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**

Варіант 13

Тип «Категорія» визначити як перерахування (enum) зі значеннями полів «холодні закуски», «перші страви», «другі страви», «десерти», «напої».

Тип «Повар» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини;

- усі функції реалізувати відповідно загальним вимогам.

Тип «Страва» визначити як клас, що містить:

- закриті поля типу символьного рядка з назвою, цілі з вартістю та тривалістю приготування, та типів «Категорія» і «Повар»;

- функції реалізувати відповідно загальним вимогам.

Тип «Замовлення» визначити як клас, що містить:

- закриті поля з назвою кафе та поточною датою (тип Date);

- інформацію про включені до даного замовлення страви оформити як динамічний масив (вказівник і розмірність, поля закриті);

- передбачити функцію для внесення до замовлення нової страви;

- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою кафе, датою та

часом очікування даного замовлення;

- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти «Замовлення» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати нову страву, вивести оновлену інформацію.

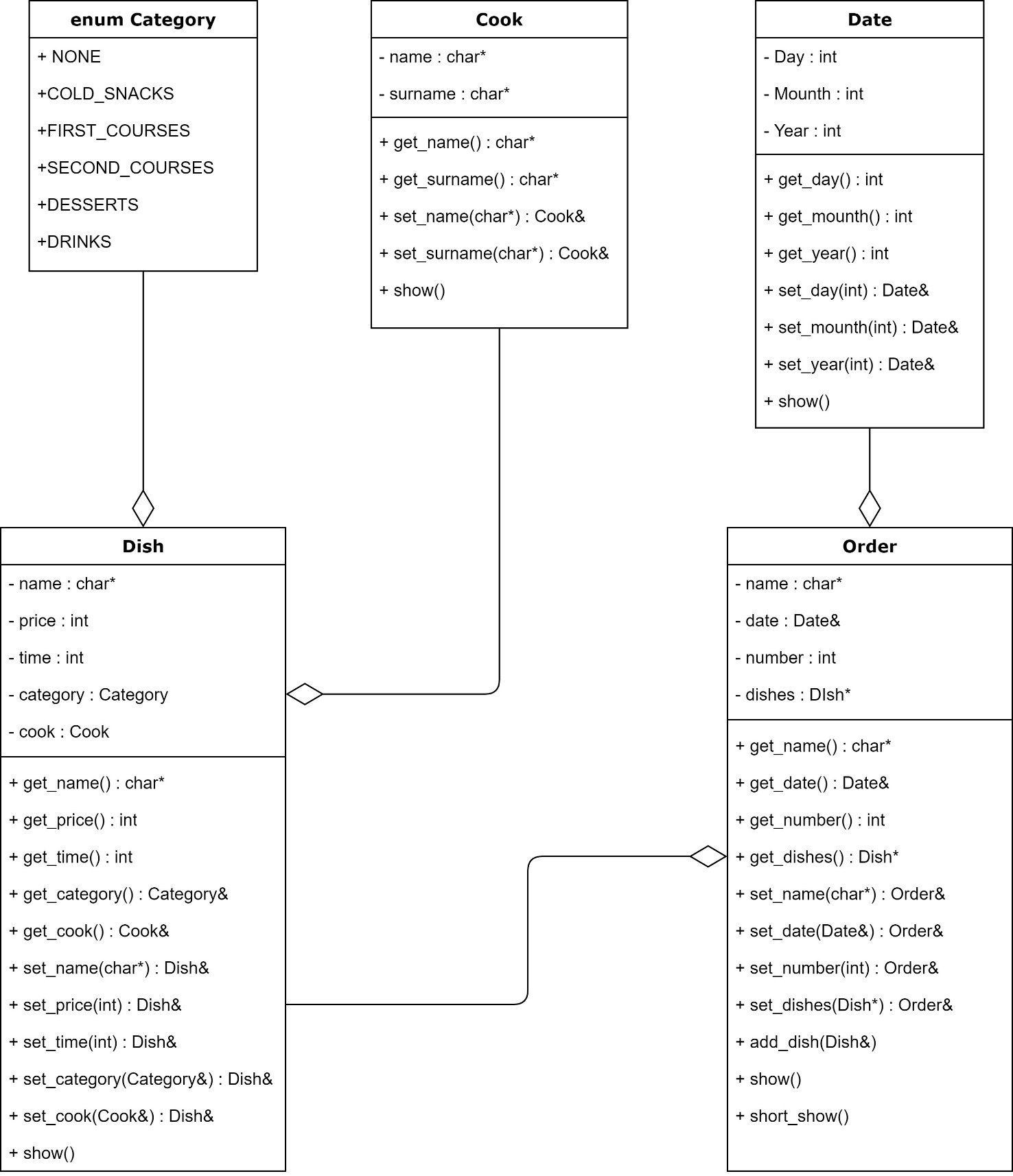
**Діаграма класів**

Рис 1. Діаграма класів

**Лістинг програми**

**Source.cpp**

#include <locale.h>

#include <iostream>

#include <conio.h>

#include "Order.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "Ukrainian");

cout << "За замовчуванням" << endl;

Order order1;

\_getch();

cout << "Повний вивiд" << endl;

order1.show();

cout << endl;

\_getch();

cout << "скорочений вивiд" << endl;

order1.short\_show();

cout << endl;

\_getch();

cout << endl << "З параметрами" << endl;

\_getch();

char\* name = new char;

cout << "Введiть iм'я замовлення" << endl;

cin >> name;

int day = 0, mounth = 0, year = 0;

cout << "Введiть день, мiсяць i рiк" << endl;

cin >> day >> mounth >> year;

Date date(day, mounth, year);

int number;

cout << "Введiть кiлькiсть страв у замовленнi" << endl;

cin >> number;

Dish\* dishes = new Dish[number+1];

char\* name\_dish = new char;

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

char\* name\_cook = new char, \* surname\_cook = new char;

