

# Лекція 5

## Об'єктно-орієнтоване програмування

Лектор: *Розова Людмила Вікторівна*

# План лекції 5

- 1 Поліморфізм
- 2 Віртуальні функції
- 3 Абстрактні класи
- 4 Віртуальний деструктор
- 5 Віртуальне успадкування

Матеріали курсу:

<https://github.com/LRozova/oop1>

# Поліморфізм

**Поліморфізм** (от греч. *polýs* — багато и *morphé* — форма) означає можливість приймати різноманітні форми, зберігаючи суть, або одночасно приймати різноманітні форми.

Поліморфізм в програмуванні проявляється, наприклад, в перевантаженні функцій, операторів і операцій.

При **успадкуванні** поліморфізм — можливість об'єктів різних класів, що пов'язані відносинами наслідування, реагувати по різному під час виклику одного й того-самого методу.

«Один інтерфейс – багато методів»

# Віртуальні функції

Реалізація динамічного *поліморфізму* в C++ здійснюється завдяки поєднанню успадкування і **віртуальних функцій (методів)** — функцій, що оголошуються в базовому класі з використанням ключового слова *virtual* і перевизначаються в одному або декількох похідних класах. Таким чином, кожний похідний клас може мати власну версію віртуальної функції.

```
virtual    тип    ім'я_функції    (параметри)  
{  
}
```

## Розглянемо приклад 1

```
#include <iostream>
using namespace std;
class A
{ public:
    void what() // оголошення функції what()
    { cout << "BASE class\n "; }
};
class A1 : public A
{ public:
    void what() // перевизначення функції what() для класу A1
    { cout << "1_DERIVED class\n "; }
};
class A2 : public A1
{ public:
    void what() // перевизначення функції what() для класу A2
    { cout << "2_DERIVED class\n "; }
};
```

```

int main() {
    A base_object;
    A *p;
    A1 a1_object;
    A2 a2_object;
    p = &base_object; // встановлюємо покажчик на об'єкт
// базового класу
    p->what(); //викликаємо метод who()
    p = &a1_object; // встановлюємо покажчик на об'єкт
// класу A1
    p->what(); //викликаємо метод who()
    p = &a2_object; // встановлюємо покажчик на об'єкт
// класу A2
    p->what(); //викликаємо метод who()
    return 0;
}

```

```

BASE class
BASE class
BASE class

Process returned 0 (0x0)   executi
Press any key to continue.

```

## Розглянемо приклад 2

```
#include <iostream>
using namespace std;
class A
{ public:
    virtual void what() // оголошення віртуальної функції
    { cout << "BASE class\n "; }
};
class A1 : public A
{ public:
    //перевизначення функції what() для класу A1 слово virtual не обов'язкове
    void what() //override
    { cout << "1_DERIVED class\n "; }
};
class A2 : public A1
{ public:
    void what() // перевизначення функції what() для класу A2
    { cout << "2_DERIVED class\n "; }
};
```

```

int main() {
    A base_object;
    A *p;
    A1 a1_object;
    A2 a2_object;
    p = &base_object; // встановлюємо покажчик на об'єкт
// базового класу
    p->what(); //викликаємо метод who() для класу A
    p = &a1_object; // встановлюємо покажчик на об'єкт
// класу A1
    p->what(); //викликаємо метод who() для класу A1
    p = &a2_object; // встановлюємо покажчик на об'єкт
// класу A2
    p->what(); //викликаємо метод who() для класу A2
    return 0;
}

```

```

BASE class
1_DERIVED class
2_DERIVED class

Process returned 0 (0x0)
Press any key to continue

```



- Важливим моментом забезпечення ідеї поліморфізму є те, що звернення до віртуальної функції відбувається **через покажчик** базового класу, який використовується в якості посилання на об'єкт похідного класу.
- У такому випадку компілятор C++ автоматично визначає, яку саме версію віртуальної функції (методу) потрібно викликати, по типу об'єкта, що адресується цим покажчиком, такий вибір відбувається під час виконання програми.
- За наявності кількох похідних класів при посиланні покажчика базового класу на різні об'єкти цих похідних класів будуть виконуватися різні версії віртуальної функції (метода).
- Також віртуальну функцію можна викликати як будь-яку іншу компонентну функцію.
- **Поліморфний клас** — клас, який включає віртуальну функцію

# Правила оголошення та використання віртуальних функцій-методів

10

1. Віртуальна функція може бути тільки методом класу.
2. Кількість і тип параметрів віртуальних функцій у базовому та похідних класах повинні точно збігатися і мати однакові прототипи, на відміну від перевантажених функцій.
3. Віртуальна функція успадковується.
4. Будь-який перевантажений метод класу можна зробити віртуальним, наприклад перевантаження операцій.
5. Статичні методи не можуть бути віртуальними.
6. Конструктори не можуть бути віртуальними.
7. Деструктори можуть (частіше - повинні) бути віртуальними - це гарантує коректне звільнення пам'яті через покажчик базового класу.

