Теория и практика программирования

Шпилёв Пётр Валерьевич

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

Лекция 4

Санкт-Петербург
2015 г.

e-mail: pitshp@hotmail.com

Перегрузка методов:

```
доступ возвращаемый_тип имя_метода(параметры /*(Набор 1)*/)
доступ возвращаемый_тип имя_метода(параметры /*(Набор 2)*/)
```

Пример:

```
using System;
class Overload
{
    public void OvlDemo()
    {
        Console.WriteLine("Без параметров") ;
    }
    // Перегрузка метода OvlDemo с одним целочисленным параметром.
    public void OvlDemo(int a)
    {
        Console.WriteLine("Один параметр: " + a);
    }
    // Перегрузка метода OvlDemo с двумя параметрами типа double.
    public double OvlDemo(double a, double b)
    {
        Console.WriteLine("Два параметра типа double: " + a + " " + b);
        return a + b;
    }
}
```

Пример 1:

```
public void MyMeth(int x)
{
    Console.WriteLine("В методе MyMeth(int): " + x);
}
public void MyMeth( ref int x)
{
    Console.WriteLine("В методе MyMeth(ref int): " + x);
}
```

Пример 1:

```
public void MyMeth(int x)
{
    Console.WriteLine("В методе MyMeth(int): " + x);
}
public void MyMeth( ref int x)
{
    Console.WriteLine("В методе MyMeth(ref int): " + x);
}
```

Пример 2:

```
// Неверно!
public void MyMeth(out int x) { // ...
public void MyMeth(ref int x) { // ...
```

Пример:

```
// Продемонстрировать перегрузку конструктора.
using System;
class MyClass
{
    public int x;
    public MyClass()
    {
        Console.WriteLine("В конструкторе MyClass().") ;
        x = 0;
    }
    public MyClass(double d)
    {
        Console.WriteLine("В конструкторе MyClass(double).") ;
        x = (int)d;
    }
    public MyClass(int i, int j)
    {
        Console.WriteLine("В конструкторе MyClass(int, int).") ;
        x = i * j;
    }
}
```

Общая форма:

```
имя_конструктора(список_параметров1) : this(список_параметров2)
{ ... }
```

Пример:

```
// Продемонстрировать вызов конструктора с помощью ключевого слова this.
using System;
class XYCoord
{
    public int x, y;
    public XYCoord(): this(0, 0)
    {
        Console.WriteLine("В конструкторе XYCoord()") ;
    }
    public XYCoord(XYCoord obj) : this(obj.x, obj.y)
    {
        Console.WriteLine("В конструкторе XYCoord(obj)") ;
    }
    public XYCoord(int i, int j)
    {
        Console.WriteLine("В конструкторе XYCoord(int, int)") ;
        x = i;
        y = j;
    }
}
```

Пример:

```
// Простой пример, демонстрирующий применение инициализаторов
объектов.
using System;
class MyClass
{
    public int Count;
    public string Str;
}
class ObjInitDemo
{
    static void Main()
    {
        // Сконструировать объект типа MyClass, используя
инициализаторы объектов.
        MyClass obj = new MyClass { Count = 100, Str = "Тестирование" };
        Console.WriteLine(obj.Count + " " + obj.Str);
    }
}
```

Пример:

```
// Простой пример, демонстрирующий применение инициализаторов
объектов.
using System;
class MyClass
{
    public int Count;
    public string Str;
}
class ObjInitDemo
{
    static void Main()
    {
        // Сконструировать объект типа MyClass, используя
инициализаторы объектов.
        MyClass obj = new MyClass { Count = 100, Str = "Тестирование" };
        Console.WriteLine(obj.Count + " " + obj.Str);
    }
}
```

Общая форма инициализации:

```
new имя_класса {имя = выражение, имя = выражение, ...}
```

Пример: // Использовать необязательный аргумент, чтобы упростить вызов метода.

```
using System;
class UseOptArgs
{
    // Вывести на экран символьную строку полностью или частично.
    static void Display(string str, int start = 0, int stop = -1)
    {
        if (stop < 0) stop = str.Length;
        // Проверить условие выхода за заданные пределы.
        if (stop > str.Length | start > stop | start < 0) return;
        for (int i = start; i < stop; i++)
        Console.Write(str[i]);
        Console.WriteLine();
    }
    static void Main()
    {
        Display("это простой тест") ;
        Display("это простой тест", 12);
        Display("это простой тест", 4, 14);
    }
}
```

