

Перегрузка операций



Перегрузка операций

- способ объявления новых операций для типа

Спецификация CLR требует, чтобы перегруженные операторные методы были

- 1) открытыми и статическими
- 2) тип одного из параметров или возвращаемого значения совпадал с типом, в котором определен операторный метод

```
public static возвращаемый_тип operator оператор(параметры)
{ }
```

```
class BigInt
{
    public int Value { get; set; }

    public static BigInt operator +(BigInt operand1,
                                    BigInt operand2)
    {
        return new BigInt
            {Value = operand1.Value + operand2.Value };
    }
    public static bool operator >(BigInt c1, BigInt c2)
    {
        if (c1.Value > c2.Value)
            return true;
        else
            return false;
    }
}

//...
```

```
}
BigInt _i1 = new BigInt { Value = 50 };
BigInt _i2 = new BigInt { Value = 105 };
Console.WriteLine(_i1 > _i2); // false
Console.WriteLine((_i1+_i2).Value); // 155
```

Операции подлежащие перегрузке

- ▶ `+, -, !, ++, --`
- ▶ `true, false` (попарно)
- ▶ `+, -, *, /, %, &, |, ^, <<, >>`
- ▶ `==, !=, <, >, <=, >=` (перегрузка парами)

Операции не подлежащие перегрузке

- ▶ [] (но есть индексатор)
- ▶ () (можно определить новые операторы преобразования)
- ▶ +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>= (но получаем автоматически в случае перегрузки бинарной операции)
- ▶ &&, ||
- ▶ =, ., ?:, ??, ->, =>, f(x), as, checked, unchecked, default, delegate, is, new, sizeof, typeof

правила:

- ▶ префиксные операции ++ и -- перегружаются парами;
- ▶ операции сравнения перегружаются парами: == и != ; < и >; <= и >=.
- ▶ Перегруженные операции обязаны возвращать значения
- ▶ Должны объявляться как public и static
- ▶ префиксная и постфиксная формы операций ++ и --, в отличие от оригинальных операций, семантически НЕ различаются.

► может быть перегружен (т.к. это метод)

```
public static BigInt operator +(BigInt operand1, BigInt  
operand2)  
{  
    return new BigInt  
    { Value = operand1.Value + operand2.Value };  
}  
public static BigInt operator +(BigInt operand1,  
double operand2)  
{  
    return new BigInt  
    { Value = operand1.Value + (int)operand2};  
}
```

если перегружаются операторы == и !=, то для этого требуется переопределить методы Object.Equals() и Object.GetHashCode().

```
class Point2D
{
    float x, y;
    Point2D()
    {
        x = 0;
        y = 0;
    }
    Point2D(Point2D key)
    {
        x = key.x;
        y = key.y;
    }
    // Перегруженные операции обязаны возвращать значения!
    // Должны объявляться как public и static.
    // При этом префиксная и постфиксная формы операций ++ и --,
    // в отличие от оригинальных операций, семантически НЕ различаются.
    // Каждая из этих операций может быть объявлена либо как префиксная:
    public static Point2D operator ++(Point2D par)
    {
        par.x++;
        par.y++;
        return par;
    }
    // либо как постфиксная!
    public static Point2D operator --(Point2D par)
    {
        Point2D tmp = new Point2D(par);
        // Скопировали старое значение.
        par.x--;
        par.y--;
        // Модифицировали исходное значение. Но возвращаем старое!
        return tmp;
    }
}
```



```
public static bool operator == ( Point2D a, Point2D b )  
{ return a.Equals( b ); }
```

```
public static bool operator !=(Point2D a, Point2D b)  
{ return ! a.Equals( b ); }
```

```
// Бинарные операции также обязаны возвращать значения!  
public static Point2D operator +(Point2D par1, Point2D par2)  
{  
    return new Point2D(par1.x + par2.x, par1.y + par2.y);  
}  
  
// Point2D + float  
public static Point2D operator +(Point2D par1, float val)  
{  
    return new Point2D(par1.x + val, par1.y + val);  
}  
  
// float + Point2D  
public static Point2D operator +(float val, Point2D par1)  
{  
    return new Point2D(val + par1.x, val + par1.y);  
}
```

```
// Перегрузка булевских операторов. Это ПАРНЫЕ операторы.
```

```
public static bool operator true(Point2D par)
{
    if (par.x == 1.0F && par.y == 1.0F) return true;
    else return false;
}

public static bool operator false(Point2D par)
{
    if (par.x == 0.0F && par.y == 0.0F) return false;
    else return true;
}
```

Point2D.false(x)? x: Point2D.!(x, y)

```
public static Point2D operator | (Point2D par1, Point2D par2)
{
    if (par1) return par1;
    if (par2) return par2;
    else return new Point2D(-1.0F, -1.0F);
}

public static Point2D operator & (Point2D par1, Point2D par2)
{
    if (par1 && par2) return par1;
    else return new Point2D(-1.0F, -1.0F);
}
```

C:\Windows\system32\cmd.exe

float значение >> 5,6

float значение >> 3,4

float значение >> 3,4

Are You sure to change the y value of object of Point2D? (y/n) >> n

p0.x == 0, p0.y == 0

p1.x == 5,6, p1.y == 3,4

p2.x == 5,6, p2.y == 3,4

#####

false!

true!

true!

true!

true!

p0.x == 1, p0.y == 1

p1.x == 6,6, p1.y == 4,4

p2.x == 5,6, p2.y == 3,4

Для продолжения нажмите любую клавишу . . .

