

Lambda izrazi

S pomoću lambda izraza kreiramo anonimne funkcije unutar metoda

Lambda operator => odvađa parametre od izraza

„stari” način

```
int KlasicnaMetoda(int x)
{
    return x * x;
}

void PozivKlasicneMetode()
{
    Console.WriteLine(KlasicnaMetoda(5));
}
```

```
PozivKlasicneMetode();
```

```
> Terminal – Lambda
```

```
25
```

Izraz (expression) lambda

```
// nismo pisali metodu već smo
// ju definirali kao varijablu
var kvadrat = (int x) => x * x;

Console.WriteLine(kvadrat(5));
```

```
> Terminal – Lambda
```

```
25
```

Izjavna (statement) lambda

```
var algoritam = (int x, int y) => {
    var t = x++ + --y;
    return x + y - t;
};

Console.WriteLine(algoritam(1, 2));
```

```
> Terminal – Lambda
```

```
1
```

Izvori:

<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-expressions>

<https://www.programiz.com/csharp-programming/lambda-expression>

Lambda izrazi

Primjeri rada s lambda

```
// niz brojeva
int[] brojevi = { 2, 7, 8, 4, 7, 5 };

// lambda expression kao parametar metode
// vraća ukupan broj elemenata koji odgovaraju uvjetu
int ukupno = brojevi.Count(x => x == 7);

Console.WriteLine("ukupno brojeva 7: " + ukupno);

Console.WriteLine("ukupno brojeva većih ili jednako 7: " +
    brojevi.Count(b => b >= 7));
```

>- Terminal – Lambda

```
ukupno brojeva 7: 2
ukupno brojeva većih ili jednako 7: 3
```

Izvori:

<https://zetcode.com/csharp/lambda-expression/>

Iteriranje nizova i listi

```
// niz brojeva
int[] brojevi = { 2, 7, 8, 4, 7, 5 };

//standardni for
for (int i=0;i<brojevi.Length;i++)
{
    Console.WriteLine(brojevi[i]);
}
Console.WriteLine("-----");

// standardni foreach
foreach (int b in brojevi){
    Console.WriteLine(b);
}
Console.WriteLine("-----");

// ForEach metoda na klasi Array

Array.ForEach(brojevi, Console.WriteLine);

Console.WriteLine("-----");

// želimo nešto posebno s svakim brojem (npr. ispisati uvećano za 1)
Array.ForEach(brojevi, b =>
{
    Console.WriteLine(b+1);
});
```

> Terminal – Lambda

```
2
7
8
4
7
5
-----
2
7
8
4
7
5
-----
2
7
8
4
7
5
-----
3
8
9
5
8
6
```

Iteriranje liste

```
var lista = new List<int> { 2, 7, 8, 4, 7, 5 };

//standardni for
for (int i = 0; i < lista.Count; i++)
{
    Console.WriteLine(lista[i]);
}
Console.WriteLine("-----");

// standardni foreach
foreach (int b in lista)
{
    Console.WriteLine(b);
}

Console.WriteLine("-----");

lista.ForEach(Console.WriteLine);

Console.WriteLine("-----");

lista.ForEach(b => Console.WriteLine(b+1));
```

