Об’єктно-орієнтоване програмування

на основі мови C++

2й семестр

Лекція 1

Викладач:

Розова Людмила Вікторівна

доцент, кандидат технічних наук

каф.ДММ, к.13

email: lyudmyla.rozova@khpi.edu.ua

курс «Об’єктно-орієнтоване програмування»:

об’єм курсу: 150 годин (5 кредитів)

1 кредит (освітня одиниця) = 30 аудиторних годин (45 хв)

З них: **Лекції** 16 годин - 0,5 пари на тиждень

Лабораторні заняття 64 години – 2 пари на тиждень

Самостійна робота 70 годин ≈ 4 години на тиждень!

Наприкінці **іспит**

Оцінювання:

Всього 100 балів

лабораторні роботи + тест = 100 балів

Література

1. **Страуструп Б.** Язык программирования C++. Специальное издание [Текст] / Бьерн Страуструп; пер. с англ. – М. : ООО Бином-Пресс, 2007. –1104 с.
2. **Шилдт Г.** C++: базовый курс [Текст] / Герберт Шилдт. – 3-е изд. –М. : Вильямс, 2009. – 624 с.
3. **Алейников, Д. В., Холодова, Е. П., Силкович, Ю.** Н. Язык программирования C++ : введение в объектно-ориентированное программирование : учебно-методическое пособие – Минск : НациональнаябиблиотекаБеларуси, 2016. – 85 с.
4. **C++. Основи програмування. Теорія та практика :** підручник **/** [О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката та ін.] ; за ред. О.Г. Трофименко. – Одеса: Фенікс, 2010. – 544 с.
5. **Павловская Т.А.** C/C++. Программирование на языке высокого уровня. — СПб.: Питер, 2001, 2003
6. **Павловская Т. А., Щупак Ю. А.** C++. Объектно-ориентированное программирование: Практикум. — СПб.: Питер, 2006. — 265 с:

План лекції

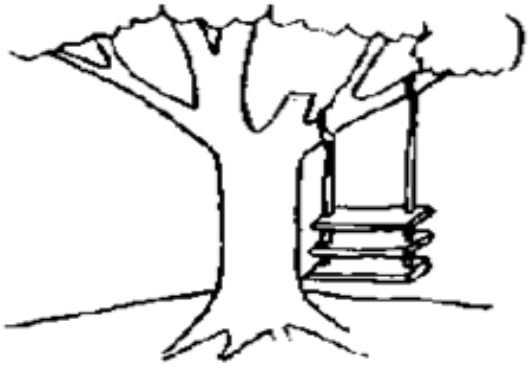
1. Введення в ООП
2. Поняття класу та об’єкту
3. Інкапсуляція, абстракція
4. Багатофайлові проекти
5. Конструктори, декструктор

Парадигми програмування

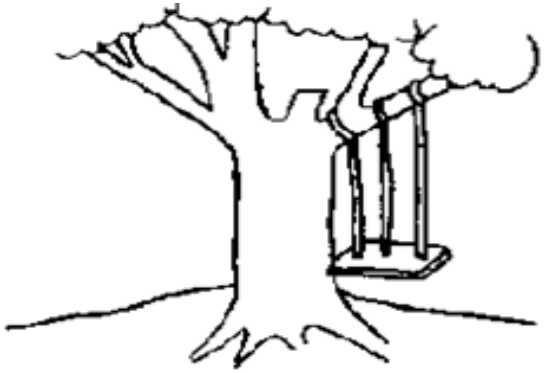
- **Структурне програмування** - проектування програми «зверху-вниз» з використанням певних алгоритмічних структур, покрокове їх виконання.
- В основі структурного підходу лежить розбиття програми на частини, підпрограми - процедурне програмування.

