Классы: основные понятия

Методы

Методы

- Метод — функциональный элемент класса, реализующий вычисления или другие действия. Методы определяют поведение класса и составляют его **интерфейс**.
- Метод — законченный фрагмент кода, к которому можно обратиться по имени. Он описывается один раз, а вызываться может столько раз, сколько необходимо.
- Один и тот же метод может обрабатывать различные данные, переданные ему в качестве аргументов.

```
double a = 0.1;
double b = Math.Sin(a);
Console.WriteLine(a);
```

Синтаксис метода

[атрибуты] [спецификаторы] тип имя_метода **(** **[параметры]** **)** тело_метода

- Спецификаторы: new, **public**, protected, internal, protected internal, private, static, virtual, sealed, override, abstract, extern.
- Метод класса имеет непосредственный доступ к его полям.
- Пример:

```
class Demo {
    double y;                          // закрытое поле класса

    public void Sety( double z ) {     // открытый метод класса
        y = z;
    }
}
… Demo x = new Demo();          // где-то в методе другого класса
  x.Sety(3.12); …               // вызов метода
```

Пример

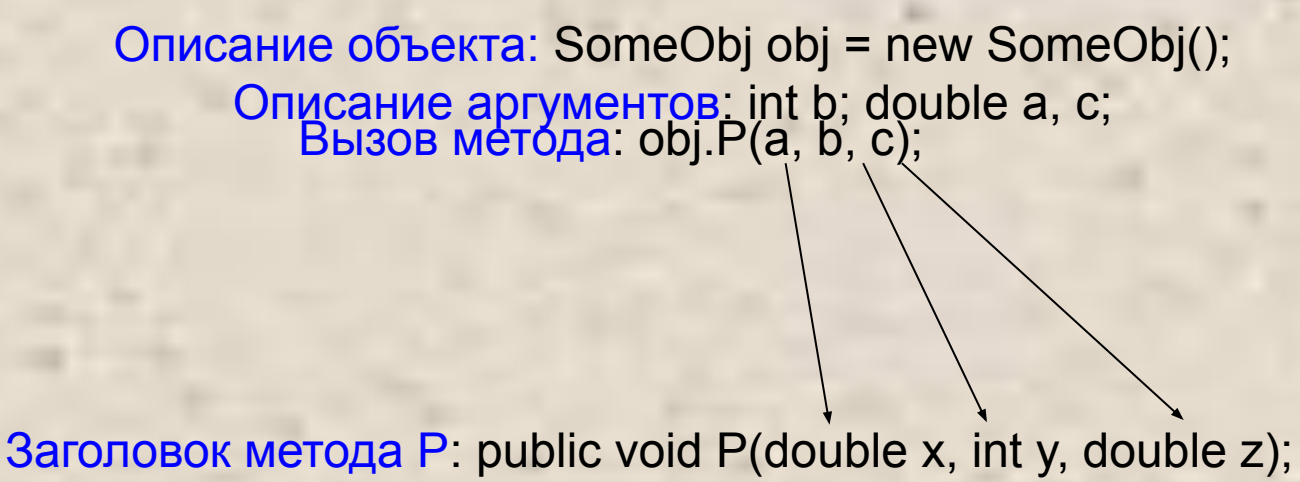
```
class Demo    {
        public int a = 1;
        public const double c = 1.66;
        static string s = "Demo";
        double y;
        public double Gety() { return y;  }            // метод получения y
        public void Sety( double y_ ){  y = y_; }      // метод установки y
        public static string Gets() { return s; }      // метод получения s
}
class Class1 {
       static void Main()
        {  Demo x = new Demo();
           x.Sety(0.12);                               // вызов метода установки y
           Console.WriteLine(x.Gety());                // вызов метода получения y
           Console.WriteLine(Demo.Gets());             // вызов метода получения s
//          Console.WriteLine(Gets());                 // вариант вызова из того же
}}                                                     // класса
```

Параметры методов

- Параметры определяют множество значений аргументов, которые можно передавать в метод.
- Список аргументов при вызове как бы накладывается на список параметров, поэтому они должны попарно соответствовать друг другу.
- Для каждого параметра должны задаваться его тип, имя и, возможно, вид параметра.
- Имя метода вкупе с количеством, типами и спецификаторами его параметров представляет собой сигнатуру метода — то, по чему один метод отличают от других.
- В классе не должно быть методов с одинаковыми сигнатурами.
- Метод, описанный со спецификатором static, должен обращаться только к статическим полям класса.
- Статический метод вызывается через имя класса, а обычный — через имя экземпляра.

