

КОНСТРУКТОРИ. ПЕРЕВАНТАЖЕННЯ КОНСТРУКТОРІВ

Конструктор – це спеціальний метод класу, що викликається при створенні об'єкту класу. Ім'я конструктора завжди співпадає з ім'ям класу. Якщо розробником класу явно не визначено жодного конструктора, то для класу автоматично створюється конструктор без аргументів. Конструктори не повертають жодних значень, тому при їх оголошенні не потрібно вказувати тип, навіть **void**

```
public class Car {                                //клас машина
    public string color;                          //колір
    public string model;                          //модель
    public int yearBuilt;                         //рік випуску

    //конструктор без аргументів
    public Car() {
        Console.WriteLine("Створюється машина");
    }
}
```

Для класу `Car` об'єкт створюємо за допомогою ключового слова **new**.

```
Car myCar = new Car();
```

При цьому при створенні об'єкту `myCar` буде визвано, явно заданий конструктор `Car()`

КОНСТРУКТОРИ. ПЕРЕВАНТАЖЕННЯ КОНСТРУКТОРІВ

Як і для інших, методів для конструкторів можливе перевантаження, тобто можна створювати конструктори з різними наборами аргументів. Найчастіше конструктори використовують, що б задавати початкові значення полів. Наприклад:

```
using System;
using System.Collections.Generic;
using System.Text;
public class Car {                                //клас машина
    public string color; //колір
    public string model; //модель
    public int yearBuilt; //рік випуску
    public Car() {
        //конструктор без аргументів
        Console.WriteLine("Створюється машина");
    }
    public Car(string c, string m, int yb) {
        //конструктор з аргументами
        color = c;
        model = m;
        yearBuilt = yb;
    }
}
class Test_60 {
    static void Main(string[] args) {
        Car c0 = new Car();
        Car c1 = new Car("Black", "BMW", 2000);
        Console.ReadKey();
    }
}
```

Використання this

Ключове слово **this** – використовується для посилання на поточний об'єкт. Одне з застосувань, те що воно дозволяє оголошувати аргументи з тими ж назвами, що і поля.

```
public Car(string color, string model,
           int yearBuilt) {

    this.color = color;
    this.model = model;
    this.yearBuilt = yearBuilt;
}
```

`this.color` – означає звертання до поля `color`, а просто `color` до аргументу конструктора `color`

Ключове слово **this** можна використовувати при посиланні на інший конструктор при перевантаженні, цей механізм дозволяє запобігати надмірному дублюванню коду програми.

```
public Car() {
    Console.WriteLine("Створюється машина");
}

public Car(string color, string model,
           int yearBuilt) : this() {

    this.color = color;
    this.model = model;
    this.yearBuilt = yearBuilt;
}
```

В цьому прикладі, для конструктору з аргументами спочатку здійсниться виклик конструктору без аргументів.

ВЛАСТИВОСТІ

Властивості – це один з механізмів інкапсуляції, вони одночасно дозволяють встановлювати та зчитувати значення полів за допомогою методів та приховувати поля від користувача. Властивості також зручно використовувати для перевірки введених даних на відповідність. Стандартний опис властивості має такий синтаксис:

```
[модификатор_доступа] возвращаемый_тип произвольное_название
{
    // код свойства
}
```

```
class Person
{
    private string name;

    public string Name
    {
        get
        {
            return name;
        }

        set
        {
            name = value;
        }
    }
}
```

Закрите поле **name** і є загальнодоступна властивість **Name** мають практично однакову назву за винятком регістра, але це не більше ніж стиль, назви у них можуть бути довільні. Через цю властивість можемо управляти доступом до змінної **name**. Стандартне визначення властивості містить блоки **get** і **set**. У блоці **get** повертаємо значення поля, а в блоці **set** встановлюємо. Параметр **value** представляє передане значення

ВЛАСТИВОСТІ

```
Person p = new Person();  
    // встановлюємо властивість - спрацьовує блок Set  
    // значення "Tom" передається до властивості value  
p.Name = "Tom";  
    // Отримуємо властивість та присваємо її  
    // змінній - спрацьовує блок Get  
string personName = p.Name;
```

Властивості дозволяють вкласти додаткову логіку, яка може бути необхідна, наприклад, при присвоєнні змінної класу будь-якого значення. Наприклад, нам треба встановити перевірку за віком:

```
class Person  
{  
    private int age;  
  
    public int Age  
    {  
        set  
        {  
            if (value < 18)  
            {  
                Console.WriteLine("Возраст должен быть больше 17");  
            }  
            else  
            {  
                age = value;  
            }  
        }  
        get { return age; }  
    }  
}
```

```
public class Car {  
    private string model;  
    public string Model {  
        //зчитує значення поля model  
        get { return model; }  
        //встановлює значення поля model  
        set { model = value; }  
    }  
}  
  
class Test 61 {  
    static void Main(string[] args) {  
  
        Car c0 = new Car();  
        c0.Model = "Audi";  
        Console.WriteLine(c0.Model);  
        Console.ReadKey();  
    }  
}
```

ВЛАСТИВОСТІ

Якщо визначено тільки метод **get** – то поле доступне тільки для читання, якщо **set** – то тільки для запису.

