Теория и практика программирования

Шпилёв Пётр Валерьевич

Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра статистического моделирования

Лекция 3

Санкт-Петербург
2015 г.

e-mail: pitshp@hotmail.com

Общий вид:

```
class имя_класса
{
    // Объявление переменных экземпляра.
    доступ тип переменная1;
    доступ тип переменная2;
    //...
    доступ тип переменнаяN;
    // Объявление методов.
    доступ возращаемый_тип метод1(параметры)
    {
        // тело метода
    }
    доступ возращаемый_тип метод2(параметры)
    {
        // тело метода
    }
    // ...
    доступ возращаемый_тип методN(параметры)
    {
        // тело метода
    }
}
```

```
доступ возвращаемый_тип имя_метода(параметры)
{
    // тело метода
    //...
    if(done) return значение; // значение имеет возвращаемый_тип
    // ...
    if(error) return значение; // значение  имеет возвращаемый_тип
}
```

```
доступ возвращаемый_тип имя_метода(параметры)
{
    // тело метода
    //...
    if(done) return значение; // значение имеет возвращаемый_тип
    // ...
    if(error) return значение; // значение  имеет возвращаемый_тип
}
```

Пример:

```
int Method1(int a, double b, float c)
{
     // ...
     return a;
    Console.WriteLine("это недоступный код");
}
```

Общий вид:

```
доступ имя_класса(список_параметров )
{
    // тело конструктора
}
```

Пример 1:

```
// Простой конструктор.
using System;
class MyClass
{
    public int x;
    public MyClass()
    {
        x = 10;
    }
}
class ConsDemo
{
    static void Main()
    {
        MyClass t1 = new MyClass();
        MyClass t2 = new MyClass();
        Console.WriteLine(t1.x +" "+ t2.x);
    }
}
```

Пример 2:

```
// Параметризированный конструктор.
using System;
class MyClass
{
    public int x;
    public MyClass(int i)
    {
        x = i;
    }
}
class ParmConsDemo
{
    static void Main()
    {
        MyClass t1 = new MyClass(10);
        MyClass t2 = new MyClass(88);
        Console.WriteLine(t1.x +" "+ t2.x);
    }
}
```

Форма оператора new

```
new имя_класса(список_аргументов)
```

Форма оператора new

new имя_класса(список_аргументов)

Пример:

```
// Использовать оператор new вместе с типом значения.
using System;
class newValue
{
    static void Main()
    {
        int i = new int(); // инициализировать переменную i нулевым
значением
        Console.WriteLine("Значение переменной i равно:" + i);
    }
}
```

Общий вид деструктора:

```
~ имя_класса()
{
    // код деструктора
}
```

Пример:

```
// Продемонстрировать применение деструктора.
using System;
class Destruct
{
    public int x;
    public Destruct(int i)
    {
        x = i;
    }
    // Вызывается при утилизации объекта.
    ~Destruct()
    {
        Console.WriteLine("Уничтожить "+ x);
    }
    // Создает объект и тут же уничтожает его.
    public void Generator(int i)
    {
        Destruct o = new Destruct(i);
    }
}
```

```
class DestructDemo
{
    static void Main()
    {
        int count;
        Destruct ob = new Destruct(0);
        /* А теперь создадим большое число объектов.
        В какой-то момент произойдет "сборка мусора".
        Примечание: для того чтобы активизировать
        "сборку мусора возможно, придется увеличить
        число создаваемых объектов. */
        for (count = 1; count < 10; count++)
        ob.Generator(count);
        Console.WriteLine("Готово!");
    }
}
```

Пример 1:
```
using System;
class Rect
{
    public int Width;
    public int Height;
    public Rect(int w, int h)
    {
        Width = w;
        Height = h;
     }
    public int Area()
    {
        return Width * Height;
    }
}
```

Пример 1:
```
using System;
class Rect
{
    public int Width;
    public int Height;
    public Rect(int w, int h)
    {
        this.Width = w;
        this.Height = h;
     }
    public int Area()
    {
        return this.Width * this.Height;
    }
}
```

Пример 2:
```
using System;
class Rect
{
    public int Width;
    public int Height;
    public Rect(int Width, int Height)
    {
        this.Width = Width;
        this.Height = Height;
     }
    public int Area()
    {
        return this.Width * this.Height;
    }
}
```

Общая форма:

тип[ ] имя_массива = new тип[размер];

где тип объявляет конкретный тип элемента массива.

