Лабораторная работа №3

Перегрузка методов и операторов

Цель работы: изучить механизм перегрузки на языке C#.

Теоретическая часть

Перегрузка

Внутри одного класса можно определить несколько методов с одним именем, которые должны отличаться количеством и/или типом входных параметров. Данный механизм называется перегрузкой (overload). Нужный метод выбирается на этапе компиляции.

Рассмотрим пример создания нескольких перегрузок метода `Info`, который выводит информацию о пользователе.

```
public void Info()
{
    Console.WriteLine("Пустой метод\n");
}
Ссылок: 0
public void Info(string name)
{
    Console.WriteLine($"Имя: {name}");
}
Ссылок: 0
public void Info(string name, string lastName)
{
    Console.WriteLine($"Имя: {name}\nФамилия: {lastName}");
}
Ссылок: 0
public void Info(string name, string lastName, int age)
{
    Console.WriteLine($"Имя: {name}\nФамилия: {lastName}\nВозраст: {age}");
}
```

Рисунок 1 – Перегрузки метода Info

Помимо методов, можно перегрузить стандартные математические операторы. Для этого нужно указать ключевое слово operator.

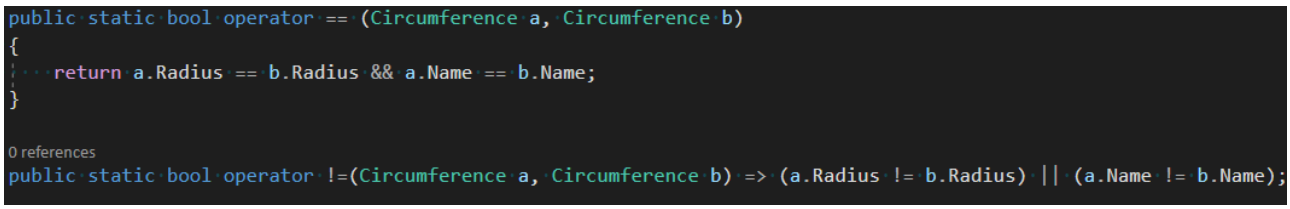

Рисунок 2 – Перегрузка операторов == и !=

Ключевое слово params

С помощью ключевого слова `params` можно указать параметр метода, принимающий переменное число аргументов. Тип параметра должен быть одномерным массивом. В объявлении метода после ключевого слова `params` дополнительные параметры не допускаются, и в объявлении метода допускается только одно ключевое слово `params`.

При вызове метода с параметром `params` можно передать следующие объекты:

- разделенный запятыми список аргументов типа элементов массива;
- массив аргументов указанного типа;
- не передавать аргументы. Если аргументы не отправляются, длина списка `params` равна нулю.

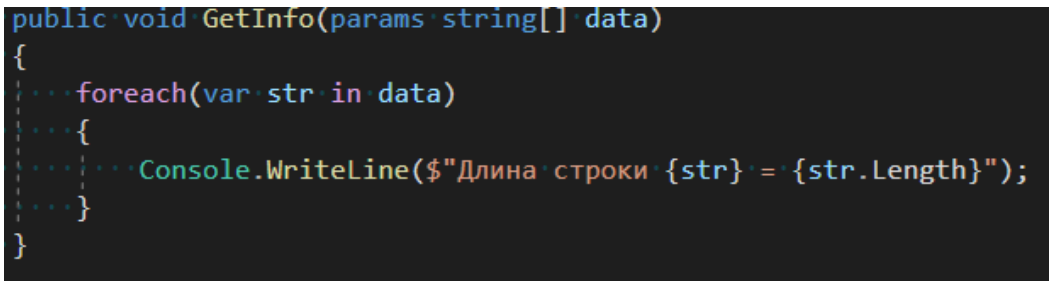

Рисунок 3 – Объявление метода с переменным числом аргументов

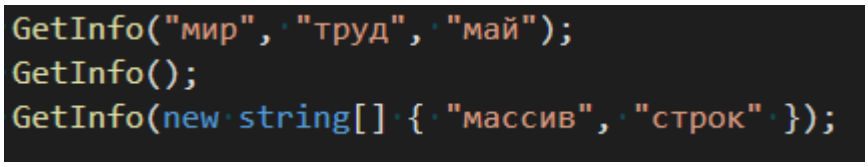

Рисунок 4 – Вызов метода с переменным числом аргументов

Практическая часть

1) Создать класс `Matrix` со свойствами `Rows` и `Columns` типа uint (неотрицательные целые, количество строк и столбцов соответственно), доступными только для чтения и с закрытым полем:

```
double [,] elements;
```

2) Создать конструктор, который принимает в качестве аргументов размерность матрицы и затем создает матрицу заданного размера, заполненную нулями:

```
public Matrix(uint rows, uint columns)
```

3) Создать копирующий конструктор:

```
public Matrix(Matrix matrix)
```

4) Создать индексатор, который позволяет получить и задать значение элемента матрицы:

```
public double this[uint row, uint column]
```

В тело индексатора нужно добавить обработку исключений в случае попытки обращения по индексам, выходящим за границы матрицы.

5) Переопределить метод `ToString()`, который будет выводить количество строк, столбцов и элементов матрицы.

6) Перегрузить метод `ToString()`, который будет выводить в строку `n` первых элементов матрицы.

7) Перегрузить операторы `+`, `-`, `*`

```
public static Matrix operator +(Matrix left, Matrix right)
{
//          TODO: реализация
}
```

8) Реализовать в классе `Matrix` интерфейс `ICloneable`.

Поскольку классы представляют ссылочные типы, то это накладывает некоторые ограничения на их использование. Например, если мы создадим две переменные

```
var a = new Matrix(2,2);
var b = a;
```

то `a` и `b` будут указывать на один и тот же объект в памяти, поэтому изменения элементов `a` затронут также и переменную `b`.

Чтобы переменная `b` указывала на новый объект, но со значениями из `a`, мы можем применить клонирование с помощью реализации интерфейса `ICloneable`.

9) Реализовать в классе `Matrix` интерфейс `IComparable`.

Содержание отчета

1. Титульный лист
2. Цель работы
3. Задание
4. Код программы
5. Результат выполнения