Метод

Формат запису методу такий:

```
доступ тип_результату ім'я (список_параметрів) {
        // Тіло методу
}
```

```
static void CalculateArea1 ( int R )
{
 if (R < 1) return;
 double area = R * R * Math.PI;
 Console.Write("area = " + area.ToString());
}

static double CalculateArea2( int R )
{
 if (R < 1) return 0.0;      // не може бути return ;
 double area = R * R * Math.PI;
 return area;
}
```


Виклик метода

```
1.  class Building {
2.    public int floors; // Кількість поверхів
3.    public int area; // Загальна площа будівлі
4.    public int occupants; // Кількість мешканців
5.     // Повернення значення площі, яка припадає на одну людину.
6.    public int areaPerPerson () {
7.      return area / occupants;
8.    }
9.  }

10. class BuildingDemo {
11.   public static void Main () {
12.     Building house = new Building ();
13.     int areaPP; // Площа, яка припадає на одну людину

14.  // Присвоюємо значення полям у об'єкті house.
15.     house.occupants = 4;
16.     house.area = 200;
17.     house.floors = 2;

18.  // Отримуємо для об'єкта house площу, яка припадає на одну людину.
19.     areaPP = house.areaPerPerson ();
20.
21.     Console.WriteLine ("Будинок має:\n {0} поверхи \n {1} мешканців \n {2} квадратних метрів
    загальної площі, з них \n {3} приходиться на одну людину", house.floors, house.occupants,
    house.area, areaPP);
22.   }
23. }
```


Повернення об'єктів

```
public class Contact2
   {
       string name ;
       string address ;
      // Private constructor.
      private Contact2 (string contactName, string contactAddress)
      {
         name = contactName;
         address = contactAddress;
      }
      // Public factory method.
      public static Contact2 CreateContact (string name, string address)
      {
            return new Contact2 (name, address);
      }
       public string ShowData()
       {
            return string.Format("name - {0}, address - {1}", name, address);
       }
   }  ----------------------------------------------------------------------------------------------------
///… static void Main() {  ...  }
Contact2 cont = Contact2.CreateContact ("Terry Adams", "123 Main St.");
Console.WriteLine(cont.ShowData());
```


Перевантаження методів

```
void StopSerive()
{
    //ця версія методу не приймає жодних параметрів
}
void StopService ( string serviceName )
{
    //ця версія методу приймає параметр типу string
}
void StopService ( int serviceID )
{
    //ця версія методу приймає параметр типу int
}
string StopSerive()
{
    //такий метод в цьому випадку - недопустимий
}
```

Виклик перевантажених методів

```
1.  class Rectangle {
2.    public int width;
3.    public int height;
4.    public Rectangle() {  }
5.    public Rectangle (int width, int height) {
6.      this.width = width;
7.      this.height = height;
8.    }
9.    public int area () {
10.     return width * height;
11.   }
12.   public int area (int K) {
13.     return width * height * K;
14.   }
15. }
16. class UseRectangle {
17.   public static void Main () {
18.     Rectangle r1 = new Rectangle ();
19.     Rectangle r2 = new Rectangle (5, 2);
20.     Console.WriteLine ("Площа прямокутника r1:" + r1.area ());
21.     Console.WriteLine ("Площа прямокутника r2:" + r2.area ());
22.     Console.WriteLine ("Площа прямокутника r2:" + r2.area (5));
23.   }
24. }
```


Необов'язкові та іменовані параметри

Коли визначається метод, що підтримує *необов'язкові параметри* дуже важливо зазначити спочатку обов'язкові параметри і тільки далі зазначити необов'язкові параметри.

```
void StopService (bool forceStop, string serviceName = null, int serviceID = 1)
{
        // код методу
} // можливий виклик методу StopService(false);
```

Наступний приклад показує як викликати метод StopService використовуючи *іменований параметр* serviceID.

```
StopService(true, serviceID: 5);
```

Ви можете комбінувати позиційні та іменовані параметри за умови, що позиційні параметри завжди зазначені перед іменованими.