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Пример:
int[ ] sample = new int[10];

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Пример:

```
int[ ] sample;
sample = new int[10];
```

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Пример:
int[ ] nums = { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Пример:

```
int[ ] nums;
nums = new int[ ] { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };
```

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Общая форма объявления многомерного массива:

тип[,...,] имя_массива = new тип[размер1, размер2, ... размерN];

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Общая форма объявления многомерного массива:

тип[,...,] имя_массива = new тип[размер1, размер2, ... размерN];

Упражнение 3.1

Написать программу для создания двумерного массива инициализированного целыми числами и посчитать его определитель.

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Общая форма объявления многомерного массива:

тип[,...,] имя_массива = new тип[размер1, размер2, ... размерN];

Общая форма инициализации двумерного массива:

тип[,] имя_массива = { {val, val, val, ..., val}, {val, val, val, ..., val}, {val, val, val, ..., val} };

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Общая форма объявления многомерного массива:

тип[,...,] имя_массива = new тип[размер1, размер2, ... размерN];

Общая форма инициализации двумерного массива:

тип[,] имя_массива = { {val, val, val, ..., val}, {val, val, val, ..., val}, {val, val, val, ..., val} };

Общая форма объявления двумерного ступенчатого массива:

тип[ ][ ] имя_массива = new тип[размер][ ];

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Пример:

```
int[ ][ ] jagged = new int[3][ ];
jagged[0] = new int[4];
jagged[1] = new int[3];
jagged[2] = new int[5];
```

Общая форма объявления двумерного ступенчатого массива:

тип[ ][ ] имя_массива = new тип[размер][ ];

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Пример:

```
int[ ][ ] jagged = new int[3][ ];
jagged[0] = new int[4];
jagged[1] = new int[3];
jagged[2] = new int[5];
```

Пример (массив массивов):

```
int[ ][ , ] jagged = new int[3][ , ];
```

Общая форма объявления двумерного ступенчатого массива:

тип[ ][ ] имя_массива = new тип[размер][ ];

Общая форма:

тип[ ] имя_массива = new тип[размер];
где тип объявляет конкретный тип элемента массива.

Общая форма инициализации одномерного массива:

тип[ ] имя_массива = {val1, val2, val3, ..., valN};

Упражнение 3.2

Используя свойство Length, поменять местами содержимое элементов массива

Общая форма объявления двумерного ступенчатого массива:

тип[ ][ ] имя_массива = new тип[размер][ ];

```
// Продемонстрировать неявно типизированный ступенчатый массив.
using System;
class Jagged
{
    static void Main()
    {
        var jagged = new [ ] {
        new [ ] { 1, 2, 3, 4 },
        new [ ] { 9, 8, 7 },
        new [ ] { 11, 12, 13, 14, 15 } };
        for(int j = 0; j < jagged.Length; j++)
        {
            for(int i=0; i < jagged[j].Length; i++)
            Console.Write(jagged[j][i] + );
            Console.WriteLine();
        }
    }
}
```

Общая форма оператора цикла foreach

foreach (тип имя_переменной_цикла in коллекция) оператор;

Общая форма оператора цикла foreach

foreach (тип имя_переменной_цикла in коллекция) оператор;

Замечание

Тип переменной цикла должен соответствовать типу элемента массива

Пример:

```
using System;
class foreachDemo
{
    static void Main()
    {
        int sum = 0;
        int[ ] nums = new int[10];
        // Задать первоначальные значения элементов массива nums.
        for (int i = 0; i < 10; i++)
        nums[i] = i;
        // Использовать цикл foreach для вывода значений
        // элементов массива и подсчета их суммы.
        foreach (int x in nums)
        {
            Console.WriteLine("Значение элемента равно: " + x);
            sum += x;
        }
        Console.WriteLine("Сумма равна: " + sum);
    }
}
```