Властивості керують доступом до полів класу. Однак, якщо є 10 і більше полів, то визначати кожне поле і писати для нього однотипну властивість незручно. Тому в фреймворк .NET були додані автоматичні властивості. Вони мають скорочене оголошення:

```
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }
}
```

Автовластивостям можна задати значення за замовчуванням

```
class Person
{
    public string Name { get; set; } = "Tom";
    public int Age { get; set; } = 23;
}

class Program
{
    static void Main(string[] args)
    {
        Person pn = new Person();
        Console.WriteLine(pn.Name); // Tom
        Console.WriteLine(pn.Age);  // 23
        Console.Read();
    }
}
```

СТАТИЧНІ КОМПОНЕНТИ

Статичні поля, методи та властивості – задаються за допомогою ключового слова **static**, та належать самому класу, а не конкретному об'єкту. Тому і звертання до них виконується **за ім'ям класу**, а не за ім'ям об'єкту.

```
class Account
{
    public static decimal bonus = 100;
    public decimal totalSum;
    public Account(decimal sum)
    {
        totalSum = sum + bonus;
    }
}
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(Account.bonus);           // 100
        Account.bonus += 200;

        Account account1 = new Account(150);
        Console.WriteLine(account1.totalSum);        // 450

        Account account2 = new Account(1000);
        Console.WriteLine(account2.totalSum);        // 1300

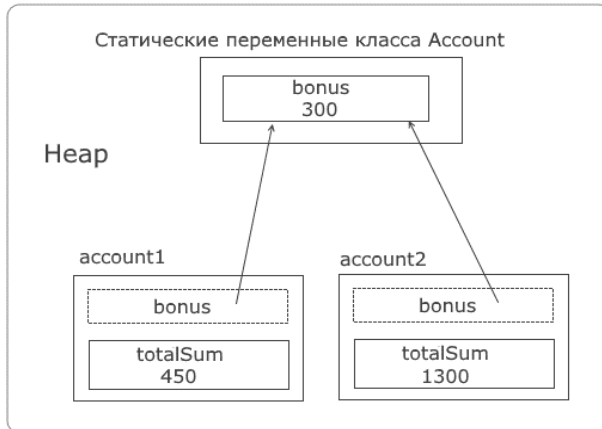
        Console.ReadKey();
    }
}
```

В даному випадку клас **Account** має два поля: **bonus** і **totalSum**. Поле **bonus** є статичним, тому воно зберігає стан класу в цілому, а не окремого об'єкта. Можемо звертатися до цього поля по імені класу:

```
Console.WriteLine(Account.bonus);
Account.bonus += 200;
```

СТАТИЧНІ КОМПОНЕНТИ

Статичні властивості і методи



Статичні члени класу є загальними для всіх об'єктів цього класу, тому до них треба звертатися по імені класу

```
Account.MinSum = 560;  
decimal result = Account.GetSum(1000, 10, 5);
```

```
class Account  
{  
    public Account(decimal sum, decimal rate)  
    {  
        if (sum < MinSum) throw new Exception("Недопустимая сумма!");  
        Sum = sum; Rate = rate;  
    }  
    private static decimal minSum = 100; // мінімальна допустима  
                                           //сума для усеґ рахунків  
    public static decimal MinSum  
    {  
        get { return minSum; }  
        set { if(value>0) minSum = value; }  
    }  
  
    public decimal Sum { get; private set; } //сума на рахунку  
    public decimal Rate { get; private set; } //ставка у відсотках  
    // визначення суми на рахунку через певний період за ставкою  
    public static decimal GetSum(decimal sum, decimal rate, int  
period)  
    {  
        decimal result = sum;  
        for (int i = 1; i <= period; i++)  
            result = result + result * rate / 100;  
        return result;  
    }  
}
```

Змінні і властивості, які зберігають стан, загальний для всіх об'єктів класу, слід визначати як статичні. І також методи, які визначають загальний для всіх об'єктів поведіння, також слід оголошувати як статичні.

СТАТИЧНІ КОМПОНЕНТИ

Слід враховувати, що **статичні методи** можуть звертатися тільки до статичних членів класу. Звертатися до нестатичних методів, полів, властивостей всередині статичного методу не можемо.

