

# ТЕМА: Віртуальні методи і поліморфізм.

## Деякі проблеми перекритих методів

Розглянемо поведінку батьківського Stat і дочірнього CharStat класів щодо методів CreateFromEven() і Print(). Нагадаємо зміст цих методів:

```
Stat * Stat::CreateFromEven()  
// створити об'єкт з елементів парних номерів  
{ // в батьківському класі  
    Stat * p = new Stat( (size+1)/2, 0);  
        // новий динамічний об'єкт  
    // . . . . .  
    return p; // результат типу Stat  
}  
  
CharStat * CreateFromEven()  
// новий об'єкт з елементів парних номерів  
{ // в дочірньому класі  
    CharStat * p = new CharStat( (size+1)/2, 0, key);  
        // новий динамічний об'єкт  
    // . . . . .  
    return p; // результат типу CharStat  
}
```



```
void Stat::Print() // друкувати поточні значення на екрані
{
    cout << "Current values:\n";
    // . . . . .
}

void Print() // для дочірнього класу
{
    Stat::Print(); // метод Print() батьківського класу
    cout << "Code= " << key << endl; // додаткове поле
}
```

Обидва методи були перекриті в дочірньому класі, щоб врахувати нове поле `char key` – характеристику масиву.

Припустимо, що є такі визначення об'єктів і виклики методів:

```
Stat base = Stat(6,-2);  
CharStat derived = CharStat(5,18,'D');  
base.Repl(0,5); base.Repl(1,7); base.Repl(4,-1);  
base.Print(); // Print() класу Stat  
derived.Repl(0,91); derived.Repl(1,100); derived.Repl(2,91);  
derived.Print(); // Print() класу CharStat  
Stat * Frombase = base.CreateFromEven(); // CreateFromEven() Stat  
Frombase->Print(); // Print() класу Stat  
CharStat * Fromderived = derived.CreateFromEven(); // CharStat  
Fromderived->Print(); // Print() класу CharStat
```

```
Current values:  
5 7 -2 -2 -1 -2  
ave=0 max=7  
Current values:  
91 100 91 18 18  
ave=63 max=100  
Code= D  
Current values:  
5 -2 -1  
ave=0 max=5  
Current values:  
91 91 18  
ave=66 max=91  
Code= D
```

Аналіз надрукованих результатів показує, що вони є правильними. Правильність зберігається, поки є виклики методів статичних об'єктів, і поки тип вказівника співпадає з типом динамічного об'єкта. Виклики методів через вказівник виконують за типом самого вказівника.


Припустимо, що є потреба працювати з батьківським і дочірнім об'єктом через єдиний вказівник. За правилом присвоєння об'єктів, це можливо за вказівником на батьківський клас (дочірній можна присвоїти батьківському, навпаки – ні):

```
Stat base = Stat(6,-2);  
CharStat derived = CharStat(5,18,'D');  
Stat * p; // може вказувати на батьківський і дочірній  
p = &base; p->Print();  
p = &derived; p->Print(); // метод дочірнього класу ?
```

```
Current values:  
-2  -2  -2  -2  -2  -2  
ave=-2  max=-2  
Current values:  
18  18  18  18  18  
ave=18  max=18
```



Як видно за отриманими результатами, в обидвох випадках викликається метод Print() батьківського класу, бо немає літери характеристики масиву. Але ж в другому випадку треба виконати метод дочірнього класу.

 Правило. Виклики методів через вказівник завжди виконують за типом самого вказівника, якщо не будувати спеціальних правил.

Такі спеціальні правила існують, їх називають «віртуальні методи».



## Віртуальні методи і поліморфізм

Означення. Віртуальний метод – це такий, виклик якого здійснюється за фактичним типом об'єкта, а не за типом вказівника.

Щоб створити віртуальний метод, достатньо записати ключове слово `virtual` один раз перед оголошенням метода в батьківському класі:

```
class Stat // визначення класу
{
    // . . . . .
    virtual void Print();
    virtual void Print(char * filename);
    virtual Stat * CreateFromEven();
    // . . . . .
};
```

Якщо дочірній клас перекриває віртуальний метод, то його віртуальність зберігається. У випадку будови наступного нисхідного класу метод знову буде віртуальним. Іншими словами, відмінити віртуальність неможливо.

