Міністерство освіти і науки України Національний

технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

**ЗВІТ**

при виконанні комп’ютерного практику №2

з дисципліни «Об’єктно-орієнтоване програмування»

Виконав:

Студент II курсу

Групи КА-03

Курилюк Р. М.

Варіант №16

Перевірив:

Шолохов О.В

Київ—2021

**Мета роботи:** Навчитись правильно описувати ієрархії об’єктів та ієрархії

класів мовою С++.

**Основні теоретичні відомості:**

*Ієрархічність та її різновиди*. Для подолання складності системи будують як ієрархічні. Ієрархія – це впорядкування абстракцій, розташування їх по рівнях. Розрізняють два різновиди ієрархій: ієрархія класів або спадковість і ієрархія об’єктів або агрегація.

*Спадковість* – це механізм отримання нового класу із існуючих шляхом

запозичення структурної або функціональної частини одного або декількох інших. Відповідно розрізняють одиночне та множинне успадкування. Спадковість створює таку ієрархію абстракцій, у якій підкласи (похідні класи, класи-нащадки) успадковують будову і поведінку одного або декількох суперкласів (базових класів, батьківських класів). Кажуть, що спадковість описується відношенням «is-a» і породжує ієрархію «узагальнення-спеціалізація».

*Агрегація* – це такий різновид ієрархії, який передбачає використання об’єктів одного класу в оголошенні іншого класу. Агрегацію (або включення) можна описати відношенням «part of» або «бути частиною». Клас, що містить поля – об’єкти інших класів, називається агрегатом або контейнером.

*Одиночна спадковість*. *Реалізація в С++* . Ідея спадковості одна з основних в об’єктно-орієнтованому програмуванні. З погляду фізичної реалізації побудова нового класу на базі старого дає можливість повторного використання старого коду і додавання нових властивостей, зміни деяких аспектів поведінки за рахунок заміни коду деяких методів (функцій) і прав доступу. Синтаксис оголошення наступний:

*class Student*

*{char\* name, \*group;..}*

*class Starosta: public Student{int level; . . . }*

Клас Starosta – похідний від класу Student, а Student – базовий для Starosta.

Клас Starosta крім своїх власних членів (level) містить і члени класу Student.

Виведення похідного класу із базового робить його підтипом базового. Тому вказівник на базовий клас можна використовувати як вказівник на похідний (але не навпаки).

Наприклад:

*void f(Student \* p, Starosta \*q)*

*{Student \*KA01 [10];*

*KA01[0] = p;*

*KA02[1] = q;...}*

Щоб клас можна було використовувати як базовий, його потрібно повністю визначити до визначення похідного, недостатньо тільки оголосити.

Для членів похідного класу доступні усі відкриті (public) та захищені (protected) члени базового класу напряму, як власні. Закриті поля (private) базового класу входять до складу похідного, але недоступні напряму – тільки через відкриті методи базового класу.

В С++ існує три різновиди успадкування: відкрите (public), захищене (protected) та за умовчанням закрите (private) За допомогою цього механізму похідний клас може змінити рівень доступу до успадкованої від базового класу частини своїх об’єктів.

Загальне правило: за допомогою похідного класу елементи базового класу не можна зробити більш відкритими (тільки більш закритими).

*Конструктори похідних класів.* В оголошенні любого конструктора похідного класу потрібно враховувати наступне.

* Конструювання об’єкту похідного класу обов’язково відбувається з викликом конструктора базового класу.
* Якщо явний виклик конструктора базового класу не передбачено у конструкторі похідного, за умовчання буде викликано конструктор без параметрів базового класу.
* Для організації виклику потрібної версії конструктора базового класу, можна скористатись списком ініціалізації, наприклад:

*Student :: Student (char\*a, char\* b): name (a), group (b) {...}*

*Starosta :: Starosta (char\*a, char\* b, int c): Student (a,b), level (c) {...}*

Об’єкти створюються згори вниз: спочатку базовий клас, тоді члени похідного класу, потім сам похідний клас. Знищуються – в протилежному порядку. Про це особливо важливо пам’ятати. коли в деструкторі необхідно явно звільняти ресурси.

Похідний клас може бути одночасно базовим для іншого класу. Такий набір зв’язаних класів називається ієрархією класів.

