ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

C#. Методи. Передача параметрів в методи

Лекція 09

План

1. Що таке метод?
2. Синтаксис оголошення методу
3. Методи і необов'язкові параметри
4. Модифікатори параметрів
5. Передача масивів у методи
6. Приклади

Що таке метод?

У C# метод - це фрагмент коду, що виконує конкретну задачу і має чітко визначену функціональність.

На відміну від деяких інших мов програмування, C# не розрізняє функції та процедури. Однак, методи, що повертають значення, можна умовно вважати функціями, а методи, що не повертають значень, - процедурами.

Коли використовувати методи?
Якщо частина коду повторюється два і більше рази, то є сенс винести інструкції в окремий метод.

Синтаксис оголошення методу

```
[модифікатори] тип_повернення Ім'яМетоду([параметри])
{
        // Тіло методу

       // Може міститися код, який виконується при виклику методу

        [return вираз;]
}
```

Приклад

```
public void SayHello()
{
    Console.WriteLine("Привіт!");
}
```

Метод не приймає параметрів і нічого не повертає, модифікатор доступу *public* робить метод доступним з будь-якого місця програми, *void* означає, що метод не повертає значення.

```
public void Greet(string name)
{
    Console.WriteLine($"Привіт, {name}!");
}
```

Метод приймає один параметр типу *string* і нічого не повертає.

Приклад

```
public int GetCurrentYear()
{
    return DateTime.Now.Year;
}
```

Метод не приймає параметрів але повертає ціле число.

```
public string ConcatenateStrings(string str1, string str2)
{
    return str1 + str2;
}
```

Метод приймає два параметри типу *string* і повертає рядок , який є конкатенацією цих двох рядків.

```
public void PrintDetails(string name = "Гість", int age = 0)
{
    Console.WriteLine($"Ім'я: {name}, вік: {age}");
}
```

Метод з параметрами за замовчуванням. Якщо при виклику методу не вказано значення для *name* або *age*, будуть використані значення за замовчуванням («Гість" та 0).

Методи

```
internal class Program
{
    static void Main(string[] args){
        string message = Hello();
        Console.WriteLine(message);

        int sum = SumFromAtoB(10, 50);
        Console.WriteLine(sum);
}

    static string Hello(){return "Привіт!";}

    static int SumFromAtoB(int a, int b)
    {   int res = 0;
        for (int i = a; i <= b; i++)
            res += i;
        return res;
    }
 }
```

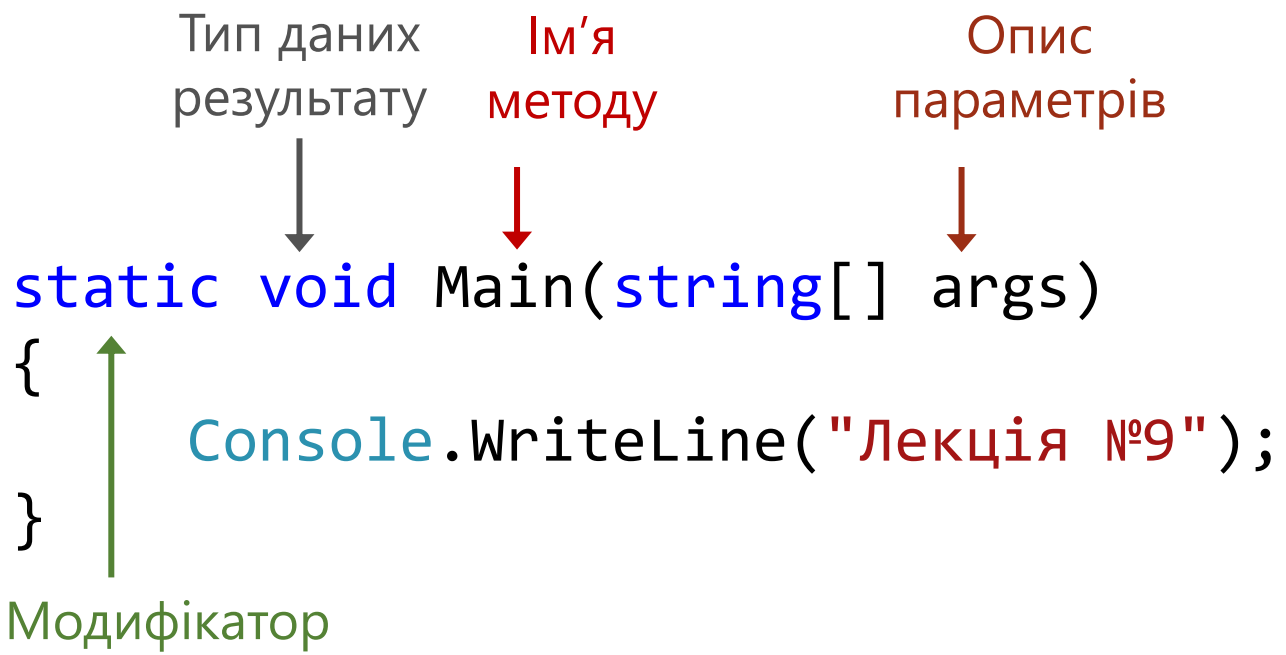
Всі три методи мають модифікатори доступу static.