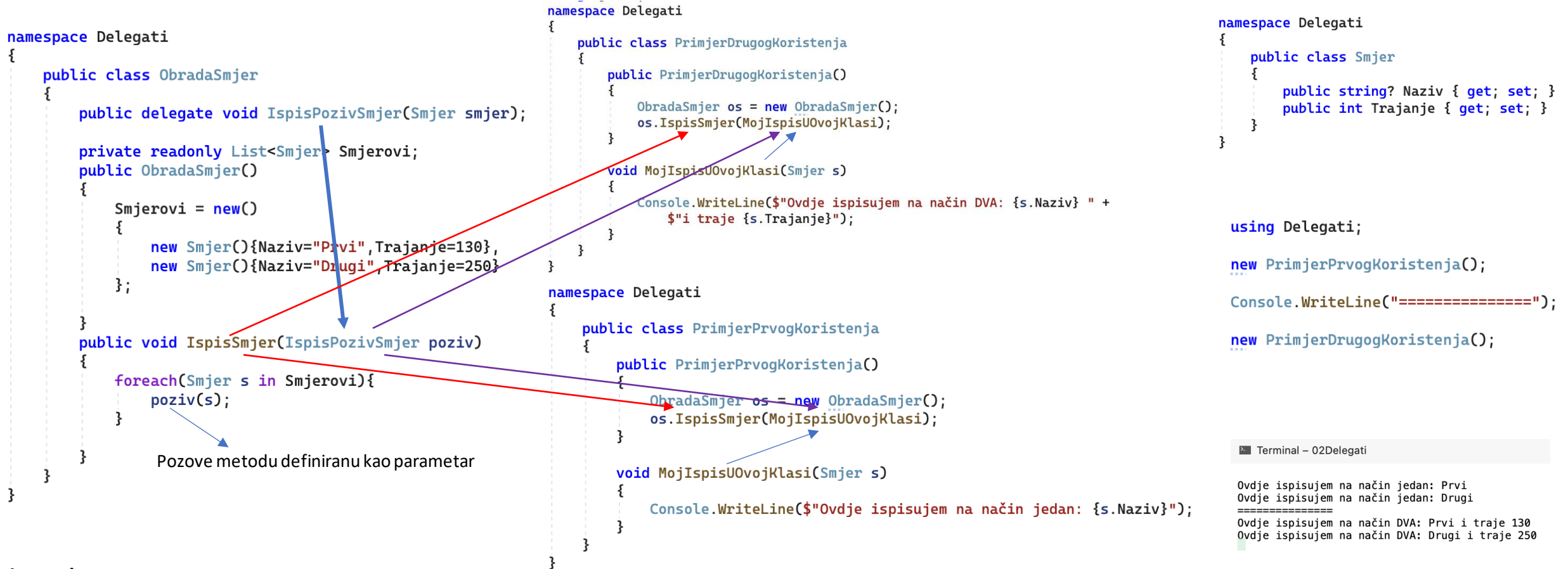
public delegate void IspisPozivSmjer(Smjer smjer);

Terminal – Lambda

2
7
8
4
7
5-----
2
7
8
4
7
5-----
2
7
8
4
7
5-----
3
8
9
5
8
6

Delegati

Delegat je tip koji predstavlja referencu na metodu s određenim popisom parametara i tipom povrata.



Izvori:

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/>

https://www.tutorialspoint.com/csharp/csharp_delegates.htm

```

namespace Delegati
{
    public class ObradaSmjer
    {
        public delegate void IspisPozivSmjer(Smjer smjer);

        private readonly List<Smjer> Smjerovi;
        public ObradaSmjer()
        {
            Smjerovi = new()
            {
                new Smjer(){Naziv="Prvi",Trajanje=130},
                new Smjer(){Naziv="Drugi",Trajanje=250}
            };
        }
        public void IspisSmjer(IspisPozivSmjer poziv)
        {
            foreach(Smjer s in Smjerovi){
                poziv(s);
            }
        }
        public void IspisSmjerPomocuAction(Action<Smjer> poziv)
        {
            foreach (Smjer s in Smjerovi)
            {
                poziv(s);
            }
        }
        public int ProcjenaTrajanje(Func<Smjer,int> poziv)
        {
            int ukupno=0;
            foreach (Smjer s in Smjerovi)
            {
                ukupno+=poziv(s);
            }
            return ukupno;
        }
    }
}

```

Delegati pomoći Action<T> i Func<T>

```

namespace Delegati
{
    public class PrimjerTrecegKoristenja
    {
        public PrimjerTrecegKoristenja()
        {
            ObradaSmjer os = new();
            Action<Smjer> action = new(MojIspisUOvojKlasi);
            os.IspisSmjerPomocuAction(action);
        }
        void MojIspisUOvojKlasi(Smjer s)
        {
            Console.WriteLine("Ovdje ispisujem na način tri: " +
                               $"{s.Naziv?[..2]}");
        }
    }
}

```

```

namespace Delegati
{
    public class PrimjerCetvrtogKoristenja
    {
        public PrimjerCetvrtogKoristenja()
        {
            ObradaSmjer os = new();
            int procjena = os.ProcjenaTrajanje(ProcjenaTrajanjeUOvojKlasi);
            Console.WriteLine($"Ovdje ispisujem na način četiri: {procjena}");
        }
        int ProcjenaTrajanjeUOvojKlasi(Smjer s)
        {
            return s.Naziv==null ? 0 : s.Naziv.Contains('P')
                ? s.Trajanje - 10 : s.Trajanje - 5;
        }
    }
}

```

```


using Delegati;

new PrimjerTrecegKoristenja();

Console.WriteLine("=====");

new PrimjerCetvrtogKoristenja();

```

 Terminal – 02Delegati

```

Ovdje ispisujem na način tri: Pr
Ovdje ispisujem na način tri: Dr
=====
Ovdje ispisujem na način četiri: 365

```

Izvori:

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/>

<https://www.infoworld.com/article/3057152/how-to-work-with-action-func-and-predicate-delegates-in-c-sharp.html>

Delegat pomoći Predicate<T>

Predikat je delegat koji prihvaća jedan ili više generičkih parametara i vraća bool vrijednost.

