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

Чабан С.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№9**

Завдання 1

1. Створіть клас "Country" (країна), який має наступні властивості та

функціональність:

Приватні поля класу:

• name (назва країни)

• capital (столиця країни)

• population (населення країни)

Публічні методи класу:

• Метод setName(), який дозволяє задати назву країни.

• Метод getName(), який повертає назву країни.

• Метод setCapital(), який дозволяє задати столицю країни.

• Метод getCapital(), який повертає столицю країни.

• Метод setPopulation(), який дозволяє задати населення країни.

• Метод getPopulation(), який повертає населення країни.

2. Створіть об'єкт класу "Country".

3. Використайте метод setName() для задання назви країни.

4. Використайте метод getName() для отримання назви країни та виведіть

її на екран.

5. Використайте метод setCapital() для задання столиці країни.

6. Використайте метод getCapital() для отримання столиці країни та

виведіть її на екран.

7. Використайте метод setPopulation() для задання населення країни.

8. Використайте метод getPopulation() для отримання населення країни та

виведіть його на екран.

9. 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 до завдання 1:**

*country.h:*

#ifndef COUNTRY\_H

#define COUNTRY\_H

#include <string>

using namespace std;

class Country {

public:

Country();

void setName(string newName);

string getName();

void setCapital(string newCapital);

string getCapital();

void setPopulation(int newPopulation);

int getPopulation();

private:

string name;

string capital;

int population;

};

#endif // COUNTRY\_H

*country.cpp:*

#include "country.h"

Country::Country() {}

void Country::setName(string newName)

{

name = newName;

}

string Country::getName()

{

return name;

}

void Country::setCapital(string newCapital)

{

capital = newCapital;

}

string Country::getCapital()

{

return capital;

}

void Country::setPopulation(int newPopulation)

{

population = newPopulation;

}

int Country::getPopulation()

{

return population;

}

main.cpp:

#include <iostream>

#include "country.h"

using namespace std;

int main()

{

Country Region;

Region.setName("Sweden");

Region.setCapital("Stockolm");

Region.setPopulation(987661);

cout << "Name Region: " << Region.getName() << endl;

cout << "Capital Region: " << Region.getCapital() << endl;

cout << "Population Region: " << Region.getPopulation() << endl;

return 0;

}

**Лістинг до завдання 2:**

*country.h:*

#ifndef COUNTRY\_H

#define COUNTRY\_H

#include <string>

using namespace std;

struct Country {

string name = "\0";

string capital = "\0";

int population = -1;

};

void setName(Country &country, const string &newName);

string getName(const Country &country);

void setCapital(Country &country, const string &newCountry);

string getCapital(const Country &country);

void setPopulation(Country &country, int newPopulation);

int getPopulation(const Country &country);

#endif // COUNTRY\_H

*country.cpp:*

#include "country.h"

void setName(Country &country, const string &newName)

{

country.name = newName;

}

string getName(const Country &country)

{

return country.name;

}

void setCapital(Country &country, const string &newCapital)

{

country.capital = newCapital;

}

string getCapital(const Country &country)

{

return country.capital;

}

void setPopulation(Country &country, int newPopulation)

{

country.population = newPopulation;

}

int getPopulation(const Country &country)

{

return country.population;

}

*country.cpp:*

#include <iostream>

#include "country.h"

using namespace std;

int main()

{

Country Region;

setName(Region, "Sweden");

setCapital(Region, "Stockolm");

setPopulation(Region, 987661);

cout << "Name Region: " << getName(Region) << endl;

cout << "Capital Region: " << getCapital(Region) << endl;

cout << "Population Region: " << getPopulation(Region) << endl;

return 0;

}

**Висновок:**

Об’єктно-орієнтований підхід дозволяє створювати класи, які поєднують дані (поля) і функціональні можливості (методи), які обробляють ці дані в межах сутності. Це допомагає краще організувати код, полегшуючи його підтримку та розширення. Важливою перевагою ООП є інкапсуляція, яка дозволяє приховати деталі внутрішньої реалізації класу та надати зрозумілий інтерфейс користувачам класу. Наприклад, у класі Країна всі дані країни зберігаються в приватних полях і доступ до них можливий лише через загальнодоступні методи. Це забезпечує контроль над тим, як ці дані змінюються та витягуються.

Структурований підхід фокусується на використанні функцій і структур для організації коду. Тут дані та функціональність існують окремо, що може ускладнити керування великими проектами. У прикладі структурованого програмування всі функції для обробки країн незалежні, і немає гарантії, що вони будуть використані правильно (наприклад, ви випадково не зміните напрямок структури, не використовуючи відповідну функцію). Крім того, структурні підходи є менш масштабованими та гнучкими, особливо при роботі зі складними системами.

Об'єктно-орієнтоване програмування надає більше можливостей для створення добре організованого, безпечного та масштабованого коду, що особливо важливо для великих проектів. Структуроване програмування є простішим і може використовуватися для невеликих програм або коли потрібно мінімізувати витрати на створення об’єктів. ООП дозволяє краще керувати складністю програм і полегшити їх розуміння та підтримку.