

ТЕМА 5: МЕТОДИ

План

5.1. Поняття методу.....	1
5.2. Модифікатори параметрів.....	1
5.3. Необов'язкові параметри.....	2
5.4. Сигнатура методу.....	3
5.5. Статичний поліморфізм.....	4

5.1. Поняття методу.

Методи виконують деякі дії у послідовності операторів. Метод може отримувати вхідні дані з коду, який його викликає, за рахунок вказування параметрів і повертати вихідні дані через вказування типу, що повертається методом, або через параметри `ref/out`. Якщо метод не повертає даних, то слід вказати тип `void`.

Методи. Можуть приймати або не приймати параметри та повертати або не повертати значення.

```
static int Add(int x, int y)
{
    int ans = x + y;
    // Вызывающий код не увидит эти изменения,
    // т.к. изменяется копия исходных данных,
    x = 10000;
    y = 88888;
    return ans;
}
```

5.2. Модифікатори параметрів.

Якщо параметр не позначений модифікатором, вважається, що він має передаватися за значенням, тобто метод отримує копію вихідних даних.

Модифікатор `out`. Має вказуватись як при реалізації, так і при виклику методу. Методи перед виходом обов'язково мають присвоїти таким параметрам значення, інакше – помилка компіляції.

```
static void Add(int x, int y, out int ans)
{
    ans = x + y;
}
```

Виклик методу:

```
int ans;
Add(90, 90, out ans);
Console.WriteLine("90 + 90 = {0}", ans);
```

Інший приклад:

```
// Возврат множества выходных параметров.
```

```
static void FillValues(out int a, out string b, out bool c)
{
    a = 9;
    b = "Enjoy your string.";
    c = true;
}
```

Виклик:

```
int i; string str; bool b;
FillTheseValues(out i, out str, out b);
```

Модифікатор *ref*. Передається посилання на існуючу змінну. Параметри мають бути ініційовані перед передачею методу.

Приклад методу, що міняє місцями дві змінні:

```
// Ссылочные параметры.
public static void SwapStrings(ref string s1, ref string s2)
{
    string tempStr = s1;
    s1 = s2;
    s2 = tempStr;
}
```

Виклик методу:

```
string str1 = "Flip";
string str2 = "Flop";
SwapStrings(ref str1, ref str2);
```

Модифікатор *params*. Масив параметрів. Підтримує лише один параметр *params*, який має бути останнім у списку параметрів.

// Возвращение среднего из некоторого количества значений double.

```
static double CalculateAverage(params double[] values)
{
    double sum = 0;
    if(values.Length == 0)
        return sum;
    for (int i = 0; i < values.Length; i++)
        sum += values[i];
    return (sum / values.Length);
}
```

Варіанти виклику:

```
// Передать разделяемый запятыми список значений double...
```

```
double average;
average = CalculateAverage(4.0, 3.2, 5.7, 64.22, 87.2);
```

```
// ...или передать массив значений double.
```

```
double [ ] data = { 4.0, 3.2, 5.7 };
average = CalculateAverage(data);
```

```
// Среднее из 0 равно 0!
```

```
Console.WriteLine("Average of data is: {0}", CalculateAverage());
```

5.3. Необов'язкові параметри.

Можна не вказувати за умови, що задовольняють стандартні значення. Розміщуються в кінці сигнатури методу.

```
static void EnterLogData(string message, string owner = "Programmer")
{
    Console.Beep ();
    Console.WriteLine("Error: {0}", message);
    Console.WriteLine("Owner of Error: {0}", owner);
}
```

Виклик:

```
EnterLogData("Oh no! Grid can't find data");
EnterLogData("Oh no! I can't find the payroll data", "CFO");
```

Значення, що присвоюється необов'язковому параметру, має бути відоме під час компіляції.

```
// Ошибка! Стандартное значение для необязательного аргумента
// должно быть известно во время компиляции!
static void EnterLogData(string message,
    string owner = "Programmer", DateTime timeStamp = DateTime.Now)
{
    Console.Beep();
    Console.WriteLine("Error: {0}", message);
    Console.WriteLine("Owner of Error: {0}", owner);
    Console.WriteLine("Time of Error: {0}", timeStamp);
}
```

Таблица 4.1. Модификаторы параметров в C#

Модификатор параметра	Описание
(отсутствует)	Если параметр не помечен модификатором, предполагается, что он должен передаваться по значению, т.е. вызываемый метод получает копию исходных данных
out	Значения выходных параметров должны присваиваться вызываемым методом и, следовательно, они передаются по ссылке. Если выходным параметрам в вызываемом методе значения не присвоены, компилятор сообщит об ошибке
ref	Значение первоначально присваивается вызывающим кодом и при желании может быть изменено в вызываемом методе (поскольку данные также передаются по ссылке). Если параметру <code>ref</code> в вызываемом методе значение не присвоено, никакой ошибки компилятор не генерирует
params	Этот модификатор позволяет передавать переменное количество аргументов как единый логический параметр. Метод может иметь только один модификатор <code>params</code> , которым должен быть помечен последний параметр метода. В реальности необходимость в использовании модификатора <code>params</code> возникает не особо часто, однако он применяется во многих методах внутри библиотек базовых классов

5.4. Сигнатура метода

(включає тип і кількість параметрів, спосіб передачі параметрів, але не включає імена параметрів і тип, що повертається методом) має бути унікальною в межах типу. З методами дозволено застосовувати наступні модифікатори:

Статичний модифікатор	<code>static</code>
Модифікатори доступу	<code>public,</code> <code>internal,</code>

Модифікатори успадковування

Модифікатор частинного методу

Модифікатор некерованого коду

private,
protected
new, virtual,
abstract,
override, sealed
partial
unsafe, extern

Тип може *перевантажувати* методи (мати декілька методів з одним і тим же іменем) за умови, що їх сигнатури будуть різними. Наприклад:

```
void MyMethod(int x) {}  
void MyMethod(int y) {}//Помилка компіляції  
void MyMethod(int x,char c) {}  
void MyMethod(char d,int y) {}  
double MyMethod(int x,char c) {}//Помилка компіляції  
void MyMethod(int[] x) {}  
void MyMethod(ref int x) {}  
void MyMethod(out int x) {}//Помилка компіляції
```

5.5. Статичний поліморфізм

Поліморфізм можна виразити як «один інтерфейс, безліч функцій». Поліморфізм може бути статичним і динамічним. **Статичний поліморфізм** реалізується шляхом перевантажень методів. Наприклад, можна мати декілька описів методу в залежності від типу вхідного параметра:

```
class Printdata  
{  
    public void print(int i)  
    {  
        Console.WriteLine("Printing int: "+i);  
    }  
    public void print(double f)  
    {  
        Console.WriteLine("Printing float: "+f);  
    }  
    public void print(string s)  
    {  
        Console.WriteLine("Printing string: "+s);  
    }  
}
```

В залежності від типу вхідного параметра буде викликатись той чи інший варіант методу `print()`.

```
public static void Main(string[] args)  
{  
    Printdata p = new Printdata();  
    p.print(5);// Printing int: 5  
    p.print(3.14);// Printing float: 3,14  
    p.print("Hello C#");// Printing string: Hello C#  
}
```