

C# Osnove i LINQ

G04Code

Marko Cvijanović
Mihailo Bubnjević

Informacije i način predavanja

- Svaki dan posle predavanja ćete imati praktični deo gde ćete rešavati relevantne zadatke
- Rešenje zadatka sa praktičnog dela ćete držati na svom [Github](#) repozitorijumu
- Rešenja pushovati najkasnije do početka sledećih predavanja
- Za C# i .NET koristićemo [Visual Studio 2022](#) (instalirati ASP.NET and web development i .NET desktop development paket)
- Za Typescript i Angular ćemo koristiti [Visual Studio Code](#)

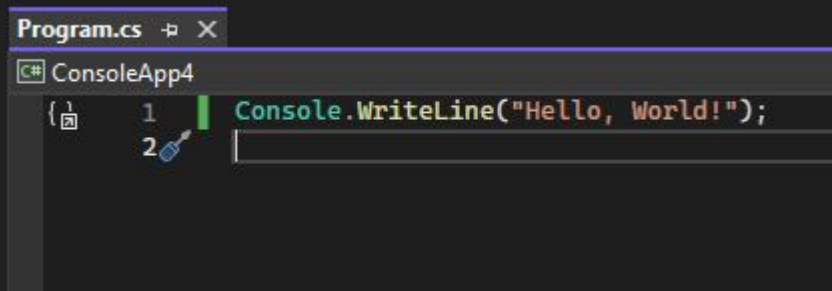
Uvod u C#

- Objektno-orijentisan jezik
- Type-safe
- Statično pisan
- Kompajliran
- [C# Dokumentacija](#)

Kreiranje C# projekta

Create a new project -> Console Application -> Imenovati Projekat i Solution

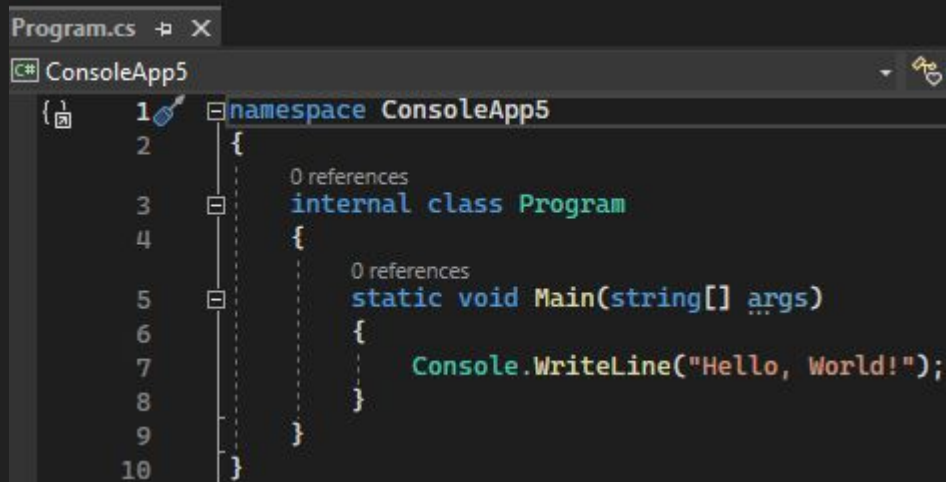
With top-level statements



The screenshot shows a Visual Studio window with a file named 'Program.cs' and a project named 'ConsoleApp4'. The code in the editor is a single line: `Console.WriteLine("Hello, World!");`. The line number 1 is visible on the left margin.

```
Program.cs [X]
C# ConsoleApp4
1 Console.WriteLine("Hello, World!");
```

Without top-level statements



The screenshot shows a Visual Studio window with a file named 'Program.cs' and a project named 'ConsoleApp5'. The code in the editor is structured with a namespace and an internal class. The line numbers 1 through 10 are visible on the left margin.

```
Program.cs [X]
C# ConsoleApp5
1 namespace ConsoleApp5
2 {
3     0 references
4     internal class Program
5     {
6         0 references
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Hello, World!");
10        }
11    }
12 }
```