Процедурне програмування

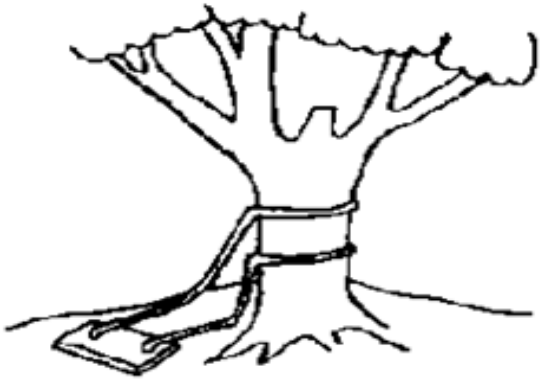
- C, Pascal, FORTRAN і інші подібні з ними мови програмування відносяться до категорії процедурних мов. Кожен оператор такої мови вказує комп'ютеру зробити деяку дію, *наприклад прийняти дані від користувача, зробити з ними певні дії і вивести результат цих дій на екран.* Програми, написані на процедурних мовах, являють собою послідовності інструкцій
- Дані + Алгоритм = Програма
- Програма, побудована на основі процедурного методу, розділена на функції, кожна з яких в ідеальному випадку виконує деяку закінчену послідовність дій і має явно виражені зв'язки з іншими функціями програми



1. Як було запропоновано організатором проекту



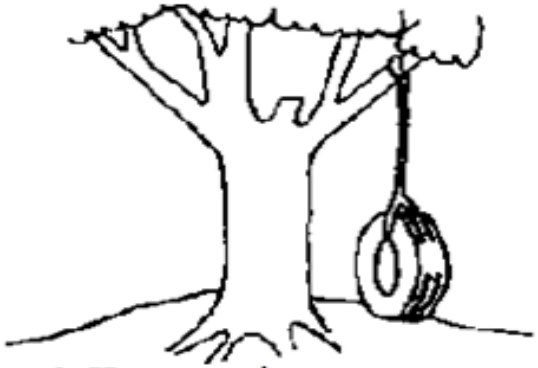
2. Як було описано в технічному завданні



3. Як було спроектовано провідним системним фахівцем

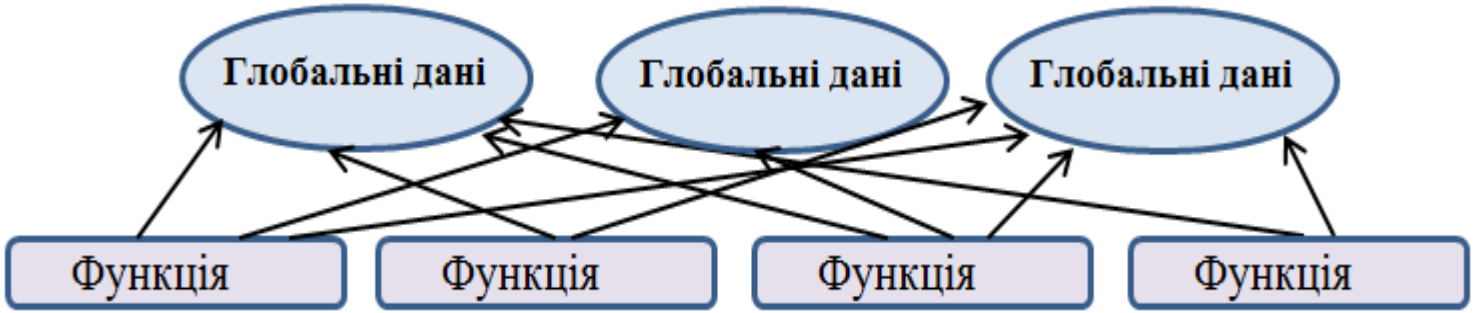
4. Як було реалізовано програмістами

5. Як було впроваджено



6. Чого хотів користувач

- У процедурній програмі, написаної, наприклад, на мові С, існує два типи даних: ***локальні і глобальні***
- **Локальні** дані знаходяться всередині функції і призначені для використання виключно цією функцією і не можуть бути змінені іншими функціями.
- Якщо існує необхідність спільного використання одних і тих ж даних декількома функціями, то дані повинні бути оголошені як **глобальні**.