Результат:

```
Значение элемента равно: 0
Значение элемента равно: 1
Значение элемента равно: 2
Значение элемента равно: 3
Значение элемента равно: 4
Значение элемента равно: 5
Значение элемента равно: 6
Значение элемента равно: 7
Значение элемента равно: 8
Значение элемента равно: 9
Сумма равна: 45
```

Результат:

```
Значение элемента равно: 0
Значение элемента равно: 1
Значение элемента равно: 2
Значение элемента равно: 3
Значение элемента равно: 4
Значение элемента равно: 5
Значение элемента равно: 6
Значение элемента равно: 7
Значение элемента равно: 8
Значение элемента равно: 9
Сумма равна: 45
```

Упражнение 3.3

Использовать оператор foreach для вычисления суммы элементов двумерного массива

Инициализация с помощью строкового литерала:

```
string str = "Строки в C# весьма эффективны.";
```

Инициализация с помощью строкового литерала:

```
string str = "Строки в C# весьма эффективны.";
```

Инициализация с помощью массива типа char:

```
char[ ] charray = {'t', 'e', 's', 't'};
string  str = new string(charray);
```

Таблица 1.Некоторые общеупотребительные методы обращения со строками

Метод	Описание
`static int Compare(string strA, string strB, StringComparison comparisonType)`	Возвращает отрицательное значение, если строка `strA` меньше строки `strB`; положительное значение, если строка `strA` больше строки `strB`; и нуль, если сравниваемые строки равны. Способ сравнения определяется аргументом `comparisonType`
`bool Equals(string value, StringComparison comparisonType)`	Возвращает логическое значение `true`, если вызывающая строка имеет такое же значение, как и у аргумента `value`. Способ сравнения определяется аргументом `comparisonType`
`int IndexOf(char value)`	Осуществляет поиск в вызывающей строке первого вхождения символа, определяемого аргументом `value`. Применяется порядковый способ поиска. Возвращает индекс первого совпадения с искомым символом или -1, если он не обнаружен
`int IndexOf(string value, StringComparison comparisonType)`	Осуществляет поиск в вызывающей строке первого вхождения подстроки, определяемой аргументом `value`. Возвращает индекс первого совпадения с искомой подстрокой или -1, если она не обнаружена. Способ поиска определяется аргументом `comparisonType`

Таблица 1.Некоторые общеупотребительные методы обращения со строками

Метод	**Описание**
`int LastIndexOf(char value)`	Осуществляет поиск в вызывающей строке последнего вхождения символа, определяемого аргументом *value*. Применяется порядковый способ поиска. Возвращает индекс последнего совпадения с искомым символом или -1, если он не обнаружен
`int LastIndexOf(string value, StringComparison comparisonType)`	Осуществляет поиск в вызывающей строке последнего вхождения подстроки, определяемой аргументом *value*. Возвращает индекс последнего совпадения с искомой подстрокой или -1, если она не обнаружена. Способ поиска определяется аргументом *comparisonType*
`string ToLower(CultureInfo. CurrentCulture culture)`	Возвращает вариант вызывающей строки в нижнем регистре. Способ преобразования определяется аргументом *culture*
`string ToUpper(CultureInfo. CurrentCulture culture)`	Возвращает вариант вызывающей строки в верхнем регистре. Способ преобразования определяется аргументом *culture*

Упражнение 3.4

Инициализировать несколько строк. Используя описанные выше методы:

а) Создать варианты строки набранные прописными и строчными буквами.

б) Вывести строку посимвольно.

в) Сравнить строки с учетом культурной среды.

г) Выделить из одной из строк подстроку и найти индекс ее первого вхождения в исходную строку.

Упражнение 3.4

Инициализировать несколько строк. Используя описанные выше методы:

а) Создать варианты строки набранные прописными и строчными буквами.

б) Вывести строку посимвольно.

в) Сравнить строки с учетом культурной среды.

г) Выделить из одной из строк подстроку и найти индекс ее первого вхождения в исходную строку.

Упражнение 3.5

Вывести отдельные цифры целого числа словами.

