Турсунов Баходурхон

#13. .NET Object-Oriented Programming (OOP). Methods Overloading. Operator Overloading. Extension Methods.

1. **Какие основные правила следует помнить при перегрузке операторов в C#? Приведите пример ситуации, в которой перегрузка операторов была бы полезна.**

Определенный пользователем тип может быть перегружен предопределенным оператором, т.е. тип может предоставлять пользовательскую реализацию операции в случае, если один или оба операнда относятся к одному типу.

Операторы могут быть перегружены с помощью ключевого слова **operator.**

Оператор перегрузки должен удовлетворять следующему правилу:

* Она включает в себя модификатор public и static.

Следующей особенностью перегрузки операторов является перегрузка неявного и явного преобразования. Таким образом, пользовательский тип может определять пользовательское преобразование или другой тип.

Для определения неявного или явного преобразования надо использовать слова operator, implicit или explicit

1. **Опишите, что такое методы расширения и почему они используются в C#.\**

Методы расширения позволяют добавлять новые методы в уже существующие типы без создания нового производного класса. Эта функциональность бывает особенно полезна, когда нам хочется добавить в некоторый тип новый метод, но сам тип (класс или структура) мы изменить не можем, поскольку у нас нет доступа к исходному коду. Либо если мы не можем использовать стандартный механизм наследования, например, если классы определены с модификатором **sealed.**

Применение методов расширения очень удобно, но при этом надо запомнить, что метод расширения никогда не будет вызван, если он имеет ту же сигнатуру, что и метод, изначально определенный в типе. Так же методы расширения действуют на уровне пространства имен. То есть, если добавить проект в другое пространство имет, метод не будет применяться к строкам, и в этом случае надо будет подключить пространство имен метода через директиву **using.**

1. Приведите фрагмент кода, демонстрирующий использование методов расширения в классе System.String.

public static class StringExtensions

{

public static string RemoveSpaces(this string input)

{

return input.Replace(" ", "");

}

}

class Program

{

static void Main()

{

string originalString = "Hello, World!";

// Использование метода расширения

string stringWithoutSpaces = originalString.RemoveSpaces();

Console.WriteLine($"Original String: {originalString}");

Console.WriteLine($"String Without Spaces: {stringWithoutSpaces}");

}

}

1. **Как вы можете отличить перегрузку методов от перегрузки операторов с точки зрения реализации и вариантов использования?**

Практическое занятие:

Очень просто. При перегрузке методов никакие ключевые слова не пишутся, а просто переопределяются входные параметры для текущего метода.

А при перегрузке операторов пишется ключевое слово **operator.**

1. **Реализуйте класс, представляющий комплексное число, на языке C#. Перегрузите операторы сложения, вычитания и умножения для работы с комплексными числами. Создайте приложение, демонстрирующее перегрузку операторов.**

using System;

public class ComplexNumber

{

public double Real { get; set; }

public double Imaginary { get; set; }

public ComplexNumber(double real, double imaginary)

{

Real = real;

Imaginary = imaginary;

}

// Перегрузка оператора сложения

public static ComplexNumber operator +(ComplexNumber a, ComplexNumber b)

{

return new ComplexNumber(a.Real + b.Real, a.Imaginary + b.Imaginary);

}

// Перегрузка оператора вычитания

public static ComplexNumber operator -(ComplexNumber a, ComplexNumber b)

{

return new ComplexNumber(a.Real - b.Real, a.Imaginary - b.Imaginary);

}

// Перегрузка оператора умножения

public static ComplexNumber operator \*(ComplexNumber a, ComplexNumber b)

{

return new ComplexNumber(

a.Real \* b.Real - a.Imaginary \* b.Imaginary,

a.Real \* b.Imaginary + a.Imaginary \* b.Real

);

}

// Перегрузка метода ToString для удобного вывода комплексных чисел

public override string ToString()

{

return $"{Real} + {Imaginary}i";

}

}

class Program

{

static void Main()

{

ComplexNumber complex1 = new ComplexNumber(2, 3);

ComplexNumber complex2 = new ComplexNumber(1, 4);

// Использование перегруженных операторов

ComplexNumber sum = complex1 + complex2;

ComplexNumber difference = complex1 - complex2;

ComplexNumber product = complex1 \* complex2;

// Вывод результатов

Console.WriteLine($"Complex1: {complex1}");

Console.WriteLine($"Complex2: {complex2}");

Console.WriteLine($"Sum: {sum}");

Console.WriteLine($"Difference: {difference}");

Console.WriteLine($"Product: {product}");

}

}

1. **Напишите метод расширения для класса массив чисел, который вычисляет среднее значение элементов списка. Используйте этот метод расширения для нахождения среднего значения списка целых чисел и списка двоек в приведенном в примере приложении.**

using System;

using System.Linq;

public static class ArrayExtensions

{

public static double CalculateAverage(this int[] array)

{

if (array == null || array.Length == 0)

throw new ArgumentException("Array is null or empty");

return array.Sum() / (double)array.Length;

}

}

class Program

{

static void Main()

{

int[] integerArray = { 1, 2, 3, 4, 5 };

double averageInteger = integerArray.CalculateAverage();

Console.WriteLine($"Average of integerArray: {averageInteger}");

int[] intArray = { 2,2,2,2,2 };

double averageDouble = intArray.CalculateAverage();

Console.WriteLine($"Average of doubleArray: {averageDouble}");

}

}

Practice

1. Реализуйте класс для представления комплексных чисел с действительной и мнимой частями. Перегрузите операторы сложения, вычитания и умножения для этого класса. Создайте приложение, демонстрирующее использование этих перегруженных операторов.