Отже, тепер той самий оператор `p->Print()`; буде виконаний по різному відповідно до типу фактичного об'єкта. Таку поведінку об'єктів і методів називають поліморфізмом.

Можна сказати ще й так: віртуальний метод в базовому класі визначає *інтерфейс функції*, тобто спосіб виклику, а перекритий метод визначає зміст для конкретного класу.

Отже, при найпершому проектуванні батьківського класу кандидатами на віртуальні методи можуть бути всі, які потенційно можуть бути перекриті в дочірніх класах.

### Загальні зауваження.

1) Конструктори не можуть бути віртуальними, вони будують об'єкт за прямим посиланням на клас.

2) Деструктори можуть бути віртуальними.

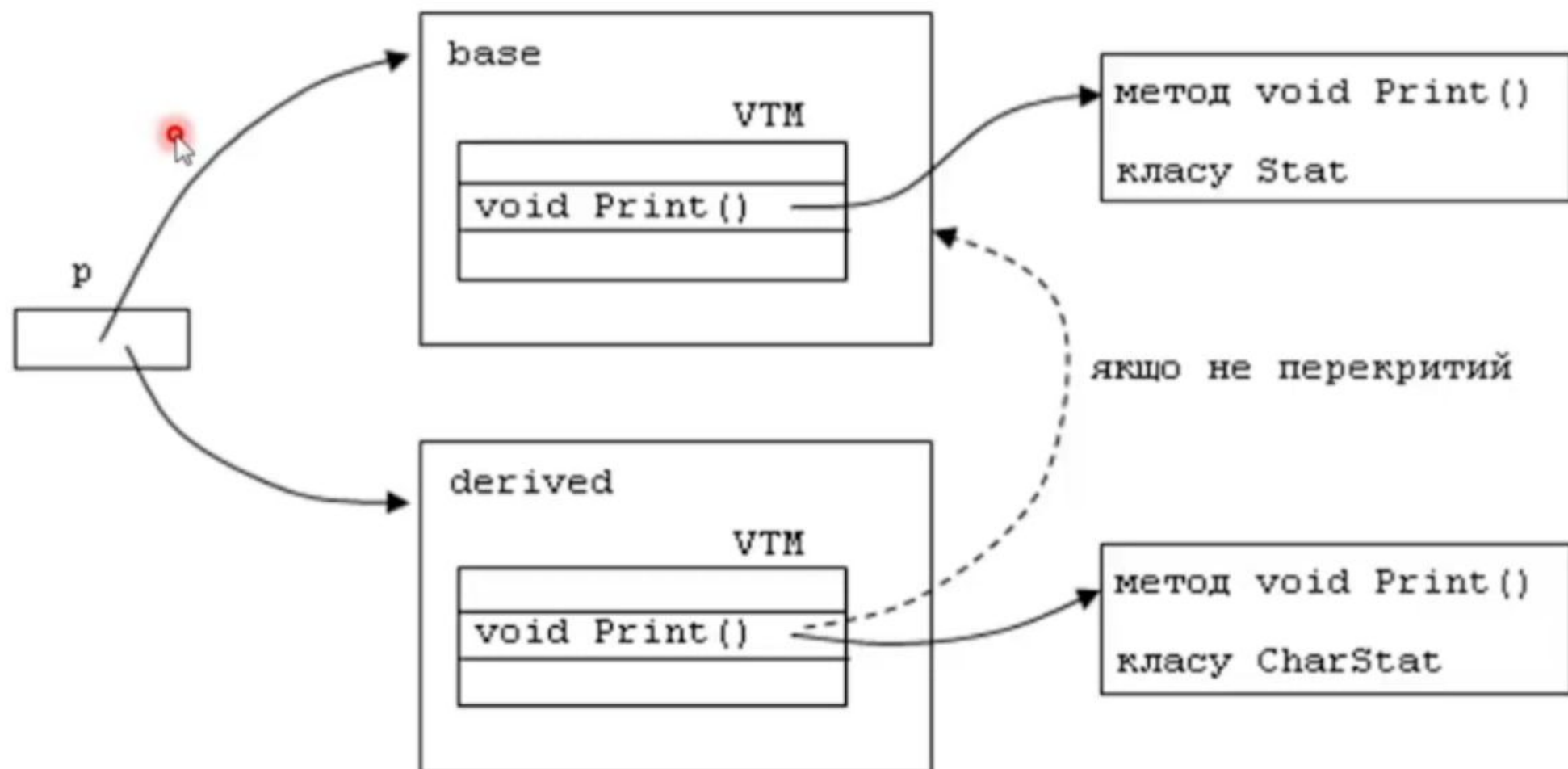
3) Перевантажені операції можуть бути віртуальними.

