Лабораторне заняття 7 (2-й семестр)

Глобальні змінні в ООП: статичні члени класу, шаблон проєктування “Одинак”

Мета роботи: вивчення способів глобального доступу до полів класів в C++.

Завдання на роботу: опис класу з глобальними та статичними членами, представлення його в UML-нотації, програмна реалізація класу, реалізація шаблону “Одинак”.

Теоретичні відомості до виконання.

Статичні та глобальні змінні.

Глобальні змінні — змінні, які було створено поза межами будь-якого блоку коду (*глобальна* або *файлова* область видимості), вони є доступними з будь-якого місця програми і зберігаються в пам’яті до завершення роботи програми. Зазвичай, глобальні змінні об’являють після блоку **#include**, але вище будь-якого іншого коду, наприклад:

```
#include <iostream>

int g_x; // Глобальна змінна g_x
const int g_y(3); // Глобальна змінна g_y

void doSomething()
{
	//  Глобальні змінні можуть бути використані в будь-якому місці програми
	g_x = 4;
	std::cout << g_y << "\n";
}

int main()
{
	doSomething();

	// Глобальні змінні можуть бути використані в будь-якому місці програми
	g_x = 7;
	std::cout << g_y << "\n";
	return 0;
}
```

На відміну від глобальних, локальні змінні об’являються всередині блоку, обмеженого фігурними дужками і є видимі тільки всередині цього блоку. Локальна змінна в деякому блоці коду завжди перекриває глобальну, яка має таке ж ім’я. Для того, щоб примусово вказати, що в даному місті блоку має бути використана глобальна змінна використовується оператор вирішення контексту “::”:

```
#include <iostream>

int value(4); //  Глобальна змінна

int main()
{
```

```
        int value = 8; // Локальна змінна перекриває значення глобальної
        // В цій точці коду value=8
        value++; // збільшується локальна змінна
        ::value--; // зменшується глобальна змінна завдяки оператору ::
        std::cout << "Global value: " << ::value << "\n";
        std::cout << "Local value: " << value << "\n";
        return 0;
}
// Локальна змінна знищується, значення глобальної залишається
// value=3
```

Забезпечення глобального доступу є і перевагою і недоліком глобальних змінних, так як їх використання може значно зменшити об'єм коду програми, але, в той же час, будь-які функції можуть змінювати значення глобальних змінних, тому у складних об'ємних проєктах слід уникати використання таких змінних, тому що це може призвести до непрогнозованих змін значень таких змінних.

Компромісним рішенням для забезпечення глобального доступу до змінних є використання *статичних змінних*, які також доступні глобально, але кожна така змінна наявна лише в одному екземплярі в пам'яті. Статичні змінні схожі до глобальних за механізмом розміщення в пам'яті, але на статичні додатково діють специфікатори доступу **public** та **private**.

Статичні поля найчастіше використовуються в таких ситуаціях:

- необхідність контролю загальної кількості об'єктів класу;
- створення єдиної глобальної змінної, до якої мають доступ усі об'єкти класу.

Статичні поля задаються ключовим словом **static**, яке може бути використано як для атрибутів, так і для методів класу. Особливістю елементів, до яких додане ключове слово **static** є те, що вони належать класу, а не об'єкту цього класу, тому можуть бути використані навіть без створення об'єкту класу, і незалежно від кількості створених об'єктів даного класу, в пам'яті буде знаходитись лише одна копія елементу, що об'явлено як статичний.

Таким чином, в пам'яті буде знаходитись завжди лише по одній копії кожної статичної змінної, а кількість копій нестатичних полів буде дорівнювати загальній кількості об'єктів класу.

Ключове слово **static** вказується перед вказанням типу даних або методів:

static тип_змінної ім'яАтрибуту;

static тип_значення_що_повертається ім'яМетоду(...);

Доступ до статичних змінних або функцій відбувається з використанням імені класу та оператору вирішення контексту "::":

Ім'яКласу :: *ім'яАтрибуту*;

Ім'яКласу :: *ім'яМетоду(...)*;

Статичні атрибути класу об'являються в об'явленні класу (або в заголовковому файлі, якщо він є), а ініціалізуються поза блоком-об'явленням в глобальній області видимості. ***Не можна*** ініціалізувати статичні змінні в тілі класу та в його методах. Такий спосіб ініціалізації потрібний для того, щоб уникнути повторної ініціалізації статичної змінної.