Передача посилань на об'єкт в метод

```
public class SampleRefType
{
   public int value;
}
public class Test {
   // блок методу Main ()

    public static void TestRefType()
    {
        SampleRefType rt = new SampleRefType();
        rt.value = 44;
        ModifyObject(rt);
        Console.WriteLine(rt.value);
    }
    static void ModifyObject (SampleRefType obj)
    {
        obj.value = 33;
    }
}
```


ref-параметр

Модифікатор параметра **ref** змушує C# організувати замість виклику за значенням виклик по посиланню. Модифікатор ref використовується при оголошенні методу і при його виклику.

```
1.  class Swap {
2.  // Цей метод міняє місцями значення своїх аргументів.
3.    public void  swap (ref int a, ref int b) {
4.      int t;
5.      t = a;
6.      a = b;
7.      b = t;
8.    }
9.  }
10. class SwapDemo {
11.   public static void Main () {
12.     Swap ob = new Swap ();
13.     int x = 10, y = 20;
14.     Console.WriteLine ("x і y перед викликом:" + x + ", " + y);
15.     ob.swap (ref x, ref y);
16.     Console.WriteLine ("x і y після виклику:" + x + ", " + y);
17.   }
18. }
```

out-параметр

Модифікатор **out** можна використовувати тільки для передачі значення з методу. Не потрібно присвоювати змінній, використовуваній як out-параметр, початкове значення до виклику методу. Але метод (до свого завершення) обов'язково повинен присвоїти цьому параметру значення. Таким чином, після звернення до методу out-параметр буде містити певне значення.

```
1.  class Decompose {
2.    /* Метод розбиває число з плаваючою точкою на цілу і дробову частини. */
3.    public int parts (double n, out double frac) {
4.      int whole;
5.      whole = (int) n;
6.      frac = n - whole; // Передаємо дробову частину допомогою параметра frac.
7.      return whole; // Повертаємо цілу частину числа.
8.    }
9.  }
10. class UseOut {
11.   public static void Main () {
12.     Decompose ob = new Decompose ();
13.     int i; double f;
14.     i = ob.parts (10.125, out f);
15.     Console.WriteLine ("Ціла частина числа дорівнює " + i);
16.     Console.WriteLine ("Дрібна частина числа дорівнює " + f);
17.   }
18. }
```


Модифікатор params

Модифікатор **params** використовується для оголошення параметра-масиву, який може отримати деяку кількість аргументів (у тому числі і "порожнє значення"). Кількість елементів у масиві буде дорівнює числу аргументів, переданих методу.

```
1.  class Min {
2.    public int minVal (params int [] nums) {
3.      int min;
4.
5.      if (nums.Length == 0) {
6.        Console.WriteLine ("Помилка: немає аргументів");
7.        return 0;
8.      }
9.
10.     min = nums [0];
11.
12.     for (int i = 1; i <nums.Length; i ++)
13.       if (nums [i] < min) min = nums [i];
14.
15.     return min;
16.   }
17. }
```

```
1.  class ParamsDemo {
2.    public static void Main () {
3.      Min ob = new Min ();
4.      int min;
5.      int a = 10, b = 20;

6.      min = ob.minVal (a, b);
7.      Console.WriteLine ("Мінімум дорівнює" + min);

8.      min = ob.minVal (a, b, -1);
9.      Console.WriteLine ("Мінімум дорівнює" + min);

10.     min = ob.minVal (18, 23, 3, 14, 25);
11.     Console.WriteLine ("Мінімум дорівнює" + min);

12.     int [] args = {45, 67, 34, 9, 112, 8};
13.     min = ob.minVal (args);
14.     Console.WriteLine ("Мінімум дорівнює" + min);
15.   }
16. }
```