Пример:

```
// Продемонстрировать управление оператором switch посредством строк.
using System;
class StringSwitch {
static void Main() {
        string[ ] strs = { "один", "два", "три", "два", "один" };
        foreach (string s in strs)
         {   switch (s)
            {
                case "один":
                    Console.Write(1);
                    break;
                case "два":
                    Console.Write(2);
                    break;
                case "три":
                    Console.Write(3);
                    break;
            }
        }
        Console.WriteLine();      } }
```

Пример:

```
// Отличия между видами доступа public и private к членам класса.
using System;
class Myclass
{
    private int alpha; // закрытый доступ, указываемый явно
    int beta; // закрытый доступ по умолчанию
    public int gamma; // открытый доступ
    // Методы, которым доступны члены alpha и beta данного класса.
   //Член класса может иметь доступ к закрытому члену этого же класса.
    public void SetAlpha(int a){ alpha = a;}
    public int GetAlpha(){ return alpha;}
    public void SetBeta(int a){ beta = a;}
    public int GetBeta(){ return beta;}
}
```

Пример:

```
class AccessDemo
{
    static void Main()
    {
        Myclass ob = new Myclass();
        // Доступ к членам alpha и beta данного класса
        // разрешен только посредством его методов.
        ob.SetAlpha(-99);
        ob.SetBeta(19);
        Console.WriteLine("ob.alpha равно " + ob.GetAlpha());
        Console.WriteLine("ob.beta равно " + ob.GetBeta ());
        // Следующие виды доступа к членам alpha и beta
        // данного класса не разрешаются.
        // ob.alpha = 10; // Ошибка! alpha - закрытый член!
        // ob.beta =9; // Ошибка! beta - закрытый член!
        // Член gamma данного класса доступен непосредственно,
        // поскольку он является открытым.
        ob.gamma = 99;
    }
}
```

Общие принципы:

- Члены, используемые только в классе, должны быть закрытыми.
- Данные экземпляра, не выходящие за определенные пределы значений, должны быть закрытыми, а при организации доступа к ним с помощью открытых методов следует выполнять проверку диапазона представления чисел.
- Если изменение члена приводит к последствиям, распространяющимся за пределы области действия самого члена, т.е. оказывает влияние на другие аспекты объекта, то этот член должен быть закрытым, а доступ к нему — контролируемым.
- Члены, способные нанести вред объекту, если они используются неправильно, должны быть закрытыми. Доступ к этим членам следует организовать с помощью открытых методов, исключающих неправильное их использование.
- Методы, получающие и устанавливающие значения закрытых данных, должны быть открытыми.
- Переменные экземпляра допускается делать открытыми лишь в том случае, если нет никаких оснований для того, чтобы они были закрытыми.

Общие принципы:

- Члены, используемые только в классе, должны быть закрытыми.
- Данные экземпляра, не выходящие за определенные пределы значений, должны быть закрытыми, а при организации доступа к ним с помощью открытых методов следует выполнять проверку диапазона представления чисел.
- Если изменение члена приводит к последствиям, распространяющимся за пределы области действия самого члена, т.е. оказывает влияние на другие аспекты объекта, то этот член должен быть закрытым, а доступ к нему — контролируемым.
- Члены, способные нанести вред объекту, если они используются неправильно, должны быть закрытыми. Доступ к этим членам следует организовать с помощью открытых методов, исключающих неправильное их использование.
- Методы, получающие и устанавливающие значения закрытых данных, должны быть открытыми.
- Переменные экземпляра допускается делать открытыми лишь в том случае, если нет никаких оснований для того, чтобы они были закрытыми.

Общие принципы:

- Члены, используемые только в классе, должны быть закрытыми.
- Данные экземпляра, не выходящие за определенные пределы значений, должны быть закрытыми, а при организации доступа к ним с помощью открытых методов следует выполнять проверку диапазона представления чисел.
- Если изменение члена приводит к последствиям, распространяющимся за пределы области действия самого члена, т.е. оказывает влияние на другие аспекты объекта, то этот член должен быть закрытым, а доступ к нему — контролируемым.
- Члены, способные нанести вред объекту, если они используются неправильно, должны быть закрытыми. Доступ к этим членам следует организовать с помощью открытых методов, исключающих неправильное их использование.
- Методы, получающие и устанавливающие значения закрытых данных, должны быть открытыми.
- Переменные экземпляра допускается делать открытыми лишь в том случае, если нет никаких оснований для того, чтобы они были закрытыми.