*Переозначення функцій в похідних класах*. Якщо в похідному класі створена функція з таким самим ім’ям, що і в базовому класі, то має місце заміщення методу. Кажуть, що в цьому випадку метод похідного класу приховує метод або методи (якщо їх декілька з однаковими іменами і різними сигнатурами) базового класу.

*Множинне успадкування*. Допустиме у С++ множинне успадкування (multiple inheritance) дає можливість отримати похідний клас від декількох базових. Для опису ієрархії множинного успадкування можна використати орієнтований ациклічний граф.

Синтаксис заголовка класу розширюється, щоб можна було використати список базових класів з атрибутами. Наприклад:

*class A: public B, public C{...}*

Створення об’єкта похідного класу відбувається з викликом конструкторів базових класів у тій послідовності, в якій визначена спадковість, навіть якщо у списку ініціалізації вони описані у зворотному порядку. Для розглянутого приклада: B, C, A.

**Завдання**

Відповідно варіанту до тексту першої лабораторної роботи потрібно внести

наступні зміни:

* визначити базовий та похідний класи відповідно таблиці 2.1.
* переписати оголошення похідного класу відповідно синтаксису для відкритого успадкування;
* переробити оголошення й реалізацію конструкторів похідного класу, у конструкторі з параметрами обов’язково передбачити використання конструктора з параметрами базового класу;
* у реалізації функції виведення на екран інформації в похідному класі використати відповідну функцію базового класу;
* до конструкторів та деструкторів базового, похідного та одного з агрегованих класів долучити виведення відповідної контрольної інформації для можливості відстеження порядку створення та знищення об’єктів;

У тестовому прикладі створити по одному об’єкту кожного з класів для

контролю етапів утворення відповідних об’єктів й викликати функції

виведення інформації. Пояснити отримані результати.

Варіант 13

Базовий клас – «Повар»

Похідний клас – «Страва»