## Раннє та пізнє зв'язування

**Статичний поліморфізм** реалізується через перевантаження функцій і операцій.

**Динамічний поліморфізм** - через використання віртуальних функцій.

Статичний і динамічний варіанти поліморфізму співвідносять з поняттями **раннього і пізнього зв'язування**.

**Зв'язування** - це процес, який використовується для конвертації ідентифікаторів (таких як імена змінних або функцій) в адреса.

# Раннє зв'язування

**Раннє зв'язування** стосується подій етапу компіляції програми, таких як виклик:

- звичайних функцій;
- перевантажених функцій;
- невіртуальних компонентних функцій;
- дружніх функцій.

При виклику перерахованих функцій вся необхідна адресна інформація відома при компіляції.

**Перевагою** раннього зв'язування є висока швидкодія виконання програм.

**Недоліком** раннього зв'язування є зниження гнучкості програм.

## Пізнє динамічне зв'язування

Стосується подій, що відбуваються в процесі виконання програми.

При виклику функцій з використанням **пізнього динамічного зв'язування** адрес функції, що буде викликатися, до початку виконання програми невідомий.

Зокрема, об'єктом пізнього зв'язування є **віртуальні функції**.

**Перевагою** пізнього зв'язування є висока гнучкість виконуваної програми, можливість реакції на події.

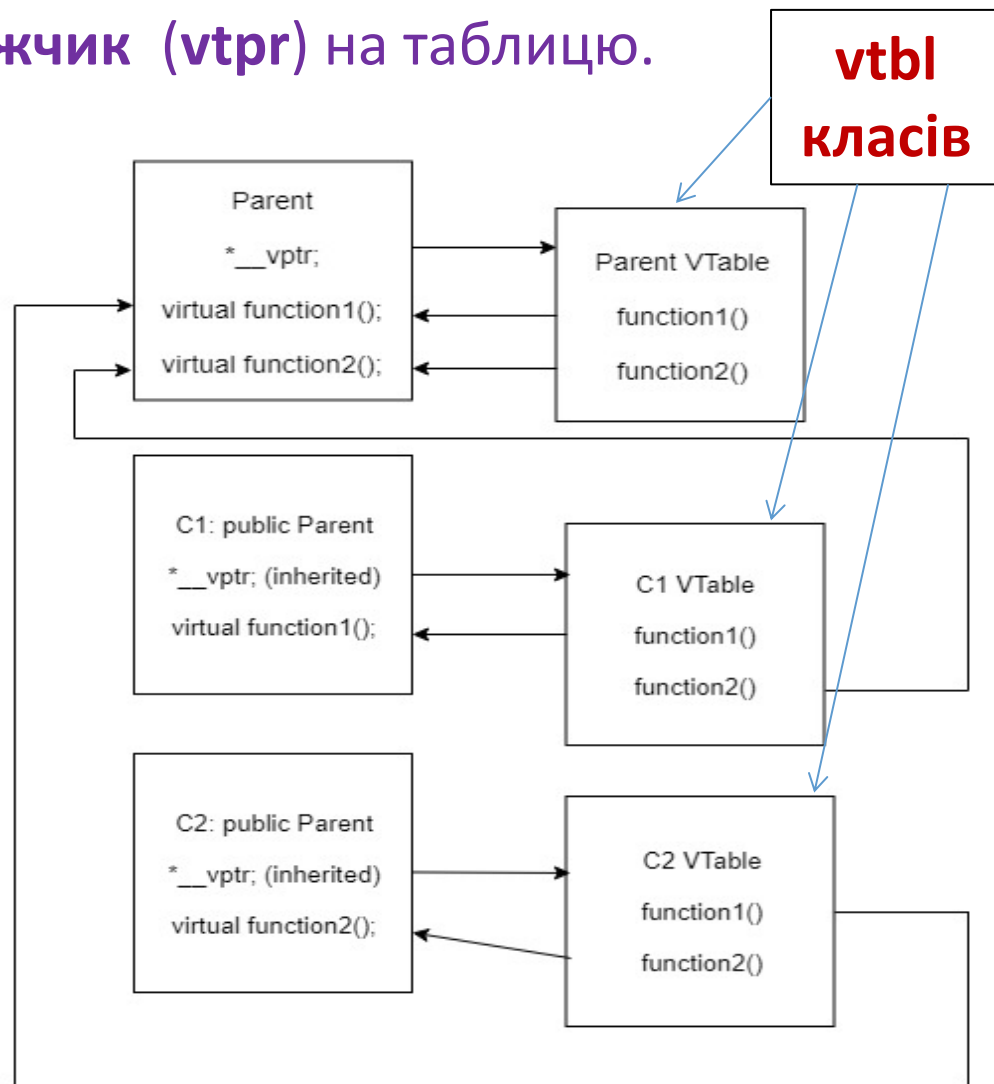
**Недоліком** є відносно низька швидкодія програми, виконуються додаткові дії.