4) Дружні функції не можуть бути віртуальними, бо вони не є членами класу.

5) Віртуальний метод перекривати не обов'язково. В цьому разі викликається метод батьківського класу.

Вибір потрібного варіанта віртуального метода програма виконує вже під час виконання. Для цього використовують так звану таблицю віртуальних методів VTM, яку будує компілятор на основі визначень класу.

Схему вибору можна спрощено показати так:





## Різниця між перекриттям і перевантаженням віртуальних методів


Віртуальний метод похідного класу буде перекритий і збережений як віртуальний, якщо має таку саму сигнатуру, як в базовому класі. В цьому разі працює схема виклику віртуальних методів, описана вище.

Проте, віртуальний метод можна не лише перекрити, а й перевантажити, тобто визначити за новою сигнатурою. Наприклад, для дочірнього класу визначимо додатково метод `Print(int a, int b)` друкування частини значень масиву, які належать інтервалу `[a,b]`:

```
void Print(int a, int b)
// перевантажений метод (додатковий, не віртуальний)
{
    // . . . . .
}
```

Отже, в дочірньому класі маємо в цілому три методи друкування:

```
void CharStat::Print()    // перекритий віртуальний
void CharStat::Print(char * filename) // перекритий віртуальний
void CharStat::Print(int a, int b)
                        // перевантажений не віртуальний
```

Тепер можна використати всі три методи для об'єктів CharStat, але для об'єктів Stat – лише перекриті віртуальні: 

```
Stat base = Stat(6,-2);
CharStat derived = CharStat(5,18,'D');
derived.Print(5,20); // Print() класу CharStat
base.Print(5,20);
    // помилка - немає Stat::Print з відповідною сигнатурою
Stat * p; // може вказувати на батьківський і дочірній
p = &base; p->Print(5,20);
    // помилка - немає Stat::Print з відповідною сигнатурою
p = &derived; p->Print(5,20);
    // помилка - немає Stat::Print з відповідною сигнатурою
CharStat * d = &derived; d->Print(5,20); // так правильно
```

Зауважимо, що новий метод Print(int a, int b) можна оголосити як віртуальний:

```
virtual void CharStat::Print(int a, int b) { . . . }
```

тоді він стає таким, починаючи від класу CharStat.



## Чисто віртуальні функції і абстрактні класи

Віртуальну функцію треба оголошувати в батьківському класі. При цьому може бути ситуація, коли батьківський клас *не має достатньо інформації* про зміст такої функції, але відомо, що вона буде потрібна. Крім того, в деяких ситуаціях треба *гарантувати*, що віртуальна функція *буде перекрита* у всіх дочірніх класах.

Для таких ситуацій в мові C++ передбачені чисто віртуальні функції.

