

Лекція 010

Об'єктно орієнтовне програмування (ООП)

ООП на Cі++

1. Абстракція та інкапсуляція
2. Вміст класу
3. Створення класу
4. Ініціалізація класу
5. Специфікатори доступу public/private
6. Конструктори та деструктори
7. Дружні функції
8. Статичні члени та методи

Клас

Клас – це користувацький *тип даних*:

Члени класу:

Тип1 поле1;

Тип2 поле2;

Методи класу:

Тип1 метод1 (аргументи1);

Тип2 метод2 (аргументи2);

Створення класу на C++

```
// клас з загальнодоступним
// доступом
struct Point2D {
    // public: // члени класу
    double x; /* 2 поля x, y типу
double */
    double y;
    //методи класу
    void show() { // метод
        cout<<"P("<<x<<" "<<y<<"")"
            <<endl;
    }
}; // завершення декларації
```

```
/* клас з встановленим
загальнодоступним
доступом */
class Point2D {
    public: /* вказуємо
загальнодоступність */
    // члени класу:
    double x;
    double y;
    //методи класу
    void show() {
        cout<<"P("<<x<<" "<<y<<"")"
            <<endl;
    }
}; // завершення декларації
```

Використання класу

```
int main(int argc, char **argv){
    Point2D p1; // визначили неініціалізований об'єкт Point2D
    Point2D p2 {1,2}; // ініціалізований об'єкт Point2D
    Point2D* p3 = new Point2D; // визначили вказівник на об'єкт Point2D
    p1.x = 11; // визначили поле x об'єкту p1 класу Point2D
    p1.y = 22; // визначили поле y об'єкту p1 класу Point2D
    (*p3).x=44; // визначили поле x через адресу об'єкту p3 класу Point2D
    p3->y = 55; // визначили поле y через адресу об'єкту p3 класу Point2D
    p3->show(); // викликали метод show() - P(44,55)
    p2.show(); // викликали метод show() - P(1,2)
    p1.show(); // викликали метод show() - P(11,22)
    delete p3; // не забули вивільнити пам'ять
}
```

Створення класів

// C++ клас точка

```
class Point{ // декларували клас Point
public: // публічний доступ
double x,y; // 2 члени x,y
Point(double x1, double y1){// конструктор
    x =x1; y=y1;
}
void show(){ // метод 1: показати
cout<<"P("<<x<<". "<<y<<")";
}
Point add (Point other){ /* метод 2: додати
вектор */
    return Point(x+other.x, y+other.y);
}
double abs(){ //метод 3: модуль
    return sqrt(x*x+y*y);
}
};
****
```

```
Point P1(1,1); // Створили об'єкт Point
P1.show(); // Викликали метод P1.show()
```

Python

клас точка

```
class Point: # декларували клас Point
```

```
def __init__(self, x, y): # конструктор
    self.x = x
    self.y = y
```

```
def show(self): # метод 1: показати
    print ("P(",self.x,".",self.y,")")
```

```
def add(self, other): # метод 2: додати вектор
    return Point(self.x+other.x, self.y+other.y)
```

```
def abs(self): # метод 3: модуль
    return sqrt(self.x**2 + self.y**2 )
```

```
P1 =Point(1,1) # Створили об'єкт Point
P1.show() # Викликали метод P1.show();
```

```
//клас точка Point.h
#ifndef __POINT_H__
#define __POINT_H__
class Point{
public:
    double x;
    double y;

    void show();
    Point add (Point other);
    double abs();
};
#endif /* end of __POINT_H__ */
```

```
// Project_test.cpp
int main(){
    Point P1;
    P1.x =1; P1.y =2;
    P1.show();
}
```

```
// файл Point.cpp
```

```
void Point::show(){
    cout<<"P("<<x<<."<<y<<")";
}

Point Point::add (Point other){
    return Point(x+other.x, y+other.y);
}

double Point::abs(){
    return sqrt(x*x+y*y);
}
```

Інкапсуляція

Рівні доступу:

public : члени та методи доступні з інших класів та функцій

private: члени та методи доступні лише в методах даного класу

protected: члени та методи доступні лише в методах даного класу та в класах-наслідниках



Приклад класу з інкапсуляцією

```
class TimeH{ // клас Час
    private: // приватні члени
        int hours;
        int minutes;
        //приватний метод
        int getTotalMinutes() {
            return hours*60 + minutes;
        }
    public: // публічні методи та члени
        void show() { // показати клас
            cout<<"H:"<<hours<<":"<<minutes<<" ";
        }
        void setHours(int x) { // встановити час
            if (0<=x && x<=23) {
                hours = x;
            }
        }
        bool setTime(int h, int m); // встановити час
    }
```

```
/* метод bool setTime(int h, int m);
   можна встановити за межами
   декларації (навіть в іншому файлі)
   */
bool TimeH::setTime(int h, int m){
    if (0>h || h>23 || 0>m || m>59) {
        return false;
    }
    hours = h;
    minutes = m;
    return true;
}
```

