

Лекція 4 (2023)

Об'єктно-орієнтоване програмування

Лектор: *Розова Людмила Вікторівна*

План лекції 4

1

Статичні елементи

2

Успадкування

2

Конструктори, деструктори при успадкуванні

Матеріали курсу:

https://github.com/LRozova/OOP_ukr_2023

Статичні змінні

В C++ є можливість доступу всіх створених об'єктів конкретного класу до однієї змінної (полю), вміст якої зберігається в одному місці. Для цього оголошують змінну:

static тип ім'я ;

- Ключове слово **static** може бути використано як для атрибутів, так і для методів класу.
- Особливістю елементів **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.
- Доступ до статичних змінних відбувається з використанням імені класу, оператору розширення видимості “**::**” та можливий **тільки після ініціалізації**:

тип ім'я_класу:: ім'я_змінної = початк.значення;

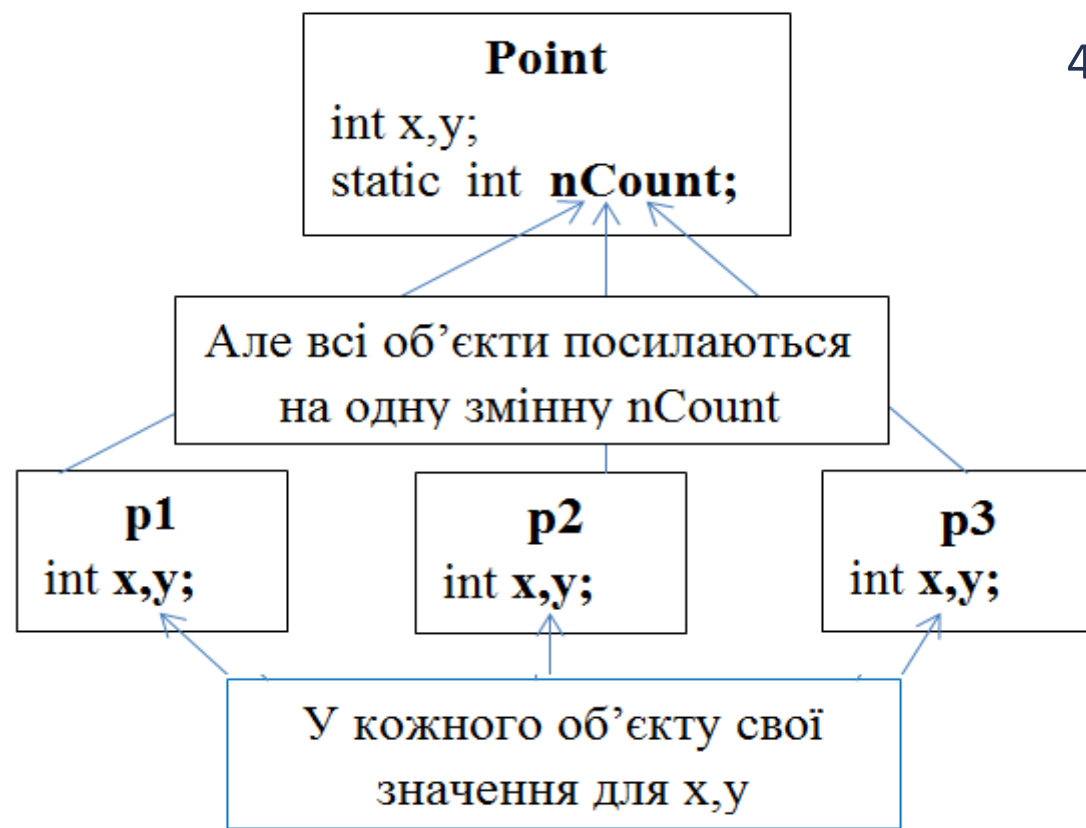
Статичні змінні

4

```
class Point {  
public:  
    int x,y;  
    static int nCount;  
    Point(){};  
};
```

```
int Point::nCount=0;
```

