

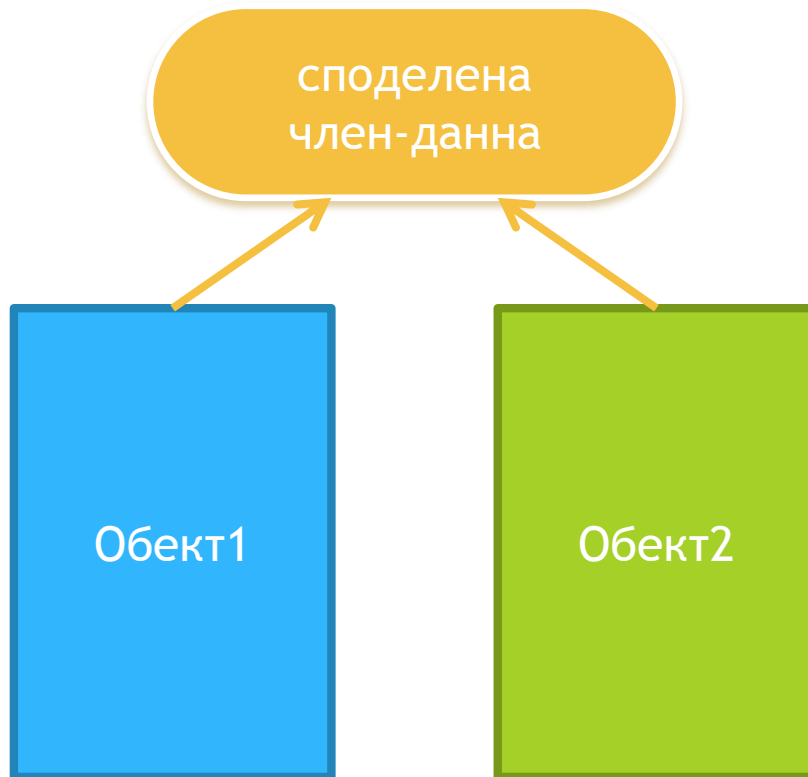
СТАТИЧНИ КОМПОНЕНТИ НА КЛАСОВЕ.

гл.ас., д-р. Нора Ангелова

СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

- Идея

Всички обекти на клас да поделят (използват едновременно) една или повече член-данни на класа.



СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

● Реализация

За да се създадат поделени от всички обекти на клас член-данни, последните се декларират като статични.

Използва се запазената дума `static`.

```
class A {  
    public:  
        static int counter;  
};
```

СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

- Дефиниция

Осъществява се **ИЗВЪН** декларацията на класа.

Преди да бъдат използвани, статичните член-данни трябва задължително да бъдат дефинирани с инициализация.

```
class A {  
    public:  
        static int counter;  
};
```

```
int A::counter = 0;
```

СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

Особености

Заделя се еднократно памет за този вид член-данни и всички обекти на класа имат достъп до тази памет.

```
class A {  
    public:  
        static int counter;  
};
```

```
int A::counter = 0;
```

```
int main() {  
    A obj1, obj2;
```

```
    cout << &obj1.counter << endl;  
    cout << &obj2.counter << endl;
```

```
    return 0;
```

```
}
```

```
00B37138
```

```
00B37138
```

```
Press any key to continue . . .
```

СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

● Особенности

Памет за статичните член-данни се заделя не в програмния стек, а в **областта за статични данни**.

Имат статут на външни променливи, поради което статичните член-данни могат да се използват и самостоятелно, а не само свързано с обектите на класа.

Достъп до статичните член-данни на клас може да се осъществява от всяка външна функция, чрез използване на пълното им име, стига статичните член-данни да са дефинирани в public секция на класа. При този достъп не е нужно статичните член-данни да са свързани с обект на класа.

```
int main() {  
    cout << A::counter << endl;  
  
    return 0;  
}
```

СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

Особености

Ако се промени статична член-данна от един обект, промяната е валидна за всеки друг обект.

```
int main() {  
    A obj1, obj2;  
  
    cout << obj1.counter << " " << obj2.counter << endl;  
    obj1.counter = 5;  
    cout << obj1.counter << " " << obj2.counter << endl;  
  
    return 0;  
}
```



0 0
5 5
Press any key to continue . . . _

СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

- Използване

```
class A {  
    public:  
    static int counter;
```

```
    A() {  
        counter++;  
    }
```

```
    ~A() {  
        counter--;  
    }
```

```
};
```

```
int A::counter = 0;
```


СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

● Използване

```
class A {  
    private:  
        static int counter;  
};
```

```
int A::counter = 0;
```

```
int main() {  
    cout << A::counter << endl;  
  
    return 0;  
}
```

Грешка по време на компилация.
Статичната член-данна counter е
декларирана в private секцията на
класа.

СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

- Област

Статичните член-данни имат за област - тази на класа, в който са дефинирани.

Това е причината достъпът до тях да е чрез оператора ::

СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

- Наследяване

Статичните член-данни се наследяват като се запазва статичността им.

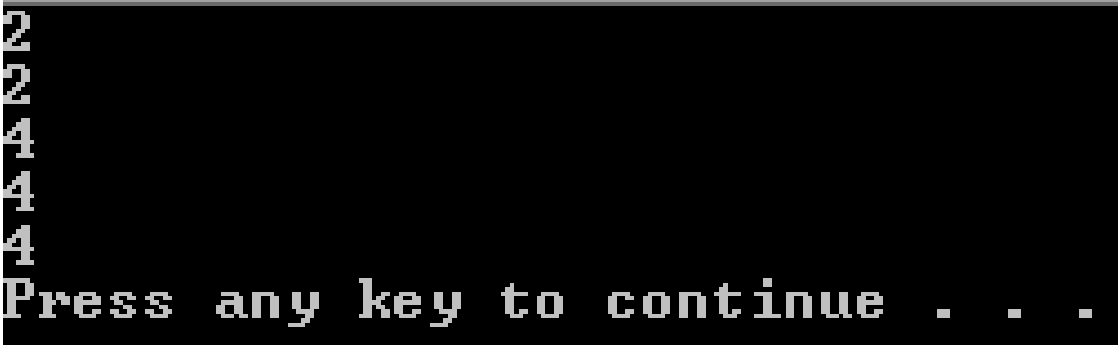
СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

- Наследяване

```
class A {  
    public:  
        A() {  
            counter++;  
        }  
        static int counter;  
};  
int A::counter = 0;
```

```
class B : public A {  
    public:  
        B() {  
            counter++;  
        }  
};
```

```
int main() {  
    A obj1, obj2;  
    cout << obj1.counter << endl << obj2.counter << endl;  
  
    B objB;  
    cout << obj1.counter << endl << obj2.counter << endl << objB.counter << endl;  
    return 0;  
}
```



```
2  
2  
4  
4  
4  
Press any key to continue . . .
```

СТАТИЧНИ ЧЛЕН-ФУНКЦИИ НА КЛАСОВЕ

СТАТИЧНО КОМПОНЕНТИ НА КЛАСОВЕ

- Статични член-функции на класове

Осъществява се чрез ключовата дума `static`, която се поставя пред типа в прототипа на член-функцията.

```
static int staticFunc(int);
```

СТАТИЧНИ ЧЛЕН-ФУНКЦИИ НА КЛАСОВЕ

Особености

Нямат неявен параметър

За тях не е дефиниран указателят `this`.

Затова:

- Статичните член-функции на клас нямат пряк достъп до нестатичните компоненти на обектите, чрез които са били извикани, т.е. в тялото на статичен метод могат да се използват само статични компоненти (член-данни и член-функции). Нестатични компоненти могат да се извикват чрез помощен указател, подобен на указателя `this`.
- Статичните член-функции на класовете могат да бъдат извиквани, както чрез обекти на класа, така и самостоятелно.

СТАТИЧНИ ЧЛЕН-ФУНКЦИИ НА КЛАСОВЕ

- Особености

```
class A {  
    public:  
        int temp;
```

```
    static void staticFunc(int tempData) {  
        temp = tempData;  
    }
```

```
};
```

Грешка

Статичен метод се обръща към
нестатична член-данна.

СТАТИЧНИ ЧЛЕН-ФУНКЦИИ НА КЛАСОВЕ

Особености

Нестатични компоненти могат да се извикват чрез помощен указател, подобен на указателя `this`.

Пример:

```
class A {  
    public:  
        int temp;  
  
        static void staticFunc(A * obj, int tempData) {  
            obj->temp = tempData; Правилна дефиниция  
        }  
};  
  
int main() {  
    A obj;  
    A::f(&obj, 5);  
    obj.f(&obj, 5);  
  
    return 0;  
}
```

СТАТИЧНИ ЧЛЕН-ФУНКЦИИ НА КЛАСОВЕ

● Ограничения

- статичен метод не може да е виртуален;
- статичен метод не може да е член-функция за достъп;
- модификаторът `static` не е основание за различаване на две член-функции с еднакви прототипи.

