

Классы и объекты

Объектно-ориентированное программирование

- ▶ Инкапсуляция и сокрытие данных
- ▶ Наследование
- ▶ Полиморфизм

Основа ООП - абстрагирование, создание типов, определенных пользователем

Следствие введения классов – расширение возможностей повторного использования кода

Основные термины

▶ Абстракция

- Отделяет назначение типа (класса) от его реализации. В основу абстракции положено определение операций над объектами данного типа

▶ Инкапсуляция

- Объединяют в единое целое данные (поля) и операции над ними (методы)

▶ Соккрытие информации

- Ограничивает возможности работы с полями и методами. Защищает классы от ошибок. Упрощает работу с классами, позволяет усовершенствовать организацию классов

Основные термины

▶ Наследование

- Позволяет создавать новые классы путем добавление новых полей и методов к существующим классам, а также изменения методов существующих классов

▶ Полиморфизм

- Позволяет выбирать одну из перегруженных операций, в зависимости от объектов к которым она применяется.

Описание класса

```
class ИмяКласса
{
    private:
        //Закрытые элементы класса
        // поля и вспомогательные методы
    public:
        //Открытые элементы
        //прототипы функций или inline реализация
};
```

} файл.h

Реализация функций (методов) класса, не определенных как inline

```
тип ИмяКласса :: имяМетода ( спецификация параметров )
{
    //Тело метода
}
```

} файл.cpp

Использование класса

- ▶ Класс – пользовательский тип данных
- ▶ Переменные типа называются объектами
- ▶ С++ две модели объектов:
 - простая
 - ссылочная
- ▶ Объявление

ИмяКласса имяОбъекта;

*ИмяКласса * имяУказателяНаОбъект;*

Пример

```
class Timer {  
    private:  
        int hour, minute, second;  
    public:  
        void setTime (int h, int m, int s);  
        void tic ();  
        int  getHour();  
        void setHour (int h);  
        void nulled();  
        int  timeInSec();  
        int  period ( Timer x);  
    . . .  
};
```

Пример

```
int Timer::period ( Timer x)
{
    return timeInSec( ) – x.timeInSec();
    . . .
}
```

Использование

```
Timer a, b;
a.setHour(15); // почему нельзя a.hour=15; ???...
b.nulled();
cout << a.period(b);
```


Особые методы

▶ Конструкторы

- не имеет возвращаемого значения
- имя конструктора совпадает с именем класса
- класс может иметь несколько перегруженных конструкторов
- класс всегда имеет 2 конструктора – без параметров и конструктор-копии

▶ Деструкторы

- не имеет возвращаемого значения и параметров
- имя начинается с ~ затем следует имя класса
- всегда один и всегда есть

Пример

```
class Timer {  
    ...  
public:  
    Timer () { hour= minute = second =0;} // inline  
    Timer ( int h, int m=0, int s=0 ) :  
        hour(h), minute(m), second(s) { }  
    Timer (const Timer & t);  
    ~Timer () { }  
};
```

- ▶ ***специальная форма записи конструктора с параметрами – список инициализации***

Timer (int h, int m=0, int s=0) :

hour(h), minute(m), second(s) { }

Когда выполняются конструкторы

- ▶ Без параметров

```
Timer a;
```

```
Timer b = Timer ();
```

```
Timer *c = new Timer;
```

```
Timer d[10];
```

```
Timer *f = new Timer[10];
```

Когда выполняются конструкторы

- ▶ С параметрами

```
Timer lect(14,10, 0);
```

```
Timer b= Timer (14, 10,0);
```

```
Timer *c=new Timer(13,30, 30);
```

```
Timer audit[2] ={Timer(10,15,0), Timer(12, 20,0)};
```

```
Timer a(13);
```

```
Timer b= Timer(12, 20);
```

```
Timer c=15;
```

Когда выполняются конструкторы

- ▶ конструктор копии всегда присутствует
- ▶ если он создается по умолчанию, то выполняет поверхностное копирование (поэлементное копирование). Поверхностное копирование создает копии простых типов и указателей
- ▶ для полного копирования (глубокого), например, массивов, строк, списков, необходимо описывать собственный конструктор копии

Когда выполняются конструкторы

- ▶ конструктор копии

```
// Timer b;
```

```
Timer a(b);
```

```
Timer a=b;
```

```
Timer a= Timer(b);
```

```
Timer *a = new Timer(b);
```

Конструктор копирования вызывается при передаче параметров в функции по значению и при возврате значения из функции

```
Timer myFunc ( Timer x);
```

Когда выполняются деструкторы

- ▶ при освобождении памяти, занимаемой объектом

```
{  
    Timer x;  
    Timer *y = new Timer;  
  
    delete y; // деструктор для y  
  
} // деструктор для x
```


Когда выполняются деструкторы

- ▶ при освобождении памяти, занимаемой объектом

```
Timer myFunc ( Timer x) // конструктор копии
{
    Timer y (x); // конструктор копии
    ....
    return y; // конструктор копии
} // деструктор для y и x
```

Указатель this

- ▶ Каждый метод класса получает один неявный параметр – указатель на тот объект, который вызвал этот метод. Этот указатель имеет имя `this`

```
void Timer ::nulled() {  
    hour = 0;  
    minute = 0;  
    second = 0;  
}
```

```
void Timer ::nulled() {  
    this->hour = 0;  
    this->minute = 0;  
    this->second = 0;  
}
```

Указатель this

```
int Timer::period (Timer x)
{
    return timeInSec( ) – x.timeInSec();
}
```

```
int Timer::period (Timer x)
{
    return this -> timeInSec( ) – x.timeInSec();
}
```

Указатель this

```
Timer Timer::maxTime ( const Timer& x)
{
    if (period(x) < 0) return x;
    else return ?????;
}
```

```
Timer Timer::maxTime ( const Timer& x)
{
    if (period(x) < 0) return x;
    else return *this;
}
```

Функции для доступа к закрытым полям класса

- ▶ Традиционно имеют имена `setXXX` и `getXXX`, где `XXX` – имя соответствующего поля.
- ▶ Отсюда их названия «сеттеры» и «геттеры».