Для реалізації пізнього динамічного зв'язування компілятор:

- створює **таблицю віртуальних методів (vtbl)** для кожного класу з віртуальним методом
- Додає приховане поле-показчик (**vtpr**) на таблицю.

Розмір класу з віртуальними функціями-методами збільшується на 4б - це розмір показника.

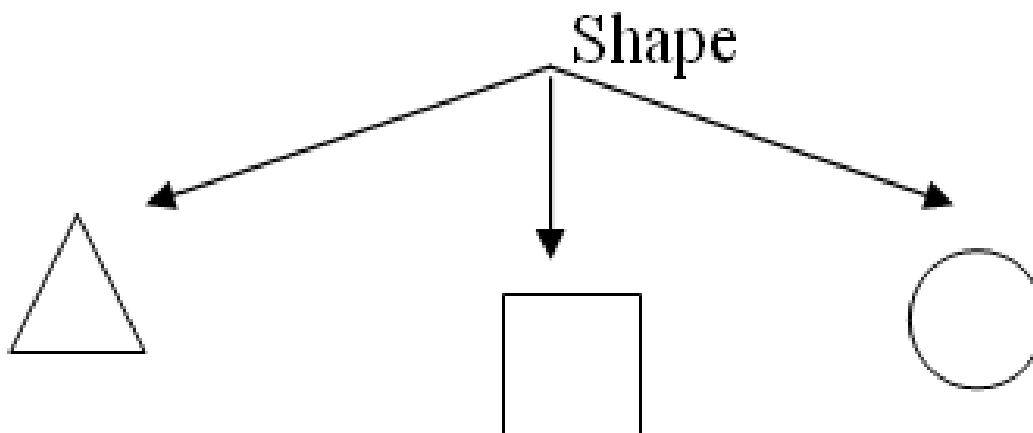
При виклику віртуального методу через показник базового класу при виконанні програми визначається тип об'єкта, на який він має посилання, та обирається версія віртуального методу для виклику (перехід на vtbl цього класу)



# Абстрактний клас

**Абстрактний клас** – це клас, який висловлює якусь загальну концепцію, яка відобразатиме основну ідею для використання в похідних класах.

Абстрактний клас створюють тільки для того, щоб на його основі створювати інші класи. Створювати екземпляри об'єктів таких класів не можна, тому їх називають абстрактними.



## Суто віртуальна функція

**Суто віртуальна функція** оголошується у базовому класі, не має чіткої реалізації, яка повинна бути обов'язково реалізована в похідних класах .

***virtual тип ім'я\_функції(список\_параметрів) = 0;***

Будь-який похідний клас від базового, що містить суто віртуальну функцію, має примусово реалізувати таку функцію.