Статичните методи се наследяват в производните класове като статични.

```

class A {
    public:
        A(int a = 2) {
            n = a;
            x = a+1;
            cout << "A: " << n << ", " << x << endl;
        }

        A(const A& p) {
            n = p.n+1;
            x = p.x+2;
            cout << "A.n: " << n << endl << "A.x: " << x << endl;
        }

        A& operator=(const A& p) {
            if (this!=&p) {
                n = p.n + 2;
                x = p.x + 3;
                cout << "A.n: " << n << endl << "A.x: " << x << endl;
            }
            return *this;
        }

        void print() const { cout << "A::x" << x << "A::n" << n << endl; }
    private:
        double x;
        int n;
};

```

```
class B {
    public:
        B(double b = 3) {
            n = b+1;
            x = b;
            cout << "B: " << n << ", " << x << endl;
        }
        void print() const {
            cout << "B::x" << x << "B::n" << n << endl;
        }
    private:
        double x;
        int n;
};
```

```
class C {
    public:
        C(double b = 4) {
            n = b+2;
            x = b+4;
            cout << "C: " << n << ", " << x << endl;
        }
        C(const C& p) {
            n = p.n + 3;
            x = p.x + 5;
            cout << "C.n: " << n << endl << "C.x: " << x << endl;
        }
        C& operator=(const C& p) {
            if (this != &p) {
                n = p.n+1;
                x = p.x+2;
                cout << "C.n: " << n << endl << "C.x: " << x << endl;
            }
            return *this;
        }
        void print() const {
            cout << "C::x" << x << "C::n" << n << endl;
        }
    private:
        int n;
        double x;
};
```

```
class D: public B, protected C, A {
    public:
        D(int x=1, int y=1, int z=1) : A(x), B(y), C(z) {
            n = z;
            m = x+y;
            cout << "D: " << n << ", " << m << endl;
        }
        void print() const {
            B::print();
            A::print();
            C::print();
            cout << "D::n" << n << "D::m" << m << endl;
        }
    private:
        int n, m;
};
```

```
int main() {  
    D x, y(2), z(1, 2), t(3, 4, 5);  
    D u = t;  
    x = t;  
    t = y;  
  
    return 0;  
}
```

B: 2, 1
C: 3, 5
A: 1, 2
D: 1, 2
B: 2, 1
C: 3, 5
A: 2, 3
D: 1, 3
B: 3, 2
C: 3, 5
A: 1, 2
D: 1, 3
B: 5, 4
C: 7, 9
A: 3, 4
D: 5, 7
C.n: 10
C.x: 14
A.n: 4
A.x: 6
C.n: 8
C.x: 11
A.n: 5
A.x: 7
C.n: 4
C.x: 7
A.n: 4
A.x: 6

```

class base {
public:
    base() {
        cout << "base()\n";
        f2();
    }
    ~base() {
        cout << "~base()\n";
        f1();
    }
    void f() const {
        cout << "f()\n";
        f1();
        base::f2();
        f3();
    }
    virtual void f1() const {
        cout << "base::f1()\n";
    }
protected:
    virtual void f3() const {
        cout << "base::f3() \n";
    }
private:
    virtual void f2() const {
        cout << "base::f2()\n";
    }
};

```

```

class der1 : public base {
public:
    der1() {
        cout << "der1()\n";
        base::f1();
        base::f3();
    }
    void f2() const {
        cout << "der1::f2()\n";
    }
protected:
    void f3() const {
        cout << "der1::f3()\n";
    }
private:
    void f1() const {
        cout << "der1::f1()\n";
    }
};

class der2 : public der1 {
protected:
    void f1() const {
        cout << "der2::f1()\n";
    }
public:
    void f3() const {
        cout << "der2::f3()\n";
    }
private:
    void f2() const {
        cout << "der2::f2()\n";
    }
};

```

```

int main() {
    base b;
    der1 d1;
    der2 d2;
    base *p = &d1;
    der1 *q = &d2;
    p->f1();
p->f2();
p->f3();
q->f1();
    q->f2();
q->f3();
    p = &d2;
    p->f1();
p->f2();
p->f3();
    der1 *r = new der2;
r->f1();
    r->f2();
r->f3();
    p->f();
    q->f();
    r->f();
    delete r;
    return 0;
}

```

```

base()
base::f2()
base()
base::f2()
der1()
base::f1()
base::f3()
base()
base::f2()
der1()
base::f1()
base::f3()
der1::f1()
der2::f2()
der2::f1()
base()
base::f2()
der1()
base::f1()
base::f3()
der2::f2()
f()
der2::f1()
base::f2()
der2::f3()
f()
der2::f1()
base::f2()
der2::f3()
f()
der2::f1()
base::f2()
der2::f3()
~base()
base::f1()

```


КРАЙ