Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра ПЗ

Лабораторна робота №7

з дисципліни «Операційні системи»

з теми «Спадчина і віртуальні функції»

Виконали: ст. 1ПІ-21Б Миронюк О.В.

Гиренко В.В.

Коцюбняк В.А.

Максименко О.В.

Перевірив: Рейда О. М.

Вінниця – 2023

**Мета роботи:** Отримати практичні навички створення об'єктів - груп та

використання методів - ітераторів.

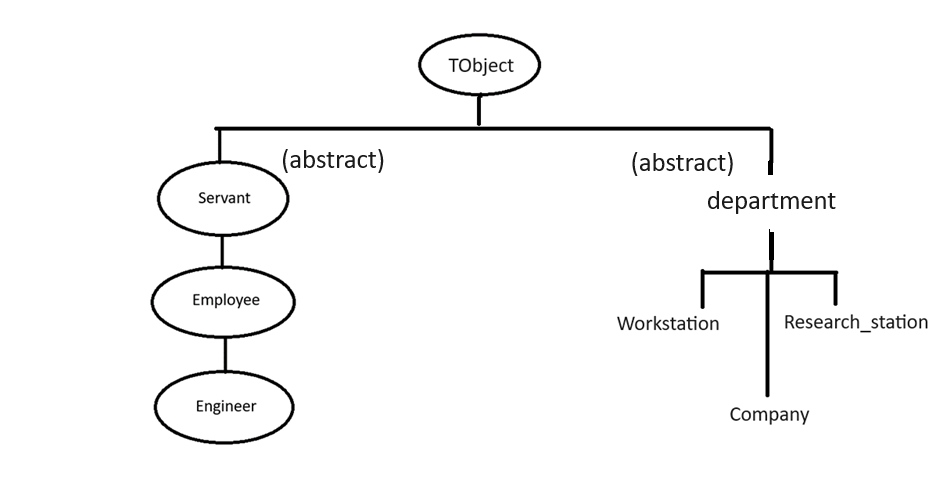
**Теоретичні відомості**

**Завдання:** Написати демонстраційну програму, в якій створюються,

показуються та руйнуються об'єкти-групи, а також демонструється

використання ітератора.

Граф ієрархії класів, де TObject – базовий клас:



**Код програми**

**Модуль TObject.h**

#pragma once

#ifndef TOBJECT\_H

#define TOBJECT\_H

class TObject

{

public:

virtual void print() = 0;

};

#endif

**Модуль SERVANT.h**

#pragma once

#include <string>

#include "TObject.h"

class servant : public TObject

{

protected:

std::string name\_;

int age\_;

int experience\_;

servant\* next;

public:

servant(std::string, int, int);

servant(servant&);

//Properties

\_\_declspec(property (put = set\_name, get = get\_name)) std::string Name;

\_\_declspec(property (put = set\_age, get = get\_age)) int Age;

\_\_declspec(property (put = set\_experience, get = get\_experience)) int Experience;

//Accessors

std::string get\_name();

void set\_name(std::string name);

int get\_age();

void set\_age(int age);

int get\_experience();

void set\_experience(int experience);

//Functions

void print();

void add(servant\* s);

static bool check\_is\_older(servant s, int n);

};

**Модуль SERVANT.cpp**

#include "servant.h"

#include <iostream>

//Accessors

std::string servant::get\_name()

{

return name\_;

}

void servant::set\_name(const std::string name)

{

name\_ = name;

}

int servant::get\_age()

{

return age\_;

}

void servant::set\_age(const int age)

{

age\_ = age;

}

int servant::get\_experience()

{

return experience\_;

}

void servant::set\_experience(const int experience)

{

experience\_ = experience;

}

//Constructors

servant::servant(std::string name = "Default\_name", int age = 18, int experience = 0)

{

name\_ = name;

age\_ = age;

experience\_ = experience;

};

servant::servant(servant& s) : servant(s.Name, s.Age, s.Experience) {}

void servant::print()

{

std::cout << "Servant: Name: " << this->Name << "; Age: " << this->Age << "; Experience: " << this->Experience << ";\n";

}

void servant::add(servant\* s) {

servant\* el = this;

while (el->next) {

el = el->next;

}

el->next = s;

}

bool servant::check\_is\_older(servant s, int n) {

if (s.Age >= n) {

return true;

}

return false;

}

**Модуль engineer.h**

#pragma once

#include "employee.h"

class engineer :

public employee

{

public:

engineer\* next;

using employee::employee;

virtual void print();

void add(engineer\* e);

};

**Модуль engineer.cpp**

#include "engineer.h"

#include <iostream>

void engineer::print()

{

std::cout << "Engineer: Name: " << this->Name << "; Age: " << this->Age << "; Experience: " << this->Experience << ";\n";

}

void engineer::add(engineer\* e) {

engineer\* el = this;

while (el->next) {

el = el->next;

}

el->next = e;