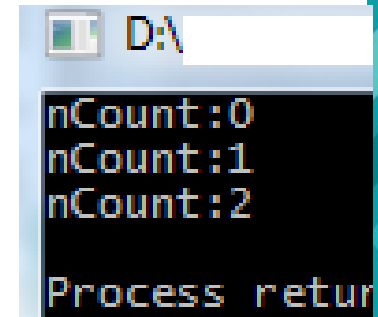
```
int main()  
{Point p1,p2,p3;  
  cout<<"nCount:"<<Point::nCount<<endl;  
  cout<<p1.nCount<<" "<<p2.nCount<<" "<<p3.nCount  
    <<endl;  
  Point::nCount=1;  
  cout<<p1.nCount<<" "<<p2.nCount<<" "  
    <<p3.nCount<< endl;  
  return 0;}
```



A screenshot of a terminal window with a black background and white text. The output shows the static variable nCount being printed as 0, followed by the instance variables x and y for three objects (p1, p2, p3) being printed as 0 0 0. Then, the static variable nCount is updated to 1, and the instance variables x and y for the three objects are printed as 1 1 1. The text "Process returne" is partially visible at the bottom.

Підрахунок створених об'єктів класу

```
class Point {  
    public:  
        int x,y;  
        static int nCount;  
        Point() {++nCount};  
};  
  
int Point::nCount=0;  
  
int main()  
{cout<<"nCount:"<<Point::nCount<<endl;  
  Point p1;  
  cout<<"nCount:"<<Point::nCount<<endl;  
  Point p2;  
  cout<<"nCount:"<<Point::nCount<<endl;  
  return 0;}
```



```
D:\  
nCount:0  
nCount:1  
nCount:2  
Process return
```

Статичні методи

- Якщо статична змінна об'явлена у розділі **private** до неї має доступ відкритий статичний метод (**static**).

static int Point::getCount ();

- Статичний метод не має покажчика *this*, тому що статичні поля і статичні методи існують незалежно від будь-яких об'єктів класу, тобто до них не прив'язані
- Метод класу може бути оголошений як **static**, якщо він не має доступ до нестатичних елементів класу.
- Статичний метод викликається з додаванням перед його ім'ям **імені класу** і бінарної операції оператора розширення видимості “**::**” **Point::getCounter();**
або через об'єкт класу **p1.getCounter();**
- Статичні поля класу створюються в єдиному екземплярі незалежно від кількості визначених в програмі об'єктів.
- Всі об'єкти (навіть створені динамічно) поділяють єдину копію статичних полів.

```
class Account //клас банківський рахунок
{private:
    double sum;
    static int rate; //процентна ставка
    const static int rate_default=5; //стат. КОНСТАНТА
public:
    Account(double sum) ;
    double getIncome() ;

//оголошення статичних методів
    static int getRate();
    static void setRate(int r) ;
};

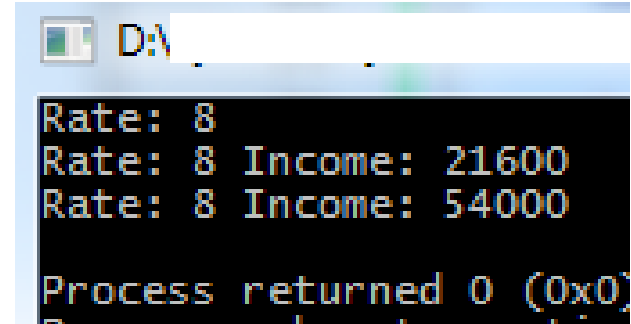
int Account::rate = 5;

Account::Account(double sum)
    {    this->sum = sum;    }

double Account::getIncome()
    {    return sum + sum * rate / 100;    }
```

//реалізація статичних методів другий раз static не вказується

```
int Account::getRate ()
{ return rate; }
void Account::setRate(int r)
{ rate = r; }
```



```
DA
Rate: 8
Rate: 8 Income: 21600
Rate: 8 Income: 54000
Process returned 0 (0x0)
```

```
int main()
{
    Account ac1(20000), ac2(50000);
    Account::setRate(8); //НОВЕ ЗНАЧЕННЯ rate
    cout <<"Rate:"<< Account::getRate() <<endl;
    cout <<"Rate:"<< ac1.getRate() <<"Income:"
        << ac1.getIncome() << endl;
    cout <<"Rate:"<<ac2.getRate() <<"Income:"
        << ac2.getIncome() << endl;
    return 0; }
```


Успадкування

Успадкування – створення нових класів на базі існуючих.