Пример: // Применить именованные аргументы.

```
using System;
class NamedArgsDemo
{
    static bool IsFactor(int val, int divisor)
    {
        if ((val % divisor) == 0) return true;
        return false;
    }
    static void Main()
    {
      // Ниже демонстрируются разные способы вызова метода IsFactor().
        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.") ;
    }
}
```

Пример: static int Main()

Пример: static int Main()

Аргументы командной строки:
static void Main(string[] args) static int Main(string[] args)

Пример: static int Main()

Аргументы командной строки:
static void Main(string[] args) static int Main(string[] args)

Упражнение 4.1

Добавить в любую из предыдущих программ передачу аргументов методу Main(), так чтобы программа выполнялась по слову "Пароль" и выдавала бы сообщение "Пароль не верен" в противном случае.

Упражнение 4.2

С помощью рекурсивного метода вывести аргументы командной строки в обратном порядке.

Пример: // Использовать модификатор static.

```
using System;
class StaticDemo
{
    public static int Val = 100;// Переменная типа static.
    public static int ValDiv2() // Метод типа static.
    {
        return Val/2;
    }
}
class SDemo
{
    static void Main()
    {
        Console.WriteLine("Исходное значение переменной " +
        "StaticDemo.Val равно" + StaticDemo.Val);
        StaticDemo.Val = 8;
        Console.WriteLine("Текущее значение переменной" +
        "StaticDemo.Val равно " + StaticDemo.Val);
        Console.WriteLine("StaticDemo.ValDiv2(): " + StaticDemo.ValDiv2());
    }
}
```

Ограничения:

Ограничения:

- В методе типа static должна отсутствовать ссылка this, поскольку такой метод не выполняется относительно какого-либо объекта.

Ограничения:

- В методе типа static должна отсутствовать ссылка this, поскольку такой метод не выполняется относительно какого-либо объекта.
- В методе типа static допускается непосредственный вызов только других методов типа static, но не метода экземпляра из того самого же класса. Дело в том, что методы экземпляра оперируют конкретными объектами, а метод типа static не вызывается для объекта. Следовательно, у такого метода отсутствуют объекты, которыми он мог бы оперировать.

Ограничения:

- В методе типа static должна отсутствовать ссылка this, поскольку такой метод не выполняется относительно какого-либо объекта.
- В методе типа static допускается непосредственный вызов только других методов типа static, но не метода экземпляра из того самого же класса. Дело в том, что методы экземпляра оперируют конкретными объектами, а метод типа static не вызывается для объекта. Следовательно, у такого метода отсутствуют объекты, которыми он мог бы оперировать.
- Аналогичные ограничения накладываются на данные типа static. Для метода типа static непосредственно доступными оказываются только другие данные типа static, определенные в его классе. Он, в частности, не может оперировать переменной экземпляра своего класса, поскольку у него отсутствуют объекты, которыми он мог бы оперировать.

Пример 1:

```
class StaticError
{
    public int Denom = 3; // обычная переменная экземпляра
    public static int Val = 1024; // статическая переменная
    /* Ошибка! Непосредственный доступ к нестатической переменной из
    статического метода недопустим. */
    static int ValDivDenom() return Val/Denom; // не подлежит
компиляции!
}
```

Пример 1:

```
class StaticError
{
    public int Denom = 3; // обычная переменная экземпляра
    public static int Val = 1024; // статическая переменная
    /* Ошибка! Непосредственный доступ к нестатической переменной из
    статического метода недопустим. */
    static int ValDivDenom() return Val/Denom; // не подлежит
компиляции!
}
```

Пример 2:

```
class AnotherStaticError
{
    void NonStaticMeth() // Нестатический метод.
    {
        Console.WriteLine("В методе NonStaticMeth().") ;
    }
    /* Ошибка! Непосредственный вызов нестатического     метода из
статического метода недопустим. */
    static void staticMeth() NonStaticMeth(); // не подлежит компиляции!
}
```

Пример 1:

```
class StaticError
{
    public int Denom = 3; // обычная переменная экземпляра
    public static int Val = 1024; // статическая переменная
    /* Ошибка! Непосредственный доступ к нестатической переменной из
    статического метода недопустим. */
    static int ValDivDenom() return Val/Denom; // не подлежит
компиляции!
}
```

Упражнение 4.3

Реализовать класс в котором используется поле типа static для подсчета экземпляров существующих объектов данного класса.

