МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 3

з навчальної дисципліни

# “ ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ ”

Успадкування, агрегація та композиція

ВИКОНАВ

студент академічної групи КБ-23

Сергій Козирь

ПЕРЕВІРИВ

Викладач

Козірова Н.Л.

Кропивницький – 2024

Лабораторна робота №3

Успадкування, агрегація та композиція

Мета: ознайомитись з основними поняттями успадкування, агрегація та композиція в ООП та навчитись їх програмно реалізовувати мовою С++.

Завдання 1

Варіант 1

1. Створіть клас "Людина" (базовий клас), який містить такі властивості:

• Ім'я

• Вік

• Клас "Адреса", який представляє адресу проживання людини (використовуйте композицію)

1. Створіть клас "Студент" (похідний клас від "Людина"), який містить додаткові властивості:

• Номер студентського квитка

• Курс

3. Створіть клас "Університет" (базовий клас), який містить такі властивості:

• Назва університету

• Список студентів (використовуйте агрегацію)

4. РеалізуйтеАгрегація є одним з важливих концепцій в ООП і відноситься до відношення між класами, де один клас представляє контейнер, що містить об'єкти іншого класу. У випадку агрегації, об'єкти-члени, які належать до контейнера, можуть існувати самостійно і не залежати від контейнера.м методи для додавання та видалення студентів зі списку університету.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Університет", "Людина" та "Студент". Додайте студентів до списку університету та виведіть інформацію про них.

6. Продемонструйте успадкування, викликавши методи базового класу та похідного класу для об'єктів типу "Студент".

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням. Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Реалізація звдання

#include <iostream>

#include <vector>

#include <string>

using namespace std;

// Клас "Адреса", який представляє адресу проживання людини

class Address {

public:

string city;

string street;

int houseNumber;

Address(string city, string street, int houseNumber)

: city(city), street(street), houseNumber(houseNumber) {}

// Метод для виведення адреси

string getAddress() const {

return street + ", " + to\_string(houseNumber) + ", " + city;

}

};

// Базовий клас "Людина"

class Person {

public:

string name;

int age;

Address address; // Композиція з класом "Адреса"

Person(string name, int age, Address address)

: name(name), age(age), address(address) {}

// Метод для виведення інформації про людину

virtual void displayInfo() const {

cout << "Ім'я: " << name << ", Вік: " << age << ", Адреса: " << address.getAddress() << endl;

}

};

// Похідний клас "Студент" від класу "Людина"

class Student : public Person {

public:

string studentID;

int course;

Student(string name, int age, Address address, string studentID, int course)

: Person(name, age, address), studentID(studentID), course(course) {}

// Перевизначення методу для виведення інформації про студента

void displayInfo() const override {

Person::displayInfo(); // Виклик методу базового класу

cout << "Студентський квиток: " << studentID << ", Курс: " << course << endl;

}

};

// Клас "Університет", що містить список студентів (агрегація)

class University {

private:

string universityName;

vector<Student> students; // Список студентів

public:

University(string universityName)

: universityName(universityName) {}

// Метод для додавання студента

void addStudent(const Student& student) {

students.push\_back(student);

cout << "Студента " << student.name << " додано до університету " << universityName << "." << endl;

}

// Метод для видалення студента за номером студентського квитка

void removeStudent(const string& studentID) {

for (auto it = students.begin(); it != students.end(); ++it) {

if (it->studentID == studentID) {

cout << "Студента " << it->name << " видалено з університету " << universityName << "." << endl;

students.erase(it);

return;

}

}

cout << "Студента з номером квитка " << studentID << " не знайдено." << endl;

}

// Метод для виведення списку студентів

void listStudents() const {

if (students.empty()) {

cout << "Немає студентів у списку." << endl;

}

else {

cout << "Список студентів університету " << universityName << ":" << endl;

for (const auto& student : students) {

student.displayInfo();

}

}

}

};

// Демонстраційна функція main()

int main() {

setlocale(LC\_ALL, "ukr");

// Створення об'єктів класу "Адреса"

Address address1("Київ", "Вулиця Шевченка", 12);

Address address2("Львів", "Проспект Свободи", 45);

// Створення об'єктів класу "Студент"

Student student1("Іван Іваненко", 20, address1, "STU12345", 2);

Student student2("Марія Петрівна", 21, address2, "STU67890", 3);

// Створення об'єкта класу "Університет"

University university("Центрально Національний Тезнічний Університет");

// Додавання студентів до університету

university.addStudent(student1);

university.addStudent(student2);

// Виведення списку студентів

university.listStudents();

// Виклик методів базового та похідного класу

cout << "\nІнформація про студента (базовий клас та похідний клас):" << endl;

student1.displayInfo();

student2.displayInfo();

