Модуль 4: Методы и параметры

Обзор

- Методы
- Параметры
- Перегрузка методов

Методы

- Определение методов
- Вызов методов
- Инструкция return
- Использование локальных переменных
- Возврат значений

Определение метода

- **Метод – это функциональный элемент класса, который реализует вычисления или другие действия, выполняемые классом или его экземпляром**
  - Метод представляет собой законченный фрагмент кода, к которому можно обратиться по имени.
  - Метод описывается один раз, а вызываться может многократно. Совокупность методов класса определяет, что конкретно может делать класс.
- **Синтаксис метода:**

```
[атрибуты] [спецификторы]
тип_возвращаемого_результаты
имя_метода ([список_параметров])
{
тело_метода;
return значение
}
```

Определение метода

- **Main – это метод**
  - Для создания собственных методов используйте тот же синтаксис

```
using System;

class ExampleClass
{
    static void ExampleMethod( )
    {
        Console.WriteLine("Example method");
    }
    static void Main( )
    {
        // ...
    }
}
```

Вызов методов

- **После создания метода, вы можете:**
  - Вызвать метод из класса, в котором он был создан

    Напишите имя метода и список параметров в круглых скобках

```
class Program
{
      static void ExampleMethod( )
      {
            Console.WriteLine("Example method");
      }
      static void Main(string[] args)
      {
            ExampleMethod( );
      }
}
```

Вызов методов

- **После создания метода, вы можете:**
  - Вызвать метод, определенный в другом классе
    Необходимо указать компилятору, какому классу принадлежит вызываемый метод
    Вызываемый метод должен быть объявлен с использованием ключевого слова **public**

```
class Program
{
    public static void MyWrite()
    {
       Console.WriteLine("Text output from function.");
    }
}

class My
{
    static void Main(string[] args)
    {
       Program.MyWrite();
    }
}
```

Вызов методов

- **После создания метода, вы можете:**
  - Использовать вложенные вызовы

    Методы могут вызывать методы, которые вызывают другие методы, и т.д.

```
static void Main()
        {
// вызовы метода Func:
int max = Func(Func(a, b), c);
  Console.WriteLine("max({0},{1},{2})={3}",a,b,c,max);
             }
```

Инструкция return

```
return[ ( ] [выражение] [ ) ];
```

- **Немедленное завершение**
- **Выход по условию**

```
static void ExampleMethod( )
{
   int numBeans;
   //...

   Console.WriteLine("Hello");
   if (numBeans < 10)
      return;
   Console.WriteLine("World");
}
```

Использование локальных переменных

- **Локальные переменные**
  - Создаются при вызове метода
  - Используются только внутри метода
  - Уничтожаются при выходе из метода
- **Совместно используемые переменные**
  - Переменные уровня класса для совместного использования
- **Конфликты области действия переменных**
  - Компилятор не предупреждает о совпадении имен локальных переменных и переменных уровня класса

Возврат значений

- **Заменить тип void типом возвращаемого значения**
- **Добавить оператор return**
  - Устанавливает значение, возвращаемое методом
  - Возвращается в точку вызова метода
- **Non-void методы должны возвращать значения**

```
static int TwoPlusTwo( )
{
    int a,b;
    a = 2;
    b = 2;
    return a + b;
}
```

```
int x;
x = TwoPlusTwo( );
Console.WriteLine(x);
```

Параметры

- Объявление и вызов методов с параметрами
- Механизмы передачи параметров
- Передача по значению
- Передача параметров по ссылке
- Возвращаемые параметры
- Список параметров переменной длины
- Рекурсивные методы

Объявление и вызов методов с параметрами

- **Объявление параметров**
  - В круглых скобках после имени метода
  - Определяйте тип данных и имя для каждого из параметров
- **Вызов методов с параметрами**
  - Передавайте значения для каждого из параметров

```
static void MethodWithParameters(int n, string y)
{
...
}

MethodWithParameters(2, "Hello, world");
```

Список формальных аргументов

- **Список формальных аргументов метода может быть:**
  - пустым (типичная ситуация для методов класса),
  - может содержать фиксированное число аргументов, разделяемых символом запятой.
- **Синтаксис объявления формального аргумента:**

```
[ref|out|params]тип_аргумента имя_аргумента
```

  - Обязательным является указание типа и имени аргумента.
  - Ограничений на тип аргумента не накладывается: он может быть любым скалярным типом, массивом, классом, структурой, интерфейсом, перечислением