Це дуже потужна можливість в ООП, що дозволяє створювати нові похідні класи, взявши за основу всі методи і елементи базового класу. Таким чином економиться час на написання і налагодження коду нової програми.



Механічна кавомолка

Електрична кавомолка

- мотор
- кнопки
- схеми

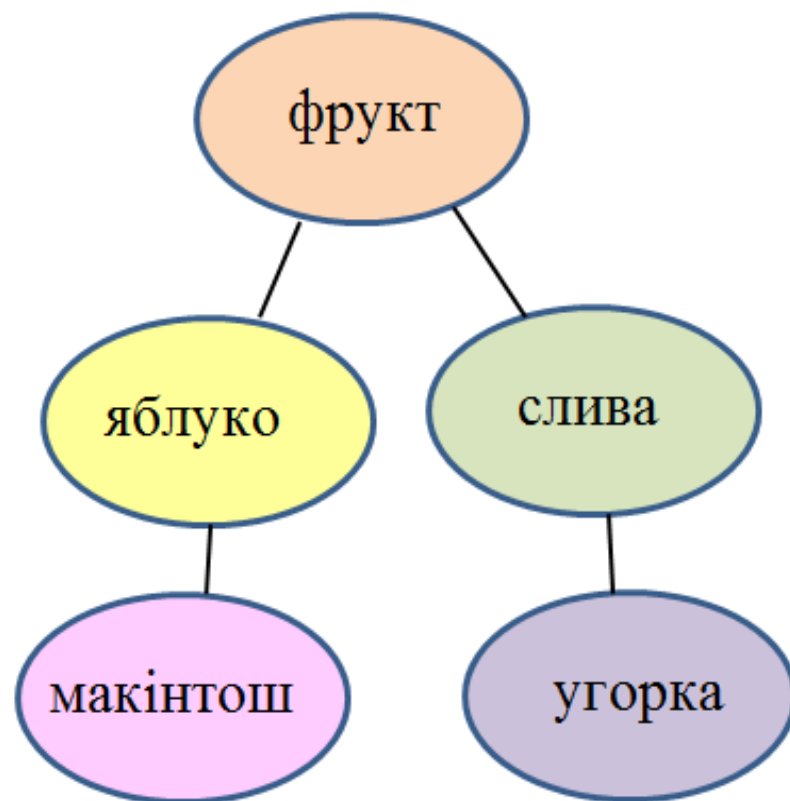
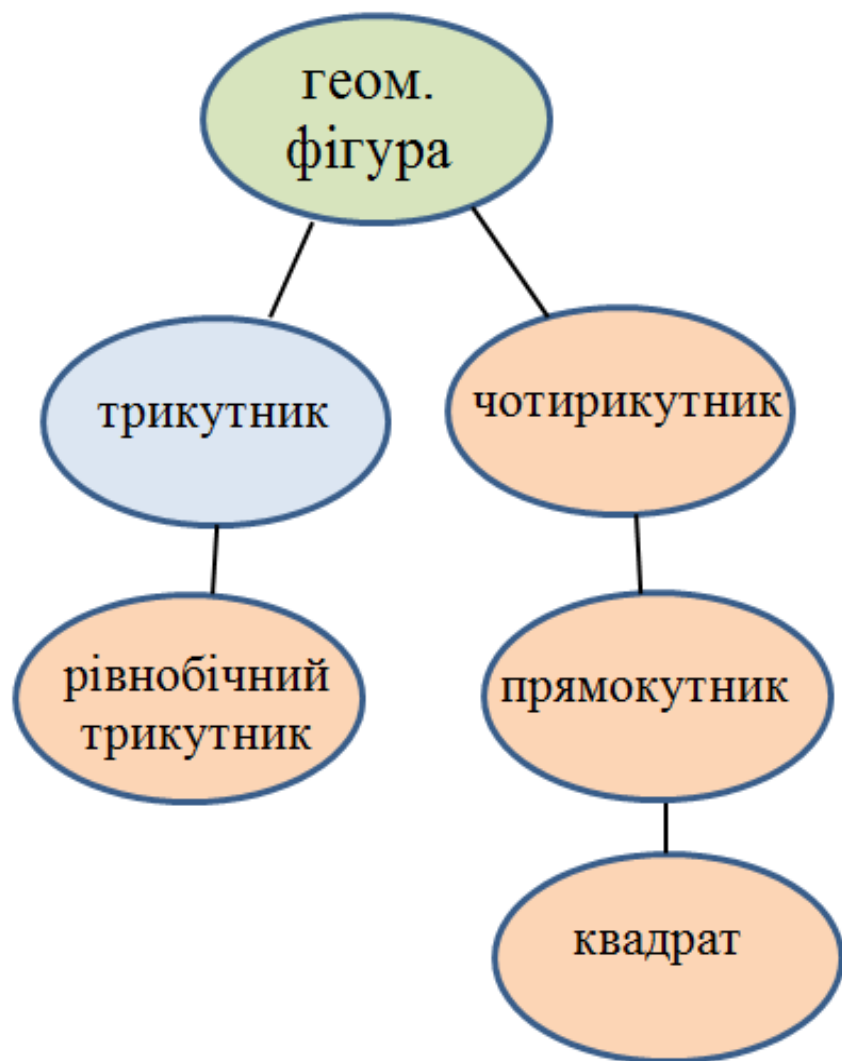
...