!!! Немає різниці в якому порядку описані методи

Модифікатор internal означає, що клас доступний лише в межах поточного збірки (assembly).

static означає, що метод можна викликати без створення екземпляра класу Program

Методи



static void Main(string[] args) - сигнатура головного методу програми C#.

Коли виконується запуск програми, середовище виконання .NET (CLR) шукає метод з такою сигнатурою і починає виконання програми з нього.

Модифікатор static означає, що метод Main належить класу, а не його екземпляру.

void означає, що метод не повертає жодного значення.

Main - ім'я методу, яке є зарезервованим для головного методу програми.

string[] args - масив рядків, який представляє аргументи командного рядка, передані програмі.

Модифікатор static в C#

Ключове слово static використовується для оголошення членів класу, які належать самому класу, а не його екземплярам. Це означає, що статичні члени є спільними для всіх об'єктів класу.

Статичні поля використовуються для зберігання даних, які є спільними для всіх екземплярів класу.

Статичні константи використовуються для визначення значень, які є незмінними та доступними для всього класу.

Статичні класи можуть містити лише статичні члени і не можуть бути інстанційовані.

Модифікатор static в C#

Статичні методи часто використовуються для створення допоміжних функцій, які не залежать від стану конкретного об'єкта.

Доступ до *статичних членів* здійснюється через ім'я класу, а не через ім'я об'єкта.

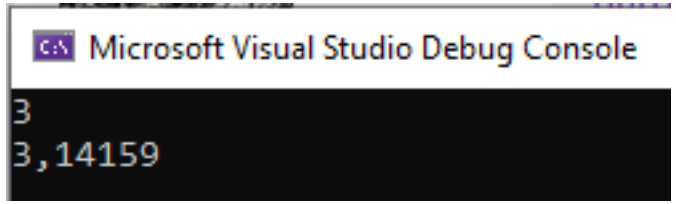
Статичні члени існують протягом усього часу існування програми.
Статичні методи не можуть звертатися до нестатичних членів класу.
Статичні класи не можуть бути інстанційовані.

Приклад

```
public class Count
{
    public static int count = 0;
    public static double PI = 3.14159;

    public Count()
    {
        count++;
    }
}
internal class Program
{
    public static void Main(string[] args)
    {
        Count c1 = new Count();
        Count c2 = new Count();
        Count c3 = new Count();

        Console.WriteLine(Count.count);
        Console.WriteLine(Count.PI);
    }
}
```

Статичні поля та константи

- належать самому класу, а не його екземплярам;
- до них можна отримати доступ, використовуючи ім'я класу, без створення об'єкта;
- вони є спільними для всіх екземплярів класу.



Приклад

```
public static void Main(string[] args)
{
    Test(25);
    string str = Zero(0);
    Console.WriteLine(str);
}
static void Test(int i){
    if (i <= 0 || i > 5){
        return; Console.Write("123");
    }
    Console.WriteLine(i);
}

static string Zero(double number)
{
    if (number < 0)
        return "Число менше нуля";
    else if (number > 0)
        return "Число більше нуля";
        return "Число рівне нулю";
}
```

Оператор return завершує виконання методу, та може повертати значення.

Інструкції розміщені після return ігноруються.

В методі дозволяється багатократне використання оператора return.

