 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №3**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Успадкування. Поліморфізм»

Варіант №15

**Виконав:**

студент гр. БС-71

Орлівський С. П.

**Перевірив:**

асистент каф. БМК

Рисін С.В.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2019

**🞏 Практичне заняття без зауважень**

**🞏 Практичне заняття має зауваження:**

**🞏 несвоєчасний захист**

**🞏 присутні зауваження до UML діаграми:**

**🞏 діаграма класу не відповідає коду**

**🞏 виконані не за стандартом:**

**🞏 атрибути**

**🞏 відношення**

**🞏 потужність**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 присутні зауваження до коду:**

**🞏 задача завдання вирішена хибно**

**🞏 код програми не компілюється**

**🞏 хибно задані специфікатори доступу**

**🞏 помилки у визначенні конструкторів / деструкторів**

**🞏 відсутні списки ініціалізації в конструкторах**

**🞏 константні методи**

**🞏 використано глобальні змінні**

**🞏 статичні змінні при роботі з масивами**

**🞏 оформлення коду**

**🞏 присутні зайві символи «{» та «}»**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 невірні відповіді на запитання:**

**🞏 №1 🞏 №2 🞏 №3 🞏 №4 🞏 №5**

**🞏 №6 🞏 №7 🞏 №8 🞏 №9 🞏 №10**

**🞏 маються інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Завдання:**

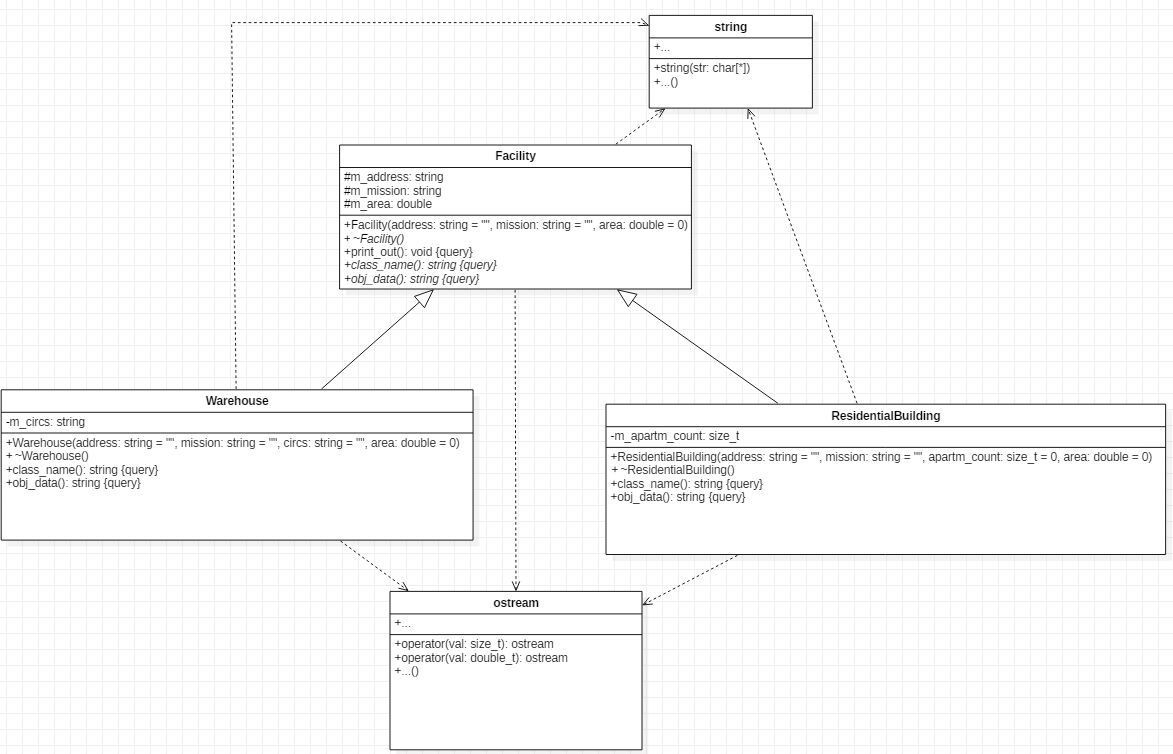
**Завдання:**

1. Вивчити теоретичні основи визначення класів, що використовують спадкування властивостей, та побудови ієрархії класів в мові С++.
2. Розробити діаграму класів в нотації UML та програмний застосунок з використанням успадкування, що реалізує ієрархію класів відповідно до свого варіанту (визначення класів та їх реалізації мають бути розташовані у файлах \*.h та \*.сpp відповідно; для ініціалізації даних класів використовувати введення з клавіатури):