Клас, який успадковується, називається **базовим** (або **предком, суперкласом**).

Клас, який успадковує базовий клас, називається **похідним** (або **нащадком, підкласом**).

Похідний клас можна використовувати в якості базового для іншого похідного класу. Таким чином вбудовується багаторівнева ієрархія класів.

Успадкування включає в собі створення нових об'єктів шляхом безпосереднього збереження властивостей і поведінки інших об'єктів, а потім їх розширення або навпаки - конкретизації.



Кожен об'єкт похідного класу є також об'єктом відповідного базового класу. Однак, зворотне невірно: об'єкт базового класу не є об'єктом класів, породжених цим базовим класом.

У нащадка можна описувати *нові поля і методи*, а також *перевизначити існуючі методи*.

Види успадкування: **просте і множинне**.

Просте успадкування - кожен клас має тільки один батьківський клас найближчого рівня.

Множинне спадкування - клас-нащадок створюється з використанням декількох базових класів-батьків

Успадкування застосовується для таких взаємопов'язаних цілей:

- виключення з програми повторюваних фрагментів коду;
- спрощення модифікації програми;
- спрощення створення нових програм на основі існуючих.

Крім того, успадкування є єдиною можливістю використовувати об'єкти, вихідний код яких недоступний, але в які потрібно внести зміни.

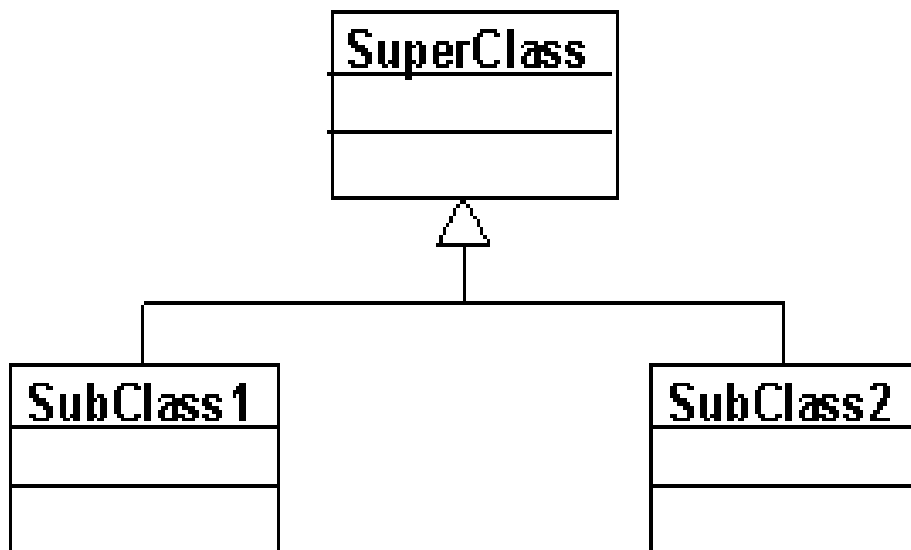
Успадкування встановлює між класами **відношення «є»**: похідний клас є частиною базового класу.

