Лекция 6. Перегрузка операторов. Индексаторы. Операторы приведения типов.

Половикова О.Н.

Перегрузка операторов

Перегрузка оператора – это реализация своего собственного функционала этого оператора для конкретного класса.

С помощью **перегрузки операторов** можно указать поведение оператора для операндов определяемого пользователем типа.

Помните про правила перегрузки!

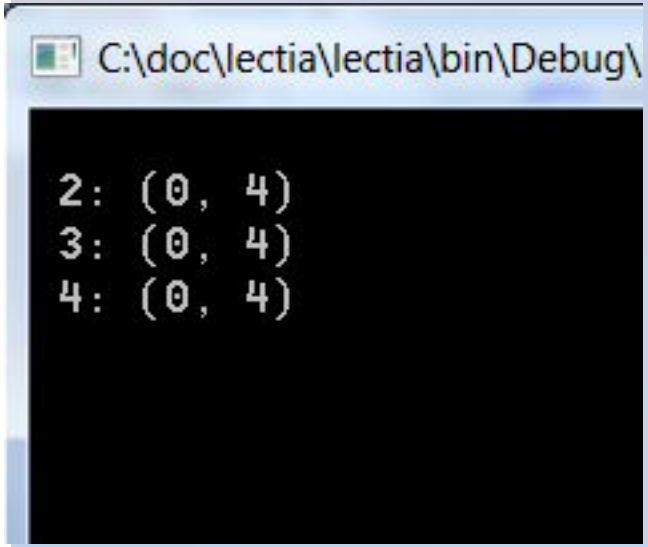
```
        public static void Main(string[] args)
        {
            math.Complex A = new math.Complex(-3, 4);
            math.Complex B = new math.Complex(2, 5);
            Console.WriteLine((A+B).ToString());
            Console.WriteLine((A+3.6).ToString());
            Console.WriteLine((3.6+B-A-7.8-(-A)+(-B)).ToString());
            Console.ReadKey(true);
        }
    }
    class Complex...
}
namespace math{
class Complex{
        double re, im;
        public override string ToString()
        {
            return string.Format("[Complex Re={0}, Im={1}]", re, im);
        }
        public void print(){
            Console.WriteLine("\n({0},{1})", re, im);
        }
        public Complex (double re=0.0, double im=0.0){
            this.re=re;
            this.im=im;
        }
        public static Complex operator-(Complex A){ return new Complex(-A.re, -A.im);}

        public  static Complex operator+(Complex A, Complex B){ return new Complex(B.re+A.re, B.im+A.im);}
        public  static Complex operator-(Complex A, Complex B){ return new Complex(A.re-B.re, A.im-B.im);}

        public  static Complex operator+(Complex A, double B){ return new Complex(A.re+B, A.im+0);}
        public  static Complex operator+( double B, Complex A){ return  new Complex(A.re+B, A.im+0);}

        public  static Complex operator-(Complex A, double B){ return new Complex(A.re-B, A.im-0);}
        public  static Complex operator-( double B, Complex A){ return  new Complex(B-A.re, 0-A.im);}

    }
```

Операция C#	Возможность перегрузки
+, -, !, ++, —, true, false	Этот набор унарных операций может быть перегружен
+, -, *, /, %, &, |, ^, <<, >>	Эти бинарные операции могут быть перегружены
==, !=, <, >, <=, >=	Эти операции сравнения могут быть перегружены. C# требует совместной перегрузки «подобных» операций (т.е. < и >, <= и >=, == и !=)
[ ]	Операция [] не может быть перегружена. Аналогичную функциональность предлагают индексаторы
( )	Операция () не может быть перегружена, но ту же функциональность предоставляют специальные методы преобразования
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	Сокращенные операции присваивания не могут перегружаться; однако вы получаете их автоматически, перегружая соответствующую бинарную операцию

План

- Перегрузка операторов (true, false, логических);
- Индексаторы
- Операторы приведения типов

```
public class Counter
{
    string name { set; get; }
    int max, min, pos;
    public void print()
    {
        Console.Write("\n {0}: ({1}, {2})", pos, min,max);
    }
    public Counter(int pos = 0, int min = 0, int max = 0){
        this.pos = pos;
        this.min = min;
        this.max = max;
    }
    public static Counter operator ++(Counter A) {
        Counter temp = new Counter(A.pos, A.min, A.max);
        if (temp.pos < temp.max) temp.pos++;
        return temp;
    }
    public static Counter operator --(Counter A)
    {
        Counter temp = new Counter(A.pos, A.min, A.max);
        if (temp.pos > temp.min) temp.pos--;
        return temp;
    }
```



