Лабораторне заняття 7 (2-й семестр)

Глобальні змінні в ООП: статичні члени класу. Шаблон проектування “Одинак”.

Статичні та глобальні змінні.

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

```
#include <iostream>
using namespace std;
int g_x; // Глобальна змінна g_x
const int g_y(3); // Глобальна константа g_y

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

int main()
{
	doSomething();

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

На відміну від глобальних, локальні змінні оголошуються всередині блока, обмеженого фігурними дужками, та є видимі тільки всередині цього блока. *Локальна змінна* в деякому блоці коду завжди перекриває глобальну, яка має таке саме ім’я. Для того щоб примусово вказати, що в цьому місці блока має бути використана глобальна змінна, використовується оператор вирішення

контексту “::”:

```
#include <iostream>
using namespace std;

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 ::
     cout << "Global value: " << ::value << "\n";
     cout << "Local value: " << value << "\n";
     return 0;
}
// Локальна змінна знищується, значення глобальної залишається  value=3
```

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’єкта класу, і незалежно від кількості створених об’єктів цього класу, в пам’яті буде знаходитись лише одна копія елемента, який оголошено як статичний.

Таким чином, у пам’яті буде знаходитись завжди лише по одній копії кожної статичної змінної, а кількість копій нестатичних полів буде дорівнювати за-

гальній кількості об’єктів класу.

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

static тип_змінної імʼяАтрибута;

static тип_значення_що_повертається імʼяМетоду(...);

Доступ до статичних змінних або функцій відбувається з використанням імені класу та оператора розширення області видимості «::»:

ІмʼяКласу :: *імʼяАтрибута*;

ІмʼяКласу :: *імʼяМетоду(...)*;

Статичні атрибути класу оголошуються в оголошені класу (або в заголовковому файлі, якщо він є), а ініціалізуються поза блоком-оголошенням в глобальній ділянці 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>
#include <string>
using namespace std;
class X // оголошення класу.
{
     static int n; //змінна-лічильник створених екземплярів.
     static string ClassName;
public:
     static int getN() { return n; }
     static string getClass() { return ClassName; }
     X() { n++; } // конструктор.
};
int X::n = 0; // ініціалізація приватного атрибута поза тілом класу через оператор
// розширення видимості.
string X::ClassName = "My Class";

int main()
{
```

```
    X a, b, c; // створюємо 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; призведе до помилки, спробі доступу до приватного члену
класу.
    return 0;
}
```

Приклад виконання програми наведено на рис. 1.

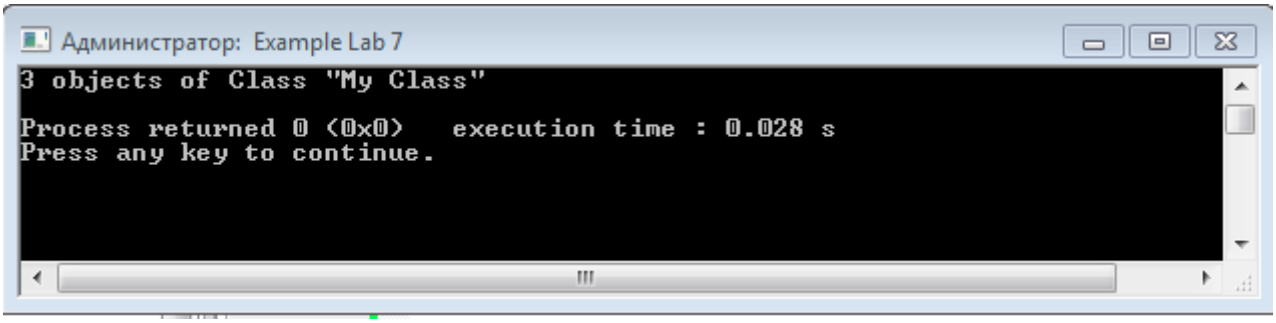

Рис.1.

Шаблон «*Одинак*». Інший приклад управління створенням об'єктів класу з використанням статичних змінних та методів — так званий шаблон проектування «*Одинак*» (*Singleton*), при використанні якого можливо створити тільки один об'єкт такого класу:

```
class Singleton
{private: // єдиний екземпляр класу та базові конструктори оголошуються в приватній
    // секції класу
    static Singleton* instance;
    Singleton() {}
    Singleton(const Singleton&);
    Singleton& operator=(const Singleton&);
public:
    static Singleton* getInstance()
    {
        if(!instance)//якщо нелове значення, тобто обїект не існує
            instance = new Singleton();
        return instance;
```

```
    }
};

Singleton* Singleton::instance = 0; // ініціалізація статичної змінної

int main()
{
    Singleton* s = Singleton::getInstance();
// отримуємо покажчик на єдиний екземпляр класу Singleton
    return 0;
}
```

Лабораторне заняття 7.

Хід виконання завдання:

1) Організувати клас для предметної області, згідно варіанту. Для нього визначити та додати наступні поля:

- глобальне;
- статичне поле для підрахування кількості екземплярів зазначеного класу.
- поле атрибут «***ім'яОб'єкта***» (не глобальне і не статичне).

2) Організувати додатково клас ***Logger*** на основі класу «*Одинак*».

- Задача цього класу — вносити поточні значення публічних полів об'єктів розробленого класу (згідно варіанту) до поля **log** з використанням методу ***addRecord()***, як рядки, наступного вигляду:

  "object1Name: name
  object1Field1: field1Value
  object1Field1: field2Value"

- метод ***saveLog()*** викликатиметься в кінці роботи програми і зберігатиме текстовий файл за назвою «***log.txt***» (дата та час створення) наступного вигляду:

  "ClassName: numberOfEntities
  time: YY.MM.DD HH:MM:SS
  object1Name: name
  object1Field1: field1Value
  object1Field2: field2Value
  ...
  time: YY.MM.DD HH:MM:SS
  object2Name: name

object2Field1: field1Value
object2Field2: field2Value
...“:

3) Написати програму, в якій користувач матиме можливість проводити маніпуляції з визначеним класом (згідно варіанту) і логувати їх (протоколювати в файл ***log.txt)*** із використанням класу **Logger**..

Довідка: ***Log*****-файли** призначаються для протоколювання інформації про виконання певний операцій.

Варіанти завдань

1) Вектор в просторі.
2) Прямокутник.
3) Літак.
4) Дата.
5) Конус.
6) Пряма.
7) Ромб.
8) Відрізок.
9) Товар.
10) Студент.
11) Маршрут.
12) Трапеція.
13) Матриця.
14) Трикутник.
15) Час.
16) Циліндр.
17) Багатокутник.
18) Сфера.
19) Поїзд
20) Ламана
21) Вектор на площині
22) Книга
23) Натуральне число

Контрольні запитання

1) Що таке статичні поля класу? Чи належать вони об’єктам класу, у якому вони оголошені?

2) Для чого використовуються статичні змінні.
3) Як оголошуються статичні поля в класі?
4) Як можна звернутися до статичного елементу класу у програмі?
5) Поясніть призначення класу «*Singleton»*.