Синтаксис успадкування

Ключі доступу

class ім'я : [private | protected | public] базовий_клас
{ тіло класу };

```
class A { ... };  
class C { ... };  
class D: public C  
{ ... };  
class B: protected A  
{ ... };
```



Успадкування встановлює між класами
відношення «є»:
похідний клас є частиною базового
класу

Правило успадкування

Режим доступу до елемента в базовому класі	Ключ доступу при успадкуванні класу	Режим доступу до елемента в похідному класі
private		недоступний
protected	public	protected
public		public
private		недоступний
protected	protected	protected
public		protected
private		недоступний
protected	private	private
public		private

```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};
```

//B успадковує A

```
class B : public A
{
    // x - public
    // y - protected
    // z - не має
        доступу
};
```

//C успадковує A

```
class C : protected A
{
    // x - protected
    // y - protected
    // z не має доступу
};
```

//D успадковує A

```
class D : private A
{
    // x - private
    // y - private
    // z - не має
        доступу
};
```

Якщо базовий клас успадковується з ключем **private**, можна вибірково зробити деякі його елементи доступними в похідному класі:

```
class Base{  
    ...  
    public: void f();  
};  
class Derived : private Base{  
    ...  
    public: Base::void f();  
};
```


Приклад. Успадкування

```
#include <iostream>
#include <string>
using namespace std;
class Person // базовий клас
{
public:
    string name;
    int age;
    void display()
    {
        cout << "Name: " << name << "\tAge: " << age
<<endl;
    }
};
```

Приклад. Успадкування. Продовження

```
class Employee : public Person //похідний  
клас
```

```
{public:  
    string company;  
};
```

```
int main()  
{    Person tom;  
    tom.name = "Tom";  
    tom.age = 23;  
    tom.display();  
    Employee bob;  
    bob.name = "Bob";  
    bob.age = 31;  
    bob.company = "Microsoft";  
    bob.display();  
    return 0;}
```

Виклик конструкторів класів при успадкуванні

Конструктори не успадковуються, тому похідний клас повинен мати власні конструктори.

Порядок виклику конструкторів при створенні об'єкту похідного класу:

- Конструктор базового класу викликається автоматично. При цьому, якщо в похідному класі явний виклик конструктора базового класу відсутній, автоматично викликається конструктор базового класу за замовчуванням.
- Потім виконується відповідний **конструктор похідного класу**.
- При багаторівневому успадкуванні конструктори викликаються по черзі походження класів.

Деструктори при успадкуванні

- Деструктори не успадковуються.
- Якщо деструктор в похідному класі не описаний, він формується **автоматично** і викликає деструктори всіх базових класів.
- Не потрібно явно викликати деструктори базових класів, це буде зроблено автоматично.
- Для ієрархії, що складається з декількох рівнів, деструктори викликаються в порядку, **строого зворотному виклику конструкторів**: спочатку викликається деструктор класу, потім - деструктори елементів класу, а потім деструктори базового класу.

Виклик конструкторів і деструкторів класів при успадкуванні

21

```
#include <iostream>
using namespace std;
class base {
public:
base() { cout <<" Створення base-об'єкту.\n"; }
~base() { cout <<" Знищення base-об'єкту.\n"; }
};
class derived1 : public base {
public:
derived1()
{ cout <<"Створення derived1-об'єкту.\n"; }
~derived1()
{ cout <<"Знищення derived1-об'єкту.\n"; }
};
```

```
class derived2: public derived1 {  
public:  
    derived2()  
    { cout <<"Створення derived2-об'єкту\n"; }  
    ~derived2()  
    { cout <<"Знищення derived2-об'єкту.\n"; }  
};  
int main()  
{  
    derived2 ob;  
    return 0;  
}
```

```
Створення base-об'єкту.  
Створення derived1-об'єкту.  
Створення derived2- об'єкту  
Знищення derived2- об'єкту.  
Знищення derived1- об'єкту.  
Знищення base- об'єкту.
```

```
Process returned 0 (0x0)   execution time : 0.049 s  
Press any key to continue.
```

Дякую за увагу!