Перегрузка true, false

```
    //если счётчик "не завершён"
    public static Boolean operator true(Counter A)
    {
        if (A.pos != A.min && A.pos != A.max) return true;
        else return false;

    }
    public static Boolean operator false(Counter A)
    {
        if (A.pos != A.min && A.pos != A.max) return false;
        else return true;

    }
}
class Program
{
    static void Main(string[] args)
    {
        Counter A = new Counter(1, 0, 4);
        for (;;)
        {
            if (A){
                A++;
                A.print();
            }else break;
        }
        Console.ReadKey(true);
```

Перегрузка оператора «!»

```
    //если счётчик "не завершён"
    public static Boolean operator true(Counter A){
        if (A.pos != A.min && A.pos != A.max) return true;
        else return false;

    }
    public static Boolean operator false(Counter A){
        if (A.pos != A.min && A.pos != A.max) return false;
        else return true;
    }
    // правильная реализация
    public static Boolean operator !(Counter A){
        if (A.pos != A.min && A.pos != A.max) return false;
        else return true;
    }
}
class Program
{
    static void Main(string[] args)
    {
        Counter A = new Counter(1, -10, 4);
        for (;;) {
            if (!A) break;
            else {
                A--;
                A.print();
            }
```

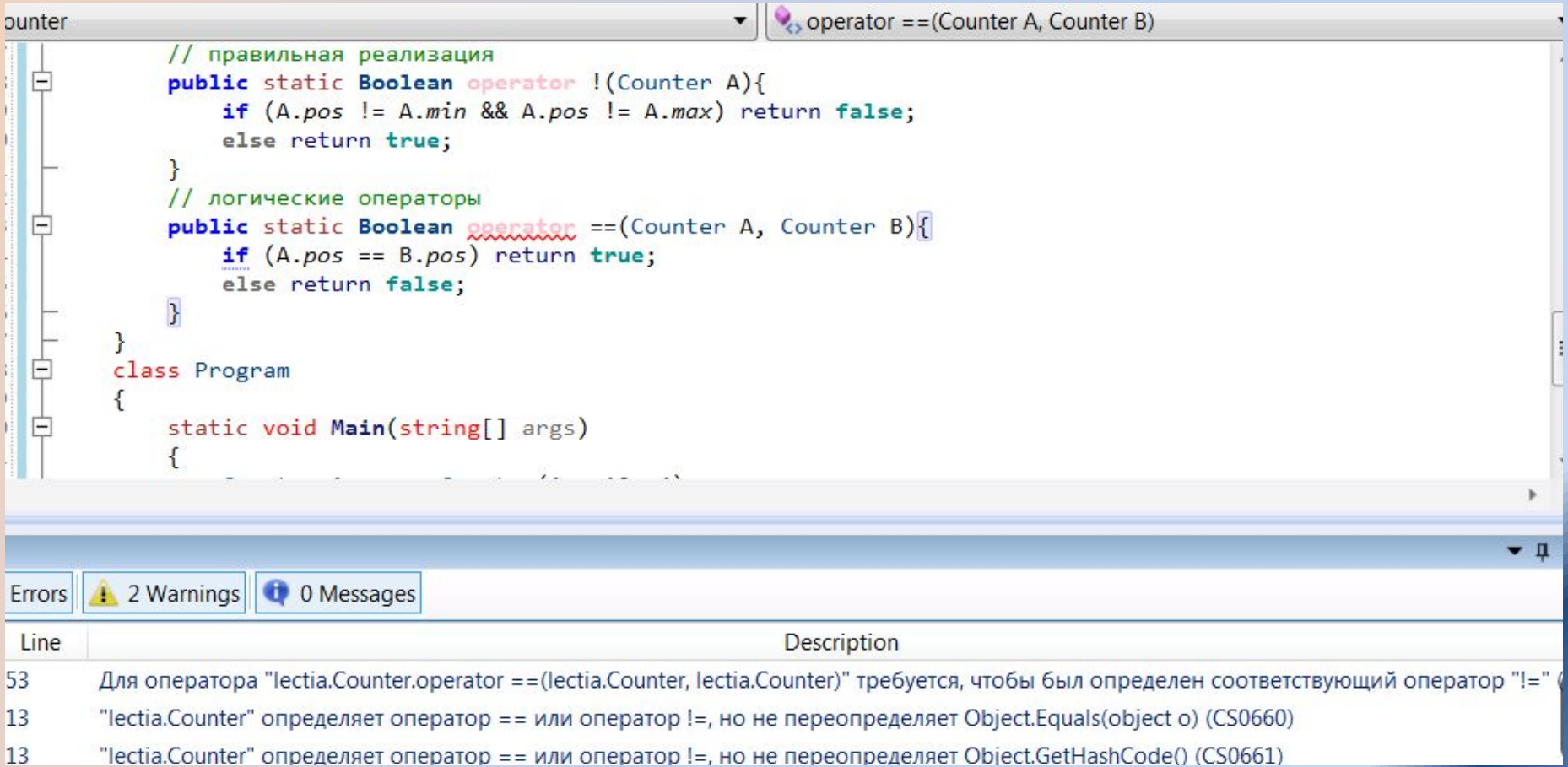
C:\doc\lectia\lectia\bin\Debug\lectia.exe

```
0: (-10, 4)
-1: (-10, 4)
-2: (-10, 4)
-3: (-10, 4)
-4: (-10, 4)
-5: (-10, 4)
-6: (-10, 4)
-7: (-10, 4)
-8: (-10, 4)
-9: (-10, 4)
-10: (-10, 4)
```