- Великі програми зазвичай містять безліч функцій і глобальних змінних. Проблема процедурного підходу полягає в тому, що число можливих зв'язків між глобальними змінними і функціями може бути дуже велике.
- Велике число зв'язків між функціями і даними породжує кілька проблем:

➢ ускладнюється структура програми

➢ в програму стає важко вносити зміни.

Моделювання реального світу

Друга, більш важлива, проблема процедурного підходу полягає в тому, що відділення даних від функцій виявляється малопридатним для відображення картини **реального світу**.

У реальному світі нам доводиться мати справу з фізичними об'єктами, такими, наприклад, як люди або машини, інше.

Ці об'єкти не можна віднести ні до даних, ні до функцій, оскільки реальні речі являють собою сукупність **властивостей** і **поведінки**.

Прикладами *властивостей* (характеристиками) для людей можуть бути колір очей або довжина волосся; для машин - потужність двигуна і кількість дверей.
Таким чином, **властивості** об'єктів рівносильні **даним** в програмах. Вони мають певне значення

Поведінка - це деяка реакція об'єкта у відповідь на зовнішній вплив.

- Посмішка у людини на шутку
- Зупинка атомобіля при натисканні на гальмо
- Лай у собаки при сторонніх

Поведінка схожа з **функцією**: ви викликаєте функцію, щоб здійснити будь-яку дію, характерну лише цьому об’єкту.

Таким чином, ні окремо взяті дані, ні окремо взяті функції не здатні адекватно відобразити об'єкти реального світу.

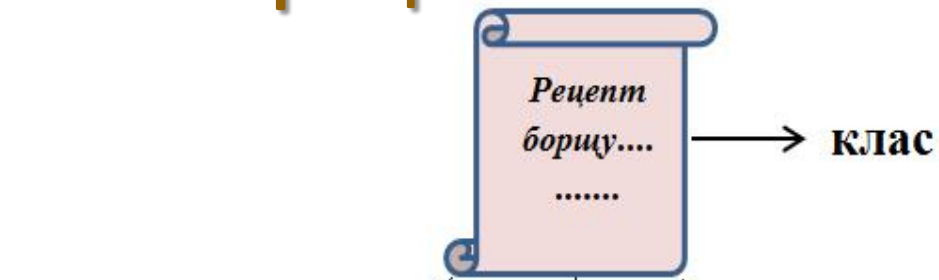
Об'єктно-орієнтований підхід

- Основною ідеєю об'єктно-орієнтованого підходу є об'єднання даних і дій, що здійснюються над цими даними в єдине ціле, яке називається **об'єктом**.
- Функції об'єкта (**методи**) зазвичай призначені для доступу до даних об'єкта. Якщо необхідно взяти будь-які дані об'єкта, потрібно викликати відповідний метод. Прямий доступ до даних неможливий.

В основі ООП лежить поняття **класа** і **об’єкта**.

- **Клас** є абстрактним типом даних, що визначається користувачем, і являє собою модель реального об'єкта у вигляді даних і функцій для роботи з ними.
- **Об'єкт** (або екземпляр класу) – це представник класу, побудований згідно створеному в класі опису. Кожному *класу* може належати одночасно декілька об'єктів, кожен з яких має унікальне ім'я. *Об'єкт* характеризується фізичним існуванням і є конкретним екземпляром *класу*.

Клас борщ



Дані класу

екземпляри класу - об'єкти

Методи класу:

- ✓ Де взяти інгредієнти;
- ✓ Помити, почистити, порізати;
- ✓ У якій послідовності застосувати, як;
- ✓ Скільки часу варити;
- ✓ …….