Означення. Чисто віртуальна функція – це віртуальна функція, яка не має визначення в базовому класі. Для її оголошення використовують таку конструкцію:

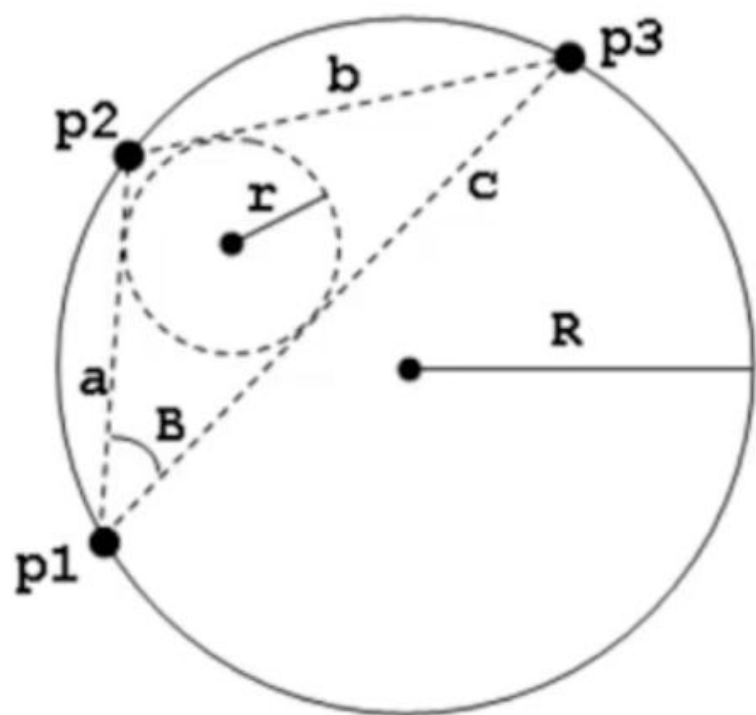
```
virtual тип ім'я ( список_параметрів ) = 0 ;
```

Оскільки функція не має визначення, її обов'язково треба перекрити в кожному похідному класі, інакше буде помилка компіляції.

*Задача. Відомо координати трьох точок площини, які належать фігурі. Вид фігури наперед не відомий. Для трикутника це є координати трьох вершин, а для кола – точки, які належать колу. Можуть бути інші фігури. Потрібно обчислювати суму площ фігур і порівнювати периметри. Побудувати класи для опрацювання геометричних фігур на площині.*

Оскільки вид фігури не відомий, то невідома наперед формула площі і периметра. Отже, ці формули є кандидатами на чисто віртуальну функцію. Маючи прототип чисто віртуальної функції, можна будувати інші методи класу.

Для випадку трьох точок, які належать колу, рисунок і обчислення такі:



p1, p2, p3 – задані точки  
a, b, c – сторони трикутника на колі  
A, B, C – протилежні кути трикутника  
r – радіус вписаного кола в трикутник  
R – радіус описаного кола  
 $p = (a+b+c) / 2$  півпериметр

$$r = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}}$$

$$\operatorname{tg} \frac{B}{2} = \frac{r}{p-b}, \quad B = 2 \operatorname{arctg} \frac{r}{p-b}$$

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R \quad (\text{теорема синусів})$$

Означення. Клас, який має хоча б одну чисто віртуальну функцію, називається абстрактним.

Об'єкти абстрактних класів створити неможливо. Абстрактні класи – це основа для похідних класів.