Логические операторы: ==, !=

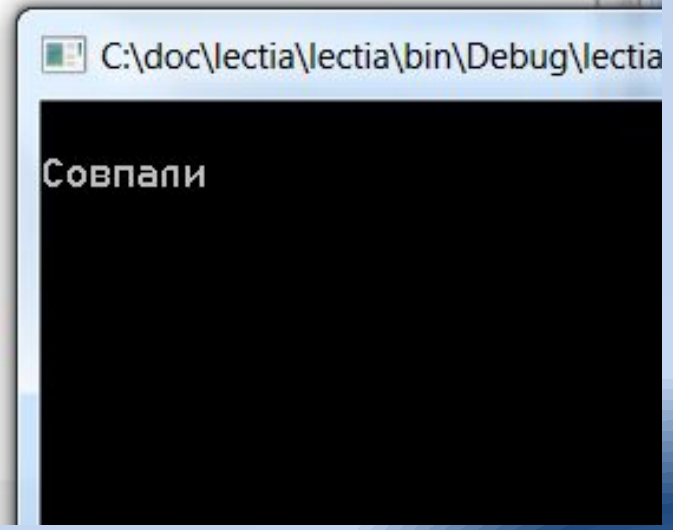
Некоторые **логические операторы перегружаются только парами**:

Если перегружен оператор «==», необходимо перегрузить и оператор «!=». Аналогично для <, >, <=, >=.



Логические операторы: ==, !=

```
    // логические операторы
    public static Boolean operator ==(Counter A, Counter B){
        if (A.pos == B.pos) return true;
        else return false;
    }

    public static Boolean operator !=(Counter A, Counter B){
        return !(A==B);
    }
}
class Program
{
    static void Main(string[] args)
    {
        Counter A = new Counter(1, -10, 4);
        Counter B = new Counter(2, -10, 4); B--;
        if (A == B) Console.Write("\nСовпали");

     //   if (A && B) Console.Write("\n оба счётчика рабочие");
        Console.ReadKey(true);
    }
}
```



Счётчики равны, если имеют равные показатели.

Нельзя перегружать

Перегружать можно только операторы, перечисленные выше.
В частности, невозможно перегрузить доступ к члену, вызов метода или

=, &&, ||, ?:, =>, checked, unchecked и new операторы.

Индексаторы

Индексаторы позволяют «нумеровать» объекты и обращаться к данным по индексу. Фактически с помощью индексаторов можно работать с объектам как с массивом.

```
возвращаемый_тип this [Тип параметр1, ...]
{
    get { ... }
    set { ... }
}
```

Синтаксис аналогичен созданию свойств со стандартными блоками get и set, которые возвращают и присваивают значение.