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

for (int i = 0; i < number; i++) {

cout << i + 1 << " страва:" << endl;

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

enum Category temp\_category;

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;

}

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

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

dishes[i].set\_name(name\_dish);

dishes[i].set\_price(price);

dishes[i].set\_time(time);

dishes[i].set\_category(temp\_category);

dishes[i].set\_cook(temp\_cook);

}

Order order2(name, date, number, dishes);

cout << endl << "Повний вивiд" << endl;

order2.show();

\_getch();

cout << endl << "Скорочений вивiд" << endl;

order2.short\_show();

cout << endl;

\_getch();

cout << endl << "Копiювання другого" << endl;

\_getch();

Order order3(order2);

cout << endl << "Повний вивiд" << endl;

order2.show();

\_getch();

cout << endl << "Скорочений вивiд" << endl;

order2.short\_show();

cout << endl;

\_getch();

cout << endl << "Додавання страви" << endl;

\_getch();

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

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

enum Category temp\_category;

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;

}

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

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

order3.add\_dish(temp\_dish);

cout << endl << "Повний вивiд" << endl;

order3.show();

\_getch();

cout << endl << "Скорочений вивiд" << endl;

order3.short\_show();

cout << endl;

\_getch();

\_getch();

}

**Date.h**

#pragma once

class Date

{

private:

long long int day, mounth, year;

public:

Date();

Date(int, int, int);

Date(Date&);

~Date();

int get\_day();

int get\_mounth();

int get\_year();

Date& set\_day(int);

Date& set\_mounth(int);

Date& set\_year(int);

void show();

};

**Date.cpp**

#include <time.h>

#include <iostream>

#include <iomanip>

#include "Date.h"

#pragma warning(disable : 4996) //Щоб не ї###о мозок

using namespace std;

Date::Date()

{

struct tm\* tim;

time\_t tt = time(NULL); //взяття часу

tim = localtime(&tt);

this->day = tim->tm\_mday;

this->mounth = tim->tm\_mon + 1;

this->year = tim->tm\_year - 100;