## Програмна реалізація задачі

```
#include <iostream>
#include <fstream>
#define _USE_MATH_DEFINES    // ця директива потрібна перед math.h
#include <math.h>
using namespace std;

// визначення допоміжної структури координат точки
struct Point {    // в мові C++ структура є різновидом класу
    float x,y;
    Point(float tx, float ty) { x=tx; y=ty; }    // конструктор структури
    Point() { x=0.0; y=0.0; }    // конструктор без параметрів
};    // struct Point

// абстрактний клас
class FlatFigures    // довільна фігура
{
protected:
    Point p1,p2,p3;    // координати точок площини
public:
    FlatFigures(Point s1, Point s2, Point s3) { p1=s1; p2=s2; p3=s3; }
    virtual float area() = 0;    // чисто вірт. функція: обчислити площу
    virtual float perimeter() = 0;    // чисто вірт. функція периметр
    bool operator< ( FlatFigures & ob )
    { return this->perimeter() < ob.perimeter(); }
};    // class FlatFigures
```

```

// клас для трикутника (координати трьох вершин)
class Triangle : public FlatFigures
{
protected:
    float a,b,c; // довжини сторін трикутника
public:
    Triangle(Point s1,Point s2,Point s3) : FlatFigures(s1,s2,s3) // констр.
    { // додатково обчислити довжини сторін
        a = sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y)*(p1.y-p2.y));
        b = sqrt((p2.x-p3.x)*(p2.x-p3.x) + (p2.y-p3.y)*(p2.y-p3.y));
        c = sqrt((p1.x-p3.x)*(p1.x-p3.x) + (p1.y-p3.y)*(p1.y-p3.y));
    }

    float perimeter() { return a+b+c; } // перекрита чисто вірт. функція

    float area() // перекрита чисто віртуальна функція
    {
        float halfp = (a+b+c) / 2; // півпериметр
        return sqrt(halfp*(halfp-a)*(halfp-b)*(halfp-c)); // площа трик.
    }

    /*
void Print() // для тестування
{   cout << a << '\t' << b << '\t' << c << '\t'
    << perimeter() << '\t' << area() << endl;
}
*/
} ; // class Triangle

```

```

// клас для кола (три точки належать колу)
class Circle : public FlatFigures
{
protected:
    float a,b,c; // довжини сторін трикутника на колі
    float R; // радіус описаного кола
public:
    Circle(Point s1, Point s2, Point s3) : FlatFigures(s1,s2,s3) // констр.
    { // обчислення радіуса описаного кола: див. рисунок в тексті
        // довжини сторін трикутника на колі
        a = sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y)*(p1.y-p2.y));
        b = sqrt((p2.x-p3.x)*(p2.x-p3.x) + (p2.y-p3.y)*(p2.y-p3.y));
        c = sqrt((p1.x-p3.x)*(p1.x-p3.x) + (p1.y-p3.y)*(p1.y-p3.y));
        float p = (a+b+c) / 2; // півпериметр трикутника
        float r = sqrt((p-a)*(p-b)*(p-c)/p); // радіус вписаного кола
        float B = 2 * atan(r/(p-b)); // кут B
        R = b / sin(B) / 2;
    }

    float perimeter() { return 2 * M_PI * R; }
                        // перекрита чисто вірт. функція

    float area() { return M_PI * R * R ; } // перекрита чисто вірт. Функ.

    float GetR() { return R; } // який радіус
} ; // class Circle

```



```

void main()
{
    /**
    cout << "for Triangle:\n";
    Triangle t1 = Triangle(Point(0.0,0.0), Point(0.0,6.0), Point(6.0,0.0));
        // (x,y) трьох точок
    Triangle t2 = Triangle(Point(1.0,1.0), Point(1.0,5.0), Point(6.0,6.0));
    // t1.Print(); t2.Print(); // перевірка правильності формул
    cout << t1.area() << '\t' << t2.area() << '\t' << t1.area()+t2.area()
        << endl;
    cout << boolalpha << (t1<t2) << '\t' << (t2<t1) << endl;
    cout << t1.perimeter() << '\t' << t2.perimeter() << endl;

    cout << "for Circle:\n";
    Circle c1 = Circle(Point(0.0,0.0), Point(0.0,6.0), Point(6.0,0.0));
        // (x,y) трьох точок
    Circle c2 = Circle(Point(1.0,1.0), Point(1.0,5.0), Point(6.0,6.0));
    cout << "radius: " << c1.GetR() << '\t' << c2.GetR() << endl;
    cout << c1.area() << '\t' << c2.area() << '\t'
        << c1.area()+c2.area() << endl;
    cout << boolalpha << (c1<c2) << '\t' << (c2<c1) << endl;
    cout << c1.perimeter() << '\t' << c2.perimeter() << endl;
    // */
    system("pause");
}

```



Результати:

```
for Triangle:  
18      10      28  
false   true  
20.4853 16.1701  
for Circle:  
radius: 4.24264 3.60555  
56.5487 40.8407 97.3894  
false   true  
26.6573 22.6543
```