Индексаторы. Пример

```
class Matrix
{
    private int[,] array = null;

    public int N{
        get {return array.GetLength(0);}
    }
    public int M{
        get {return array.GetLength(1);}
    }
    public Matrix(int n, int m)
    {
        array = new int[n, m];
    }
    public void print(){
        Console.Write ("\n");
        for (int i=0; i<N; i++){
            Console.WriteLine();
            for (int j=0; j<M; j++)
                    Console.Write (" {0,3:D}",array[i, j]);
         }
    }
    public void set(int max){
        Random R = new Random(DateTime.Now.Millisecond);
        for (int i=0; i<N; i++)
            for (int j=0; j<M; j++)
                array[i, j] = R.Next(-max, max);
    }
```

Рассмотрим работу индексаторов на классе Matrix

Индексаторы. Пример (продолжение)

```
        }
        public int this[int i, int j]
        {
            get{  if (i < N && j < M) return array[i, j];else return array[0, 0]; }
            set{  if (i < N && j < M) array[i,j] = value;}
        }
       public int this[int i]
        {
            get {
                int a = 0;
                for (int j = 0; j < N; j++ ) a += array[i, j];
                return a;
            }
         }
}
  class Program{
       public static void Main(string[] args)
       {
            Matrix A = new Matrix(6,6);
            A.print();
            A.set(10);
            //элементы на главной диагонали 1
            for(int i=0; i<A.N; i++) A[i,i] = 1;
            A.print();

            Console.ReadKey(true);
       }
```

C:\doc\lectia\matrix\bin\Debug\matrix.exe

```
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0

  1  -2  -5 -10  -2 -10
  6   1   2  -5 -10   5
  6   2   1  -1  -4   4
  7  -5   8   1   4   4
  2 -10  -9  -5   1   4
 -3   5   7  -2  -5   1
```

Создано два индексатора для данного класса

Индексаторы

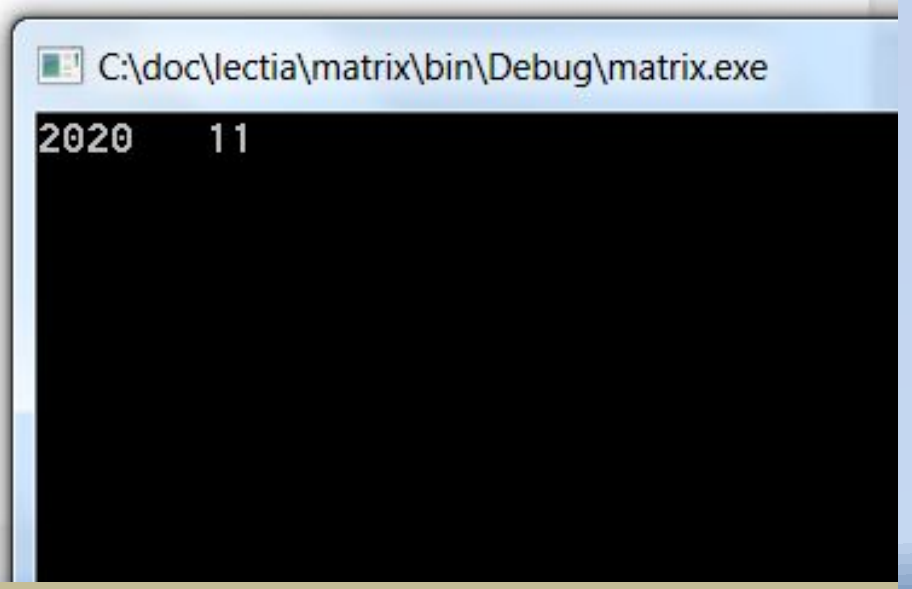
Конструкция public int this[int i, int j] представляет индексатор.

Индексатор получает набор индексов в виде параметров. Параметры (индексы) могут не быть типа *int*. Например, строка:

```
public string this[string  a]
        {  get{
                if (a = “….")
                    return …;
                else …;
            }

        }
```

Индексаторы. Пример

```
class Temp{
    DateTime DateB;
    public Temp(){
        DateB = new DateTime();
        DateB = DateTime.Now;
    }
    public string this[string a]{
        get {if (a.Equals("y")) return DateB.Year.ToString();
            else return DateB.Month.ToString();
        }
    }
}
class Program{
    public static void Main(string[] args)
    {
        Temp A = new Temp();
        Console.Write(A["y"]+ "    "+A["m"]);

        Console.ReadKey(true);
    }
}
```



Следует учитывать, что индексатор не может быть статическим и применяется только к экземпляру класса.
Можно ограничивать доступ к блокам get и set, используя модификаторы доступа. Например, сделаем блок set приватным. В этом случае изменить объект через индексатор снаружи будет невозможно.
Количество индексаторов определяется потребностями разработчиков (можно перегружать).