Tipovi podataka (Data types)

- Vrednosni tipovi

- int
- long
- float
- double
- char
- byte
- decimal
- short
- bool

- Referentni tipovi

- object
- string
- dynamic
- array
- class
- interface
- delegate

Kastovanje tipova (Type casting)

- Implicitno
 - char -> int -> long -> float -> double
- Eksplicitno
 - double -> float -> long -> int -> char

```
int myInt = 9;  
double myDouble = myInt;  
// Automatsko kastovanje: int u double
```

```
double myDouble = 9.78;  
int myInt = (int)myDouble;  
// Eksplicitno kastovanje: double u int
```

Iskazi i petlje (Statements & loops)

- if / else / else if
- ternary operator

```
if (condition)
{
    // uradi ako je condition True
}
else if(otherCondition)
{
    // uradi ako je condition False a otherCondition True
}
else
{
    // uradi ako su oba False
}
```

```
var result = condition ? doIfTrue : doIfFalse;
```


Iskazi i petlje (Statements & loops)

- switch case

```
string season = "winter";
switch (season)
{
    case "spring":
        Console.WriteLine("spring");
        break;
    case "summer":
        Console.WriteLine("summer");
        break;
    case "autumn":
        Console.WriteLine("autumn");
        break;
    case "winter":
        Console.WriteLine("winter");
        break;
}
```

Iskazi i petlje (Statements & loops)

- for / for each

```
for (int i = 0; i < 5; i++)  
{  
    Console.WriteLine(i);  
}  
// Output: 0 1 2 3 4
```

```
int[] numbers = { 0, 1, 2, 3 };  
foreach (int number in numbers)  
{  
    Console.WriteLine(number);  
}  
// Output: 0 1 2 3
```

Iskazi i petlje (Statements & loops)

- while

```
while (condition)
{
    // code block to be executed
}
```

```
do
{
    // code block to be executed
}
while (condition);
```

Iskazi i petlje (Statements & loops)

- break / continue

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine("looking for a 7...");
    if(i == 7)
    {
        Console.WriteLine("found a 7!");
        break;
    }
}
```

```
int[] numbers = { 0, 1, 2, 3, 4 };
foreach (int number in numbers)
{
    if (number % 2 != 0)
    {
        continue;
    }
    Console.WriteLine("Its even");
}
```

Iskazi i petlje (Statements & loops)

- try / catch / finally

```
try
{
    // execute statement
}
catch (Exception e)
{
    // handle execeptions
}
finally
{
    // always execute
}
```

Objektno orijentisano programiranje u C#

Stubovi objekto orijentisanog programiranja:

- Enkapsulacija
- Apstrakcija
- Nasledjivanje
- Polimorfizam

Objektno orijentisano programiranje u C#

- Enkapsulacija - sprovodi se putem access modifier-a kao što su:
 - private
 - protected
 - internal

```
class Car
{
    private int speed;
    0 references
    public int Speed
    {
        get { return speed; }
        set { speed = value; }
    }
}
```

Objektno orijentisano programiranje u C#

- Apstrakcija

```
// Abstract class
1 reference
abstract class Animal
{
    // Abstract method (does not have a body)
    1 reference
    public abstract void animalSound();
    // Regular method
    0 references
    public void sleep()
    {
        Console.WriteLine("Zzz");
    }
}

// Derived class (inherit from Animal)
0 references
class Cat : Animal
{
    1 reference
    public override void animalSound()
    {
        Console.WriteLine("Meow");
    }
}
```


Objektno orijentisano programiranje u C#

- Nasledjivanje

```
// Abstract class
1 reference
abstract class Animal
{
    // Abstract method (does not have a body)
    1 reference
    public abstract void animalSound();
    // Regular method
    0 references
    public void sleep()
    {
        Console.WriteLine("Zzz");
    }
}

