ОБЕКТНО ОРИЕНТИРАНО ПРОГРАМИРАНЕ (C#)

доц. д-р Ивайло Дончев

катедра Информационни технологии

кабинет 501

Литература

- Albahari, J., Albahari, B., C# 7.0 in a Nutshell: The Definitive Reference, 1st Edition, O'Reilly, 2017
- Hilyard, J., Teilhet, St., C# 6.0 Cookbook: Solutions for C# Developers, 4th Edition, O'Reilly, 2015
- Nagel, Chr., Professional C# 7 and .NET Core 2.0, 7th Edition, Wrox, 2017
- Perkins, B., Hammer, J., Reid, J., Beginning C# 7 Programming with Visual Studio 2017, 1st Edition, Wrox, 2018
- Powers, L., Snell, M., Microsoft Visual Studio 2015 Unleashed, 3rd Edition, Sams Publishing, 2015
- Наков, С., Колев, В. и др., Принципи на програмирането със C#, Фабер, 2018
- C# documentation (form Microsoft) https://docs.microsoft.com/en-us/dotnet/csharp/

Лекция 1: Базови елементи на езика

- Пространства на имената (*Namespaces*)
- Преобразуване на типовете
- Функции
  - Сигнатура на функция
  - Параметри и аргументи
  - Функции с променлив брой аргументи
  - Предаване на параметрите по стойност и по референция
  - `out` параметри
  - Референции към променливи, дефинирани другаде
  - `in` параметри и `ref readonly` върнати сойности от функции
  - Предефиниране на функции
  - Параметри с подразбиращи се стойности
  - Именувани аргументи
- Делегати
- Изброявания (*Enums*)
- Switch изрази
- Индекси и диапазони

Пространства на имената (*Namespaces*)

- Чрез тях .NET осигурява контейнери за код на приложенията, така че съдържащите се в този код елементи да могат да бъдат еднозначно идентифицирани.
- Служат и за категоризиране на елементите на .NET Framework, повечето от които са дефиниции на типове: `System.Int32`, `System.String`
- Глобално пространство на имената (*global namespace*)
- Ключовата дума `namespace`
- Квалифицирани имена

Пространства на имената (*Namespaces*)

```
System.Collections.Generic.List<string> list;
```

- Операторът `using`

```
using System.Collections.Generic;
List<string> list;
```

- Операторът `using static`

```
using static System.Console;
WriteLine("It's OK");
```

Преобразуване на типовете

- Всички данни се записват в поредици от битове.
- Различните типове използват различни схеми за представяне на данните.

  `char` (`System.Char`) и `ushort` (`System.UInt16`)

- *Косвено (Implicit)* преобразуване

Всеки тип A, чийто диапазон на допустими стойности попада изцяло в диапазона допустими стойности на тип B, може да бъде косвено преобразуван до тип B.

Implicit numeric conversions table

Преобразуване на типовете. Явно (explicit) преобразуване.

```
(<destinationType>) <sourceVar>
```

- Възможна е загуба на данни
- Проверка за аритметично препълване – `checked` и `unchecked`.
- Преобразуване със `System.Convert`.

Функции

- Сигнатура – името и параметрите (без тип на резултата)
- Конвенцията *PascalCase*
- Общ вид на дефиниция:

```
[<static>] <тип_на_резултата> <Име>(<тип> <параметър>, ...)
{
    ...
    return <стойност>;
}
```

- Параметри и аргументи

Функции.
Expression-bodied methods.

```
public static int Product(int a, int b) => a * b;
```

вместо

```
public static int Product(int a, int b)
{
    return a * b;
}
```

Функции

- Функции с променлив брой аргументи
  - Специален параметър (*parameter array*)

```
static int MaxValue(params int[] intArray)
```

Може да се извиква така:

```
int maxVal = MaxValue(1, 8, 3, 6, 2, 5, 9, 3);
```

- Предаване на параметри по референция и по стойност.

```
static void Swap(ref int x, ref int y)
```

- Предаването на параметър по референция има две ограничения:
  - аргументът не може да бъде константа;
  - аргументът трябва да бъде **инициализирана** променлива.

Функции

- out параметри: Използва се модификатора `out`
  - може да се използва неинициализирана променлива като out аргумент;
  - функцията третира `out` параметъра като неинициализиран. Тоест дори аргумента да има някаква стойност в момента на извикването, тази стойност се губи, когато започне изпълнението на функцията.

```
static int MaxValue(int[] intArray, out int maxIndex)
```

- И в обръщението към функциите се добавят модификаторите `out` или `ref`

Референции към променливи, дефинирани другаде

- Знаейки позицията, на която се намира търсеният елемент, можем да променяме неговата стойност:

```
myArray[maxIndex] = 100;
foreach(var e in myArray)
   Console.Write($"{e} ");
Conole.WriteLine();
```

- Можем да получим референция към този елемент и без да използваме индекса му в клиентския код. Трябва да променим функцията така:

```
static ref int MaxValue(int[] intArray)
{
     //----------------------
     return ref intArray[maxIndex];
}
```

