
Лекция 4

Объекты и классы

Понятие класса

Класс - новый элемент программы на языке C++, который объединяет понятия структуры и функции. Класс внутри себя может содержать разнотипные данные (как структура C/C++), а также функции для манипулирования этими данными и их обработки.

Смысл использования класса

Класс служит программной моделью какого-либо понятия из предметной области. Класс позволяет описать **свойства** объекта (через поля структуры) и его **поведение** (через набор функций).

Простейший класс

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

class MyClass
{
public:
    int data1;
    int data2;
    void set(int a1, int a2)
        { data1 = a1; data2 = a2; }
    void show()
        { cout << data1 << " "
              << data2 << endl; }
};
```

```
int main()
{
    MyClass s1, s2;
    s1.set(10, 5);
    s2.data1 = 17;
    s2.data2 = 21;

    s1.show();
    s2.show();
    return 0;
}
```

Объединение данных и функций для работы с ними - основной прием объектно-ориентированной разработки программ.

Класс MyClass

данные

```
int data1;  
int data2;  
...
```

функции

```
void set(int, int);  
void show();  
...
```

Определение класса

```
class имя_класса
{
    спецификатор_доступа_1:
        поле1;
        поле2;
        ...
        метод1;
        метод2;
        ...

    спецификатор_доступа_2:
        поле3;
        метод3;
        ...
};
```

Поля - элементы данных, **методы** - функции для манипулирования данными.

Использование класса

1) Создание объекта класса

```
имя_класса имя_объекта;
```

Примеры:

```
MyClass x,y;           // x,y - объекты  
MyClass *pointer;     // указатель на объект  
MyClass A[10];        // массив объектов
```

2) Доступ к полям данных объекта (только в случае, если доступ открыт)

```
имя_объекта.имя_поля = значение;
```

ИЛИ

```
указатель_на_объект->имя_поля = значение;
```

Примеры:

```
x.data1 = y.data2 + 5;
```

```
pointer->data1 = 20;
```

```
A[3].data2 = 0;
```

3) Вызов метода объекта

(только в случае, если доступ открыт)

```
имя_объекта.метод(аргументы);
```

ИЛИ

```
указатель_на_объект->метод(аргументы);
```

Примеры:

```
int a = x.get ();  
pointer->set(7,2);  
A[7].show();
```


Соккрытие данных (инкапсуляция)

Данные внутри класса (поля) могут быть защищены от несанкционированного доступа. Защита производится с использованием механизмов разграничения доступа.

Спецификаторы доступа:

- **private** (по умолчанию) - данные доступны только внутри этого класса,
- **protected** - данные доступны внутри класса и из его потомков
- **public** - данные доступны внутри класса, из его потомков, и извне.

Инкапсуляция данных (пример)

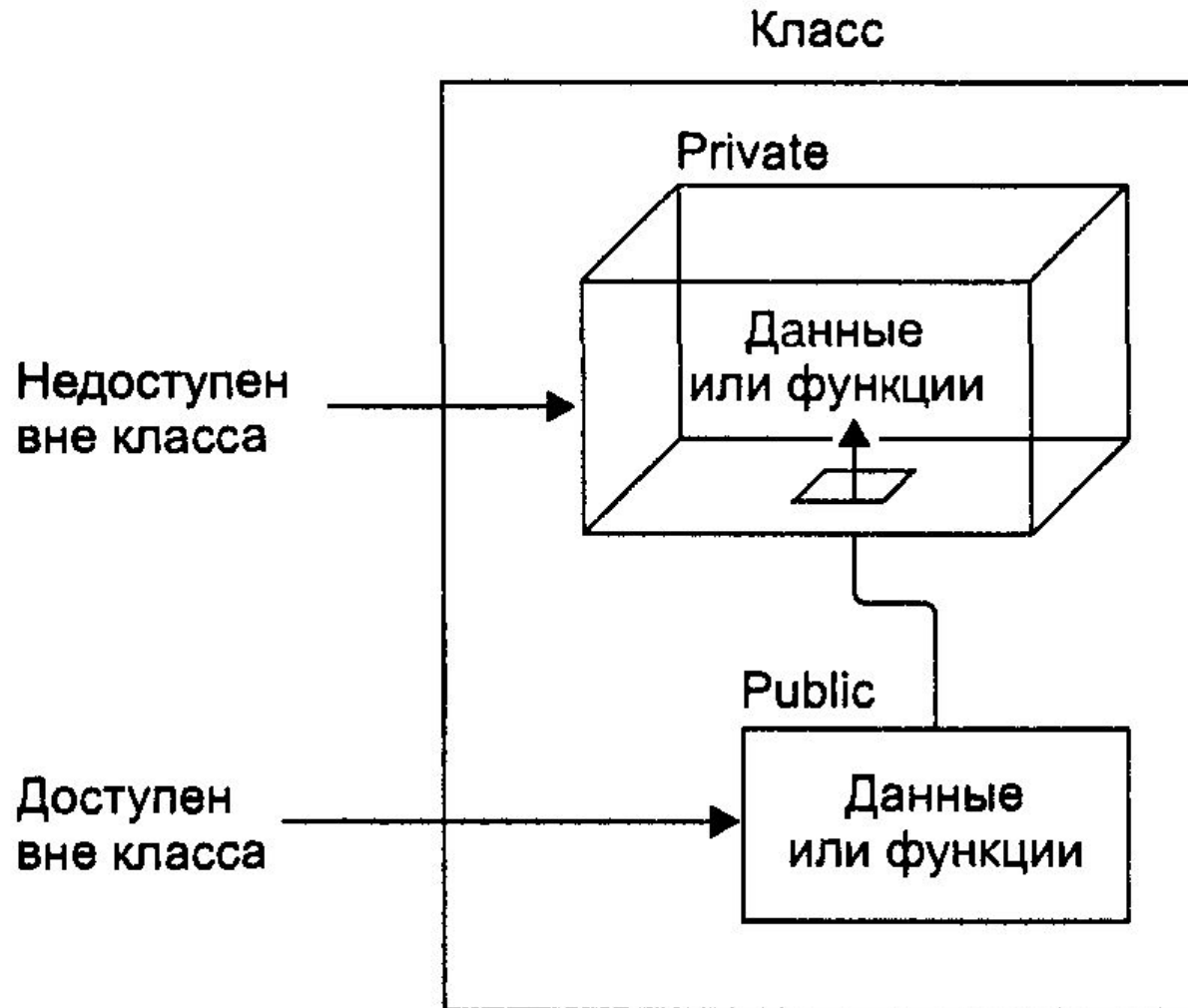
ОШИБКА
ВРЕМЕНИ
КОМПИЛЯЦИИ
(доступ к полям data1
и data2 извне закрыт)