}

Date::Date(int day, int mounth, int year)

{

this->day = day;

this->mounth = mounth;

this->year = year;

}

Date::Date(Date& temp)

{

this->day = temp.day;

this->mounth = temp.mounth;

this->year = temp.year;

}

Date::~Date()

{

}

void Date::show()

{

cout << this->day << "." << this->mounth << "." << this->year;

}

int Date::get\_day()

{

return this->day;

}

int Date::get\_mounth()

{

return this->mounth;

}

int Date::get\_year()

{

return this->year;

}

Date& Date::set\_day(int day)

{

this->day = day;

return \*this;

}

Date& Date::set\_mounth(int mounth)

{

this->mounth = mounth;

return \*this;

}

Date& Date::set\_year(int year)

{

this->year = year;

return \*this;

}

**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()

{

this->name = new char[7];

strcpy\_s(this->name, sizeof "noname", "noname");

this->surname = new char[10];

strcpy\_s(this->surname, sizeof "nosurname", "nosurname");

}

Cook::Cook(char\* name, char\* surname)

{

this->name = new char[strlen(name) + 1];

strcpy\_s(this->name, strlen(name) + 1, name);

this->surname = new char[strlen(surname) + 1];

strcpy\_s(this->surname, strlen(surname) + 1, surname);

}

Cook::Cook(Cook& cook)

{

this->name = new char[strlen(cook.name) + 1];

strcpy\_s(this->name, strlen(cook.name) + 1, cook.name);

this->surname = new char[strlen(cook.surname) + 1];

strcpy\_s(this->surname, strlen(cook.surname) + 1, cook.surname);

}

Cook::~Cook()

{

}

char\* Cook::get\_name()

{

return this->name;

}

char\* Cook::get\_surname()

{

return this->surname;

}

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

{

this->name = new char[strlen(name) + 1];

strcpy\_s(this->name, strlen(name) + 1, name);

return \*this;

}

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

{

this->surname = new char[strlen(surname) + 1];

strcpy\_s(this->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

{

private:

char\* name;

int price, time;

Category category;

Cook cook;

public:

Dish();

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

Dish(Dish&);

~Dish();

char\* get\_name();

int get\_price();

int get\_time();

Category& get\_category();

Cook& get\_cook();

Dish& set\_name(char\*);

Dish& set\_price(int);

Dish& set\_time(int);

Dish& set\_category(Category&);

Dish& set\_cook(Cook&);

void show();

};

**Dish.cpp**

#include <string>

#include <iostream>

#include "Dish.h"

using namespace std;

Dish::Dish()

{

this->name = new char[7];

strcpy\_s(this->name, sizeof "noname", "noname");

price = 0;

time = 0;

this->category = NONE;

}

Dish::Dish(char\* name, int price, int time, Category& category, Cook& cook)

{

this->name = new char[strlen(name) + 1];

strcpy\_s(this->name, strlen(name) + 1, name);

this->price = price;

this->time = time;

this->category = category;

this->cook = cook;

}

Dish::Dish(Dish& dish)

{

this->name = new char[strlen(dish.name) + 1];

strcpy\_s(this->name, strlen(dish.name) + 1, dish.name);

this->price = dish.price;

this->time = dish.time;

this->category = dish.category;

this->cook = dish.cook;

}

Dish::~Dish()

{

}

char\* Dish::get\_name()

{

return this->name;

}

int Dish::get\_price()

{

return this->price;

}

int Dish::get\_time()

{

return this->time;

}

Category& Dish::get\_category()

{

return this->category;

}

Cook& Dish::get\_cook()

{

return this->cook;

}

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

{

delete[] this->name;

this->name = new char[strlen(name) + 1];

strcpy\_s(this->name, strlen(name) + 1, name);

return \*this;

}

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

{

this->price = price;

return \*this;

