ^ e) {{//(Кнопка «Удалить» в бирже)

try {//Удаление с биржи

int a = Convert::ToInt64(numericUpDown1->Value) - 1;

if (comboBox2->SelectedIndex < 0 || numericUpDown1->Value > birja[comboBox2->SelectedIndex].size() || numericUpDown1->Value == 0) { throw 1; }

for (unsigned int i = 0, j = 0; i < boss.table[comboBox2->SelectedIndex].size() && j == 0; i++)//Проверка наличия сотрудника в списке директора

{

int bf1;

bf1 = boss.table[comboBox2->SelectedIndex][i]->getId();

if (bf1 == birja[comboBox2->SelectedIndex][a]->getFIOZarplata()->getId())//Проверка на совпадение по уникальному id

{

boss.table[comboBox2->SelectedIndex].erase(boss.table[comboBox2->SelectedIndex].begin() + Convert::ToInt64(i)); //Его удаление из списка

i--;

}

}

delete birja[comboBox2->SelectedIndex][a]; //его удаление из биржы

birja[comboBox2->SelectedIndex].erase(birja[comboBox2->SelectedIndex].begin() + a);//Убираем место занимаемое им

}

catch (...) {

MessageBox::Show(

"Попытка удалить несуществующий элемент",

"Сообщение об ошибке",

MessageBoxButtons::OK,

MessageBoxIcon::Error,

MessageBoxDefaultButton::Button1,

MessageBoxOptions::RightAlign);

}

}

private: System::Void Button3\_Click(System::Object ^ sender, System::EventArgs ^ e) {{//(Кнопка «Показать биржу труда»)

string s1 = "";//Вывод биржы

msclr::interop::marshal\_context con;

for (unsigned int i = 0; i < birja.size(); i++)

{

switch (i)

{

case 0:s1 = "Секретари:\n"; break;

case 1:s1 += "Водители:\n"; break;

case 2:s1 += "Программисты:\n"; break;

case 3:s1 += "СуперПрограммисты:\n"; break;

}

if (birja[i].size() > 0) {

for (unsigned int j = 0; j < birja[i].size(); j++)

{

s1 += "#" + con.marshal\_as<string>((j + 1).ToString()) + " \t" + (birja[i][j]->stringAllInfo()) + "\n";

//Вывод всей информации о сотруднике с его номером в бирже

}

}

else { s1 += "Пусто!\n\n"; }

}

MessageBox::Show(//Вывод в MessageBox

gcnew System::String(s1.c\_str()),

"Биржа труда",

MessageBoxButtons::OK,

MessageBoxIcon::Asterisk,

MessageBoxDefaultButton::Button1,

MessageBoxOptions::DefaultDesktopOnly);

}

private: System::Void Button4\_Click(System::Object ^ sender, System::EventArgs ^ e) {//(Кнопка «Показать список директора»)

try

{

string s1;//Вывод таблицы босса нанятым секретарем

if (boss.table[0].empty()) { throw 1; }

s1 = ((Secretar\*)(birja[0][0]))->vivodInTable(boss);

MessageBox::Show(

gcnew System::String(s1.c\_str()),

"Таблица Директора",

MessageBoxButtons::OK,

MessageBoxIcon::Asterisk,

MessageBoxDefaultButton::Button1,

MessageBoxOptions::DefaultDesktopOnly);

}

catch (...)

{

MessageBox::Show(

gcnew System::String("Чтобы выводить таблицу директора, наймите секреторя!"),

"Таблица Директора",

MessageBoxButtons::OK,

MessageBoxIcon::Error,

MessageBoxDefaultButton::Button1,

MessageBoxOptions::DefaultDesktopOnly);

}

}

private: System::Void Button7\_Click(System::Object^ sender, System::EventArgs^ e) {//(Кнопка «Изменить»)

try {//Изменение полей сотрудников на бирже

int a = Convert::ToInt64(numericUpDown1->Value) - 1;

if (comboBox2->SelectedIndex < 0 || numericUpDown1->Value > birja[comboBox2->SelectedIndex].size() || numericUpDown1->Value == 0) { throw 1; }

string s1, s2, s3, s5, s6;

double s4;

msclr::interop::marshal\_context con;

s5 = con.marshal\_as<string>(textBox5->Text);//Поле для водители и программиста

s6 = con.marshal\_as<string>(textBox6->Text);//Поле для водители и программиста

s1 = con.marshal\_as<string>(textBox1->Text);//Ф

s2 = con.marshal\_as<string>(textBox2->Text);//И

s3 = con.marshal\_as<string>(textBox3->Text);//О

s4 = Convert::ToDouble(numericUpDown3->Value);//Зарплата

textBox5->Text = "";//Зануление полей

textBox6->Text = "";

textBox1->Text = "";

textBox2->Text = "";

textBox3->Text = "";

numericUpDown3->Value = 0;

switch (comboBox2->SelectedIndex)

{

case 0:

birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown1->Value) - 1]->setFIO(s1, s2, s3);

birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown1->Value) - 1]->setZarplata(s4);

break;

case 1:

case 2:

case 3:

birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown1->Value) - 1]->setFIO(s1, s2, s3);

birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown1->Value) - 1]->setZarplata(s4);

birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown1->Value) - 1]->setString1Info(s5);

birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown1->Value) - 1]->setString2Info(s6);

break;

default:

break;

}

}

catch (int a) {

MessageBox::Show(

"Попытка изменить несуществующий элемент",

"Сообщение об ошибке",

MessageBoxButtons::OK,

MessageBoxIcon::Error,

MessageBoxDefaultButton::Button1,

MessageBoxOptions::RightAlign);

}

//numericUpDown1->Value = 0;

}

private: System::Void NumericUpDown1\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {//(Ползунок «Индекс работника» (В бирже труда))

try {//Показывание информации о перелистываемых сотрудниках (для удаления и изменения)

if (comboBox2->SelectedIndex < 0 || numericUpDown1->Value > birja[comboBox2->SelectedIndex].size() || numericUpDown1->Value == 0) { throw 1; }

string s5, s6;

FIOZarplata\* A = birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown1->Value) - 1]->getFIOZarplata();

textBox1->Text = gcnew System::String(A->firstName.c\_str());//фамилия

textBox2->Text = gcnew System::String(A->name.c\_str());//Имя

textBox3->Text = gcnew System::String(A->secondName.c\_str());//Отчество

numericUpDown3->Value = Convert::ToInt32(A->zarplata);//Зарплата

switch (comboBox2->SelectedIndex)

{

case 1:

case 2:

case 3:

textBox5->Text = gcnew System::String(birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown1->Value) - 1]->getString1Info().c\_str());//Спец поля

textBox6->Text = gcnew System::String(birja[comboBox2->SelectedIndex][Convert::ToInt32(numericUpDown1->Value) - 1]->getString2Info().c\_str());//Спец поля

break;

default:

break;

}

}

catch (...) {

MessageBox::Show(

"Попытка выйти за пределы существующих элементов",

"Сообщение об ошибке",

MessageBoxButtons::OK,

MessageBoxIcon::Error,

MessageBoxDefaultButton::Button1,

MessageBoxOptions::RightAlign);

}

}

};

}