Приклад класу з інкапсуляцією: використання

```
TimeH q1; // створили об'єкт
TimeH* q4 = new TimeH; // створили вказівник та виділили пам'ять
//q1.hours = 12; q1.minutes=25; // error: int TimeH::hours' is private within this context
//q4->hours = 23; q4->minutes = 33; // error: int TimeH::hours' is private
q1.setHours(12); // встановили години
q1.setMinutes(25); // встановили мінути
(*q4).setHours(23); // встановили години через доступ до вказівника
q4->setMinutes(33); // встановили мінути через доступ до вказівника(syntax sugar)
q1.show(); // викликали метод show
q4->show(); // викликали show
//q1.getTotalMinutes(); //error: getTotalMinutes() is private within this context
int h,m;
cin>>h>>m; // ввели час
if(q1.setTime(h,m)) {    q1.show(); } // показали якщо клас гарно ініціалізували
else {    cout<<"Uncorrect initialization"; } // та якщо ні вивели попередження
```

Спеціальні методи класу (методи за замовченням)

- 1) Конструктор
- 2) Конструктор копіювання
- 3) Деструктор

```
class NameOfTheClass{  
    **** // приватні члени та методи  
    ****  
  
    public: // публічні члени та методи  
  
    NameOfTheClass(){} // конструктор за замовченням  
                        // приймає 0 аргументів — створює неініціалізований клас  
  
    NameOfTheClass(const NameOfTheClass& x){} /* конструктор копіювання - створює  
копію екземпляру класу */  
  
    ~NameOfTheClass(){} // деструктор — знищує об'єкт класу  
  
};
```

Конструктор класу - це спеціальний метод класу для **створення нових об'єктів** цього класу
Конструктор — там сама *назва*, що і у *класу*. *Немає типу повернення*.

```
class ClassExample{
    int member1;
    char member2;
    double member3[10];
public:
    ClassExample(){} // 1 : варіант за замовченням - пустий об'єкт
    // 2 : встановлює всі члени в 0
    ClassExample(){ member1=0; member2=0;
                    for (int i=0;i<10;++i)member3[i]=0; }
    //3 : встановлює member1, member2 заданими значеннями
    ClassExample( int x, char y ){ member1= x; member2= y; }
    // 4 : встановлює всі члени
    ClassExample( int x, char y, double* z; size_t n ){ member1= x; member2= y;
                    for (int i=0;i<min(10, n);++i) member3[i]=0; }
    // 5 : встановлює всі члени – альтернативний синтаксис
    ClassExample( int x, char y ): member1(x), member2(y) { for (int i=0;i<10;++i)member3[i]=0; }
};
```

Конструктори

```
class TimeH{
    // private:
    int hours;  int minutes;
    int getTotalMinutes() {      return hours*60 + minutes;  }
    public:
    // Constructor(s)
    //1
    //TimeH(int h, int m){ setTime(h,m);}
    //2
    //TimeH():hours(0),minutes(0){}
    //3
    TimeH(int h=0, int m=0) {      setTime(h,m);  }
    ****
    //4    TimeH(int m=0);
};
```

```
TimeH::TimeH(int m) {  h = m / 60 ; m %=60; }
```

Q. Чи може бути конструктор private? protected?

Слайд 10:
результат?

Конструктор, Конструктор(копія), деструктор

```
class Sample2{
public:
    size_t member1;
    float * member3;
    Sample2(): member1(0),
member3(nullptr){}

    Sample2(size_t n): member1(n){ member3 =
new float[n]; }

    Sample2(const Sample2& x){
        member1 = x.member1;
        member2 = new float[member1];
        strcpy
(member2,x.member2,sizeof(float)*member1);
    }
    ~Sample2() {delete[] member2;}
};
```

```
int main(){

    Sample2 s1;
    Sample2 s2[2];

    Sample2 s3(3);
    s3.member2={1,2,3};

    Sample2 s4(s3);
}
```

Приклад

```
/*
 * Class Polynome
 */
class Poly{
    unsigned n; ///< size of Polynome
    double* a; ///< array of Polynome coefficients
public:
    // default constructor: create empty polynome
    Poly();
    /* constructor: create polynome of given size
     * @param unsigned n - size of polynome
     */
    Poly(unsigned n);
    /* constructor: create polynome with given array
     *  $P(x) = a_0 * x^{n-1} + a_1 * x^{n-2} + \dots + a_{n-1}$ 
     * @param unsigned n - size of polynome
     * @param double* ptr_a - array of coefficients  $a_0, a_1, \dots, a_n$ 
     */
    Poly(unsigned n, double* ptr_a);
```

Приклад

```
// destructor: free memory
~Poly();
/* copy-constructor: create another polynome from the given
 * @param Poly p - polynome
 */
Poly(const Poly& p);
/* Calculate the value of the polynome
 * @param double x - argument of polynome
 */
double value(double x);
// display polynome
void show();

};
```


Приклад

```
#include <cstring>    // memmove, memcpy
#include "Poly.h"
```

```
Poly::Poly(){
    n=0; a = nullptr;
};
```

```
Poly::Poly(unsigned m){
    n = m;
    a = new double[n];
}
```

```
Poly::Poly(unsigned n, double* ptr_a){
    this->n =n;
    a = new double[n];
    memmove(a, ptr_a, sizeof(*a)*n);
}
```

```
Poly::~~Poly(){
    delete[] a;
}
```