Клас (class)

Дані класу називаються **полями** (по аналогії з полями структури),

Функції класу - **методами.**

Поля і методи називаються **елементами класу**.

Опис класу в першому наближенні виглядає так:

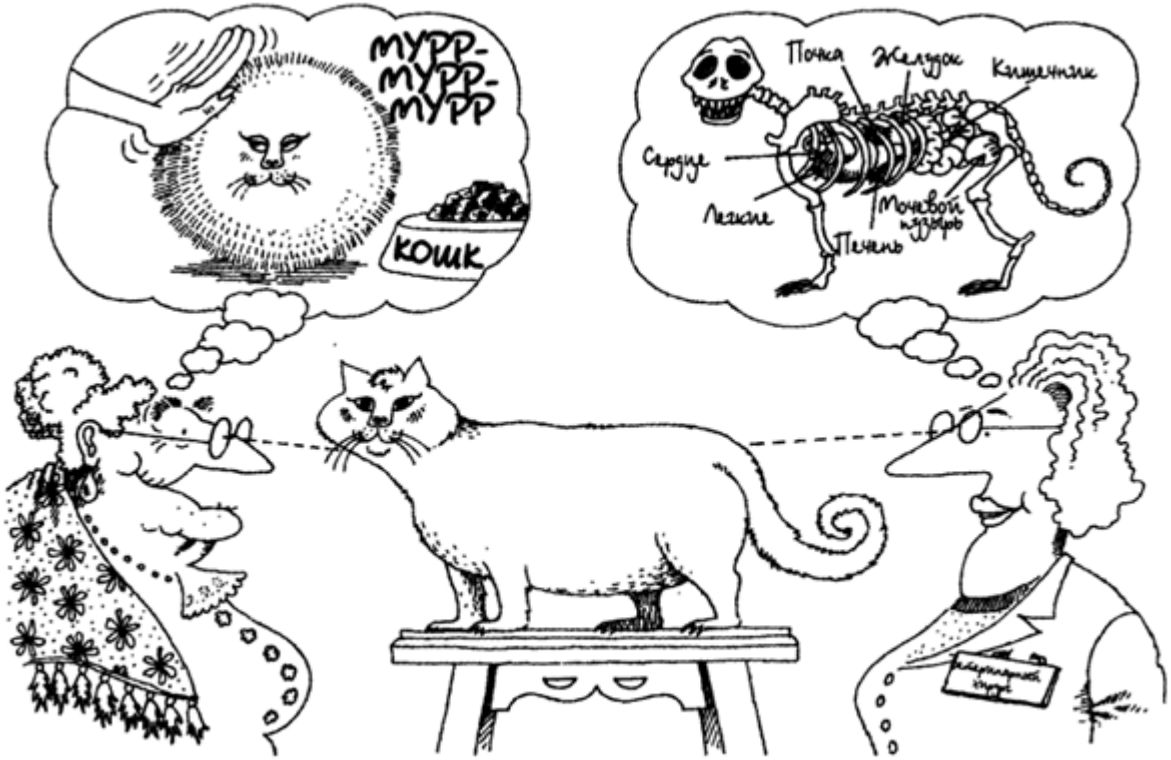
```
class <Ім'я> {
 private:
 <Опис прихованих елементів>
 public:
 <Опис доступних елементів>
 protected:
 <Опис захищених елементів>
}; // Опис закінчується крапкою з комою
```

Інкапсуляція

- об'єднання даних і методів для роботи з ними в один об'єкт.
- Інкапсуляція також реалізує приховування даних від зовнішнього впливу, що захищає їх від випадкових змін.