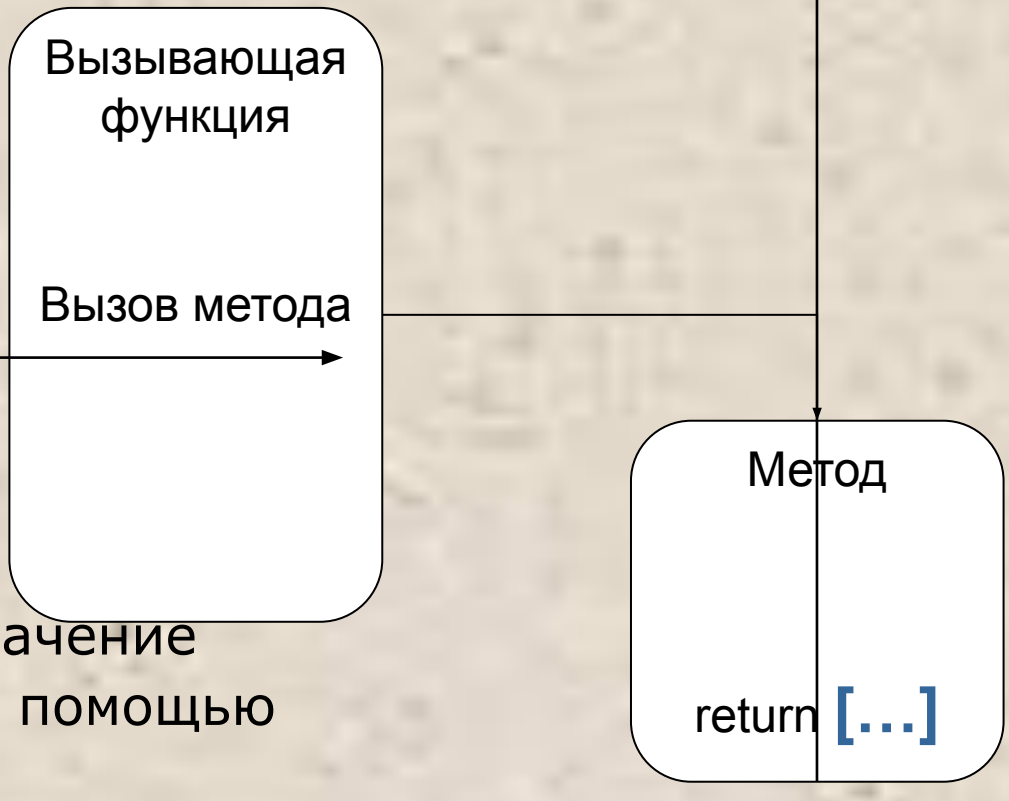
Вызов метода

1. Вычисляются выражения, стоящие на месте аргументов.
2. Выделяется память под параметры метода.
3. Каждому из параметров сопоставляется соответствующий аргумент. При этом проверяется соответствие типов аргументов и параметров и при необходимости выполняется их преобразование. При несоответствии типов выдается диагностическое сообщение.
4. Выполняется тело метода.
5. Если метод возвращает значение, оно передается в точку вызова; если метод имеет тип void, управление передается на оператор, следующий после вызова.



Примеры методов

```
public void Sety(double z) {
    y = z;
}
public double Gety() {
   return y;
}
```



- Тип метода определяет, значение какого типа вычисляется с помощью метода
- Параметры используются для обмена информацией с методом. Параметр - локальная переменная, которая при вызове метода принимает значение соответствующего аргумента.

```
x.Sety(3.12);
double t = x.Gety();
```

Способы передачи параметров и их типы

Способы передачи параметров: по значению и по ссылке.

- *При передаче по значению* метод получает копии значений аргументов, и операторы метода работают с этими копиями.
- *При передаче по ссылке* (*по адресу*) метод получает копии адресов аргументов и осуществляет доступ к аргументам по этим адресам.

В C# четыре типа параметров:

- параметры-значения;
- параметры-ссылки (**ref**);
- выходные параметры (**out**);
- параметры-массивы (**params**).

Ключевое слово предшествует описанию типа параметра. Если оно опущено, параметр считается параметром-значением. Пример:

```
public int Calculate( int a, ref int b, out int c, params int[] d ) …
```

Пример передачи параметров

```
class Class1
   {   static int Max(int a, int b)      // выбор макс. значения
       {
    a=20; b=40;
    return b;
       }
       static void Main()
       {
           int a = 2, b = 4;
    Console.WriteLine( a, b);
           int x = Max( a, b );        // вызов метода Max
           Console.WriteLine( a, b);                  // результат: 4
           short t1 = 3, t2 = 4;
           int y = Max( t1, t2 );      // пар-ры совместимого типа
           Console.WriteLine( y );                    // результат: 4
           int z = Max( a + t1, t1 / 2 * b );   // выражения
           Console.WriteLine( z );                    // результат: 5
}}
```

