

## Розділ 6. Детальніше про класи

### 6.1. 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**. Приклад [5]:

```
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 описаного 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()`.

Як і для інших, методів для конструкторів можливе перевантаження, тобто можна створювати конструктори з різними наборами аргументів. Найчастіше конструктори використовують, щоб 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();
    }
}

```

## 6.2. 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. Так конструктор з попереднього прикладу можна переписати наступним чином:

```

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

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

```

тут, `this.color` – означатиме звертання до поля `color`, а просто `color` – до аргументу конструктора `color`. Також 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. Інший приклад:

```

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

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

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

```

Демонструє, як на базі існуючого створити конструктор з меншою кількістю аргументів.

### 6.3. 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.

Для властивостей можна одночасно визначати два методи **get** та **set** (**get** – для отримання значення, **set** – для встановлення) або по одинці кожен з них. Приклад:

```

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();
    }
}

```

При компіляції, ці методи перетворюються на **get\_Model** та **set\_Model** відповідно, тому при написанні програм не рекомендується створення методів з подібними іменами.

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

#### 6.4. Статичні компоненти. Деструктори

**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. Прикладами статичних полів та методів можуть слугувати поля та методи вже знайомого нам класу **Math** (такі як **Math.PI**, **Math.Abs(x)**, **Math.Pow(x, y)** і т. д.).

**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. Наприклад:

```

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();
    }
}

```

В цьому прикладі статичне поле **numOfCars** збільшується на одиницю при створенні об'єкту класу **Car**, та зменшується на одиницю при знищенні об'єкту. Проте при виводі значення поля **numOfCars**, найвирогідніше, в обох випадках буде число 2. Це відбуватиметься через те, що в середовищі **.Net** за виділення та звільнення пам'яті відповідає спеціальна служба **збирання сміття** (англ. garbage collection), яка звільняє пам'ять виділену під об'єкт не одразу ж при виході з зони видимості змінної, а при потребі. Для керування процесом збирання сміття в середовищі **.Net** створено спеціальний клас **System.GC**.

**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. Приклад:

```

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

```

## 6.5. Простори імен

*Простори імен* (ключове слово **namespace**) – призначені для локалізації імен ідентифікаторів, і попередження їх конфліктів. В .Net простори імен в основному використовуються для:

- групування споріднених класів:
- для розв'язання конфлікту імен для різних класів з однаковими іменами.

Синтаксис:

```
namespace <ім'я_простору> {  
    // Описова частина  
}
```

В якості прикладу розглянемо класи для двовимірної та тривимірної точки:

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

Обидві описані класами `Point`, але двовимірна в просторі імен `Graphics2D`, а тривимірна – в `Graphics3D`.

Тепер, 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();  
//...
```

або скористатися оператором **using**:

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

**Зауваження:** Простори імен можуть бути вкладеними один в один, створюючи таким чином ієрархію.

Наприклад:

```
namespace name1{  
    namespace name2{  
        //...  
    }  
}
```

Або те саме з використанням крапки:

```
namespace name1.name2{  
    //...  
}  
}
```

### **Контрольні запитання та завдання**

1. Поняття конструктору. Приклади створення перевантажених конструкторів.
2. Які використовується ключове слово `this`?
3. Що таке властивості? Приклади використання властивостей.
4. Які поля та методи називаються статичними? Навести приклади використання статичних компонентів.
5. Для чого використовуються простори імен?