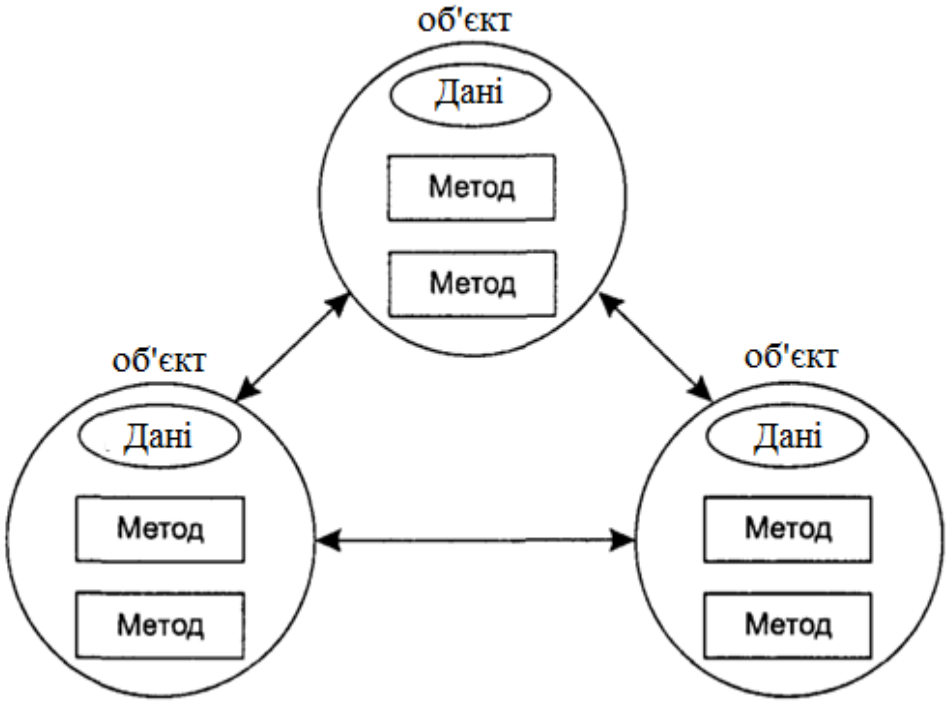
Для моделювання класу необхідно застосувати метод **абстракції -** виділення істотних характеристик деякого об'єкта, що відрізняють його від всіх інших видів об'єктів і таким чином, чітко описує його концептуальні межі з точки зору спостерігача.

Абстракція - один з основних методів, що дозволяють впоратися зі складністю вихідного завдання, бо основна задача проектувальника створити ілюзію простоти



Абстракція концентрує увагу на суттєвих властивостях об'єкта з точки зору спостерігача

Об'єктно-орієнтоване програмування ООП (object-oriented programming - OOP) - це метод програмування, заснований на представленні програми у вигляді сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії при успадкування.



Моделі реальних об'єктів (абстракції) +
+ поведінка об'єктів (взаємодії) = програма

Що може бути об’єктом реального світу (класом)?

Фізичні об'єкти:

- Автомобілі під час моделювання вуличного рухи.
- схемні елементи при моделюванні ланцюга електричного струму.
- країни при створенні економічної моделі.
- літаки при моделюванні диспетчерської системи.

Елементи інтерфейсу:

- вікна.
- меню.
- графічні об`єкти (лінії, прямокутники, кола).
- миша, клавіатура, дискові пристрої, принтери.

Структури даних

- масиви.
- стеки.
- пов'язані списки.
- бінарні дерева.

Що може бути об’єктом реального світу (класом)?

Групи людей

- Співробітники.
- Студенти.
- Покупці.
- Продавці.

Сховища даних

- Описи інвентарю.
- Списки співробітників.
- Словники.
- Географічні координати міст світу.

Призначені для користувача типи даних

- Час.
- Величини кутів.
- Комплексні числа.
- Точки на площині.

Учасники комп'ютерних ігр

- Автомобілі в гонках.
- Позиції в настільних іграх (шашки, шахи).
- Друзі та вороги в пригодницьких іграх.

Та інше

Режими доступу **private** і **public**

- специфікатори доступу **private** і **public** керують видимістю елементів класу.