// Derived class (inherit from Animal)
0 references
class Cat : Animal
{
    1 reference
    public override void animalSound()
    {
        Console.WriteLine("Meow");
    }
}
```

Objektno orijentisano programiranje u C#

- Polimorfizam
 - Compile Time Polymorphism / Static Polymorphism
 - Run-Time Polymorphism / Dynamic Polymorphism

```
3 references
class Polygon
{
    // method to render a shape
    2 references
    public virtual void render()
    {
        Console.WriteLine("Rendering Polygon...");
    }
}

0 references
class Square : Polygon
{
    // overriding render() method
    2 references
    public override void render()
    {
        Console.WriteLine("Rendering Square...");
    }
}
```

Kolekcije

- ArrayList - kolekcija objekata
- List<T> - generic kolekcija tipa T
- Dictionary<TKey, TValue> - generic kolekcija parova ključ (tipa TKey) i vrednost (tipa TValue)

IEnumerable i LINQ

- Language-Integrated Query
- IEnumerable
 - interfejs koji omogućava iteriranje
 - podrška za LINQ
 - sadrži extension metode

LINQ

Definicija klase koju ćemo koristiti u narednim primerima

```
1 reference
class Person
{
    public string name;
    public int age;
    public Gender gender;

    0 references
    public Person(string name, int age, Gender gender)
    {
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
}

2 references
enum Gender
{
    Male,
    Female
}
```

LINQ

- Where - filtriranje

```
// upit sintaksa
var resultQuery =
    from person in people
    where person.gender == Gender.Male
    select person;

// metod sintaksa
var resultMethod = people.Where(person => person.gender == Gender.Male);
```

LINQ

- Select - projekcija

```
// upit sintaksa  
var resultQuery =  
    from person in people  
    select person.name;
```

```
// metod sintaksa  
var resultMethod = people.Select(person => person.name);
```

LINQ

- Select - transformacija

```
// upit sintaksa
var resultQuery =
    from person in people
    select new Person(person.name, person.age + 5, person.gender);

// metod sintaksa
var resultMethod = people.Select(person => new Person(person.name, person.age + 5, person.gender));
```


LINQ

- OrderBy - sortiranje

```
// upit sintaksa
var resultQuery =
    from person in people
    orderby person.age ascending
    select person;

// metod sintaksa
var resultMethod = people.OrderBy(person => person.age);
```

LINQ

- GroupBy - grupisanje

```
// upit sintaksa
var resultQuery =
    from person in people
    group person by person.gender into genderGroup
    select genderGroup;

// metod sintaksa
var resultMethod = people.GroupBy(person => person.gender);
```

Zadaci

1. Napraviti konzolnu aplikaciju koja će se ponašati kao kalkulator za množenje, gde će od korisnika primati brojeve i ispisivaće njegov kvadrat sve dok korisnik ne unese "x" u kom momentu aplikacija treba da završi sa radom.
2. Napraviti konzolnu aplikaciju koja od korisnika traži pozitivan broj n i ispisuje n brojeva u fibonačijevom nizu.
3. Iz liste brojeva `numbers` prikazati brojeve koji su deljivi sa brojem n koji unosimo preko konzole. Za inicijalizaciju liste brojeva koristiti ugradjenu funkciju `Enumerable.Range(start, count)` koja vraća "count" brojeva pocev od "start".
4. Kreirati klasu "Osoba" koja sadrzi polja koja oznacavaju ime, starost i pol osobe. Pol implementirati kao enum.

Zadaci

6. Za kolekciju objekata tipa "Osoba" iz zadatka 2 napisati LINQ upit koji vraca listu osoba sortiranih po broju godina, opadajuce.

7. Za kolekciju objekata tipa "Osoba" iz zadatka 2 napisati LINQ upit koji vraca imena i godine osoba grupisanih po polu osobe.