Якщо метод має тип void – оператор return може використовуватись для дострокового виходу

Методи і необов'язкові параметри

```
static void PrintUserInfo(string name, int age = 18, string city = "Невідомо")
 {
     Console.WriteLine($"Ім'я: {name}, Вік: {age}, Місто: {city}");
 }
 public static void Main(string[] args)
 {
     Console.OutputEncoding = Encoding.Unicode;
     Console.InputEncoding = Encoding.Unicode;

     PrintUserInfo("Іван");
     PrintUserInfo("Марія", 25);
     PrintUserInfo("Петро", 30, "Київ");

 }
```

```
Ім'я: Іван, Вік: 18, Місто: Невідомо
Ім'я: Марія, Вік: 25, Місто: Невідомо
Ім'я: Петро, Вік: 30, Місто: Київ
```

У C# методи можуть мати необов'язкові параметри, що дозволяє викликати метод, не передаючи значення для всіх параметрів.

Необов'язкові параметри оголошуються із значенням за замовчуванням.

Якщо значення не передається при виклику методу, використовується значення за замовчуванням.

Методи і необов'язкові параметри

```
static void PrintUserInfo(string name = "Аліна", int age, string city = "Невідомо")
{
    Console.WriteLine($"Ім'я: {name}, Вік: {age}, Місто: {city}");
}
```

!!! Помилка

Необов’язкові параметри мають з’являтися після всіх обов’язкових параметрів.

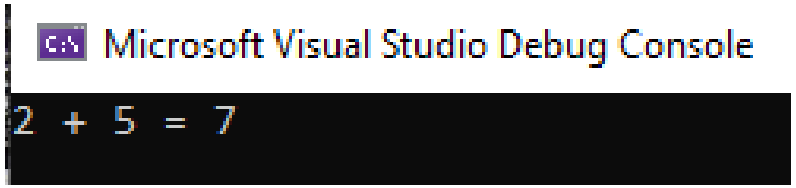
Не надано жодного аргументу, який відповідає обов’язковому параметру «age» для «Program.PrintUserInfo(string, int, string)»

Приклад

```
static int Add(int x1, int x2)
{
    x1 += x2;
    return x1;
}

static void Main(string[] args)
{
    int num1 = 2, num2 = 5;

    int sum = Add(num1, num2);

    Console.WriteLine($"{num1} + {num2} = {sum}");
}
```

Передача параметрів по значенню

Значення змінних «ззовні» функції не змінюються!



Модифікатори параметрів

У мові C# модифікатори параметрів використовуються для визначення способу передачі аргументів методам. Вони впливають на те, як метод отримує та обробляє значення переданих змінних.

Основні модифікатори параметрів у C#:

ref (reference parameters) передає параметр за посиланням.
out (output parameters) передає вихідний параметр за посиланням.
in передає параметр за посиланням, але забороняє методу змінювати значення аргументу.
params дозволяє методу приймати змінну кількість аргументів певного типу.

Модифікатори параметрів

```
static void Main(string[] args)
{
    Console.OutputEncoding = Encoding.Unicode;
    Console.InputEncoding = Encoding.Unicode;

    int number = 5;
    ModRef(ref number);
    Console.WriteLine($"Значення після виклику методу: {number}");
}
static public void ModRef(ref int x)
{
    x = x + 10;
    Console.WriteLine($"Значення всередині методу: {x}");
}
```

Щоб передати змінну за посиланням (через покажчик) використовується модифікатор ref

```
Значення всередині методу: 15
Значення після виклику методу: 15
```

Модифікатори параметрів

Змінні, які передаються через ref обов’язково повинні мати значення.

В середині методу можна змінювати значення ref-змінних і вони змінюються у викликаючому коді.

Модифікатори параметрів

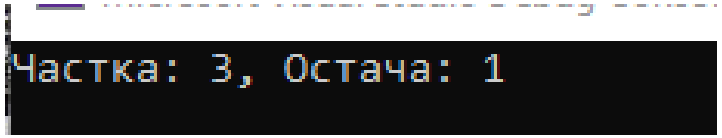
Вихідний параметр – це параметр, у який записуватиметься значення і яке буде доступне у зовнішньому коді.

В середині методу забороняється використовувати значення змінних, переданих як out.

Усім параметрам, описаним з модифікатором out повинні бути присвоєні значення в середині методу.

Приклад

```
static void Main(string[] args)
{
    Console.OutputEncoding = Encoding.Unicode;
    Console.InputEncoding = Encoding.Unicode;

    int resultQ, resultR;

    Divide(10, 3, out resultQ, out resultR);
    Console.WriteLine($"Частка: {resultQ}, Остача: {resultR}");
}
static public void Divide(int a, int b, out int q, out int r)
{
    q = a / b;
    r = a % b;
}
```



Приклад

```
static void Main(string[] args)
{

    int result = 5;

    Divide(out result);

    Console.WriteLine($" {result} ");
}

static public void Divide(out int result)
{

    result += 15;
}
```

!!! Помилка.