Операции преобразования типа

- ▶ преобразует объект исходного класса в другой тип
- ▶ явная и неявная форма - будет ли этот алгоритм выполняться неявно или необходимо будет явным образом указывать необходимость соответствующего преобразования.

▶ implicit operator тип (параметр)
// неявное преобразование

преобразование вызывается
автоматически

в который выполняется преобразование

▶ explicit operator тип (параметр)
// явное преобразование

преобразование вызывается в
том случае, когда выполняется
приведение типов

тип, который преобразуется

Преобразуемые типы не должны быть связаны
отношениями наследования

```
public int x, y, z;
```

```
public Point3D()
```

```
{
```

```
    x = 0;
```

```
    y = 0;
```

```
    z = 0;
```

```
}
```

```
public Point3D(int xKey, int yKey, int zKey)
```

```
{
```

```
    x = xKey;
```

```
    y = yKey;
```

```
    z = zKey;
```

```
}
```

```
// Операторная функция, в которой реализуется алгоритм преобразования  
// значения типа Point2D в значение типа Point3D.
```

```
// Это преобразование осуществляется НЕЯВНО.
```

```
public static implicit operator Point3D(Point2D p2d)
```

```
{
```

```
    Point3D p3d = new Point3D();
```

```
    p3d.x = p2d.x;
```

```
    p3d.y = p2d.y;
```

```
    p3d.z = 0;
```

```
    return p3d;
```

```
}
```

```
public Point2D()  
{  
    x = 0;  
    y = 0;  
}
```

```
public Point2D(int xKey, int yKey)  
{  
    x = xKey;  
    y = yKey;  
}
```

```
// Операторная функция, в которой реализуется алгоритм преобразования  
// значения типа Point3D в значение типа Point2D. Это преобразование  
// осуществляется с ЯВНЫМ указанием необходимости преобразования.  
// Принятие решения относительно присутствия в объявлении ключевого  
// слова explicit вместо implicit оправдывается тем, что это  
// преобразование сопровождается потерей информации. По мнению  
// разработчика классов об этом обстоятельстве следует напоминать  
// каждый раз, когда данное преобразование случается в программе.
```

```
public static explicit operator Point2D(Point3D p3d)  
{  
    Point2D p2d = new Point2D();  
    p2d.x = p3d.x;  
    p2d.y = p3d.y;  
    return p2d;  
}
```



```
class TestClass
{
    static void Main(string[] args)
    {
        Point2D p2d = new Point2D(125, 125);
        Point3D p3d; // Сейчас это только ссылка!
        // Этой ссылке присваивается значение в результате
        // НЕЯВНОГО преобразования значения типа Point2D к типу Point3D
        p3d = p2d;

        // Изменили значения полей объекта.
        p3d.x = p3d.x * 2;
        p3d.y = p3d.y * 2;
        p3d.z = 125; // появилась новая информация,
        // которая неизбежно будет потеряна в случае присвоения значения типа Point3D
        // значению типа Point2D. Ключевое слово explicit в объявлении соответствующего
        // метода преобразования приводит к тому, что программист всякий раз вынужден
        // повторять, что он в курсе возможных последствий.
        p2d = (Point2D)p3d;
    }
}
```

- Ключевые слова `implicit` и `explicit` в сигнатуру не включаются

Ограничения на операторы преобразования

- ▶ Исходный или целевой тип преобразования должен относиться к классу, для которого объявлено данное преобразование
- ▶ Нельзя указывать преобразование в/из класс `object` или же из этого класса
- ▶ Для одних типов данных нельзя указывать одновременно явное и неявное преобразование
- ▶ Нельзя указывать преобразование базового класса в производный класс
- ▶ Нельзя указывать преобразование в/из интерфейс

Вложенные типы

- Тип, определенный внутри класса называется вложенным типом

МОЛЧАНИЮ ЯВЛЯЮТСЯ `private`

```
class Person
{
    public class Date
    {
        public Date() { }
    }
}
```

```
Person.Date today = new Person.Date();
```

- ▶ Вложенный тип может получить доступ к внешнему типу, а внутренний тип — к внешнему
- ▶ Вложенный тип имеет доступ ко всем членам, которые доступны вмещающему типу



► Вложение или включение классов модель включения-делегирования.

```
class Hora
{
    public void Марш()
    {
        Console.WriteLine( "топ!!" );
    }
}
```

```
class Человек
{
    public Человек()
    {
        левая = new Hora();
        правая = new Hora();
    }
    public void Побежали()
    {
        левая.Марш();
        правая.Марш();
    }
}
```

```
    Hora левая, правая;
```

```
class Class1
{
    static void Main()
    {
        Человек Вася = new Человек();
        Вася.Побежали();
    }
}
```

C:\Windows\system32\cmd.exe

```
топ!!
топ!!
Для продолжения нажмите любую клавишу
```

Вложенные объекты

```
public class Date
{
    public Date() { }
}
class Person
{
    Date birthday;

}
```

```
Person anna = new Person();
anna.birthday = null;
```

#region

```
class Person
{
    #region Peson filed
    public Date birthday;
    #endregion
    #region Person construc
    #endregion
    #region Person operator
    #endregion
    #region Person proterty
    #endregion
}
```

```
class Person
{
    Peson filed
    Person construc
    Person operator
    Person proterty
}
```

- Сворачивание и разворачивания блоков кода