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | Будівля (адреса, загальна площа, призначення) | Жилий будинок (кількість квартир) | Використовуючи віртуальні функції, не знаючи з об’єктом якого класу ви працюєте, виведіть на екран швидкість та компанію виробника для засобу пересування |
| Сховище (умови зберігання) |

1. Скласти і захистити звіт.

**UML діаграма:**

****

**Код програми:**

#pragma once

#include <string>

#include <iostream>

#include <cmath>

class **Facility**

{

public:

**Facility**(std::string address = "", std::string mission = "", double area = 0);

void **print\_out**() const;

virtual std::string ***class\_name***()const = 0 ;

virtual std::string ***obj\_data***() const = 0;

virtual ~***Facility***() = 0;

protected:

std::string m\_address, m\_mission;

double m\_area;

};

**facility.cpp:**

#include "facility.h"

Facility::**Facility**(std::string address, std::string mission, double area)

: m\_address(address), m\_mission(mission), m\_area(area)

{

}

Facility::~***Facility***()

{

}

void Facility::**print\_out**() const

{

std::cout << *class\_name*();

std::cout << *obj\_data*();

}

std::string Facility::***obj\_data***() const

{

char buffer[57];

sprintf(buffer, "address: %s\nmission: %s\narea: %f\n", m\_address.c\_str(),

m\_mission.c\_str(), m\_area);

return buffer;

}

**warehouse.h:**

#pragma once

#include "facility.h"

class **Warehouse** : public Facility

{

public:

**Warehouse**(std::string address = "", std::string mission = "",

std::string circs = "", double area = 0);

std::string ***class\_name***() const override;

std::string ***obj\_data***() const override;

~***Warehouse***() override;

private:

std::string m\_circs;

};

**warehouse.cpp:**

#include "warehouse.h"

Warehouse::**Warehouse**(std::string address, std::string mission, std::string circs, double area)

:Facility (address, mission, area), m\_circs(circs)

{

}

std::string Warehouse::***class\_name***() const

{

return "Class name: Warehouse\n";

}

std::string Warehouse::***obj\_data***() const

{

std::string base\_info(Facility::obj\_data());

char buffer[125];

sprintf(buffer, "Circumstances: %s\n", m\_circs.c\_str());

base\_info += (std::string)buffer;

return base\_info;

}

Warehouse::~***Warehouse***()

{

}

**residentialbuilding.h:**

#pragma once

#include "facility.h"

class **ResidentialBuilding** : public Facility

{

public:

**ResidentialBuilding**(std::string address = "", std::string mission = "",

size\_t apartm\_count = 0, double area = 0);

std::string ***class\_name***() const override;

std::string ***obj\_data***() const override;

~***ResidentialBuilding***() override;

private:

size\_t m\_apartm\_count;

};

**residentialbuilding.cpp:**

#include "residentialbuilding.h"

ResidentialBuilding::**ResidentialBuilding**(std::string address, std::string mission, size\_t apartm\_count,

double area)

: Facility(address, mission, area), m\_apartm\_count(apartm\_count)

{

}

std::string ResidentialBuilding::***obj\_data***() const

{

std::string base\_info(Facility::obj\_data());

char buffer[57];

sprintf(buffer, "Number of apartments: %d\n", m\_apartm\_count);

base\_info += (std::string)buffer;

return base\_info;

}

std::string ResidentialBuilding::***class\_name***() const

{

return "Class: ResidentialBuilding\n";

}

ResidentialBuilding::~***ResidentialBuilding***()

{

}

**main.cpp:**

#include <iostream>

#include "facility.h"

#include "warehouse.h"