}

**Модуль employee.h**

#pragma once

#include "servant.h"

class employee :

public servant

{

public:

using servant::servant;

~employee(){}

employee\* next;

virtual void print();

void add(employee\*);

};

**Модуль employee.cpp**

#include "employee.h"

#include <iostream>

void employee::print()

{

std::cout << "Employee: Name: " << this->Name << "; Age: " << this->Age << "; Experience: " << this->Experience << ";\n";

}

void employee::add(employee\* e) {

employee\* el = this;

while (el->next) {

el = el->next;

}

el->next = e;

}

**Модуль workstation.h**

#pragma once

#include "department.h"

#include "employee.h"

class workstation :

public department

{

protected:

employee\* employee\_;

public:

using department::department;

workstation();

workstation(std::string name, employee\* employee);

~workstation() = default;

void print();

void iterate(bool (f)(servant s, int), int a) override;

};

**Модуль workstation.cpp**

#include "workstation.h"

#include <iostream>

void workstation::iterate(bool (f)(servant e, int), int a) {

employee\* e = employee\_;

do {

if (f(\*e, a)) {

e->print();

} e = e->next;

} while (e);

}

workstation::workstation(std::string n, employee\* employee)

{

name = n;

employee\_ = employee;

}

void workstation::print() {

std::cout << "Workstation" << std::endl;

}

**Модуль research\_station.h**

#pragma once

#include "department.h"

#include "engineer.h"

class research\_station :

public department

{

protected:

engineer\* engineer\_;

public:

using department::department;

research\_station(std::string, engineer\*);

~research\_station() = default;

void print();

void iterate(bool (f)(servant e, int), int a) override;

};

**Модуль research\_station.cpp**

#include "research\_station.h"

#include <iostream>

#include "engineer.h"

void research\_station::iterate(bool (f)(servant e, int), int a) {

engineer\* e = engineer\_;

do {

if (f(\*e, a)) {

e->print();

} e = e->next;

} while (e);

}

void research\_station::print() {

std::cout << "Workstation" << std::endl;

}

research\_station::research\_station(std::string name\_, engineer\* engineer)

{

name = name\_;

engineer\_ = engineer;

}

**Модуль department.h**

#pragma once

#include "servant.h"

#include "TObject.h"

class department :

public TObject

{

protected:

std::string name;

public:

department\* next;

department(std::string);

department(department&);

department();

virtual ~department();

std::string get\_name();

void set\_name(char\* NAME);

void add(department\* s);

void show();

void print();

virtual void iterate(bool (f)(servant w, int), int a) = 0;

};

**Модуль department.cpp**

#include "department.h"

#include <iostream>

department::department(std::string name\_)

{

name = name\_;

};

department::department(department& d) : department(d.name) {};

department::department() {};

department::~department(){};

std::string department::get\_name()

{

return name;

};

void department::set\_name(char\* NAME)

{

name = NAME;

};

void department::add(department\* d)

{

department\* el = this; while (el->next) {

el = el->next;

}

el->next = d;

};

void department::print()

{

}

void department::iterate(bool f(servant s, int), int a)

{

std::cout << "boo";

}

**Модуль company.h**

#pragma once

#include "department.h"

class company :

public department

{

public:

using department::department;

department\* department\_;

company();

company(std::string n, department\* d);

~company() override;

void print();

void iterate(bool (f)(servant m, int), int a) override;

};

**Модуль company.cpp**

#include "company.h"

company::company(std::string n, department\* d) {

name = n;

department\_ = d;

}

void company::print() {}

void company::iterate(bool (f)(servant m, int), int a) {

department\* u = department\_;

do {

u->iterate(f, a);

u = u->next;

} while (u);

}

company::~company() = default;

**Модуль main.cpp**

#include <iostream>

#include "company.h"

#include "employee.h"

#include "engineer.h"

#include "research\_station.h"

#include "servant.h"

#include "workstation.h"

using namespace std;

int main()

{

employee empl1 ("rob", 25, 6);

employee empl2 ("rob1", 26, 7);

employee empl3 ("rob2", 56, 8);

empl1.add(&empl2);

empl2.add(&empl3);

engineer engin1("chris", 30, 10);

engineer engin2("chris1", 35, 15);

engineer engin3("chris2", 300, 18);

engin1.add(&engin2);

engin2.add(&engin3);

workstation workstation\_("Workstation1", &empl1);

research\_station research\_station\_("Research1", &engin1);

workstation\_.add(&research\_station\_);

company company\_("company", &workstation\_);

company\_.iterate(servant::check\_is\_older, 30);

}

**Результат тестування роботи програми**

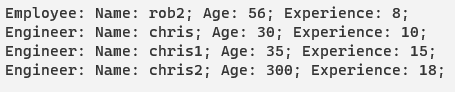


Рисунок 1.1 – Результат тесутвання програми

Висновок: ми навчились створювати програму на мові програмування С++, працювати з класами, їх ієрархією, вказівниками та віртуальними функціями.