**Діаграма класів**

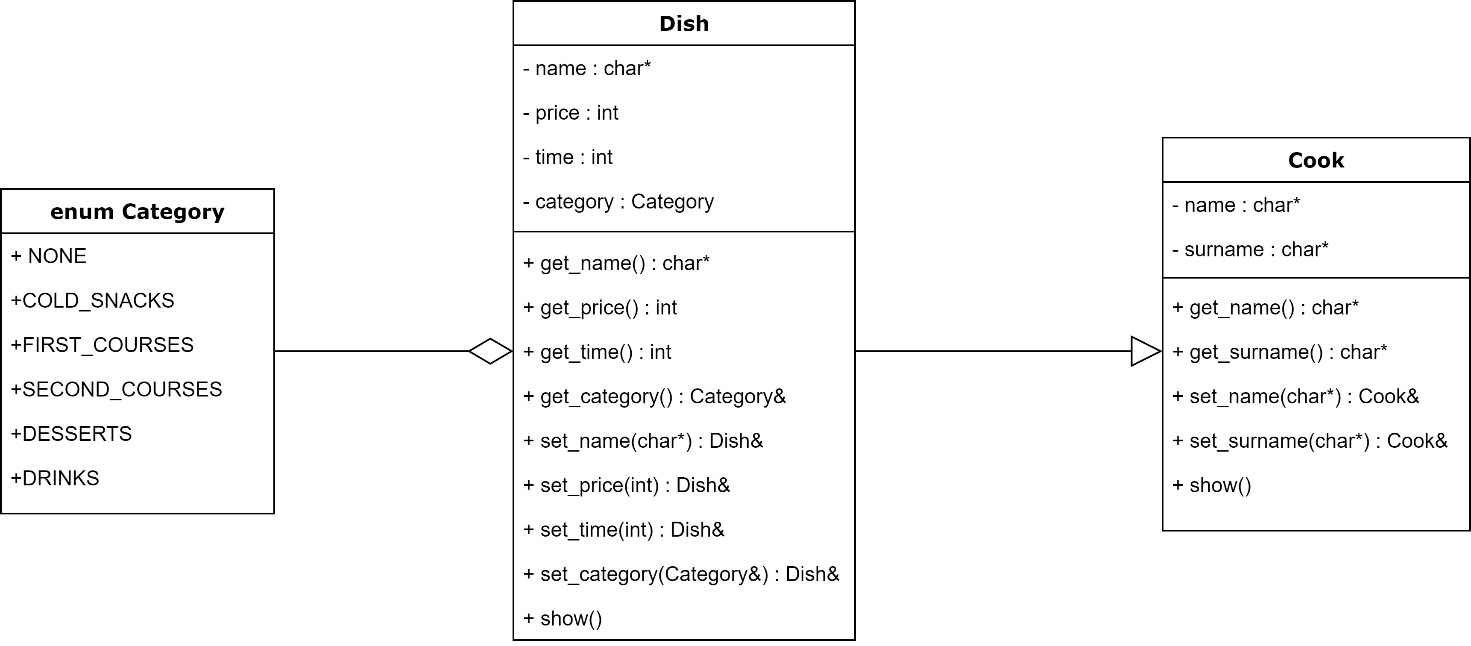
****

Рис 1. Діаграма класів

**Лістинг програми**

**Source.cpp**

#include <locale.h>

#include <iostream>

#include <conio.h>

#include "Dish.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "Ukrainian");

cout << "Введiть iм'я i прiзвище кухаря" << endl;

char\* name\_cook = new char;

char\* surname\_cook = new char;

cin >> name\_cook >> surname\_cook;

Cook cook(name\_cook, surname\_cook);

cook.show();

cout << endl << "Введiть iм'я страви, цiну, час приготування, категорiю, iм'я i прiзвище кухаря" << endl;

char\* name\_dish = new char;

int price = 0, time = 0, category = 0;

Category temp\_category;

cin >> name\_dish >> price >> time >> category >> name\_cook >> surname\_cook;

switch (category) {

case 0:

temp\_category = NONE;

break;

case 1:

temp\_category = COLD\_SNACKS;

break;

case 2:

temp\_category = FIRST\_COURSES;

break;

case 3:

temp\_category = SECOND\_COURSES;

break;

case 4:

temp\_category = DESSERTS;

break;

case 5:

temp\_category = DRINKS;

break;

default:

break;

}

Dish dish(name\_dish, price, time, temp\_category, name\_cook, surname\_cook);

dish.show();

\_getch();

return 0;

}

**Category.cpp**

enum Category {

NONE,

COLD\_SNACKS,

FIRST\_COURSES,

SECOND\_COURSES,

DESSERTS,

DRINKS

};

**Cook.h**

#pragma once

class Cook {

private:

char\* name, \* surname;

public:

Cook();

Cook(char\*, char\*);

Cook(Cook&);

~Cook();

char\* get\_name();

char\* get\_surname();

Cook& set\_name(char\*);

Cook& set\_surname(char\*);

void show();

};

**Cook.cpp**

#include <string>

#include <iostream>

#include "Cook.h"

using namespace std;

Cook::Cook()

{

name = new char[7];

strcpy\_s(name, sizeof "noname", "noname");

surname = new char[10];

strcpy\_s(surname, sizeof "nosurname", "nosurname");

cout << "Конструктор Cook за замовчування " << this << endl;

}

Cook::Cook(char\* \_name, char\* \_surname)

{

name = new char[strlen(\_name) + 1];

strcpy\_s(name, strlen(\_name) + 1, \_name);

surname = new char[strlen(\_surname) + 1];

strcpy\_s(surname, strlen(\_surname) + 1, \_surname);

cout << "Конструктор Cook з параметрами " << this << endl;

}

Cook::Cook(Cook& \_cook)

{

name = new char[strlen(\_cook.name) + 1];

strcpy\_s(name, strlen(\_cook.name) + 1, \_cook.name);

surname = new char[strlen(\_cook.surname) + 1];

strcpy\_s(surname, strlen(\_cook.surname) + 1, \_cook.surname);

cout << "Конструктор Cook копіювання " << this << endl;

}

Cook::~Cook()

{

cout << "Деструктор Cook " << this << endl;

}

char\* Cook::get\_name()

{

return name;

}

char\* Cook::get\_surname()

{

return surname;

}

Cook& Cook::set\_name(char\* \_name)

{

name = new char[strlen(\_name) + 1];

strcpy\_s(name, strlen(\_name) + 1, \_name);

return \*this;

