**МІНІСТЕРСТВО ОСВІТИ І НАУКИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАІНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

**ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ**

**КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І СПЕЦІАЛІЗОВАНИХ КОМП`ЮТЕНИХ СИСТЕМ**

**ЛАБОРАТОРНАЯ РОБОТА №2**

з дисципліни «**Об’єктно-орієнтоване програмування 2**»

Тема: «**Просте успадкування. Додаток Database**»

|  |  |
| --- | --- |
|  | Виконав  студент 2 курсу  гр. КВ-43  Вітовщик Михайло Васильович  Перевірив(ла):  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Київ – 2016

***Постановка задачі***

Написати додаток Database, яке зберігає відомості про співробітників компанії. У додатку використовується наступна ієрархія поліморфних класів.

Базовим є абстрактний клас

//An abstract class

class Person{

public:

Person(){};

Person(string \_f\_name, string \_l\_name, int \_age) : age(\_age),

f\_name(\_f\_name), l\_name(\_l\_name){ }

virtual void Display(bool) = 0;

protected:

string f\_name;//first name

string l\_name;//last name

int age;

};

Відомості про рядових співробітників представлені класом

class Employee : public Person{

public:

Employee(){};

Employee(string \_f\_name, string \_l\_name, int \_age, int \_id) :

Person(\_f\_name, \_l\_name, \_age), id(\_id){};

Employee(const Employee &e);

Employee& operator=(const Employee &e);

void SetSalary(int s);

void SetDepartment(string dept);

void SetId(int n);

int GetId();

string GetDepartment();

virtual void Display();

//Add here whatever you need

protected:

string department;

int salary;

int id;

};

Відомості про менеджерів представлені класом

class Manager : public Employee {

public:

Manager(){};

Manager(string \_f\_name, string \_l\_name, int \_age, int \_id) :

Employee(\_f\_name, \_l\_name, \_age, \_id){};

Manager(const Manager &m;

Manager& operator=(const Manager &m);

virtual void Display(bool);

//add an employee to the subordinates list

Person\* AddSubordinate(Person \*p);

void DisplaySubordinates();

//Add here whatever you need

private:

list<Person \*> subordinates;//список подчиненных

};

Тобто, крім відомостей, спільних із класом Employee, клас Manager містить список його підлеглих (усі вони працюють у тому ж відділі, що й менеджер).

База даних представлена класом

class Database{

public:

Database(){};

~Database(){};//no need in destructor

//creates “flat” database

bool LoadFromFile(const char \*file);

//arranges "flat" database after loading from the file

void ArrangeSubordinates();

//hire a new employee

Person\* HireEmployee(Person \*p);

void DisplayDepartmentEmployees(string \_department);

//fire the employee

bool FireEmployee(int id);

void DisplayAll();

//Add here whatever you need

private:

vector<Person\*> employees;

};

***Текст програми***

**Employee.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <string>

#include <iostream>

#include <fstream>

#include <stdio.h>

#include <string.h>

using namespace std;

//An abstract class

class Person{

public:

Person(){};

Person(string \_f\_name, string \_l\_name, int \_age): f\_name(\_f\_name), l\_name(\_l\_name), age(\_age){};

virtual void Display(bool) = 0;

protected:

string f\_name; // first name

string l\_name; // last name

int age;

};

class Employee : public Person{

public:

Employee(){};

Employee(string \_f\_name, string \_l\_name, int \_age, int \_id) : Person(\_f\_name, \_l\_name, \_age), id(\_id) { salary = 0; };

Employee(const Employee &e);

Employee& operator = (const Employee &e);

void SetSalary(int s);

void SetDepartment(string dept);

void SetId(int n);

int GetId() const;

string GetDepartment() const;

void Display(bool bl) override;

protected:

string department;

int salary;

int id;

};

**Employee.cpp**

#include "Employee.h"

Employee::Employee(const Employee &e){

(\*this) = e;

}

Employee& Employee::operator = (const Employee &e){

f\_name = e.f\_name;

l\_name = e.l\_name;

age = e.age;

department = e.department;

salary = e.salary;

id = e.id;

return \*this;