Розглянемо приклад використання статичної змінної для підрахунку загальної кількості створених об’єктів класу (файл «main.cpp»):

```
#include <iostream>
using namespace std;
class X // Об’явлення класу
{
	static int n; //Змінна-лічильник створених екземплярів
	static char ClassName[30];
public:
	static int getN() { return n; }
	static char* getClass() { return ClassName; }
	X() { n++; } // конструктор
};
// Використання класу
int X::n = 0; // Ініціалізація приватного атрибуту поза тілом класу через оператор
                                                        // вирішення контексту
char X::ClassName[] = "My Class";

int main()
{
	X a, b, c; // Об’являємо 3 об’єкти класу X
	cout << X::getN() << " objects of Class \"" << X::getClass() << "\""  << endl;
                                                        // Звертання до методів
                                                        // класу також через
                                                        // оператор “::”
	//cout << X::n <<  endl; помилка, спроба доступу до приватного члену класу
	return 0;
}
```

Шаблон “Одинак”. Інший приклад управління створенням об’єктів класу з використанням статичних змінних та методів — так званий шаблон проєктування “Одинак” (Singleton), при використанні якого можливо створити тільки один об’єкт такого класу:

```
// Файл Singleton.h
class Singleton
{
private: // Єдиний екземпляр класу та базові конструктори об’являються в приватній
                                                        // секції класу
	static Singleton* instance;
	Singleton() {}
	Singleton(const Singleton&);
	Singleton& operator=(const Singleton&);
public:
	static Singleton* getInstance()
	{
		if(!instance)
			instance = new Singleton();
		return instance;
	}
};
```

```
// Файл Singleton.cpp
#include “Singleton.h”
Singleton* Singleton::instance = 0; // Ініціалізація об’єкту

// Файл main.cpp
#include “Singleton.h”
int main()
{
        Singleton* s = Singleton::getInstange(); // Отримуємо покажчик на єдиний
                                                 // екземпляр класу Singleton
        return 0;
}
```

Лабораторне заняття 7.

Хід виконання завдання:

1. Для класу з лабораторного заняття 3 визначити поле, яке має бути глобальним, також додати статичне поле для підрахунку кількості екземплярів вказаного класу. Обов’язковим полем (не глобальним і не статичним) має бути атрибут “ім’яОб’єкту”.
2. Реалізувати клас в вигляді програмного коду з UML-діаграмою з урахуванням п.1.
3. Реалізувати клас **Logger** на основі шаблону “Одинак”:

Logger
- instance: Logger* + log: string[]
- необхідні конструктори та оператори + getInstance(): Logger* + addRecord(obj: покажчик на об’єкт Класу з п.1) + saveLog()

- Задача даного класу — вносити поточні значення публічних полів об’єктів класу з п.1 до поля **log** з використанням методу addRecord(), в вигляді строки виду:
  “object1Name: name
  time: YY.MM.DD HH:MM:SS
  object1Field1: field1Value
  object1Field1: field2Value”
- метод **saveLog()** викликатиметься в кінці роботи програми і зберігатиме текстовий файл з назвою “log.txt” (дата та час створення) такого виду:
  “ClassName: numberOfEntities
  object1Name: name
  time: YY.MM.DD HH:MM:SS
  object1Field1: field1Value
  object1Field2: field2Value
  ...
  object2Name: name
  time: YY.MM.DD HH:MM:SS
  object2Field1: field1Value
  object2Field2: field2Value
  ...“

4. Написати програму, в якій користувач матиме можливість проводити маніпуляції з визначеним класом і логувати їх з використанням класу **Logger**.

Варіанти завдань.

1. Вектор в просторі.
2. Прямокутник.
3. Раціональне число.
4. Дата.
5. Конус.
6. Пряма.
7. Мішане число.
8. Відрізок.
9. Комплексне число.
10. Ламана.
11. Вектор на площині.
12. Трапеція.
13. Матриця.
14. Трикутник.
15. Час.
16. Циліндр.
17. Багатокутник.
18. Сфера.