Общие принципы:

- Члены, используемые только в классе, должны быть закрытыми.
- Данные экземпляра, не выходящие за определенные пределы значений, должны быть закрытыми, а при организации доступа к ним с помощью открытых методов следует выполнять проверку диапазона представления чисел.
- Если изменение члена приводит к последствиям, распространяющимся за пределы области действия самого члена, т.е. оказывает влияние на другие аспекты объекта, то этот член должен быть закрытым, а доступ к нему — контролируемым.
- Члены, способные нанести вред объекту, если они используются неправильно, должны быть закрытыми. Доступ к этим членам следует организовать с помощью открытых методов, исключающих неправильное их использование.
- Методы, получающие и устанавливающие значения закрытых данных, должны быть открытыми.
- Переменные экземпляра допускается делать открытыми лишь в том случае, если нет никаких оснований для того, чтобы они были закрытыми.

Общие принципы:

- Члены, используемые только в классе, должны быть закрытыми.
- Данные экземпляра, не выходящие за определенные пределы значений, должны быть закрытыми, а при организации доступа к ним с помощью открытых методов следует выполнять проверку диапазона представления чисел.
- Если изменение члена приводит к последствиям, распространяющимся за пределы области действия самого члена, т.е. оказывает влияние на другие аспекты объекта, то этот член должен быть закрытым, а доступ к нему — контролируемым.
- Члены, способные нанести вред объекту, если они используются неправильно, должны быть закрытыми. Доступ к этим членам следует организовать с помощью открытых методов, исключающих неправильное их использование.
- Методы, получающие и устанавливающие значения закрытых данных, должны быть открытыми.
- Переменные экземпляра допускается делать открытыми лишь в том случае, если нет никаких оснований для того, чтобы они были закрытыми.

Общие принципы:

- Члены, используемые только в классе, должны быть закрытыми.
- Данные экземпляра, не выходящие за определенные пределы значений, должны быть закрытыми, а при организации доступа к ним с помощью открытых методов следует выполнять проверку диапазона представления чисел.
- Если изменение члена приводит к последствиям, распространяющимся за пределы области действия самого члена, т.е. оказывает влияние на другие аспекты объекта, то этот член должен быть закрытым, а доступ к нему — контролируемым.
- Члены, способные нанести вред объекту, если они используются неправильно, должны быть закрытыми. Доступ к этим членам следует организовать с помощью открытых методов, исключающих неправильное их использование.
- Методы, получающие и устанавливающие значения закрытых данных, должны быть открытыми.
- Переменные экземпляра допускается делать открытыми лишь в том случае, если нет никаких оснований для того, чтобы они были закрытыми.

Общие принципы:

- Члены, используемые только в классе, должны быть закрытыми.
- Данные экземпляра, не выходящие за определенные пределы значений, должны быть закрытыми, а при организации доступа к ним с помощью открытых методов следует выполнять проверку диапазона

Упражнение 3.7

Используя модификаторы доступа, реализовать стек из элементов char. Добавить методы с помощью которых можно:

а) поместить символ в стек

б) извлечь символ из стека

в) проверить заполнен стек или нет

г) возвратить общую емкость стека

д) возвратить количество объектов, находящихся в данный момент в стеке.

- Переменные экземпляра допускается делать открытыми лишь в том случае, если нет никаких оснований для того, чтобы они были закрытыми.

Две причины по которым значение простого типа иногда требуется передавать по ссылке:

- разрешить методу изменить содержимое его аргументов
- возвратить несколько значений

Две причины по которым значение простого типа иногда требуется передавать по ссылке:

- разрешить методу изменить содержимое его аргументов
- возвратить несколько значений

Пример:

```
// Использовать модификатор ref для
// передачи значения обычного типа по ссылке.
using System;
class RefTest
{
     // Этот метод изменяет свой аргумент. Обратите
     // внимание на применение модификатора ref.
     public void Sqr(ref int i)
    {
      i = i * i;
    }
}
class RefDemo
{
    static void Main()
    {
      RefTest ob = new RefTest();
      int a = 10;
      Console.WriteLine("a до вызова: " + a);
      ob.Sqr(ref a); // обратите внимание на применение модификатора ref
      Console.WriteLine("a после вызова: " + a);
    }
}
```