Константы в описании класса

```
class MyClass{
private:
const int N=5; //только int
const int MAX=999;
/* создается для каждого объекта, поэтому не может
использоваться для определения размеров
массива но может использоваться в методах, хотя
это накладно*/
int arr[N]; //ошибка
public:
bool wellFormed(){ return x<MAX; } // правильно
...
};
```

Статические константы

```
class MyClass{  
private:  
static const int N=5;  
static const int MAX=999;  
// одна константа для всех объектов  
int arr1[N];  
public:  
bool wellFormed(){ return x<MAX; }  
  
};
```

```
class MyClass{
int arr1[N];
    public:
static const int N=5;
static const int MAX=999;
// открытые, но принадлежат не объектам, а
    классу
};

cout<< "max value="<<MyClass::MAX<<endl;
```


Статические методы

```
class MyClass{
static const int N=5;
// одна константа для всех объектов
int arr1[N];
public:
    int sizeOfArr(){ return N; }
//независимо от объекта возвращает одинаковые
    значения
};
MyClass x;
cout<<"ArraySize="<<x.sizeOfArr()<<endl;
```

Статические методы

```
class MyClass{
private:
    static const int N=5;
    // одна константа для всех объектов
    int arr1[N];
public:
    static int  sizeOfArr() {return N;}
    //принадлежит классу, а не объектам
    //работает только со статическими элементами класса
};

cout<<"ArraySize="<<MyClass::sizeOfArr()<<endl;
```

Константные методы

- ▶ Не изменяют *this
- ▶ Вернемся к примеру Timer

```
int Timer::period (Timer x) // почему плохо?  
{  
    return timeInSec() - x.timeInSec();  
}
```

Константные методы

- ▶ Пусть нам нужна не копия, а ссылка на параметр

```
int  Timer :: period (const Timer& x)
{
    return timeInSec() –
           x.timeInSec(); //ошибка , может x изменится?
}
```

В результате преобразования теряются
квалификаторы

Константные методы

- ▶ мы знаем, что `timeInSec()` не меняет вызвавший его объект

```
int timeInSec() const ;
```

Композиция классов

- ▶ Классы могут содержать в качестве полей, объекты (указатели на объекты) других классов или структуры данных, содержащие объекты (указатели на объекты) других классов.
- ▶ Такой способ построения сложных классов называется ***композицией***

Примеры

```
class Point {  
    private:  
        int x,y;  
    public:  
    . . .};  
class T {  
    private:  
        Point a, b, c;  
    public :  
    . . .  
}
```

Классы, использующие динамическую память

```
class Array {  
private:  
    int n;  
    int *A;  
public:  
    Array();                //конструктор без параметров  
    Array(int _n, int x=0); //конструктор с параметрами  
    Array(const Array &B); //конструктор копии  
    int max( );            // максимальный элемент  
    void addInt (int x);  
    void addArray (const Array& x);  
    bool equal (const Array& x);  
    ~Array ( );  
};
```


Конструктор без параметров

```
Array :: Array( ) {  
    n=10;  
    A=new int[n]; // выделяем память  
    for (int i=0; i<n; ++i)  
        A[i]=0;  
}
```

Конструктор с параметрами

```
Array::Array(int _n, int x) { // синтаксис!!!  
    n=_n;  
    A=new int[n];  
    for (int i=0; i<n; ++i)  
        A[i]=x;  
}
```

Конструктор копии

```
Array::Array (const Array &x) {  
    n=x.n;  
    A=new int[n];  
    for (int i=0; i<n; ++i)  
        A[i]=x.A[i];  
}
```

При отсутствии, будет использован конструктор копии по умолчанию, выполняющий поверхностное копирование. Память выделена не будет, а скопируется указатель на массив в оригинале.

Дектсруктор

```
Array::~~Array( ){  
    delete[] A;  
}
```

Максимальный элемент

```
int Array::max( ) {  
    int maxi = A[0];  
    for (int i=1; i<n; ++i)  
        if (A[i]>maxi)  
            maxi=A[i];  
    return maxi;  
}
```

Добавить целое

```
void Array::addInt ( int x) {  
    for (int i=0; i<n; ++i)  
        A[i]=A[i]+x;  
}
```

Добавить другой массив

```
void Array::addArray (const Array &x) {  
    if ( n!= x.n ) return;  
    for ( int i=0; i<n; ++i)  
        A[i]=A[i]+x.A[i];  
  
}
```

Сравнить на равенство

```
bool Array::equal (const Array& x){  
    if ( ! this || !(&x)) return false;  
    if (this == &x) return true;  
    if (n!= x.n) return false;  
    for (int i=0; i<n; ++i)  
        if (A[i]!= x.A[i]) return false;  
    return true;  
}
```


Дружественные функции

- ▶ класс разрешает функции, а не методу иметь доступ к закрытым частям класса

```
class Array {  
private:  
    . . .  
public:  
    . . .  
friend void print(const Array& x);  
//хотя бы один параметр – объект класса  
};
```

```
void print (const Array& x){  
    for (int i=0; i<x.n; ++i) cout<< x.A[i]<<"  ";  
    cout<<endl;  
}
```

ВЫЗОВ – обычная функция

```
Array a1;  
print(a1);
```