}

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

{

this->time = time;

return \*this;

}

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

{

this->category = category;

return \*this;

}

Dish& Dish::set\_cook(Cook& cook)

{

this->cook = cook;

return \*this;

}

void Dish::show()

{

cout << name << ", " << 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();

}

**Order.h**

#pragma once

#include "Date.h"

#include "Dish.h"

class Order

{

private:

char\* name;

Date date;

int number;

Dish\* dishes;

public:

Order();

Order(char\*, Date, int, Dish\*);

Order(Order&);

~Order();

char\* get\_name();

Date& get\_date();

int get\_number();

Dish\* get\_dishes();

Order& set\_name(char\*);

Order& set\_date(Date&);

Order& set\_number(int);

Order& set\_dishes(Dish\*);

void add\_dish(Dish&);

void show();

void short\_show();

};

**Order.cpp**

#include <string>

#include <iostream>

#include "Order.h"

using namespace std;

Order::Order()

{

this->name = new char[7];

strcpy\_s(this->name, sizeof "noname", "noname");

this->number = 1;

this->dishes = new Dish[2];

}

Order::Order(char\* name, Date date, int number, Dish\* dishes)

{

this->name = new char[strlen(name) + 1];

strcpy\_s(this->name, strlen(name) + 1, name);

this->date = date;

this->number = number;

delete[] this->dishes;

this->dishes = new Dish[number + 1];

for (int i = 0; i < this->number; i++) {

this->dishes[i].set\_name(dishes[i].get\_name());

this->dishes[i].set\_price(dishes[i].get\_price());

this->dishes[i].set\_time(dishes[i].get\_time());

this->dishes[i].set\_category(dishes[i].get\_category());

this->dishes[i].set\_cook(dishes[i].get\_cook());

}

}

Order::Order(Order& order)

{

this->name = new char[strlen(order.name) + 1];

strcpy\_s(this->name, strlen(order.name) + 1, order.name);

this->date = order.date;

this->number = order.number;

this->dishes = new Dish[number + 1];

for (int i = 0; i < this->number; i++) {

this->dishes[i] = order.dishes[i];

}

}

Order::~Order()

{

}

char\* Order::get\_name()

{

return this->name;

}

Date& Order::get\_date()

{

return this->date;

}

int Order::get\_number()

{

return this->number;

}

Dish\* Order::get\_dishes()

{

return this->dishes;

}

Order& Order::set\_name(char\* name)

{

this->name = new char[strlen(name) + 1];

strcpy\_s(this->name, strlen(name) + 1, name);

return \*this;

}

Order& Order::set\_date(Date& date)

{

this->date = date;

return \*this;

}

Order& Order::set\_number(int number)

{

this->number = number;

return \*this;

}

Order& Order::set\_dishes(Dish\*)

{

for (int i = 0; i < this->number; i++) {

this->dishes[i] = dishes[i];

}

return \*this;

}

void Order::add\_dish(Dish& dish)

{

dishes[this->number] = dish;

this->number++;

}

void Order::show()

{

cout << name << ", ";

this->date.show();

cout << ", " << number << ", " << endl << "Страви:" << endl;

for (int i = 0; i < this->number; i++) {

dishes[i].show();

}

}

void Order::short\_show()

{

cout << name << ", ";

this->date.show();

cout << ", ";

int temp = 0;