}

Cook& Cook::set\_surname(char\* \_surname)

{

surname = new char[strlen(\_surname) + 1];

strcpy\_s(surname, strlen(\_surname) + 1, \_surname);

return \*this;

}

void Cook::show()

{

cout << name << ", " << surname << endl;

}

**Dish.h**

#pragma once

#include "Category.cpp"

#include "Cook.h"

class Dish :

public Cook

{

private:

char\* name\_dish;

int price, time;

Category category;

public:

Dish();

Dish(char\*, int, int, Category&);

Dish(char\*, int, int, Category&, Cook&);

Dish(char\*, int, int, Category&, char\*, char\*);

Dish(Dish&);

~Dish();

char\* get\_name();

int get\_price();

int get\_time();

Category& get\_category();

Dish& set\_name(char\*);

Dish& set\_price(int);

Dish& set\_time(int);

Dish& set\_category(Category&);

void show();

};

**Dish.cpp**

#include <string>

#include <iostream>

#include "Dish.h"

using namespace std;

Dish::Dish()

{

name\_dish = new char[7];

strcpy\_s(name\_dish, sizeof "noname", "noname");

price = 0;

time = 0;

category = NONE;

cout << "Конструктор Dish за замовчування " << this << endl;

}

Dish::Dish(char\* \_name\_dish, int \_price, int \_time, Category& \_category) :

Cook()

{

name\_dish = new char[strlen(\_name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_name\_dish) + 1, \_name\_dish);

price = \_price;

time = \_time;

category = \_category;

cout << "Конструктор Dish з параметрами " << this << endl;

}

Dish::Dish(char\* \_name\_dish, int \_price, int \_time, Category& \_category, Cook& \_cook) :

Cook(\_cook)

{

name\_dish = new char[strlen(\_name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_name\_dish) + 1, \_name\_dish);

price = \_price;

time = \_time;

category = \_category;

cout << "Конструктор Dish з параметрами " << this << endl;

}

Dish::Dish(char\* \_name\_dish, int \_price, int \_time, Category& \_category, char\* \_name, char\* \_surname) :

Cook(\_name, \_surname)

{

name\_dish = new char[strlen(\_name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_name\_dish) + 1, \_name\_dish);

price = \_price;

time = \_time;

category = \_category;

cout << "Конструктор Dish з параметрами " << this << endl;

}

Dish::Dish(Dish& \_dish):

Cook(\_dish.get\_name(), \_dish.get\_surname())

{

name\_dish = new char[strlen(\_dish.name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_dish.name\_dish) + 1, \_dish.name\_dish);

price = \_dish.price;

time = \_dish.time;

category = \_dish.category;

cout << "Конструктор Dish копіювання " << this << endl;

}

Dish::~Dish()

{

cout << "Деструктор Dish " << this << endl;

}

char\* Dish::get\_name()

{

return name\_dish;

}

int Dish::get\_price()

{

return price;

}

int Dish::get\_time()

{

return time;

}

Category& Dish::get\_category()

{

return category;

}

Dish& Dish::set\_name(char\* \_name\_dish)

{

delete[] name\_dish;

name\_dish = new char[strlen(\_name\_dish) + 1];

strcpy\_s(name\_dish, strlen(\_name\_dish) + 1, \_name\_dish);

return \*this;

}

Dish& Dish::set\_price(int price)

{

price = price;

return \*this;

}

Dish& Dish::set\_time(int time)

{

time = time;

return \*this;

}

Dish& Dish::set\_category(Category& category)

{

category = category;

return \*this;

}

void Dish::show()

{

cout << name\_dish << ", " << price << " грн, " << time << "хв, ";

switch (category)

{

case NONE:

cout << "NONE" << ", ";

break;

case COLD\_SNACKS:

cout << "холоднi закуски" << ", ";

break;

case FIRST\_COURSES:

cout << "перша страва" << ", ";

break;

case SECOND\_COURSES:

cout << "друга страва" << ", ";

break;

case DESSERTS:

cout << "десерт" << ", ";

break;

case DRINKS:

cout << "напiй" << ", ";

break;