Механизмы передачи параметров

- **Три способа передачи параметров**

	По значению
ref	По ссылке
out	Возвращаемые параметры

Передача по значению

- **Механизм, используемый по умолчанию:**
  - Значение аргумента копируется в формальный параметр метода
  - Значение параметра может меняться внутри метода
  - Это не влияет на значение аргумента, используемого при вызове
  - Типы параметра и аргумента должны быть одинаковыми или совместимыми

```
static void AddOne(int x)
{
      x++; // Increment x
}
static void Main( )
{
      int k = 6;
      AddOne(k);
      Console.WriteLine(k); // Display the value 6, not 7
}
```

Передача по ссылке

- **Что такое параметры, передаваемые по ссылке?**
  - Ссылка на область памяти
- **Использование параметров, передаваемых по ссылке**
  - При объявлении и вызове метода используйте ключевое слово **ref**
  - Типы параметра и аргумента должны совпадать
  - Изменения параметра отразятся на аргументе, используемый при вызове метода
  - При попытке передать ссылку на неинициализированный параметр, компилятор выдаст ошибку

Возвращаемые параметры

- **Что такое возвращаемые параметры?**
  - Для получения значения из метода
- **Использование возвращаемых параметров**
  - Похожи на **ref**, но не передают значения в метод
  - При объявлении и вызове метода используйте ключевое слово **out**

```
static void OutDemo(out int p)
{
      // ...
}
int n;
OutDemo(out n);
```

Список параметров переменной длины

- Используйте ключевое слово params
- Объявляется в виде массива в конце списка параметров
- Можно передавать только по значению

```
static long AddList(params long[ ] v)
{
    long total, i;
    for (i = 0, total = 0; i < v.Length; i++)
        total += v[i];
    return total;
}
static void Main( )
{
    long x = AddList(63,21,84);
}
```

Необязательные и именованные параметры C# 4.0

- **Необязательные (optional) параметры позволяют опускать аргументы функции**

```
public void optionalParamFunc(int p1, int p2 = 2, int p3 = 3);
```

- **при вызове метода можно передать явно два параметра или опустить второй и третий параметр:**

```
optionalParamFunc(1);
//это эквивалентно optionalParamFunc(1,2,3);
```

- **Именованные (named) параметры разрешают передавать аргументы по названию параметра**

```
optionalParamFunc(1, p3:10);
//это эквивалентно optionalParamFunc(1,2,10);
```

Рекурсивные методы

- **Метод может вызывать себя**
- **Напрямую**
  - метод с прямой рекурсией обычно содержит следующую структуру:

```
if (<условие>)
<оператор>;
else <вызов данного метода с другими параметрами>;
```

- **Косвенно**
  - метод вызывает себя в качестве вспомогательного не непосредственно, а через другой вспомогательный метод

Метод с прямой рекурсией

- **Вычисление факториала**

```
static long F(int n)  //рекурсивный метод
   {
      if (n==0 || n==1)
       return 1;    //нерекурсивная ветвь
       else return n*F(n-1);  //шаг рекурсии –
             //повторный вызов метода с другим параметром
    }
    static void Main()
    {
      Console.Write("n=");
       int n =int.Parse( Console.ReadLine());
       long f=F(n); //нерекурсивный вызов метода F
       Console.WriteLine("{0}!={1}",n, f);
      }
```

Перегрузка методов

- Объявление перегруженного метода
- Сигнатура метода
- Использование перегруженных методов

Объявление перегруженного метода

- **В одном классе можно использовать несколько методов с одним именем**
  - Списки параметров должны отличаться по типу и/или количеству

```
class OverloadingExample
{
    static int Add(int a, int b)
    {
        return a + b;
    }
    static int Add(int a, int b, int c)
    {
        return a + b + c;
    }
    static void Main( )
    {
        Console.WriteLine(Add(1,2) + Add(1,2,3));
    }
}
```

Сигнатура метода

- **Сигнатуры методов должны быть уникальны в пределах класса**
- **Определение сигнатуры**

Определяет сигнатуру

- Имя метода
- Тип параметра
- Количество параметров

Не влияет на сигнатуру

- Имя параметра
- Тип возвращаемого методом значения

Использование перегруженных методов

- **Используйте перегруженные методы, если:**
  - У вас имеются похожие методы, различающиеся типом и количеством параметров
  - Вы хотите добавить новую функциональную возможность в уже существующий код
- **Не переусердствуйте, т.к.:**
  - Сложно отлаживать
  - Сложно поддерживать