Елементи, описані після службового слова ***private***, видимі тільки всередині класу. Цей вид доступу прийнятий в класі за замовчуванням. Елементи, описані після слова *public*, доступні зовні.

- Інтерфейс класу описується після специфікатора public.

Дія будь-якого специфікатор поширюється до наступного специфікатора або до кінця класу. Можна, задавати кілька секцій private і public, Порядок їх слідування значення не має.

```
class Employee
{private:
    char name[25];//ім'я
    int age; //вік
    char position[25]; //посада
    int stage; //стаж
public:
    void setName(char *n);
    void setAge(int s) { age = s;}
    int getAge() { return age;}
    int getStage(int currentYear, int
currentMonth, int currentDay);
};
void Employee::setName(char* n)
{
    strcpy(name, n);
}
```

Опис об’єктів (екземплярів класу)

```
Employee John;
Employee managers[20];
Employee *chef;
```

Доступ до елементів об'єкта

```
John.setName(name);
managers[2].setAge(30);
chef->getStage(2020, 2, 18);
```

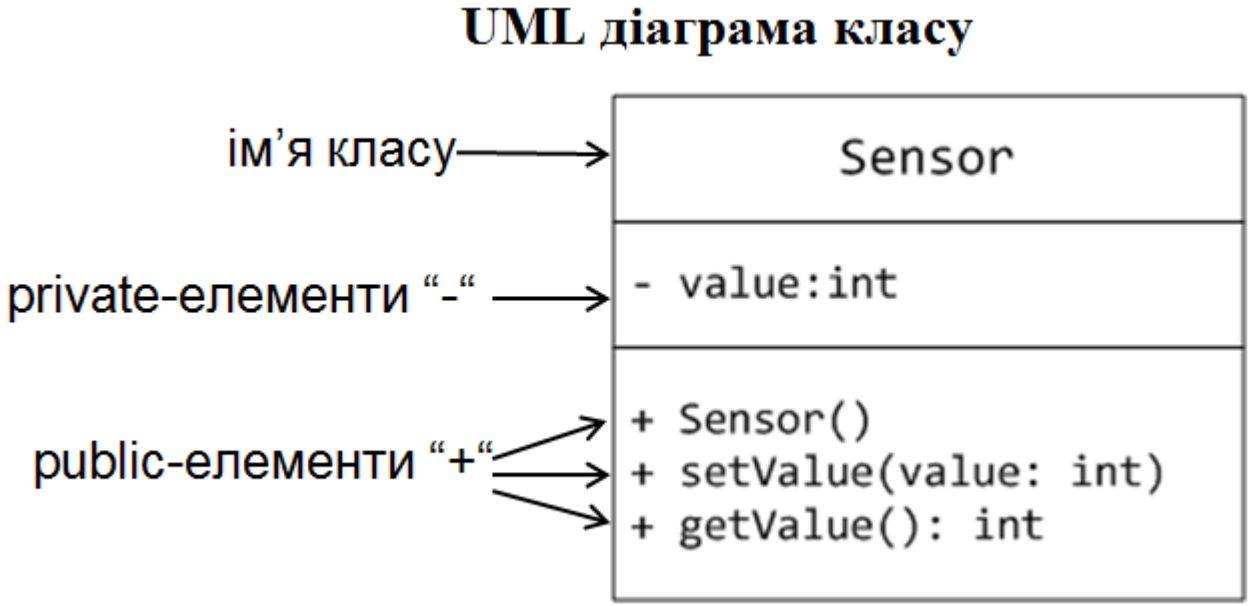
Геттери і сеттери - методи класу, які організовують доступ до приватних елементів класу, застосовуючи певну логіку доступу, враховуючі небажані ситуації

UML діаграми класів

Одним з способів опису та представлення об'єктно-орієнтованих програм та класів зокрема, є використання ***UML-нотації***.