Референции към променливи, дефинирани другаде

- Трябва да се промени и функцията `Main()`

```
ref int maximum = ref MaxValue(myArray);
Console.WriteLine($"The maximum is {maximum}");
maximum = 200;
```

- Функцията `MaxValue()` може да се извиква и без модификатора `ref`, ако е необходимо само копие на стойността на максималния елемент, а не референция към него.

```
Console.WriteLine($"The maximum is {MaxValue(myArray)}");
int max = MaxValue(myArray);
```

- Функции, които връщат референции могат да се използват и като леви стойности – да стоят отляво на операцията за присвояване.

```
MaxValue(myArray) = 300;
```

`in` параметри и `ref readonly` върнати стойности от функции

- В C# 7.2, освен модификаторите `out` и `ref` за предаване на параметри по референция, с цел избягване на ненужно копиране, е добавен модификаторът `in`.
- `in` параметрите се предават по референция, но функцията няма право да променя техните стойности. Така аргументите не се копират и се подобрява ефективността на кода.
- `static int Sum(in int a, in int b) => a + b;`
- Обръщението към такива функции, за разлика от тези с `ref` и `out` аргументи, не изисква използване на `in`. Също така е допустимо извикването с константи и/или литерали.
- Има възможност функция да върне по референция резултат от стойностен тип (*value type*), но да забрани на клиентския код да модифицира тази стойност. Това се обявява в декларацията на функцията с модификатора `ref readonly` към типа на резултата.

in параметри и ref readonly върнати стойности от функции

- Да се върнем на примера с максималния елемент на масива. За да осигурим защита от презапис на елемента в него, само заглавният ред на функцията трябва да се промени:

```
static ref readonly int MaxValue(int[] intArray)
```

- В този случай обаче можем да използваме върнатия от функцията резултат само като дясна стойност:

```
ref readonly int maximum = ref MaxValue(myArray);
Console.WriteLine($"The maximum is {MaxValue(myArray)}");
maximum = 100;  // Грешка! Променливата е readonly.
MaxValue(myArray) = 200;   // Грешка!
```

- Лесно можем да направим копие на ref readonly върнатата стойност. Трябва просто да я присвоим на променлива, която не е декларирана като ref readonly.

Функции. Предефиниране.

- Това е възможността да имаме няколко функции с едно и също име, но с различни параметри (по брой и тип).
- Коя функция да бъде изпълнена решава компилаторът в зависимост от аргументите, с които е извикана.
- Възможно е параметрите на функциите да се различават само по това дали параметрите им се предават по стойност или по референция.

Функции. Параметри с подразбиращи се стойности

- Позволява асоцииране на параметъра с **константна** стойност в декларацията на метода. Това позволява извикване на метода с пропускане на подразбиращите се параметри.

```
public static int Power(int a, int n=2)
```

- Параметрите с подразбиращи се стойности трябва да са разположени след изискваните параметри (тези, които нямат подразбиращи се стойности).

```
static void Main(string[] args){
    Console.WriteLine(Power(2,3));  // 8
    Console.WriteLine(Power(5));    // 25
}
```

Функции. Именувани аргументи.

- При извикването на функцията може изрично да се посочи името на параметъра, на който да се присвои стойността, а не да се разчита на съответствие в подредбата на аргументите и параметрите.
- Това е удобно при функции с много параметри с подразбиращи се стойности.
- Недостатък е, че при промяна на името на параметъра трябва да се променя и клиентския код.

```
DisplayGreeting(lastName: "Donchev",
firstName:"Ivaylo");
```

Делегати

- Делегатът (*delegate*) е тип, чиито стойности са референции към функции.
- Типичното приложение на делегатите е при работата със събития (*event handling*).
- Декларирането на делегат е подобно на това на функция, но без тяло. Използва се ключовата дума `delegate`.
- Декларацията определя тип на резултата и списък с параметри на делегата.
- След декларирането на делегата могат да се декларират променливи от тип този делегат и те да се инициализират с референции на функции, които имат същия тип на резултата и списък с параметри.
- След това променливата-делегат може да се използва като функция. Това позволява чрез делегати функции да се предават като параметри на други функции.

Изброявания (*enums*)

```
enum Orientation { North, South, East, West }
```

- Изброяването дефинира тип от краен брой стойности, които ние задаваме. След това могат да се дефинират променливи от този тип и да им се присвояват стойности от изброяването.

```
Orientation orientation = Orientation.North;
```

- Изброяванията имат базисен тип, използван за съхраняване на стойностите. По подразбиране той е `int`.

Изброявания – общ вид

```
enum <typeName> : <underlyingType>
{
    <value1>,
    <value2>,
    <value3>,
    ...
    <valueN>
}
```

- Базисни типове могат да бъдат byte, sbyte, short, ushort, int, uint, long, и ulong.

```
enum Orientation : byte { North, South, East, West }
```

Изброявания

- По подразбиране на всяка стойност на изброяването се присвоява кореспондираща стойност на базисния тип, започвайки от нула.
- С оператора = могат да се задават други стойности.
- Стойностите може да се повтарят. Ако при тези присвоявания се получи цикъл, това предизвиква грешка.
- Възможно е преобразуване на типовете от изброяване към базисен и обратно.
- Възможно е преобразуване от `string` до изброяване с командата `Enum.Parse()`.

