# ТЕМА 2: СТВОРЕННЯ ОБ'ЄКТІВ. ІНІЦІАЛІЗАЦІЯ

## План



### 1.1. Константні поля та поля лише для читання.

Ключове слово const застосовується для полів, які ніколи не будуть змінюватися:

```
public const double PI = 3.14;
```

Поля, що доступні лише для читання, подібно до константних полів можуть після присвоєння значення лише зчитуватися. Але присвоєння може відбуватися лише у конструкторі на етапі виконання програми.

```
class MyMathClass
{
// Поля только для чтения могут присваиваться
// в конструкторах, но нигде более,
public readonly double PI;
public MyMathClass ()
{
PI = 3.14;
}
}
```

### 1.2. Властивості

Властивості класу виглядають як поля, але при зчитуванні або встановленні значень містять логіку. Реалізується ця логіка за допомогою відповідно блоків `get` і `set`:

```
class price
{
      private decimal cina;
      public decimal Cina
      {
            get {return cina;} //виконується при зчитуванні
```

```
            set {cina=value;} // виконується при присвоєнні
      }
}
```

Засіб доступу `get` виконується при зчитуванні властивості, а `set` – при присвоєнні. Засіб доступу `set` має неявний параметр `value` з типом властивості, що присвоюється. Хоча доступ до властивостей відбувається таким самим способом, як і до полів, проте властивості відрізняються тим, що надають програмісту повний контроль над отриманням і встановленням їх значень. Це дозволяє обирати будь-яке необхідне внутрішнє представлення, не показуючи внутрішні деталі користувачу.

Якщо вказати лише засіб доступу `get`, то властивість буде доступна лише для читання, а якщо лише `set` – тільки для запису.

З властивостями дозволено застосовувати наступні модифікатори:

| | |
|---|---|
| Статичний модифікатор | `static` |
| Модифікатори доступу | `public, internal, private, protected` |
| Модифікатори успадковування | `new, virtual, abstract, override, sealed` |
| Модифікатор некерованого коду | `unsafe, extern` |

Модифікатори доступу можна окремо застосовувати для засобів доступу `get` і `set`:

```
class price
{
      private decimal cina;
      public decimal Cina //властивість більш доступна
      {
            get {return cina;}
            private set {cina=value;} // засіб доступу менш доступний
      }
}
```

В C# елементом програмування є клас. Об’єкти створюються з цих класів, а функції інкапсулюються в класах як методи.

### 1.3. Методи.

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;
}
```

### 1.4. 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);
```

### 1.5. 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 sl, ref string s2)
{
string tempStr = sl;
sl = s2;
s2 = tempStr;
}
```

Виклик методу:

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

### 1.6. 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());
```

### 1.7. 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 `ref` 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 `params`, которым должен быть помечен последний параметр метода. В реальности необходимость в использовании модификатора `params` возникает не особо часто, однако он применяется во многих методах внутри библиотек базовых классов |

## 1.8. Сигнатура методу

Сигнатура методу (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 | `static` |
| Модифікатори доступу | `public`, `internal`, `private`, `protected` |
| Модифікатори успадковування | `new`, `virtual`, `abstract`, `override`,`sealed` |
| Модифікатор частинного методу | `partial` |
| Модифікатор некерованого коду | `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
```

### 1.9. 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#
}
```