**Абстрактний клас** — клас, який містить хоча б одну суто віртуальну функцію.

**В абстрактного класу не може бути об'єктів!**



## Приклад 3

```
#include <iostream>
using namespace std;
class Shape //абстрактний клас
{public:
    virtual double perimetr()=0; //суто віртуальна функція
};

class Circle: public Shape
{ double rad;
public:
    Circle (double r)
    { rad=r; }
    double perimetr() override //визначення вірт.функції
    { return 2*3.1415*rad; }
};
```

```
class Triangle_eq: public Shape
```

```
{ double a;
```

```
public:
```

```
    Triangle_eq (double aa)
```

```
    { a=aa; }
```

```
    double perimetr() override //визначення віртуальної
```

```
// функції для класу Triangle_eq
```

```
    { return 3*a; }
```

```
};
```

```
class Rectangle: public Shape
```

```
{ double a;
```

```
  double b;
```

```
public:
```

```
    Rectangle (double aa,double bb)
```

```
    { a=aa;b=bb; }
```

```
    double perimetr() override //визначення віртуальної
```

```
// функції для класу Rectangle
```

```
    { return 2*a*b; }
```

```
};
```

```

int main()
{
    Shape *ptr; // покажчик на об'єкт абстрактного класу
// є допустимим.
    Circle C(1);
    Triangle_eq T(1);
    Rectangle R(1,2);
    ptr=&C; // посилення на об'єкт класу Circle
    cout<<"Perim for circle="<<ptr->perimetr()<<endl;
    ptr=&T; // посилення на об'єкт класу Triangle_eq
    cout<<"Perim for triangle= "<<ptr->perimetr()
<<endl;
    ptr=&R; // посилення на об'єкт класу Rectangle
    cout<<"Perim for rectangle = "<<ptr->perimetr()
<<endl;
}

```

```

Perim for circle= 6.283
Perim for triangle= 3
Perim for rectangle= 4

Process returned 0 (0x0)
Press any key to continue.

```

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

class Sport
{public:
    virtual void play()=0;
};

class Football: public Sport
{public:
    void play() //override визначення метода play()
    { cout<<"play football"<<endl;
    }
};

class Tennis: public Sport
{public:
    void play() //override визначення метода play()
    { cout<<"play tennis"<<endl;
    }
};
```

```
class Basketball: public Sport
```

!1

```
{public:
```

```
    void play() //override визначення метода play()
```

```
{
```

```
    cout<<"play basketball"<<endl;
```

```
}
```

```
};
```

```
class Player
```

```
{private:
```

```
    Sport *sp;
```

```
public:
```

```
    Player () {sp=NULL;}
```

```
    void setSport(Sport *sp) {this->sp=sp;}
```

```
    void play() //свій метод play()
```

```
{
```

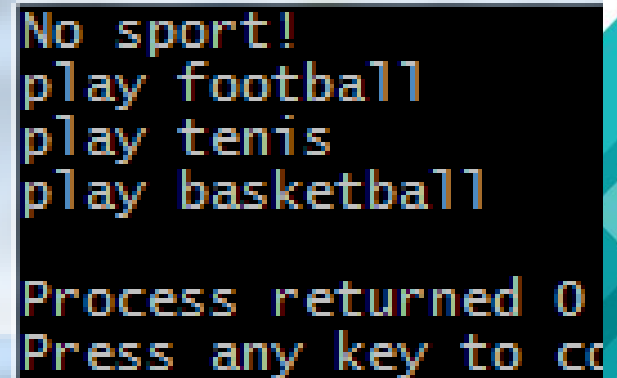
```
    if (sp==NULL) cout<<"No sport!"<<endl;
```

```
    else sp->play(); //викликає метод play() згідно посилання
```

```
}
```

```
};
```

```
int main()
{
    Sport *ptr;
    Player student;
    student.play();
    Football F;
    ptr=&F;
    student.setSport(ptr);
    student.play();
    Tennis T;
    Basketball B;
    student.setSport(&T);
    student.play();
    student.setSport(&B);
    student.play();
    return 0;
}
```



