Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

Кафедра кібербезпеки та програмного забезпечення

Дисципліна: ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

**Лабораторна робота №1**

**Тема: «ОСНОВНІ ПОНЯТТЯ ООП. КЛАСИ ТА ОБ’ЄКТИ. ФУНКЦІЇ ДОСТУПУ. ВКАЗІВНИК THIS.»**

|  |
| --- |
| Виконав: ст. гр. КI-23 |
| Пєніна А.О. |
| Перевірив викладач:  Козірова Н. Л. |
|  |

Кропивницький 2024

**Тема:** ОСНОВНІ ПОНЯТТЯ ООП. КЛАСИ ТА ОБ’ЄКТИ. ФУНКЦІЇ ДОСТУПУ. ВКАЗІВНИК THIS.

**Мета:**ознайомитись з основними поняттями ООП. Вивчити поняття клас, об’єкт, сеттер, геттер та навчитись їх програмно реалізовувати мовою С++.

**Варіант 5**

**Завдання:**

*Завдання 1*

1. Створіть клас "Student" (студент), який має наступні властивості та функціональність:

Приватні поля класу:

• name (ім'я студента)

• age (вік студента)

• major (спеціальність студента)

Публічні методи класу:

• Метод setName(), який дозволяє задати ім'я студента.

• Метод getName(), який повертає ім'я студента.

• Метод setAge(), який дозволяє задати вік студента.

• Метод getAge(), який повертає вік студента.

• Метод setMajor(), який дозволяє задати спеціальність студента.

• Метод getMajor(), який повертає спеціальність студента.

2. Створіть об'єкт класу "Student".

3. Задайте значення полів об'єкта за допомогою відповідних методів.

4. Виведіть інформацію про студента на екран, використовуючи методи для отримання значень полів.

5. Реалізувати програму за допомогою роздільної компіляції. У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

*Завдання 2*

Реалізувати вище наведену задачу за допомогою структурного програмування. У висновку описати різницю цих методів.

**Лістинг student.h:**

#ifndef STUDENT\_H

#define STUDENT\_H

#include <string>

using namespace std;

class Student

{

private:

string name;

int age;

string major;

bool tester (string test);

bool testers(int tests);

public:

Student();

bool setName(string newName);

string getName() { return name; }

bool setAge(int newAge);

int getAge() { return age; }

bool setMajor(string newMajor);

string getMajor() { return major; }

};

#endif // STUDENT\_H

**Лістинг strukture.h:**

#ifndef STRUKTURE\_H

#define STRUKTURE\_H

#include <string>

using namespace std;

struct StudentS {

string name;

int age;

string major;

};

bool setName(StudentS& student, const string& newName);

string getName(const StudentS& student);

bool setAge(StudentS& student, int newAge);

int getAge(const StudentS& student);

bool setMajor(StudentS& student, const string& newMajor);

string getMajor(const StudentS& student);

bool tester (const string& test);

bool testers(int tests);

#endif // STRUKTURE\_H

**Лістинг student.cpp:**

#include "student.h"

#include <iostream>

Student::Student() {}

bool Student::setName(string newName)

{

if (tester(newName))

{

this->name = newName;

return true;

}

return false;

}

bool Student::setMajor(string newMajor)

{

if (tester(newMajor))

{

this->major = newMajor;

return true;

}

return false;

}

bool Student::setAge(int newAge)

{

if (testers(newAge))

{

this->age = newAge;

return true;

}

return false;

}

bool Student::tester(string test)

{

if (test.length() > 0)

{

return true;

}

return false;

}

bool Student::testers(int tests)

{

if (tests > 14)

{

return true;

}

else

{

std::cout << "Вік замалий для студента" << std::endl;

}

return false;

}

**Лістинг strukture.cpp:**

#include "strukture.h"

bool setName(StudentS& student, const string& newName) {

if (tester(newName)) {

student.name = newName;

return true;

}

return false;

}

string getName(const StudentS& student) {

return student.name;

}

bool setAge(StudentS& student, int newAge) {

if (testers(newAge)) {

student.age = newAge;

return true;

}

return false;

}

int getAge(const StudentS& student) {

return student.age;

}

bool setMajor(StudentS& student, const string& newMajor) {

if (tester(newMajor)) {

student.major = newMajor;

return true;

}

return false;

}

string getMajor(const StudentS& student) {

return student.major;

}

bool tester(const string& test) {

return !test.empty();

}

bool testers(int tests) {

return tests > 0;

}

**Лістинг main.cpp:**

#include <QCoreApplication>

#include "student.h"

#include "strukture.h"

#include <iostream>

int main(int argc, char \*argv[])

{

QCoreApplication a(argc, argv);

char order = '\0';

int InputNum;

string Input;

string Innput;

Student \*student = new Student;

StudentS IStudent;

cout << "OOP/Strukt (O/S)?" << endl;

cin >> order;

cin.ignore();

switch (order) {

case 'o':

case 'O':

cout << "Enter name: " << endl;

getline(cin, Input);

if (student->setName(Input))

{

cout << "Name: " << student->getName() << endl;

}

else

{

cout << "Eror 1" << endl;

}

cout << "Enter age " << endl;

cin >> InputNum;

if (student->setAge(InputNum))

{

cout << "Age: " << student->getAge() << endl;

}

else

{

cout << "Eror 2" << endl;

}

cout << "Enter major: " << endl;

cin >> Innput;

if (student->setMajor(Innput))

{

cout << "Major: " << student->getMajor() << endl;

}

else

{

cout << "Eror 1" << endl;

}

return 0;

case 's':

case 'S':

cout << "Введіть імя: " << endl;

getline(cin, Input);

if (setName(IStudent, Input)) {

cout << "Імя: " << getName(IStudent) << endl;

}

else {

cout << "Некоректні дані" << endl;

}

cout << "Зазначте вік студента: " << endl;

cin >> InputNum;

if (setAge(IStudent, InputNum)) {

cout << "Вік: " << getAge(IStudent) << endl;

}

else {

cout << "Вік замалий для студента" << endl;

}

cout << "Введіть спеціальність: " << endl;

getline(cin, Innput);

if (setName(IStudent, Innput)) {

cout << "Спеціальність: " << getName(IStudent) << endl;

}

else {

cout << "Некоректні дані" << endl;

}

return 0;

default:

cout << "Некоректний запит" << endl;

delete student;

return 0;

}

delete student;

return a.exec();

}

**Висновок:**

При використанні парадигми ООП дані, приховано від сторонього чи прямого доступу і зміненюються лише через спеціальні методи. У структурному підході дані завжди доступні напряму.

Використання класу дозволяє легко додавати нові функціональні можливості (наприклад, додавання нових методів або зміна існуючих), тоді як у структурному підході це зробити складніше.

Також код написаний з використанням класів, часто легше читати і підтримувати, особливо у великих проектах. Структурний підхід може бути простішим для невеликих програм, але складнішим при спробі покращити чи змінити проект.Тому вибір між підходами залежить від задачі.