```
    { data1 = a1; data2 = a2; }  
    void show()  
    { cout << data1 << endl;  
      cout << data2 << endl;  
    };
```

```
int main()  
{  
    MyClass s1, s2;  
    s1.set(10, 5);  
    s2.data1 = 17;  
    s2.data2 = 21;  
  
    s1.show();  
    return 0;  
}
```

ОШИБКИ НЕТ
(метод show() имеет доступ
к полям data1 и data2)

Разграничение доступа к данным: доступ внутри класса и извне



Файл "parts.h"

```
class part
{
private:
    int modelnum;
    int partnum;
    float cost;
public:
    void setpart(int modelnum, int partnum, float cost)
    {
        modelnum = modelnum;
        partnum = partnum;
        cost = cost;
    }
    void showpart()
    {
        cout << "Model " << modelnumber;
        cout << ", part " << partnumber;
        cout << ", costs $" << cost << endl;
    }
};
```

Файл "driver.cpp"

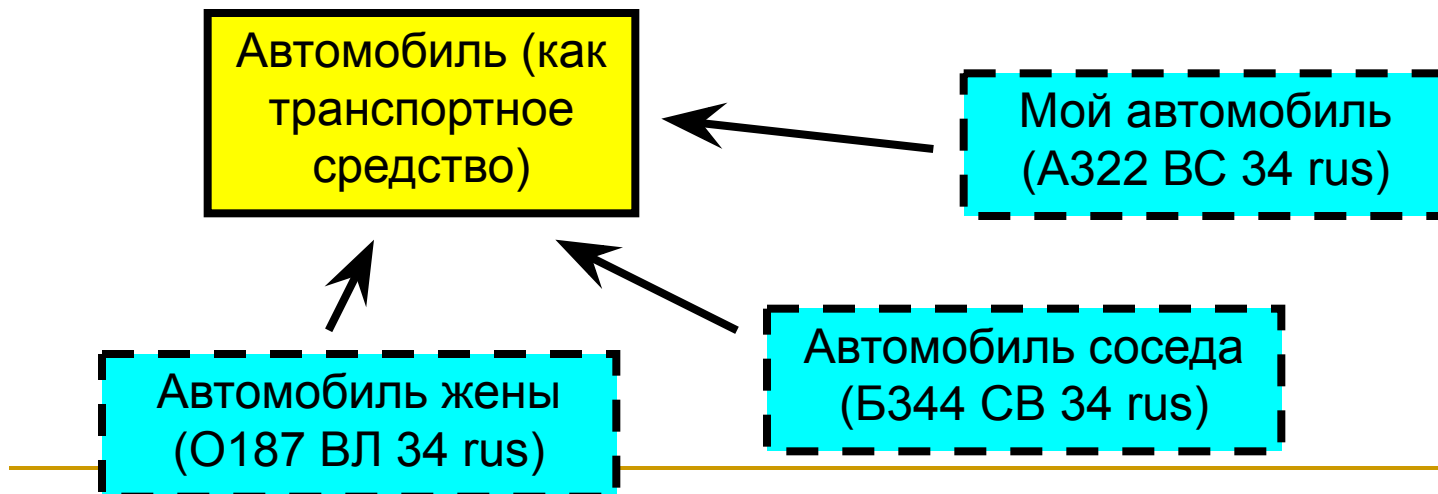
```
int main()
{
    part part1; // создаем объект
                // вызываем методы
    part1.setpart(624, 37, 217.5F);
    part1.showpart();
    return 0;
}
```

Классы и объекты

Отношение объекта к своему классу такое же, как отношение переменной к своему типу.

- **MyClass** - новый пользовательский тип (класс),
- **s1** и **s2** - переменные данного типа (объекты класса).

Пример отношения класса и объекта



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