Пример: // Использовать модификатор ref для
// передачи значения обычного типа по ссылке.

```
using System;
```

Результат:

```
a до вызова: 10
a после вызова: 100
```

```
class RefDemo
{
    static void Main()
    {
      RefTest ob = new RefTest();
      int a = 10;
      Console.WriteLine("a до вызова: " + a);
      ob.Sqr(ref a); // обратите внимание на применение модификатора ref
      Console.WriteLine("a после вызова: " + a);
    }
}
```

```
using System;
class Decompose
{ /* Разделить числовое значение с плавающей точкой на целую и
дробную части. */
    public int GetParts(double n, out double frac)
    {
        frac = n - (int)n;//передать дробную часть числа через параметр frac
        return (int)n; // возвратить целую часть числа
     }
}
class UseOut
{
static void Main()
    {
        Decompose ob = new Decompose();
        int i;
        double f;
        i = ob.GetParts(10.125, out f);
        Console.WriteLine("Целая часть числа равна " + i);
        Console.WriteLine("Дробная часть числа равна " + f);
    }
}
```

Упражнение 3.8

Написать метод который, во-первых, определяет общий множитель (кроме 1) для двух целых чисел, возвращая логическое значение true, если у них имеется общий множитель, а иначе — логическое значение false. И, во-вторых, возвращает посредством параметров типа out наименьший и наибольший общий множитель двух чисел, если таковые обнаруживаются

Метод с переменным числом параметров:

```
доступ тип имя_метода(params тип[ ] имя_параметра)
```

Метод с переменным числом параметров:

доступ тип имя_метода(params тип[ ] имя_параметра)

Упражнение 3.9

Реализовать метод, обнаруживающий наименьшее среди ряда значений.

Метод с переменным числом параметров:

доступ тип имя_метода(params тип[ ] имя_параметра)

Упражнение 3.9

Реализовать метод, обнаруживающий наименьшее среди ряда значений.

Общий вид:

доступ тип имя_метода(тип имя_параметра1, тип имя_параметра2,...,
тип имя_параметра N-1, params тип[ ] имя_параметраN)

Пример:

```
using System;
class Myclass
{
    public void ShowArgs(string msg, params int[] nums)
    {
        Console.Write(msg + " ");
        foreach(int i in nums)
        Console.Write(i + " ");
        Console.WriteLine();
    }
}
class ParamsDemo2
{
    static void Main()
    {
        Myclass ob = new Myclass();
        ob.ShowArgs("Это ряд целых чисел",1, 2, 3, 4, 5);
        ob.ShowArgs("А это еще два целых числа", 17, 20);
    }
}
```

Пример:

Результат:

Это ряд целых чисел: 1, 2, 3, 4, 5
А это еще два целых числа: 17, 20

```
class ParamsDemo2
{
    static void Main()
    {
        Myclass ob = new Myclass();
        ob.ShowArgs("Это ряд целых чисел",1, 2, 3, 4, 5);
        ob.ShowArgs("А это еще два целых числа", 17, 20);
    }
}
```

Общий вид:

доступ возвращаемый_тип метод(параметры)

Общий вид:

доступ возвращаемый_тип метод(параметры)

Упражнение 3.10

Создать фабрику класса (т.е. метод, предназначенный для построения объектов его же класса).

Общий вид:

доступ возвращаемый_тип метод(параметры)

Упражнение 3.10

Создать фабрику класса (т.е. метод, предназначенный для построения объектов его же класса).

Возврат массива из метода:

доступ возвращаемый_тип[ ] метод(параметры)

Общий вид:

доступ возвращаемый_тип метод(параметры)

Упражнение 3.10

Создать фабрику класса (т.е. метод, предназначенный для построения объектов его же класса).

Возврат массива из метода:

доступ возвращаемый_тип[ ] метод(параметры)

Упражнение 3.11

Реализовать метод, который возвращает массив, содержащий множители переданного ему аргумента