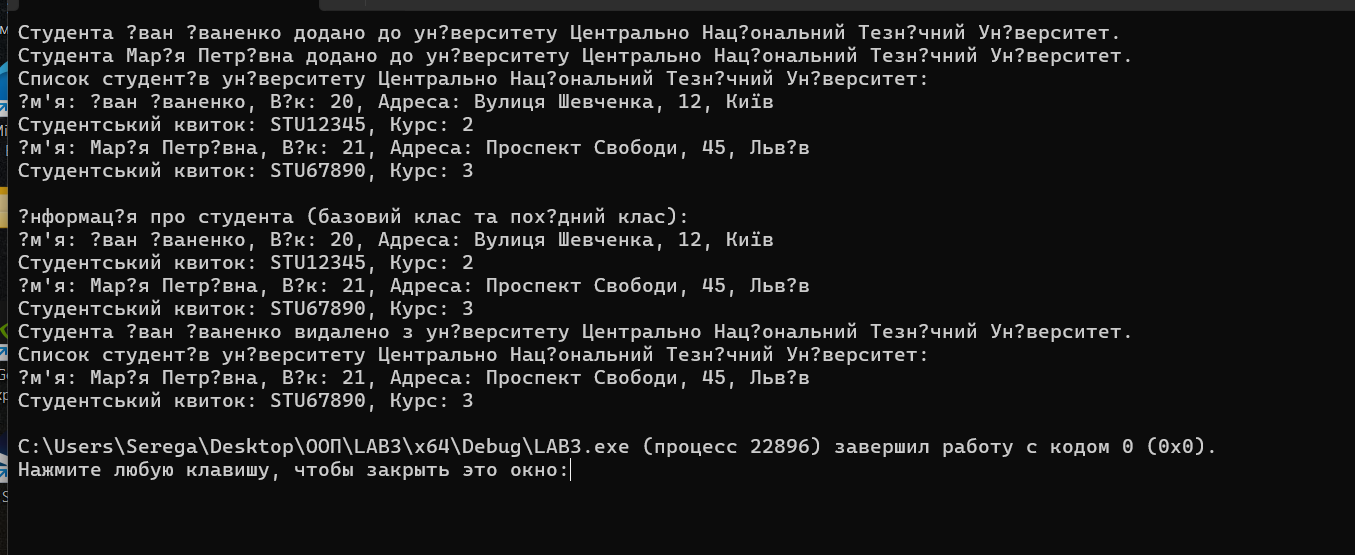
// Видалення студента

university.removeStudent("STU12345");

// Виведення списку студентів після видалення

university.listStudents();

return 0;



**Завдання 2** Створіть використовуючи композицію два класи, які матимуть свої властивості, перший реалізуйте за допомогою включення (composition), а другий за допомогою посилання (reference), контейнером для цих класів використайте клас з лабораторної роботи №1, створіть також третій – похідний клас, використовуючи наслідування, базовим класом може бути або клас з лр№1, або класи які ви створили за допомогою композиції. За бажанням можете зобразити схематично як ваші класи залежать один від одного

Реалізація

main.cpp

#include <iostream>

#include "Book.h"

#include "Library.h"

#include "DerivedLibrary.h"

int main() {

// Створення книги

Book book1("1984", "George Orwell", 1949);

Book book2("To Kill a Mockingbird", "Harper Lee", 1960);

// Створення бібліотеки з секцією та полицею

Library library(book1, &book2);

// Виведення інформації про бібліотеку

library.displayLibraryInfo();

// Створення розширеної бібліотеки

AdvancedLibrary advancedLibrary("Central Library", book1, &book2);

// Виведення інформації про розширену бібліотеку

advancedLibrary.displayAdvancedLibraryInfo();

return 0;

}

Book.cpp

// Book.cpp

#include "Book.h"

// Конструктор за замовчуванням

Book::Book() : title(""), author(""), year(0) {}

// Конструктор з параметрами

Book::Book(std::string t, std::string a, int y) : title(t), author(a), year(y) {}

// Методи доступу до полів

void Book::setTitle(const std::string& t) {

title = t;

}

std::string Book::getTitle() const {

return title;

}

void Book::setAuthor(const std::string& a) {

author = a;

}

std::string Book::getAuthor() const {

return author;

}

void Book::setYear(int y) {

year = y;

}

int Book::getYear() const {

return year;

}

Book.h

#ifndef BOOK\_H

#define BOOK\_H

#include <string>

class Book {

private:

std::string title;

std::string author;

int year;

public:

// Конструктори

Book(); // Конструктор за замовчуванням

Book(std::string title, std::string author, int year); // Конструктор з параметрами

// Методи доступу

void setTitle(const std::string& title);

std::string getTitle() const;

void setAuthor(const std::string& author);

std::string getAuthor() const;

void setYear(int year);

int getYear() const;

};