namespace HW13\_1  
{  
  
 class Program  
 {  
 static void Main()  
 {  
 ComplexNumber complex1 = new ComplexNumber(2, 3);  
 ComplexNumber complex2 = new ComplexNumber(1, 4);  
  
 ComplexNumber sum = complex1 + complex2;  
 ComplexNumber difference = complex1 - complex2;  
 ComplexNumber product = complex1 \* complex2;  
  
 Console.WriteLine($"Sum: {sum}");  
 Console.WriteLine($"Difference: {difference}");  
 Console.WriteLine($"Product: {product}");  
 }  
  
 public class ComplexNumber  
 {  
 public double Real { get; set; }  
 public double Imaginary { get; set; }  
  
 public ComplexNumber(double real, double imaginary)  
 {  
 Real = real;  
 Imaginary = imaginary;  
 }  
  
 public static ComplexNumber operator +(ComplexNumber a, ComplexNumber b)  
 {  
 return new ComplexNumber(a.Real + b.Real, a.Imaginary + b.Imaginary);  
 }  
  
 public static ComplexNumber operator -(ComplexNumber a, ComplexNumber b)  
 {  
 return new ComplexNumber(a.Real - b.Real, a.Imaginary - b.Imaginary);  
 }  
  
 public static ComplexNumber operator \*(ComplexNumber a, ComplexNumber b)  
 {  
 return new ComplexNumber(  
 a.Real \* b.Real - a.Imaginary \* b.Imaginary,  
 a.Real \* b.Imaginary + a.Imaginary \* b.Real  
 );  
 }  
  
 public override string ToString()  
 {  
 return $"{Real} + {Imaginary}i";  
 }  
 }  
 }  
  
}

2. Реализуйте класс с именем для представления матрицы 2x2 из целых чисел. Перегрузите операторы сложения и умножения для этого класса. Создайте приложение, демонстрирующее использование этих перегруженных операторов.

namespace HW13\_2  
{  
 using System;  
  
 public class Matrix2x2  
 {  
 public int[,] Elements { get; }  
  
 public Matrix2x2(int a, int b, int c, int d)  
 {  
 Elements = new int[2, 2] { { a, b }, { c, d } };  
 }  
  
 public static Matrix2x2 operator +(Matrix2x2 m1, Matrix2x2 m2)  
 {  
 return new Matrix2x2(  
 m1.Elements[0, 0] + m2.Elements[0, 0],  
 m1.Elements[0, 1] + m2.Elements[0, 1],  
 m1.Elements[1, 0] + m2.Elements[1, 0],  
 m1.Elements[1, 1] + m2.Elements[1, 1]  
 );  
 }  
  
 public static Matrix2x2 operator \*(Matrix2x2 m1, Matrix2x2 m2)  
 {  
 return new Matrix2x2(  
 m1.Elements[0, 0] \* m2.Elements[0, 0] + m1.Elements[0, 1] \* m2.Elements[1, 0],  
 m1.Elements[0, 0] \* m2.Elements[0, 1] + m1.Elements[0, 1] \* m2.Elements[1, 1],  
 m1.Elements[1, 0] \* m2.Elements[0, 0] + m1.Elements[1, 1] \* m2.Elements[1, 0],  
 m1.Elements[1, 0] \* m2.Elements[0, 1] + m1.Elements[1, 1] \* m2.Elements[1, 1]  
 );  
 }  
  
 public override string ToString()  
 {  
 return $"[{Elements[0, 0]}, {Elements[0, 1]}\n {Elements[1, 0]}, {Elements[1, 1]}]";  
 }  
 }  
  
 class Program  
 {  
 static void Main()  
 {  
 Matrix2x2 matrix1 = new Matrix2x2(1, 2, 3, 4);  
 Matrix2x2 matrix2 = new Matrix2x2(1, 2, 3, 4);  
  
 Matrix2x2 sum = matrix1 + matrix2;  
 Matrix2x2 product = matrix1 \* matrix2;  
  
 Console.WriteLine($"Sum:\n{sum}");  
 Console.WriteLine($"Product:\n{product}");  
 }  
 }  
  
}

3. Напишите метод расширения для массива, который вычисляет и возвращает сумму всех элементов в списке. Используйте этот метод расширения в приложении для нахождения суммы элементов в списке целых чисел.

using System;  
using System.Linq;  
  
public static class ArrayExtensions  
{  
 public static int SumAll(this int[] array)  
 {  
 return array.Sum();  
 }  
}  
  
class Program  
{  
 static void Main()  
 {  
 int[] numbers = { 1, 2, 3, 4, 5 };  
 int sum = numbers.SumAll();  
  
 Console.WriteLine($"Sum of array elements: {sum}");  
 }  
}

4. Напишите метод расширения для System.String, который подсчитывает количество вхождений указанной подстроки в строку. Используйте этот метод расширения для поиска вхождений определенной подстроки в заданную строку.

using System;  
  
public static class StringExtensions  
{  
 public static int CountOccurrences(this string source, string substring)  
 {  
 int count = 0;  
 int index = 0;  
  
 while ((index = source.IndexOf(substring, index, StringComparison.OrdinalIgnoreCase)) != -1)  
 {  
 index += substring.Length;  
 count++;  
 }  
  
 return count;  
 }  
}  
  
class Program  
{  
 static void Main()  
 {  
 string inputString = "Programming is fun. Programming is creative. Programming is powerful.";  
 string substringToSearch = "Programming";  
  
 int occurrences = inputString.CountOccurrences(substringToSearch);  
  
 Console.WriteLine($"Occurrences of '{substringToSearch}': {occurrences}");  
 }  
}