```
No sport!
play football
play tennis
play basketball

Process returned 0
Press any key to co
```

# Віртуальний деструктор

- **Деструктор** класу може бути **віртуальним**.
- Коли деструктор базового класу віртуальний, то і деструктори всіх похідних класів також.
- Деструктор необхідно оголошувати віртуальним, якщо доступ до динамічного об'єкту похідного класу виконується через покажчик базового класу. В цьому випадку при знищенні об'єкта через покажчик базового класу викликається деструктор похідного класу, а він викликає деструктор базового класу.
- Якщо деструктор базового класу не віртуальний, то при знищенні об'єкта похідного класу через покажчик базового класу викликається деструктор тільки базового класу.

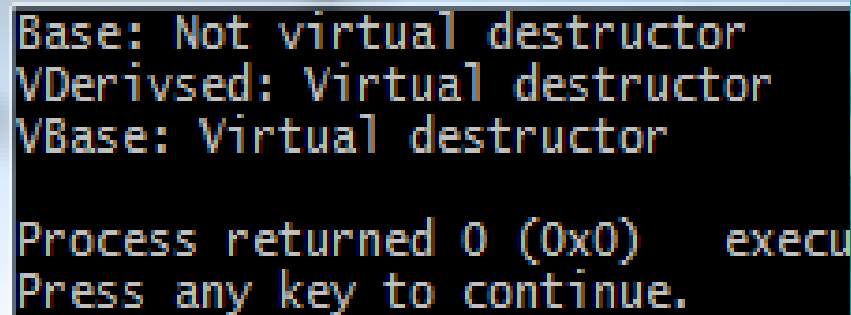
## Приклад 5

```
#include <iostream>
using namespace std;
class Base
{public:
    ~Base()
    {cout<<"Base: Not virtual destructor"<<endl; }
};
class Derived: public Base
{public:
    ~Derived()
    {cout<<"Derivsed: Not virtual destructor"
    <<endl; }
};
class VBase
{public:
    virtual ~VBase()
    {cout<<"VBase: Virtual destructor"<<endl; }
};
```



```
class VDerived: public VBase
{public:
    ~VDerived()
    {cout<<"VDerivsed: Virtual destructor"<<endl;
    }
};
```

```
int main()
{Base *bp=new Derived;
 delete bp;
 VBase *vbp=new VDerived;
 delete vbp;
 return 0;
}
```



```
Base: Not virtual destructor
VDerivsed: Virtual destructor
VBase: Virtual destructor

Process returned 0 (0x0)   execu
Press any key to continue.
```

Деструктор може бути оголошений як  
**суто віртуальний:**

***virtual ~VBase() = 0;***

Клас, в якому визначено **суто віртуальний деструктор**, є **абстрактним**, і створювати об'єкти цього класу заборонено.

Деструктори не успадковуються.

При оголошенні чисто віртуального деструктора потрібно написати і його визначення в похідних класах

# Віртуальне успадкування

## Множинне успадкування і проблема ромба

**class A** // Базовий клас

```
{protected:
```

```
    int x;
```

```
public:
```

```
    A(int xx=0)    { x=xx; }
```

```
};
```

**class B : virtual public A**

```
{...           //віртуальне успадкування
```

```
};
```

**class C : virtual public A**

```
{...           //віртуальне успадкування
```

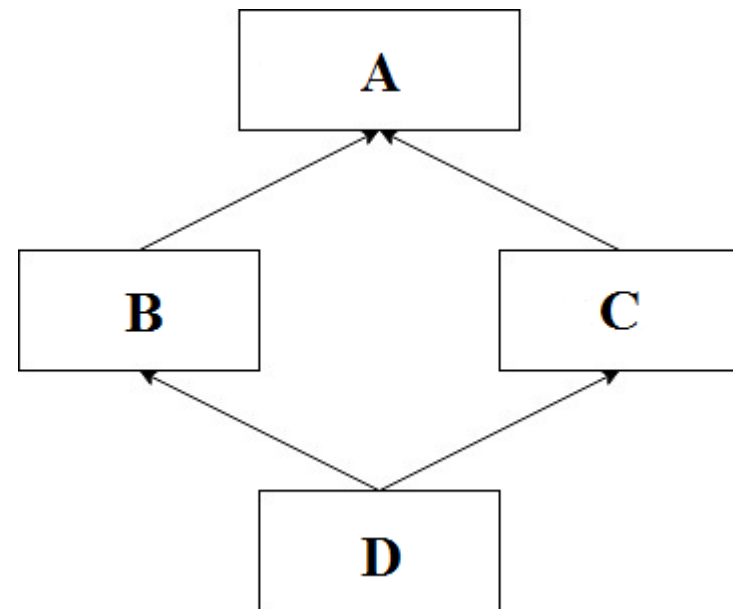
```
};
```

**class D : public B, public C** //завдяки віртуального успадкування в клас D буде додана лише одна копія класу A

```
{    void showX()
```

```
    {cout<<"x="<<x<<endl;
```

```
};
```



Дякую за увагу!