UML (*unified modeling language*) — узагальнена мова опису предметних областей, зокрема, програмних систем, об'єктно-орієнтованого аналізу і проєктування. Використовується для візуалізації, специфікації, конструювання і документування програмних систем. В контексті ООП в UML містяться такі базові компоненти, як сутності, зв'язки між ними, та набори сутностей

Структурна сутність клас уявляє собою опис набору об'єктів з однаковими властивостями (атрибутами), операціями (методами), зв'язками та поведінкою



Відділення інтерфейсу від реалізації

Один з найбільш фундаментальних принципів розробки хорошого програмного забезпечення полягає в відділенні інтерфейсу від реалізації:

при побудові програми на С ++ для зручності подальшої підтримки та сприйняття об'єктно-орієнтованого коду програми, слід розділяти об'явлення класу та його реалізацію, Для цього об'явлення класу слід надавати в заголовкових файлах, а реалізацію — в файлах вихідного коду

Заголовки - імя_класса.h

Файл вихідного коду - імя_класса.cpp

Заголовки включаються (через #include) в кожен файл, в якому використовується клас, а файли з вихідними кодами компілюються і компонуються з файлом, що містить головну програму (main).

Конструктори класу

Конструктор - метод класу, який служить ***для створення*** і ***ініціалізації екземпляра*** класу. Ім'я конструктора завжди збігається з ім'ям класу

- ✓ **Ім’я конструктора – це ім’я класу**. Отже, компілятор відрізняє конструктор від інших методів класу

```
Employee();
```

- ✓ Виконується автоматично в момент створення об’єкта
- ✓ Для створення екземпляра класу потрібно, щоб конструктор був методом типу *public*
- ✓ Конструктор не має типу, не повертає жодного значення, навіть типу void
- ✓ **Клас** може мати **кілька конструкторів** з різними параметрами для різних видів ініціалізації
- ✓ Конструктори не успадковуються

Деструктор класу

Деструктор - метод класу, який служить для видалення екземпляра класу.

- ✓ **Ім'я деструктора** складається з символу **~ (тильда) і імені класу**

```
~Employee();
```

- ✓ Деструктор не має аргументів і значення, що повертається.
- ✓ Деструктор викликається автоматично, коли об'єкт виходить з області видимості:

- для локальних об'єктів - при виході з блоку, в якому вони оголошені;
- для глобальних - як частина процедури виходу з main;
- для об'єктів, заданих через покажчики, деструктор викликається неявно при використанні операції delete.

```
//Файл CreateAndDestroy.h
class CreateAndDestroy
{
public:
        CreateAndDestroy(int); // Конструктор
        ~CreateAndDestroy(); // Деструктор
private:
        int data;
};
//-----------------Конец файла-----

//Файл CreateAndDestroy.cpp
#include <iostream>
using namespace std;
CreateAndDestroy::CreateAndDestroy(int value)
{
    data = value;
    cout << "Object "<< data << " constructor" << endl;
}
CreateAndDestroy::~CreateAndDestroy()
{    cout << "Object "<< data << " destructor" << endl;
}
//-------Кінець файлу-----------
```

```
// Файл main.cpp
#include <iostream>
#include "CreateAndDestroy.h"
using namespace std;
CreateAndDestroy first(1);
int main()
{   cout << "main: Hello" << endl;
    CreateAndDestroy second(2);
    cout << "main: second created"<< endl;
    CreateAndDestroy* third = new CreateAndDestroy(3);
    cout << "main: third created"<< endl;
    delete third;
    cout << "main: third deleted" << endl;
    cout << "Exit from main" << endl;
    return 0;
}
```

```
Object 1 constructor
main: Hello
Object 2 constructor
main: second created
Object 3 constructor
main: third created
Object 3 destructor
main: third deleted
Exit from main
Object 2 destructor
Object 1 destructor
```

Дякую за увагу!