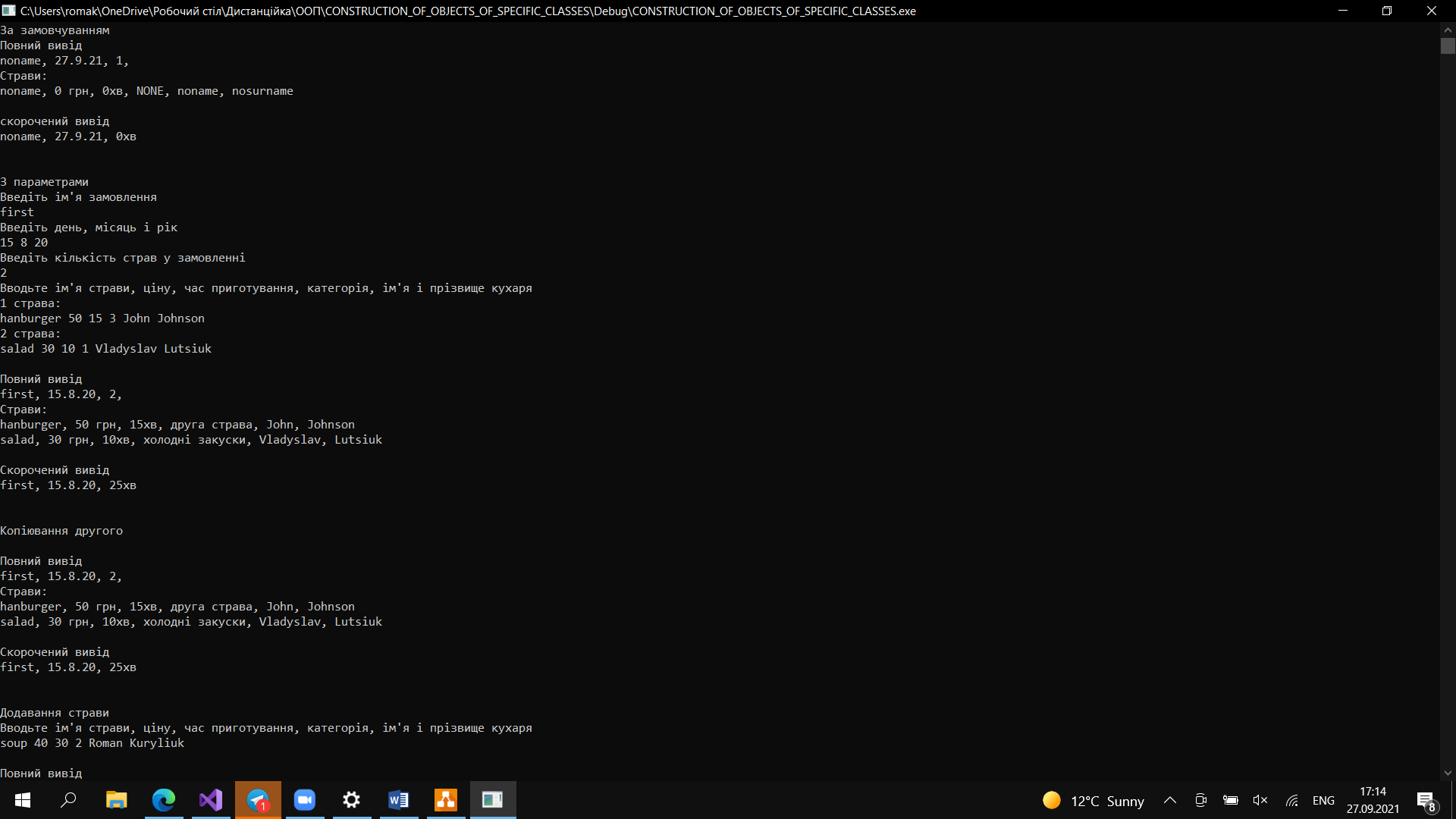
for (int i = 0; i < this->number; i++) {