#include "residentialbuilding.h"

#include <algorithm>

int **main**()

{

const size\_t SIZE = 5;

Facility \*\*facility\_arr = new Facility\*[SIZE];

facility\_arr[0] = new Warehouse("56783", "edsfgkjfdsj", "good", 345.4);

facility\_arr[1] = new Warehouse("11111", "1111111111", "verygood", 1111);

facility\_arr[2] = new ResidentialBuilding("351234513", "wqereqwrt", 1124, 234.345);

facility\_arr[3] = new ResidentialBuilding("22222222", "smthsmth", 12.4, 123.5);

facility\_arr[4] = new Warehouse("00000", "000000", "0good", 555.9);

std::random\_shuffle(facility\_arr, facility\_arr + SIZE);

for (size\_t i = 0; i < SIZE; ++i)

{

std::cout << "\n";

facility\_arr[i]->print\_out();

delete facility\_arr[i];

std::cout << "\n";

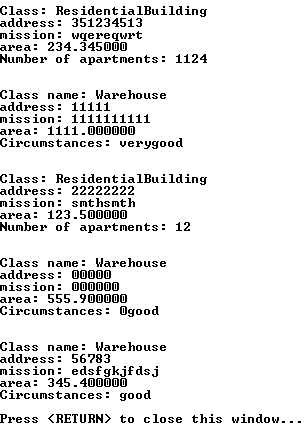
}

delete [] facility\_arr;

return 0;

}

**Результат роботи програми:**

****

**Контрольні запитання**

* 1. Що означає поняття успадкування?

Поняття успадкування означає, що один клас може успадкувати усі методи та поля (поведінку) класу від якого він наслідується (супер/базового/батьківського класу)

* 1. Який клас називається базовим, а який є класом нащадком?

Базовий клас – клас від якого унаслідуваний хоча б один клас. Нащадок – клас, що наслідується (є похідним) від базового класу.

* 1. Скільки базових класів може мати похідний клас?

Від одного до безлічи.

* 1. Чи може похідний клас бути базовим?

Так, базовим для класів, котрі наслідуються від нього.

* 1. Як змінюється доступ до елементів базового класу при спадкуванні з різними специфікаторами доступу?

На такий випадок я маю табличку, в котрій все набагато зрозуміліше виглядає, ніж, якщо я почну словесно усе розписувати:



До private та protected полів ми не маємо доступу з функцій поза класом та зі сторонніх класів, що не наслідуються від данного. Хоча навіть у випадку наслідування до private поля базового класу ми не маємо доступу з класів нащадків (як можна побачити в 3й колонці таблички. Клас нащадок має ці приватні поля у собі, однак прямого доступу до них не має). Тобто користувач буде мати прямий доступ з програми лише до public секції супер класу та класу нащадка тільки при public наслідуванні. Для отримання доступ до полів супер класу з класу нащадка ми можемо використати усі 3 специфікатори доступу (до яких саме полів отримаємо доступ залежить вже, як і було сописано вище, від конкретного специфікатору)

* 1. У чому різниця між простим та множинним наслідуванням?

При множинному наслідуванні клас копіює поведінку усіх класів від яких він наслідується (усі поля та методи). При одинармому наслідуванні відбувається теж саме з поправкою, що базовий клас всього один.

Nf

7. Чи можна з класу-нащадка одержати доступ до private частини базового класу, якщо специфікатор доступу при спадкуванні private?

До приватної частини базового класу не можна доступитися з класів нащадків при жодному специфікаторі наслідування.

8. Чи успадковуються конструктори та деструктори?

Ні.

9. Що таке віртуальна функція?

Віртуальна функція – це така функція базового класу, що може бути перевизначена у класах нащадках. При цьому така функція викликається під час run time. При виклику такої функції через вказівник на базовий клас, що вказує на клас-нащадок, буде викликана саме перевизначена функція класу нащадка, якщо така є (на відміну від не віртуальних функцій).

10.Чи можна віртуальну функцію визначити як static, відповідь пояснити.

Ні, не можна. Адже віртуальна функція працює з конкретним об’єктом того класу, що її перевизначив, а staic функція може бути викликана взагалі без об’єктів.