Приклад

```
Poly::Poly(const Poly& p){
    n = p.n;
    a = new double[n];
    memmove(a, p.a, n*sizeof(*a));
}
double Poly::value(double x){
    if (n==0) {std::cerr<<"Empty Polynome"; return 0;};
    double res = a[0];
    for(unsigned i=1;i<n;++i){
        res *= x;
        res += a[i];
    }
    return res;
}
void Poly::show(){
    std::clog<<"P{";
    for(unsigned i=0;i<n;++i){
        std::clog<<a[i]<<" ";
    }
    std::clog<<"}\n";
}
```

Приклад

```
double a[] = {1,2,3};  
Poly polin(3,a);  
cout<<"P="<<polin.value(1);  
  
Poly polin2(polin);  
polin2.show();  
cout<<"P2="<<polin2.value(2);  
  
Poly polin3{polin};  
polin3.show();  
cout<<"P3="<<polin3.value(3);
```

```
Poly pol[10];  
  
pol[0] = polin;  
pol[0].show();  
cout<<"P4="<<pol[0].value(4);  
  
Poly* pol2 = new Poly[2];  
pol2[0] = pol2[1] = polin2;  
pol2[1].show();  
cout<<"P5="<<pol2[1].value(5);  
  
delete[] pol2;
```

Дружній клас

```
class A {  
private:  
    int a;  
public:  
    A() { a=0; }  
    friend class B;    // Friend Class  
};
```

```
class B {  
private:  
    int b;  
public:  
    void showA(A& x) {  
        // Оскільки B дружній к A, то  
        // є доступ до приватних в A  
        std::cout << "A::a=" << x.a;  
    }  
};
```

```
A a;  
B b;  
b.showA(a);
```

Дружній метод класу

```
class B1;
```

```
class A1{  
public:  
    void showB(B1& );  
};
```

```
class B1{  
private:  
    int b;  
public:  
    B1() { b = 0; }  
    friend void A1::showB(B1& x); // Дружній метод  
};
```

```
void A1::showB(B1 &x){  
    // Так як show() дружній B, він  
    // має доступ до B  
    std::cout << "B::b = " << x.b;  
}
```

```
A1 a1;  
B1 x1;  
a1.showB(x);
```

Дружня функція

```
class A2{  
    int a;  
public:  
    A2() {a = 0;}  
    friend void showA(A2&); // друга функція  
};
```

```
void showA(A2& x) {  
    // Оскільки showA() друга функція  
    // вона має доступ до A  
    std::cout << "A::a=" << x.a;  
}
```

```
A2 a2;  
showA(a2);
```

```
class Box {
    public:    // Constructor definition
        Box(double l = 2.0, double b = 2.0, double h = 2.0):
            length (l), breadth(b), height(h){};

        double Volume() {
            return length * breadth * height;
        }
        int compare(Box box) {
            return this->Volume() > box.Volume();
        }
    private:
        double length;    // Length of a box
        double breadth;    // Breadth of a box
        double height;    // Height of a box
};
```

```
int main(void) {
    Box Box1(3.3, 1.2, 1.5); // Визначили box1
    Box Box2(8.5, 6.0, 2.0); // Визначили box2
    Box *ptrBox;             // Визначили вказівник на клас
    // Взяли значення класу по адресу
    ptrBox = &Box1;
    // Викликаємо метод за допомогою оператора доступу
    cout << "Volume of Box1: " << ptrBox->Volume() << endl;

    // Беремо інший об'єкт
    ptrBox = &Box2;

    // Викликаємо метод за допомогою оператора доступу
    cout << "Volume of Box2: " << ptrBox->Volume() << endl;


    return 0;
}
```



```
class Monomial{
private:
    int deg;
    double aval;
    static int count;
public:
    Monomial();
    ~Monomial();
    Monomial(int n, double a): deg(n),aval(a){ count++;};

    static int show_count(){
        return count;
    }
    static double power(double x, unsigned n);

    static double pi() { return 3.14159265; }
```



```
void setDegree(int m){  
    deg = m;  
}
```

```
void setAval(double p){  
    aval = p;  
}
```

```
int getDegree(){  
    return deg;  
}
```

```
double getAval(){  
    return aval;  
}
```

```
double value(double x){  
    return aval*power(x, deg);  
}
```

```
};
```






```
#include "Monomial.h"
```

```
Monomial::Monomial(){  
    count++;  
}
```

```
Monomial::~~Monomial(){  
    count--;  
}
```

```
double Monomial::power(double x, unsigned n){  
    if(n==1) return x;  
  
    double res = power(x, n/2);  
    if(n & 1){  
        return x*res*res;  
    }  
    return res * res;  
}
```



```
Monomial g1(5, 2.56);  
cout<<"Pwr="<<g1.power(2,3);  
cout<<"Pwr="<<Monomial::power(2,3);  
cout<<"Res="<<g1.value(2);  
  
cout<<"Cnt="<<g1.show_count();
```