Пример передачи объектов

```
class Test
{
  public int a, b;
   public Test(int i, int j)  {  a = i;     b = j;  }
 public void Change(Test ob) {
    ob.a = ob.a + ob.b;
    ob.b = -ob.b;   }
}
 class CallByRef {
  static void Main() {
    Test ob = new Test(15, 20);
     Console.WriteLine("до вызова: "+ ob.a + " " + ob.b);
     ob.Change(ob);
     Console.WriteLine("после вызова: " +  ob.a + " " + ob.b);
  } }
```

Пример: параметры-значения и ссылки ref

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void P( int a, ref int b )
        {
            a = 44; b = 33;
            Console.WriteLine( "внутри метода {0} {1}", a, b );
        }
        static void Main()
        {
            int a = 2, b = 4;
            Console.WriteLine( "до вызова     {0} {1}", a, b );
            P( a, ref b );
            Console.WriteLine( "после вызова  {0} {1}", a, b );
}}}
```

Результат работы программы:
до вызова 2 4
внутри метода 44 33
после вызова 2 33

Пример: выходные параметры out

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void P( int x, out int y )
        {
            x = 44; y = 33;
            Console.WriteLine( "внутри метода {0} {1}", x, y );
        }
        static void Main()
        {
            int a = 2, b;         // инициализация b не требуется

            P( a, out b );
            Console.WriteLine( "после вызова  {0} {1}", a, b );
}}}
```

Результат работы программы:

внутри метода 44 33
после вызова 2 33

Использование модификаторов ref и out для ссылок на объекты

```
class RefSwap {
  int a, b;
  public RefSwap(int i, int j){     a = i;     b = j;  }

  public void Swap(ref RefSwap ob1, ref RefSwap ob2) {
    RefSwap t;   t = ob1;     ob1 = ob2;     ob2 = t;  }
 public void Show() {Console.WriteLine (a + “  “+b);}
}
 class RefSwapDemo {
  static void Main()   {
 RefSwap x = new RefSwap(1, 2);
RefSwap y = new RefSwap(3, 4);
x.Show(); y.Show();
    x.Swap(ref x, ref y);
Console.Write("x после вызова: ");     x.Show();
 Console.Write("y после вызова: ");     y.Show();   } }
```

Правила применения параметров

1. Для параметров-значений используется передача по значению. Этот способ применяется для исходных данных метода.

- При вызове метода на месте параметра, передаваемого по значению, может находиться выражение (а также его частные случаи — переменная или константа). Должно существовать неявное преобразование типа выражения к типу параметра.

2. Параметры-ссылки и выходные параметры передаются по адресу. Этот способ применяется для передачи побочных результатов метода.

- При вызове метода на месте параметра-ссылки **ref** может находиться только имя инициализированной переменной точно того же типа. Перед именем параметра указывается ключевое слово ref.
- При вызове метода на месте выходного параметра **out** может находиться только имя переменной точно того же типа. Ее инициализация не требуется. Перед именем параметра указывается ключевое слово out.

Использование переменного числа аргументов

```
class Min
{
  public int MinVal(params int[] nums)
  {
    int m = nums[0];
    for(int i=1; i < nums.Length; i++)  if(nums[i] < m) m = nums[i];
    return m; }}

class ParamsDemo
{
  static void Main()
  {
    Min ob = new Min();
    int min, a = 10, b = 20;

min = ob.MinVal(a, b, -1); Console.WriteLine("Наименьшее значение равно " + min);

min = ob.MinVal(18, 23, 3, 14, 25); Console.WriteLine("Наименьшее значение " + min);

int[] args = { 45, 67, 34, 9, 112, 8 };
    min = ob.MinVal(args);
    Console.WriteLine("Наименьшее значение равно " + min);   } }
```

Необязательные аргументы

```
static void OptArgMeth(int alpha, int beta=10, int gamma = 20)
{}

// Передать все аргументы явным образом.
OptArgMeth(1, 2, 3);

// Сделать аргумент gamma необязательным.
OptArgMeth(1, 2);

// Сделать оба аргумента beta и gamma необязательными.
OptArgMeth(1);

int Sample(string name = "пользователь", int userId) { }// Ошибка!
int Sample(int accountId, string name = "пользователь", int
userId)  { }  //Ошибка!
```

Именованные аргументы

Для указания аргумента по имени служит следующая форма синтаксиса.

***имя_параметра* : значение**

```
class NamedArgsDemo
{

  static bool IsFactor(int val, int divisor){
    if((val % divisor) == 0) return true;
    return false; }

  static void Main()   {
    if(IsFactor(10, 2))      Console.WriteLine("2 множитель 10.");

    if(IsFactor(val :10, divisor: 2))   Console.WriteLine("2 множитель 10.");

    if(IsFactor(divisor: 2, val: 10))   Console.WriteLine("2 множитель 10.");

    if(IsFactor(10, divisor: 2))    Console.WriteLine("2 множитель 10.");
  }  }
```