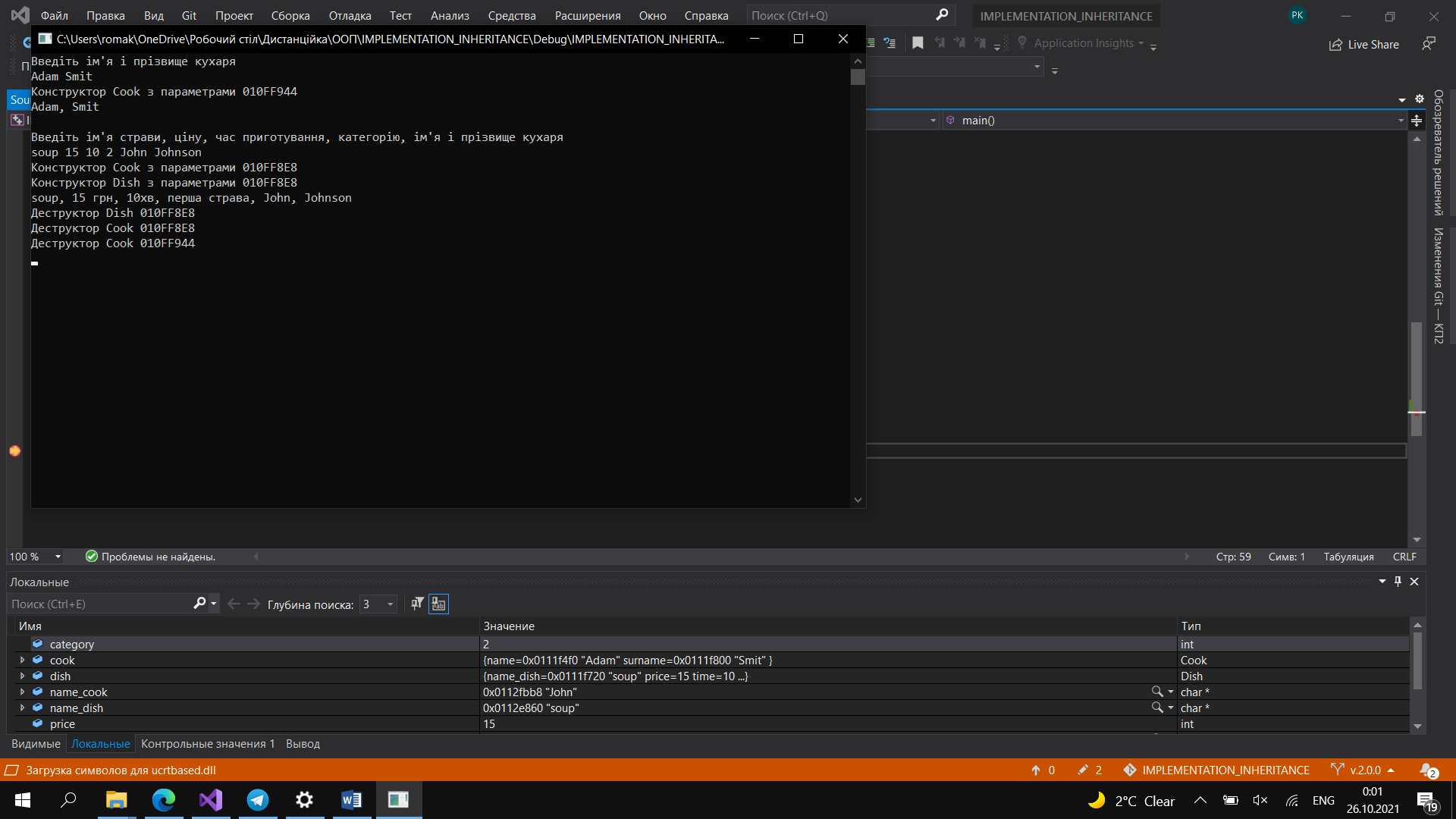
default:

break;

}

Cook::show();

}

**Результати роботи програми**

**Висновок**

Під час виконання комп'ютерного практикуму я навчився описувати ієрархії об’єктів та ієрархії класів мовою С++.

Було перероблено тип “Страва” як похідний від класу “Кухар”. Наданий тестовий приклад відповідно до загальних вимог та з виведенням інформації про об'єкти класу “Кухар” та “Страва”, а також повідомлення про конструювання або деструкцію.