Пример 1:

```
class StaticError
{
    public int Denom = 3; // обычная переменная экземпляра
    public static int Val = 1024; // статическая переменная
    /* Ошибка! Непосредственный доступ к нестатической переменной из
    статического метода недопустим. */
    static int ValDivDenom() return Val/Denom; // не подлежит
компиляции!
}
```

Упражнение 4.3

Реализовать класс в котором используется поле типа static для подсчета экземпляров существующих объектов данного класса.

Упражнение 4.4

Реализовать статическую фабрику класса.

Общая форма:

```
static class имя_класса { // ...
```

Общая форма:

```
static class имя_класса { // ...
```

Замечание (1)

Объекты статического класса создавать нельзя. Все члены класса должны быть объявлены как static.

Общая форма:

```
static class имя_класса { // ...
```

Замечание (1)

Объекты статического класса создавать нельзя. Все члены класса должны быть объявлены как static.

Замечание (2)

Несмотря на то, что объекты статического класса создавать нельзя у статического класса может быть статический конструктор.

Общая форма:

```
static class имя_класса { // ...
```

Замечание (1)

Объекты статического класса создавать нельзя. Все члены класса должны быть объявлены как static.

Замечание (2)

Несмотря на то, что объекты статического класса создавать нельзя у статического класса может быть статический конструктор.

Упражнение 4.5

Реализовать статический класс MyMath с методами отсутствующими в классе Math. Например, добавить возможность вычисления определенного интеграла, округления вверх и т.п. (3-4 метода)

Общая форма перегрузки унарного оператора

```
public static возвращаемый_тип operator op(тип_параметра операнд)
{ // операции }
```

Общая форма перегрузки унарного оператора

```
public static возвращаемый_тип operator op(тип_параметра операнд)
{ // операции }
```

Общая форма перегрузки бинарного оператора

```
public static возвращаемый_тип operator op(тип_параметра1 операнд1,
тип_параметра2 операнд2)
{ // операции }
```

Общая форма перегрузки унарного оператора

```
public static возвращаемый_тип operator op(тип_параметра операнд)
{ // операции }
```

Общая форма перегрузки бинарного оператора

```
public static возвращаемый_тип operator op(тип_параметра1 операнд1,
тип_параметра2 операнд2)
{ // операции }
```

Замечание

Вместо "op" подставляется перегружаемый оператор, например "+" или "/"

Общая форма перегрузки унарного оператора

```
public static возвращаемый_тип operator op(тип_параметра операнд)
{ // операции }
```

Общая форма перегрузки бинарного оператора

```
public static возвращаемый_тип operator op(тип_параметра1 операнд1,
тип_параметра2 операнд2)
{ // операции }
```

Замечание

Вместо "op" подставляется перегружаемый оператор, например "+" или "/"

Замечание

Тип операнда унарных операторов должен быть таким же, как и у класса, для которого перегружается оператор. А в бинарных операторах хотя бы один из операндов должен быть такого же типа, как и у его класса.

Упражнение 4.6

Реализовать перегруженный бинарный оператор для сложения и вычитания заданных векторов из пространства $\Re^3$

Упражнение 4.6

Реализовать перегруженный бинарный оператор для сложения и вычитания заданных векторов из пространства $\Re^3$

Упражнение 4.7

Реализовать перегруженные операторы инкремента $(++)$ и декремента $(--)$ для заданных векторов из пространства $\Re^3$

Упражнение 4.6

Реализовать перегруженный бинарный оператор для сложения и вычитания заданных векторов из пространства $\Re^3$

Упражнение 4.7

Реализовать перегруженные операторы инкремента $(++)$ и декремента $(--)$ для заданных векторов из пространства $\Re^3$

Упражнение 4.8

Реализовать перегруженный бинарный оператор "+" для сдвига заданного вектора из пространства $\Re^3$ покоординатно на целое число.

Упражнение 4.6

Реализовать перегруженный бинарный оператор для сложения и вычитания заданных векторов из пространства $\Re^3$

Упражнение 4.7

Реализовать перегруженные операторы инкремента $(++)$ и декремента $(--)$ для заданных векторов из пространства $\Re^3$

Упражнение 4.8

Реализовать перегруженный бинарный оператор "+" для сдвига заданного вектора из пространства $\Re^3$ покоординатно на целое число.

Упражнение 4.9

Реализовать перегруженные бинарные операторы отношений ($<$ и $>$) для сравнения двух симметричных, положительно определенных квадратных матриц A и B $\in \Re^{3\times3}$.