Статичні поля застосовуються для зберігання лічильників

```
class User
{
    private static int counter = 0;
    public User()
    {
        counter++;
    }

    public static void DisplayCounter()
    {
        Console.WriteLine($"Створено {counter} об'єктів User");
    }
}
class Program
{
    static void Main(string[] args)
    {
        User user1 = new User();
        User user2 = new User();
        User user3 = new User();
        User user4 = new User();
        User user5 = new User();

        User.DisplayCounter(); // 5

        Console.ReadKey();
    }
}
```

ДЕСТРУКТОРИ

Деструктор – це спеціальний метод класу, що викликається при знищенні об'єкту. Оголошується за ім'ям класу та за допомогою символу ~, без модифікаторів. Може використовуватись, наприклад, для закриття пов'язаних з об'єктом файлів, або для підрахунку створених об'єктів класу.

```
public class Car {
    public static int numOfCars = 0; //кількість машин

    public Car() { //конструктор
        ++numOfCars;
    }
    ~Car() { //деструктор
        --numOfCars;
    }
}

class Test 62 {
    static void Main(string[] args) {
        Car c0 = new Car();
        {
            Car c1 = new Car();
            Console.WriteLine(Car.numOfCars);
        }
        Console.WriteLine(Car.numOfCars);
        Console.ReadKey();
    }
}
```

СТАТИЧНИЙ КОНСТРУКТОР

Статичний конструктор – оголошується за допомогою ключового слова **static**, без модифікаторів та не має аргументів. Викликається один раз перед створенням першого екземпляру класу. Може використовуватись для ініціалізації статичних полів та різного типу налаштувань класу.

```
static Car() {  
    Console.WriteLine("Static Car");  
}
```

Статичні конструктори використовуються для ініціалізації статичних даних, або ж виконують дії, які потрібно виконати тільки один раз.

```
class User  
{  
    static User()  
    {  
        Console.WriteLine("Створений І користувач");  
    }  
}  
  
class Program  
{  
    static void Main(string[] args)  
    {  
        User user1 = new User();  
        // тут спрацює статичний конструктор  
        User user2 = new User();  
  
        Console.Read();  
    }  
}
```

БАГАТОВИМІРНІ МАСИВИ

Багатовимірні масиви

```
int[, ,]matrix3 = new int[5, 7, 5];  
int[, , ,]matrix4 = new int[4, 4, 3, 3];
```

Складність з перебором масиву

```
int[,] mas = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } };  
foreach (int i in mas)  
    Console.WriteLine($"{i} ");  
Console.WriteLine();
```

```
1 2 3 4 5 6 7 8 9 10 11 12
```

У кожного масиву є метод **GetUpperBound(dimension)**, який повертає індекс останнього елемента в певній розмірності. За допомогою виразу **mas.GetUpperBound(0) + 1** можна отримати кількість рядків таблиці, представленої двовірним масивом. Через **mas.Length / rows** можна отримати кількість елементів в кожному рядку:

```
1 int[,] mas = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } };  
2  
3 int rows = mas.GetUpperBound(0) + 1;  
4 int columns = mas.Length / rows;  
5 // или так  
6 // int columns = mas.GetUpperBound(1) + 1;  
7  
8 for (int i = 0; i < rows; i++)  
9 {  
10     for (int j = 0; j < columns; j++)  
11     {  
12         Console.WriteLine($"{mas[i, j]} \t");  
13     }  
14     Console.WriteLine();  
15 }
```

```
1 2 3  
4 5 6  
7 8 9  
10 11 12
```

СТАТИЧНІ КЛАСИ

Статичні класи оголошуються з модифікатором *static* і можуть містити тільки статичні поля, властивості і методи. Прикладами статичного класу є клас **Math** (*Math.PI*, *Math.Abs(x)*, *Math.Pow(x, y)*)

*Клас **Account** має тільки статичні змінні, властивості і методи, тому оголошений як статичний*

```
static class Account
{
    private static decimal minSum = 100;
    // мінімальна допустима сума для усіх рахунків
    public static decimal MinSum
    {
        get { return minSum; }
        set { if(value>0) minSum = value; }
    }

    // визначення суми на рахунку через певний період
    //за ставкою
    public static decimal GetSum(decimal sum, decimal rate, int
period)
    {
        decimal result = sum;
        for (int i = 1; i <= period; i++)
            result = result + result * rate / 100;
        return result;
    }
}
```

ПРОСТОРИ ІМЕН

Простори імен (ключове слово **namespace**) – призначені для локалізації імен ідентифікаторів, і попередження їх конфліктів. В .Net простори імен використовуються для: групування споріднених класів, для розв'язання конфлікту імен для різних класів з однаковими іменами.

```
namespace Graphics2D {  
    public class Point{  
        public int x,y;  
    }  
}  
  
namespace Graphics3D {  
    public class Point{  
        public double x,y,z;  
    }  
}
```

Обидва простори описані класами **Point**, але двовимірний в просторі імен **Graphics2D**, а тривимірний – в **Graphics3D**. Щоб використати в своєму коді двовимірний варіант, потрібно: або використовувати повне ім'я класу разом з ім'ям простору до якого він відноситься:

```
//...  
Graphics2D.Point p = new Graphics2D.Point();  
//...
```

```
using Graphics2D;  
//...  
Point p = new Point();  
//...
```