Switch изрази

- Това е нововъведение в C#8, което позволява по-сбит и интуитивен синтаксис – с по-малко повторения на ключовите думи case и break и по-малко скоби. Често операторът switch произвежда стойност във всеки от case блоковете си.

```
public enum Rainbow { Red,
                      Orange,
                      Yellow,
                      Green,
                      Blue,
                      Indigo,
                      Violet }
```

- Приемаме, че в програмата ни има дефиниран тип `RGBColor`, който конструира цвят от 3 компонента – цели числа. Искаме да преобразуваме цвят от тип Rainbow в тип `RGBColor`.

Switch изрази

С класически оператор switch можем да напишем следната функция:

```
public static RGBColor FromRainbowClassic(Rainbow colorBand)
        {
            switch (colorBand)
            {
                case Rainbow.Red:
                    return new RGBColor(0xFF, 0x00, 0x00);
                case Rainbow.Orange:
                    return new RGBColor(0xFF, 0x7F, 0x00);
                case Rainbow.Yellow:
                    return new RGBColor(0xFF, 0xFF, 0x00);
                case Rainbow.Green:
                    return new RGBColor(0x00, 0xFF, 0x00);
                case Rainbow.Blue:
                    return new RGBColor(0x00, 0x00, 0xFF);
                case Rainbow.Indigo:
                    return new RGBColor(0x4B, 0x00, 0x82);
                case Rainbow.Violet:
                    return new RGBColor(0x94, 0x00, 0xD3);
                default:
                    throw new ArgumentException(message: "invalid enum
value", paramName: nameof(colorBand));
            }

        }
```

Switch изрази

С помощта на switch израз същата функция изглежда така:

```
public static RGBColor FromRainbow(Rainbow colorBand) =>
    colorBand switch
    {
        Rainbow.Red => new RGBColor(0xFF, 0x00, 0x00),
        Rainbow.Orange => new RGBColor(0xFF, 0x7F, 0x00),
        Rainbow.Yellow => new RGBColor(0xFF, 0xFF, 0x00),
        Rainbow.Green => new RGBColor(0x00, 0xFF, 0x00),
        Rainbow.Blue => new RGBColor(0x00, 0x00, 0xFF),
        Rainbow.Indigo => new RGBColor(0x4B, 0x00, 0x82),
        Rainbow.Violet => new RGBColor(0x94, 0x00, 0xD3),
        _ => throw new ArgumentException(message: "invalid
enum value", paramName: nameof(colorBand)),
            };
```

Индекси и диапазони

- Индексите и диапазоните осигуряват кратък синтаксис за достъп до отделни елементи или диапазони в последователности.
- В езика са въведени два нови типа и две операции:
  - типът `System.Index` представя индекс в последователност;
  - операцията „индекс от края“ (*index from end operator*) `^`, определя индекса като относителен спрямо края на последователността;
  - типът `System.Range` представя поддиапазон в последователност;
  - операцията диапазон (*range*) `..` чрез своите операнди определя началото и края на диапазона .

Индекси и диапазони

- Нека имаме дефиниран масив

```
int[] sequence = { 1,2,3,4,5};
```

- Индекс `0` се отнася до елемента `sequence[0]`.
- Индекс `^0` съответства на `sequence[sequence.Length]`.
- За всяко число `n` индексът `^n` е същият като `sequence.Length - n`.
- Диапазонът се определя от начало и край.
- Началото попада в диапазона, но краят - не.
- Така диапазонът `[0..^0]` представя целия диапазон, подобно на `[0..sequence.Length]`.

Индекси и диапазони (пример)

```
var words = new string[]
        {
                        // index from start    index from end
            "The",      // 0                   ^9
            "quick",    // 1                   ^8
            "brown",    // 2                   ^7
            "fox",      // 3                   ^6
            "jumped",   // 4                   ^5
            "over",     // 5                   ^4
            "the",      // 6                   ^3
            "lazy",     // 7                   ^2
            "dog"       // 8                   ^1
        };              // 9 (or words.Length) ^0
```

```
Console.WriteLine($"The last word is {words[^1]}"); // writes "dog"
            var quickBrownFox = words[1..4]; // quick, brown, fox
            var lazyDog = words[^2..^0];     // lazy, dog
```

Индекси и диапазони (примери)

```
var allWords = words[..];//contains "The" through "dog".
var firstPhrase = words[..4];//contains "The" through "fox"
var lastPhrase = words[6..];//contains "the", "lazy" and "dog„
```

- Диапазони могат да се дефинират и като променливи:

```
Range phrase = 1..4;
var text = words[phrase];
```

- Още един пример: Обхожда масива и извежда елементите му в обратен ред

```
int[] mas = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
for(int i=1; i<=mas.Length; i++)
     Console.Write($"{ mas[^i]} ");  //  index from end operator ^
Console.WriteLine();
```

Благодаря за вниманието!

- Въпроси?