Упражнение 4.6

Реализовать перегруженный бинарный оператор для сложения и вычитания заданных векторов из пространства $\Re^3$

Упражнение 4.7

Реализовать перегруженные операторы инкремента $(++)$ и декремента $(--)$ для заданных векторов из пространства $\Re^3$

Упражнение 4.8

Реализовать перегруженный бинарный оператор "+" для сдвига заданного вектора из пространства $\Re^3$ покоординатно на целое число.

Упражнение 4.9

Реализовать перегруженные бинарные операторы отношений ($<$ и $>$) для сравнения двух симметричных, положительно определенных квадратных матриц A и B $\in \Re^{3\times3}$.

Замечание

Операторы отношения должны перегружаться попарно.

Общая форма:

```
public static bool operator op(тип_параметра операнд)
{
// Возврат логического значения true или false.
}
```

Упражнение 4.10

Реализовать перегрузку операторов true и false для заданной матрицы A $\in \Re^{3\times 3}$ (считать (A) имеет значение false если $\det A = 0$).

Общая форма:

```
public static bool operator op(тип_параметра операнд)
{
// Возврат логического значения true или false.
}
```

Упражнение 4.10

Реализовать перегрузку операторов true и false для заданной матрицы A $\in \Re^{3\times 3}$ (считать (A) имеет значение false если $\det A = 0$).

Упражнение 4.11

Реализовать перегрузку операторов &, | и ! для заданных матриц A и B $\in \Re^{3\times 3}$

Четыре правила необходимых для использования && и ||:

- в классе должна быть произведена перегрузка логических операторов & и |.
- перегружаемые методы операторов & и | должны возвращать значение того же типа, что и у класса, для которого эти операторы перегружаются
- каждый параметр должен содержать ссылку на объект того класса, для которого перегружается логический оператор
- для класса должны быть перегружены операторы true и false

Четыре правила необходимых для использования && и ||:

- в классе должна быть произведена перегрузка логических операторов & и |.
- перегружаемые методы операторов & и | должны возвращать значение того же типа, что и у класса, для которого эти операторы перегружаются
- каждый параметр должен содержать ссылку на объект того класса, для которого перегружается логический оператор
- для класса должны быть перегружены операторы true и false

Четыре правила необходимых для использования && и ||:

- в классе должна быть произведена перегрузка логических операторов & и |.
- перегружаемые методы операторов & и | должны возвращать значение того же типа, что и у класса, для которого эти операторы перегружаются
- каждый параметр должен содержать ссылку на объект того класса, для которого перегружается логический оператор
- для класса должны быть перегружены операторы true и false

Четыре правила необходимых для использования && и ||:

- в классе должна быть произведена перегрузка логических операторов & и |.
- перегружаемые методы операторов & и | должны возвращать значение того же типа, что и у класса, для которого эти операторы перегружаются
- каждый параметр должен содержать ссылку на объект того класса, для которого перегружается логический оператор
- для класса должны быть перегружены операторы true и false

Четыре правила необходимых для использования && и ||:

- в классе должна быть произведена перегрузка логических операторов & и |.
- перегружаемые методы операторов & и | должны возвращать значение того же типа, что и у класса, для которого эти операторы перегружаются
- каждый параметр должен содержать ссылку на объект того класса, для которого перегружается логический оператор
- для класса должны быть перегружены операторы true и false

Упражнение 4.12

Переделать упражнения 4.10 и 4.11 с учетом сформулированных выше правил и добиться реализации операторов && и ||

Две формы операторов преобразования:

```
public static explicit operator целевой_тип(исходный_тип v) {return значение;}
public static implicit operator целевой_тип(исходный_тип v) {return значение;}
```

Две формы операторов преобразования:

public static explicit operator целевой_тип(исходный_тип v) {return значение;}
public static implicit operator целевой_тип(исходный_тип v) {return значение;}

Упражнение 4.13

Реализовать явный оператор преобразования матрицы A($\in \Re^{3\times 3}$) из класса матриц в число типа double ($= \det A$).

Две формы операторов преобразования:

public static explicit operator целевой_тип(исходный_тип v) {return значение;}
public static implicit operator целевой_тип(исходный_тип v) {return значение;}

Упражнение 4.13

Реализовать явный оператор преобразования матрицы A($\in \Re^{3\times 3}$) из класса матриц в число типа double ($= \det A$).

Упражнение 4.14

Реализовать кольцо на множестве целых чисел по модулю 7