```
class Distance {
private:
    int feet;
    float inches;
public:
    void setdist(int f, float i);
    void showdist();
};

int main()
{
    Distance dist1, dist2;

    dist1.setdist(11, 6.25);
    dist2.getdist();

    cout << "\ndist1 = "; dist1.showdist();
    cout << "\ndist2 = "; dist2.showdist();

    cout << endl;
    return 0;
}

void Distance::setdist(int f, float i)
{ feet = f; inches = i; }

void Distance::showdist()
{ cout << feet << " feet - " << inches << " inches\n"; }
};
```

Конструктор класса (class constructor)

Для инициализации полей объекта могут использоваться специально разработанные методы класса. Однако гораздо удобнее инициализировать поля объекта **автоматически в момент его создания**. Такой способ реализуется с помощью особого метода класса, который называется **конструктором**.

Определение. Конструктор - это метод класса, который выполняется автоматически в момент создания объекта.

Свойства конструктора

Конструктор похож на обычный метод класса, но имеет также особенности:

1. не имеет возвращаемого типа данных (даже void)
 2. имеет то же имя, что и класс, в котором он объявлен,
 3. допускается перегрузка конструктора (одно и то же имя, но различные параметры),
 4. объявляется со спецификатором доступа public.
-

Задачи конструктора:

- 1) создание объекта (размещение его в памяти)
- 2) инициализация полей объекта и определение **инварианта класса**.

В случае некорректности инварианта конструктор должен сообщить об ошибке.

Таким образом, корректно написанный конструктор всегда оставляет объект в «правильном» состоянии.

Инвариант класса — утверждение, которое должно быть истинно для любого объекта данного класса в любой момент времени

Синтаксис определения конструктора

```
class имя
{
    ...
    public:
        имя (<аргументы>) :<список_иниц-ции>
        {
            <тело_конструктора>
        }
    ...
};
```

Пример 1: конструктор без списка инициализации.

```
class Distance
{
    private:
        int feet;
        float inches;

    public:
        Distance(int ft, float in)
        {
            feet = ft;
            inches = in;
        }
        ...
};
...
```

Пример 2: конструктор со списком инициализации.

```
class Distance
{
    private:
        int feet;
        float inches;

    public:
        Distance(int ft, float in):
            feet(ft), inches(in) { }
        ...
};
...
```

Использование конструктора

Конструктор почти никогда не вызывается явным образом. Вызов конструктора происходит неявно всегда, когда создается объект данного класса.

А) Для инициализации объекта используется конструктор без параметров

```
имя_класса имя_объекта ;
```

ИЛИ

```
имя_класса имя_объекта ( ) ;
```

Использование конструктора

Б) Используется конструктор с параметрами

```
имя_класса имя_объекта (параметры) ;
```

Примеры неявного использования конструктора:

```
MyClass A,B,C;
```

```
Distance X(15, 2.85);
```

Деструктор класса (class destructor)

Определение. Деструктор - это метод класса, который вызывается автоматически в момент удаления объекта из памяти.

Назначение деструктора – выполнить действия, необходимые для корректного освобождения объекта (освобождение памяти, закрытие файлов и т.д.).

Аналогично конструктору, деструктор не возвращает никакого значения, и имеет то же имя, что и класс (только со знаком «тильда»)

Пример: деструктор класса

```
class Distance
{
    private:
        int feet;
        float inches;

    public:
        Distance(int ft, float in)
        {
            feet = ft;
            inches = in;
        }
        ~Distance()
        {
            feet = 0;
            inches = 0;
        }
};
...
```


Конструкторы/деструкторы и динамическая память

При размещении объекта в динамической памяти с помощью оператора `new` его конструктор вызывается **автоматически**.

При удалении объекта из памяти его деструктор также вызывается **автоматически**.

```
Distance *p;  
p = new Distance X(15, 2.85);  
...  
delete p;
```