Неможна використовувати значення змінної *result*, оскільки вона може не містити значення

Різниця між ref та out

Характеристика	ref	out
Призначення	Передає змінну за посиланням, дозволяє змінювати оригінальну змінну.	Передає змінну за посиланням, але вимагає від методу присвоїти їй значення.
Ініціалізація змінної	Повинна бути ініціалізована перед викликом методу.	Не обов'язково повинна бути ініціалізована перед викликом методу.
Вхідні/вихідні дані	Може використовуватися для передачі як вхідних, так і вихідних даних.	Використовується для передачі вихідних даних.
Гарантія присвоєння	Метод не зобов'язаний присвоювати нове значення змінній.	Метод зобов'язаний присвоїти нове значення змінній перед завершенням.

Передача аргументів у метод (in)

Оператор in в C# використовується для передачі аргументів методу за посиланням, але з метою запобігання їх модифікації всередині методу. Це означає, що метод може читати значення аргументу, але не може його змінювати.

in використовується для оптимізації продуктивності, особливо при передачі великих структур або об'єктів, щоб уникнути їх копіювання.

```
[модифікатори] тип_повернення Ім'яМетоду([in ТипПараметра назваПараметра])
{
        // Тіло методу, де можна читати значення назваПараметра
         // але не можна присвоювати йому нове значення
}
```

Приклад

```
static void Main(string[] args)
{
    Console.OutputEncoding = Encoding.Unicode;
    Console.InputEncoding = Encoding.Unicode;

    int number = 5;
    OutInfo(in number);
}
static public void OutInfo(in int x)
{
    Console.WriteLine($"Значення всередині методу: {x}");

    // x = x + 10; // помилка, не можна змінювати параметр in
}
```

Корисно для передачі великих структур чи об'єктів, щоб уникнути копіювання, але при цьому гарантувати, що дані не будуть змінені.

```
Значення всередині методу: 5
```

Передача змінної кількості аргументів у метод

```
static void Main(string[] args)
{
    double sum = Summa(1, 2, 3);
    Console.WriteLine(sum);

    double sum2 = Summa(1, 2, 3, 4, 5, 6, 7);
    Console.WriteLine(sum2);

}
static  double Summa(params int [] arr)
{
    double res = 0.0;
    //for (int i = 0; i < arr.Length; i++)
    //    res += arr[i];

    // res = arr.Sum();

    res = arr.Aggregate(0, (Sum, nextEl) => Sum + nextEl);
    return res;
}
```

Дозволяють передавати змінну кількість аргументів одного типу.

Аргументи передаються як масив.

```
6
28
```

Приклад

```
public class Example
{
    public static void MethodRef(ref int x)
    { x = 10; }
    public static void MethodOut(out int y)
    { y = 20; }
    public static void MethodIn(in int z)
    {
        Console.WriteLine($"z = {z}");
    }
    public static void MethodParams(params int[] numbers)
    {
        int sum = 0;
        foreach (int number in numbers)
            sum += number;
            Console.WriteLine($" sum= {sum}");
    }
}
```

```
a після MethodRef: 10
b після MethodOut: 20
z = 15
 sum= 6
 sum= 9
```

```
int a = 5;
Example.MethodRef(ref a);
Console.WriteLine($"a після MethodRef: {a}");

int b;
Example.MethodOut(out b);
Console.WriteLine($"b після MethodOut: {b}");

int c = 15;
Example.MethodIn(c);
Example.MethodParams(1, 2, 3);
Example.MethodParams(new int[] { 4, 5 });
```

Передача масивів у методи

```
static void Main(string[] args)
{
    int[] arr = { 1, 2, 3, 4 };
    arr.ToList().ForEach(a => Console.Write(a + " "));

    Reverse(arr);     Console.WriteLine();

    arr.ToList().ForEach(a => Console.Write(a + " "));
}
static void Swap(ref int a, ref int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
static void Reverse(int[] arr)
{
    for (int i = 0; i < arr.Length / 2; i++)
        Swap(ref arr[i], ref arr[arr.Length - i - 1]);
}
```

Масиви передаються як звичайні змінні