#endif

DerivedLibrary.h

#define DERIVEDLIBRARY\_H

#include "Library.h"

class AdvancedLibrary : public Library {

private:

std::string libraryName;

public:

AdvancedLibrary(const std::string& name, const Book& sectionBook, Book\* shelfBook)

: Library(sectionBook, shelfBook), libraryName(name) {}

void displayAdvancedLibraryInfo() const {

std::cout << "Library Name: " << libraryName << std::endl;

displayLibraryInfo();

}

};

#endif

Library.h

#ifndef LIBRARY\_H

#define LIBRARY\_H

#include "Book.h"

// Клас, що використовує композицію (composition)

class LibrarySection {

private:

Book book; // Включає об'єкт класу Book

public:

LibrarySection(const Book& book) : book(book) {}

void displaySectionInfo() const {

std::cout << "Book in this section: " << book.getTitle() << " by " << book.getAuthor() << std::endl;

}

};

// Клас, що використовує посилання (reference)

class LibraryShelf {

private:

Book\* book; // Вказівник на об'єкт класу Book

public:

LibraryShelf(Book\* book) : book(book) {}

void displayShelfInfo() const {

if (book) {

std::cout << "Book on this shelf: " << book->getTitle() << " by " << book->getAuthor() << std::endl;

}

}

};

// Контейнерний клас Library, що містить класи LibrarySection і LibraryShelf

class Library {

private:

LibrarySection section;

LibraryShelf shelf;

public:

Library(const Book& sectionBook, Book\* shelfBook)

: section(sectionBook), shelf(shelfBook) {}

void displayLibraryInfo() const {

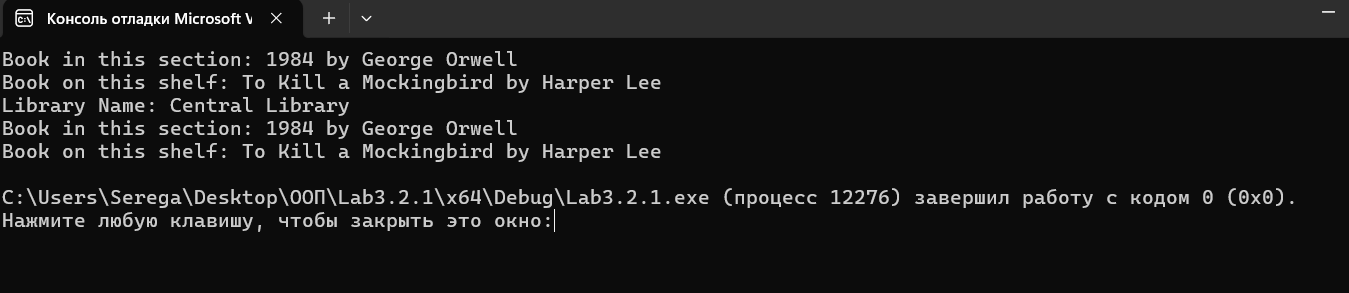
section.displaySectionInfo();

shelf.displayShelfInfo();

}

};

#endif

****

**Агрегація** є одним з важливих концепцій в ООП і відноситься до відношення між класами, де один клас представляє контейнер, що містить об'єкти іншого класу. У випадку агрегації, об'єкти-члени, які належать до контейнера, можуть існувати самостійно і не залежати від контейнера. Зазвичай агрегація відбувається за допомогою включення (composition) або посилання (reference).

**Композиція** - це тип відношення між класами, коли один клас (відомий як контейнер або композит) містить об'єкти іншого класу (відомі як компоненти). У випадку композиції, компоненти не можуть існувати без контейнера, тобто вони залежать від його існування і життєвого циклу.

Основна відмінність між агрегацією і композицією полягає в тому, що в композиції компоненти є прямими членами контейнера і створюються та знищуються разом з ним. Це означає, що коли контейнер створюється або знищується, його компоненти також автоматично створюються або знищуються. Крім того, контейнер відповідає за створення та управління життєвим циклом своїх компонентів.

**Успадкування** (іноді його називають також наслідуванням) в об'єктноорієнтованому програмуванні є одним із фундаментальних концепцій. Воно дозволяє створювати нові класи на основі вже існуючих (батьківських або базових класів), утворюючи ієрархію класів.

Один з основних принципів успадкування - це утворення спеціалізованих класів на базі загального базового класу.

Однією з головних переваг успадкування є полегшення повторного використання коду. Базовий клас може містити загальні функції і властивості, які можуть бути успадковані і використовані у всіх похідних класах. Це забезпечує ефективніше управління кодом і зменшення дублювання.

Успадкування є важливим інструментом в об'єктно-орієнтованому програмуванні, оскільки дозволяє створювати ієрархії класів з поліморфними можливостями та забезпечує повторне використання коду. Воно сприяє модульності, розширюваності і зрозумілості програмного коду.