}

void Employee::SetSalary(int s){

salary = s;

}

void Employee::SetDepartment(string dept){

department = dept;

}

void Employee::SetId(int n){

id = n;

}

int Employee::GetId() const{

return id;

}

string Employee::GetDepartment() const{

return department;

}

void Employee::Display(bool bl){

if(bl)cout << "\t\t";

cout << "Employee type: employee\n";

if (bl) cout << "\t\t";

cout << "id: " << id << endl;

if (bl) cout << "\t\t";

cout << f\_name << " " << l\_name << " age: " << age << " salary: " << salary << endl;

if (!bl) cout << "department: " << department << endl;

cout << endl;

}

**Manager.h**

#pragma once

#include "Employee.h"

#include <list>

class Manager : public Employee{

public:

Manager(){};

Manager(string \_f\_name, string \_l\_name, int \_age, int \_id) : Employee(\_f\_name, \_l\_name, \_age, \_id){};

Manager(const Manager &m);

Manager& operator = (const Manager &m);

Person\* AddSubordinate(Person \*p);

void DellSubordinate(int id);

void DisplaySubordinates();

void Display(bool) override;

private:

list <Person \*> subordinates; // список підлеглих

};

**Manager.cpp**

#include "Manager.h"

Manager::Manager(const Manager &m){

(\*this) = m;

}

Manager& Manager::operator = (const Manager &m){

f\_name = m.f\_name;

l\_name = m.l\_name;

age = m.age;

department = m.department;

salary = m.salary;

id = m.id;

subordinates = m.subordinates;

return \*this;

}

Person\* Manager::AddSubordinate(Person \*p){

list<Person\*>::iterator it = subordinates.begin();

while(it != subordinates.end()){

if (dynamic\_cast<Employee\*>(\*it)->GetId() == dynamic\_cast<Employee\*>(p)->GetId())

return nullptr;

++it;

}

subordinates.push\_back(p);

return p;

}

void Manager::DisplaySubordinates(){

list <Person \*>::iterator it;

if (subordinates.empty()){

cout << "\t\tnone\n\n\n";

return;

}

for(it = subordinates.begin(); it != subordinates.end(); ++it)

(\*it)->Display(true);

}

void Manager::Display(bool){

cout << "Employee type: manager\n";

cout << "id: " << id << endl;

cout << f\_name << " " << l\_name << " age: " << age << " salary: " << salary << endl;

cout << "department:" << department << endl;

cout << "Subordinates:\n";

DisplaySubordinates();

}

void Manager::DellSubordinate(int id){

list<Person\*>::iterator it = subordinates.begin();

while(it != subordinates.end()){

if (id == dynamic\_cast<Employee\*>(\*it)->GetId()){

subordinates.erase(it);

break;

}

++it;

}

}

**Database.h**

#pragma once

#include "Manager.h"

#include <vector>

class DataBase{

public:

DataBase(){};

~DataBase(){}; // no need in destructor

//creates "flat" database

bool LoadFromFile(const char \*file);

//arrages "flat" database after loading from the file

void ArrangeSubordinates();

//hire a new employee

Person\* HireEmployee(Person \*p);

void DisplayDepartmentEmployees(string \_department);

//fire the employee

bool FireEmployee(int id);

void DisplayAll();

private:

vector <Person \*> employees;

};

**Database.cpp**

#include "Database.h"

#define MAX\_STR 255

bool DataBase::LoadFromFile(const char \*file){

Employee \*empl;

Manager \*mang;

string \*str = new string[7];

int i;

ifstream fin(file);

employees.clear();

if (!fin.is\_open()) return false;

while(!fin.eof()){

for (i = 0; i < 6; i++)

getline(fin, str[i], ';');

getline(fin, str[6]);

if (atoi(str[0].c\_str())){

mang = new Manager(str[2], str[3], atoi(str[4].c\_str()), atoi(str[1].c\_str()));

mang->SetDepartment(str[5]);

mang->SetSalary(atoi(str[6].c\_str()));

employees.push\_back(mang);