Операторы преобразования типов (операторы приведения типов)

или операторы пользовательского преобразования типов

Если создали свой класс (новый тип данных, *A*) можно задать (определить) неявное или явное преобразование в другой тип (*B*) или из другого типа в созданный нами тип:

A-->B;

B-->A

Примеры преобразований для базовых типов:

```
int a = (int)3.14; //явное
float f = 3.14f;
double d = f; //неявное
```

Операторы преобразования типов (операторы приведения типов)

Существуют две формы операторов преобразования: явная и неявная:

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

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

где целевой_тип обозначает тот тип, в который выполняется преобразование;

исходный_тип -- тот тип, который преобразуется;

Операторы преобразования **возвращают данные, имеющие целевой_тип**, **возвращать другой тип данных нельзя**.

Операторы преобразования типов. Пример

```
    // из матрицы в число - явно
    public static explicit operator int(Matrix A){
     return A[0,0];
    }
    //из числа в матрицу -  явно
     public static explicit operator Matrix(int A){
     Matrix B = new Matrix(2,2);
     for (int i=0; i<B.N; i++)
             for (int j=0; j<B.M; j++)
                 B.array[i, j] = A;

     return B;
    }
}

class Temp...
class Program{
    public static void Main(string[] args)
    {
        Matrix A = new Matrix(2,3);
        int i = (int)A;
        Console.Write(" \n число:{0}", i);
        Console.Write(" \n матрица\n");
        i = 5;
        ((Matrix)i).print();
        Console.ReadKey(true);
    }
```

```
C:\doc\lectia\matrix\bin\Debug\matrix.exe

число:0
матрица


  5   5
  5   5
```

Операторы явного преобразования для класса Matrix

Операторы преобразования типов. Пример

```
    // из матрицы в число - неявно
    public static implicit operator int(Matrix A){
     return A[0,0];
    }
    //из числа в матрицу -  неявно
     public static implicit operator Matrix(int A){
     Matrix B = new Matrix(Math.Abs(A), Math.Abs(A));
     for (int i=0; i<B.N; i++)
             for (int j=0; j<B.M; j++)
                 B.array[i, j] = A;

     return B;
    }
}

class Temp...
class Program{
    public static void Main(string[] args)
    {
        Console.Write(" \n матрица\n");
        int i = 7;
        Matrix B = i;
        B.print();
        i = B;
        Console.Write(" \n число:{0}", i);
        Console.ReadKey(true);
    }
```

```
C:\doc\lectia\matrix\bin\Debug\matrix.exe

матрица

  7   7   7   7   7   7   7
  7   7   7   7   7   7   7
  7   7   7   7   7   7   7
  7   7   7   7   7   7   7
  7   7   7   7   7   7   7
  7   7   7   7   7   7   7
  7   7   7   7   7   7   7
число:7
```

Операторы неявного преобразования для класса Matrix

Операторы преобразования типов. Пример ошибок

Matrix | implicit operator Matrix(int A)

```
58          //из числа в матрицу -  неявно
59          public static implicit operator Matrix(int A){
60          Matrix B = new Matrix(Math.Abs(A), Math.Abs(A));
61          for (int i=0; i<B.N; i++)
62                  for (int j=0; j<B.M; j++)
63                      B.array[i, j] = A;
64
65          return B;
66          }
67      }
68
69      class Temp...
81      class Program{
82          public static void Main(string[] args)
83          {
84              Console.Write(" \n матрица\n");
85              int i = 7;
86              i.print();
87              Console.Write(" \n число:{0}", i);
88              Console.ReadKey(true);
```

Errors

1 Errors | 0 Warnings | 0 Messages

!	Line	Description	File	Path
⊗	86	"int" не содержит определение для "print". Не удалось найти метод расширения "...	Program.cs	C:\doc\lectia\...

Операторы преобразования типов

Для одних и тех же исходных и целевых типов данных **нельзя** указывать оператор преобразования одновременно **в явной и неявной форме**.

Неявные преобразования не требуют специального синтаксиса для вызова и могут происходить в различных ситуациях, **например при присваивании**.

Предопределенные неявные преобразования на C# всегда завершаются успешно (для базовых типов) и никогда не вызывают исключение или потерю данных.

Неявные пользовательские преобразования должны вести себя таким же образом.

Если пользовательское преобразование может вызвать исключение или привести к потере данных, определите его как явное преобразование.

Операторы преобразования типов. Ограничения

1. **Исходный или целевой тип преобразования должен относиться к классу, для которого объявлено данное преобразование**.
2. Для одних и тех же исходных и целевых типов данных нельзя указывать одновременно явное и неявное преобразование.
3. Нельзя указывать преобразование базового класса в производный класс (наследование).