Тема 5. Методы и параметры

Парадигмы программирования. C#

План

- Использование методов
- Использование параметров
- Использование перегруженных методов

Использование методов

Часть 1

Декларирование методов

- Main – это метод
- Используйте аналогичный синтаксис для объявления собственных методов

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

Синтаксис декларирования методов без параметров

```
static void MethodName( )
{
    Тело метода
}
```

Вызов методов

- Варианты вызова метода
  - Из того же класса
    - Указать название метода и список фактических параметров в скобках
    - Скобки указывать обязательно, даже если метод не имеет формальных параметров
  - Из другого класса
    - При декларировании метода требуется указать модификатор доступа public
    - При вызове нужно указать, из какого класса (структуры) или объекта будет вызываться метод
  - С использованием вложенных вызовов
    - Методы могут вызывать другие методы и т.д.

Использование оператора return

- Используется для безусловного завершения работы выполняющегося метода и возврата к выполнению вызывавшего метода
- При возврате из метода может возвращаться его результат
  - В этом случае использование `return` обязательно
- При использовании условного оператора возврат можно сделать условным

```
static void ExampleMethod( )
{
    int numBeans;
    //...
    Console.WriteLine("Hello");
    if (numBeans < 10)
        return;
    Console.WriteLine("World");
}
```

Использование локальных переменных

- Локальные переменные
  - Создаются при начале работы метода в стеке
  - Являются частными (private) для метода
  - Автоматически уничтожаются при завершении работы метода
- Общие переменные
  - Переменные, объявленные на уровне класса используются как глобальные переменные внутри методов класса
- Конфликты областей
  - Компилятор не предупреждает о совпадении имён локальных переменных метода и переменных класса

Возврат значений

- Задекларируйте тип результата метода – не void
- Обязательно добавьте оператор return с выражением, результат которого будет возвращён из метода

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

...

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

Использование параметров

Часть 2

Декларирование и вызов параметров

- Декларирование параметров
  - Разместите параметры между круглыми скобками
  - Укажите типы и имена всех параметров
- Вызов метода с параметрами
  - Укажите значение для каждого параметра

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

MethodWithParameters(2, "Lemon tree");
```

Механизмы передачи параметров

- in: передача по значению
- in/out: передача по ссылке
- out: выходной параметр

Передача по значению

- Это механизм передачи параметров по умолчанию
  - Значение параметра копируется
  - Параметр может быть изменён внутри метода
  - Это изменение не сказывается на значении вне метода
  - Фактический параметр должен быть того же типа, что и формальный, или совместимого типа

```
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 ссылка на параметр?
  - Ссылка на расположение параметра в памяти
- Использование ссылочных параметров
  - При декларировании и вызове используйте ключевое слово ref
  - Типам формальных параметров должны соответствовать типы фактических параметров
  - Изменения значений формальных параметров видны вызывающему методу
  - Фактические параметры должны быть проинициализированы перед вызовом метода

```
static void AddOne(ref int x)
{
    x++;
}
static void Main( )
{
    int k = 6;
    AddOne(ref k);
    Console.WriteLine(k); // 7
}
```

Выходные параметры

- Что такое выходные параметры?
  - Возвращают значения, но не принимают их на вход
- Использование выходных параметров
  - Как ref, но значение не передаётся в метод
  - Используйте ключевое слово out при декларировании метода и его вызове

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

int n;
OutDemo(out n);
```

Использование списков параметров переменной длины

- Используйте ключевое слово params
- Декларируйте последний формальный параметр как массив
- Всегда передаются 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);
}
```

Рекомендации по использованию параметров методов

- Механизмы
  - Передача по значению является самой распространённой
  - Возвращаемое значение метода используется для возврата единичных значений
  - Используйте ref или out для возврата нескольких значений
  - Используйте ref если данные передаются и на вход, и на выход
- Эффективность
  - Передача по значению является самой эффективной

Рекурсия

- Метод может вызывать сам себя
  - Непосредственно
  - Косвенно (через другие методы)
- Рекурсия удобна для реализации некоторых алгоритмов
  - Числа Фибоначи
  - Поиск наибольшего общего делителя двух чисел методом Евклида

Использование перегруженных методов

Часть 3

Объявление перегруженных методов

- Перегруженными методами являются методы с одинаковым идентификатором, размещённые в одном классе или структуре
- Недопустимо пытаться использовать один и тот же идентификатор для обозначения метода и поля
- Перегруженные методы отличаются списками параметров

```
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));
    }
}
```

Сигнатуры методов

- Любые методы, даже перегруженные должны отличаться друг от друга сигнатурой
- Состав сигнатуры
  - Идентификатор метода
  - Типы параметров
  - Модификаторы параметров
- В сигнатуру не входят
  - Идентификаторы параметров
  - Тип возвращаемого значения

Использование перегруженных методов

- Рекомендуемое использование
  - Для методов, похожих по поведению, но имеющих разные параметры
  - Для добавления нового функционала к существующему коду без значительного его изменения
- Не рекомендуемое слишком частое использование
  - Трудности отладки
  - Трудности поддержки

Выводы

- Все методы должны являться членами классов или структур
- Для ограничения доступа к методам используйте модификаторы доступа
- Параметры в методы могут передаваться по значению, по ссылке и могут являться выходными
- Хотя C# поддерживает перегрузку методов, помните, что перегруженные методы должны отличаться сигнатурами

Заключение

- Спасибо за внимание!