}

else{

empl = new Employee(str[2], str[3], atoi(str[4].c\_str()), atoi(str[1].c\_str()));

empl->SetDepartment(str[5]);

empl->SetSalary(atoi(str[6].c\_str()));

employees.push\_back(empl);

}

}

fin.close();

return true;

}

void DataBase::ArrangeSubordinates(){

vector<Person\*>::iterator it = employees.begin(), p\_it;

while (it != employees.end()) {

if (typeid(\*\*it) == typeid(Manager))

for (p\_it = employees.begin(); p\_it != employees.end(); ++p\_it)

if (typeid(\*\*p\_it) == typeid(Employee) &&

dynamic\_cast<Manager\*>(\*it)->GetDepartment() == dynamic\_cast<Employee\*>(\*p\_it)->GetDepartment())

dynamic\_cast<Manager\*>(\*it)->AddSubordinate(\*p\_it);

++it;

}

}

Person\* DataBase::HireEmployee(Person \*empl){

vector <Person\*>::iterator it = employees.begin();

while (it != employees.end()) {

if (typeid(\*empl) == typeid(Manager) && typeid(\*\*it) == typeid(Manager) &&

dynamic\_cast<Manager\*>(\*it)->GetDepartment() == dynamic\_cast<Manager\*>(empl)->GetDepartment())

return nullptr;

if (dynamic\_cast<Employee\*>(\*it)->GetId() == dynamic\_cast<Employee\*>(empl)->GetId())

return nullptr;

++it;

}

employees.push\_back(empl);

ArrangeSubordinates();

return empl;

}

bool DataBase::FireEmployee(int id){

vector<Person\*>::iterator it = employees.begin(), \_it;

while(it != employees.end()){

if(id == dynamic\_cast<Employee\*>(\*it)->GetId()){

if (typeid(\*\*it) == typeid(Employee))

for (\_it = employees.begin(); \_it != employees.end(); ++\_it) {

if (typeid(\*\*\_it) == typeid(Manager) && dynamic\_cast<Employee\*>(\*it)->GetDepartment() == dynamic\_cast<Manager\*>(\*\_it)->GetDepartment()) {

dynamic\_cast<Manager\*>(\*\_it)->DellSubordinate(id);

break;

}

}

employees.erase(it);

break;

}

++it;

}

return true;

}

void DataBase::DisplayDepartmentEmployees(string \_department){

vector<Person\*>::iterator it = employees.begin();

while(it != employees.end()){

if(\_department == dynamic\_cast<Employee\*>(\*it)->GetDepartment())

if (typeid(\*\*it) == typeid(Manager))

dynamic\_cast<Manager\*>(\*it)->Display(false);

else if (typeid(\*\*it) == typeid(Employee))

dynamic\_cast<Employee\*>(\*it)->Display(false);

++it;

}

}

void DataBase::DisplayAll(){

vector<Person\*>::iterator it = employees.begin();

while (it != employees.end()){

(\*it)->Display(0);

++it;

}

}

**main.cpp**

#include "Database.h"

int main(){

DataBase db;

Manager \*empl2;

Employee \*empl;

db.LoadFromFile("input.csv");

db.DisplayAll();

system("pause");

cout << "----------------------------------------------------------------\n\n";

db.ArrangeSubordinates();

db.DisplayAll();

system("pause");

cout << "----------------------------------------------------------------\n\n";

empl = new Employee("John", "Wolt", 19, 13);

empl->SetDepartment("PZ");

empl->SetSalary(600);

if(!(db.HireEmployee(empl))) cout << "Error" << endl;

empl2 = new Manager("Jon", "Snow", 21, 55);

empl2->SetDepartment("PZ");

empl2->SetSalary(2000);

if (!(db.HireEmployee(empl2))) cout << "Error" << endl;

db.DisplayAll();

system("pause");

cout << "----------------------------------------------------------------\n\n";

db.FireEmployee(0);

db.DisplayAll();

system("pause");

cout << "----------------------------------------------------------------\n\n";

db.DisplayDepartmentEmployees("IT");

return 0;

}