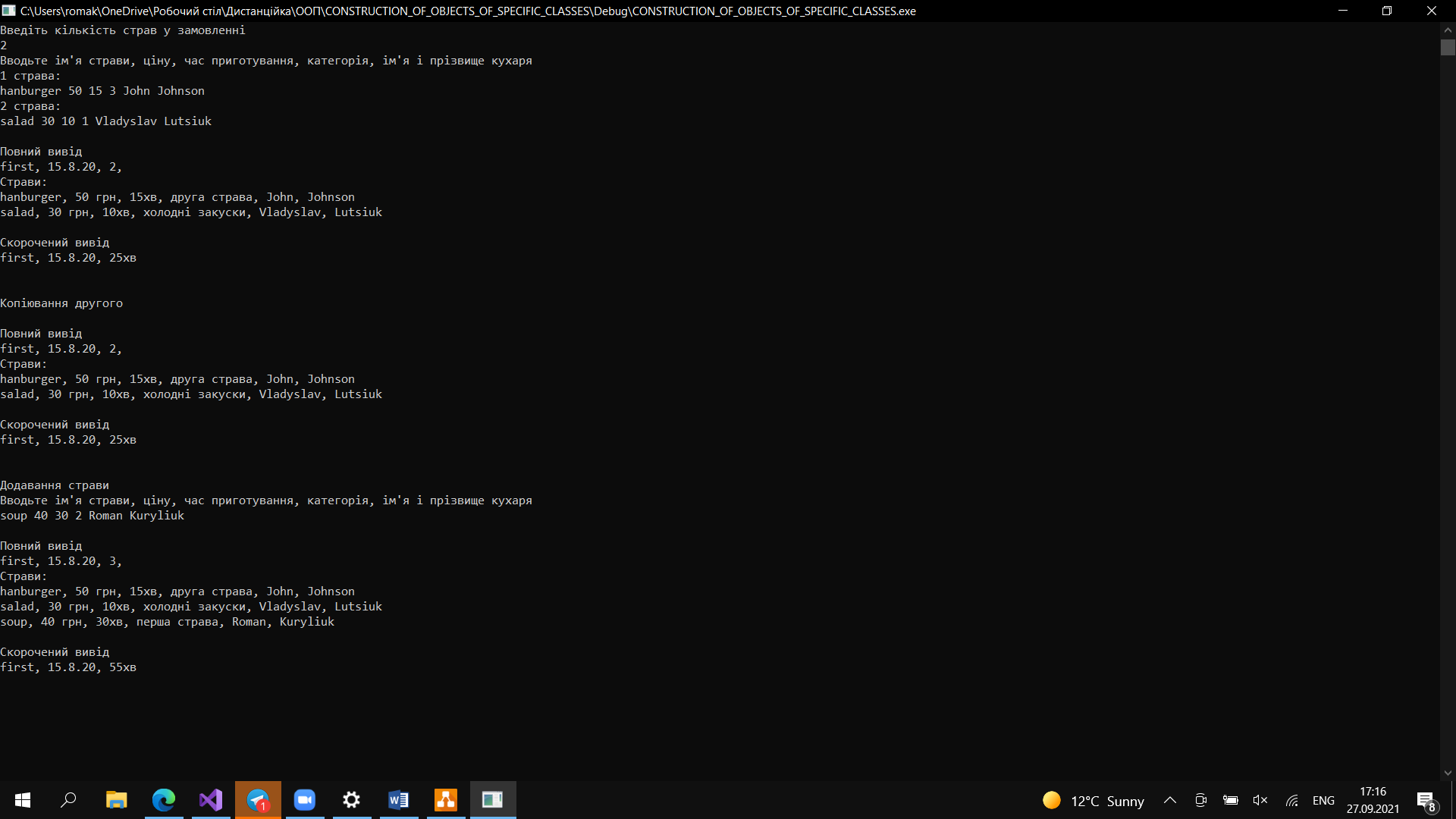
temp += this->dishes[i].get\_time();

}

cout << temp << "хв" << endl;

}

**Результати роботи програми**



**Висновок**

Під час виконання комп'ютерного практикуму я навчився описувати абстракції предметної області з використанням принципу інкапсуляції мовою С++.

Було створено тип “Категорія” як перерахування enum, типи “Дата”, “Кухар” та “Страва”, “Замовлення”, що перебувають в стані інкапсуляції. Наданий тестовий приклад відповідно до загальних вимог та з виведенням повної та скороченої інформації про об'єкти класу “Замовлення”.