Predikatni delegati obično se koriste za izvođenje operacija pretraživanja na temelju skupa kriterija.

Promijeniti potpis liste u ObradaSmjer.cs

```
public List<Smjer> Smjerovi { get; }

namespace Delegati
{
    public class PrimjerPetogKoristenja
    {
        public PrimjerPetogKoristenja()
        {
            ObradaSmjer os = new();
            Predicate<Smjer> Trajanje = s => s.Trajanje == 130;
            Smjer? s = os.Smjerovi.Find(Trajanje);
            Console.WriteLine(s?.Naziv);
        }
    }
}
```

Za podsjetnik kako izgledaju podaci

```
Smjerovi = new()
{
    new Smjer(){Naziv="Prvi",Trajanje=130},
    new Smjer(){Naziv="Drugi",Trajanje=250}
};
```

```
using Delegati;
```

```
new PrimjerPetogKoristenja();
```

```
> Terminal – 02Delegati
```

```
Prvi
```

Izvori:

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/>

<https://www.infoworld.com/article/3057152/how-to-work-with-action-func-and-predicate-delegates-in-c-sharp.html>

Ekstenzije

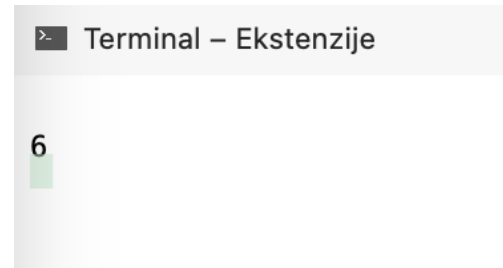
Omogućuje dodavanje metoda objektima bez definiranja metode na klasi iz koje je instanciran objekt

```
namespace Ekstenzije
{
    public static class Ekstenzije
    {
        public static int SumVeceOdNula(this int[] niz)
        {
            var suma = 0;
            foreach( int b in niz)
            {
                if (b > 0)
                {
                    suma += b;
                }
            }
            return suma;
        }
    }
}
```

```
using Ekstenzije;

int[] brojevi = { 2,2,-2,2};

Console.WriteLine(brojevi.SumVeceOdNula());
```



```
> Terminal – Ekstenzije

6
```

Izvor:

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods>


```
namespace Ekstenzije
{
    public class Smjer
    {
        // možemo postaviti vrijednost da nikada ne bude null
        public string Naziv { get; set; } = "";
        public int Trajanje { get; set; }
    }
}
```

```
namespace Ekstenzije
{
    public class Grupa
    {
        // ? označava kako Smjer može biti null
        public Smjer? Smjer { get; set; }
    }
}
```

```
using Ekstenzije;
```

```
Grupa g = new();
g.PostaviSmjer();
Console.WriteLine("Naziv: {0}, trajanje: {1}", g.Smjer?.Naziv, g.Smjer?.Trajanje);
```

```
g = new();
g.PostaviSmjer(s =>
{
    s.Naziv = "Test";
    s.Trajanje = 135;
});
Console.WriteLine("Naziv: {0}, trajanje: {1}", g.Smjer?.Naziv, g.Smjer?.Trajanje);
```

```
namespace Ekstenzije
{
    public static class Ekstenzije
    {
        public static Grupa PostaviSmjer(this Grupa g,
            Action<Smjer>? actionSmjer = null)
        {
            Smjer s = new();

            /*
            if (actionSmjer != null)
            {
                actionSmjer.Invoke(s);
            }
            */

            // ako nije null actionSmjer pozovi na njemu invoke
            // sljedeća linija ekvivalent prethodnom if-u
            actionSmjer?.Invoke(s);
            g.Smjer = s;

            return g;
        }
    }
}
```

Terminal – Ekstenzije

```
Naziv: , trajanje: 0
Naziv: Test, trajanje: 135
```

```

namespace Ekstenzije
{
    public interface ISucelje
    {
        public void Posao();
    }
}

namespace Ekstenzije
{
    public class Smjer : ISucelje
    {
        // možemo postaviti vrijednost da nikada ne bude null
        public string Naziv { get; set; } = "";
        public int Trajanje { get; set; }

        public void Posao()
        {
            Console.WriteLine("Odrađujem posao na smjeru, " +
                "npr ispisujem trajanje smjera: {0}", Trajanje);
        }
    }
}

namespace Ekstenzije
{
    public class Grupa : ISucelje
    {
        // ? označava kako Smjer može biti null
        public Smjer? Smjer { get; set; }

        public void Posao()
        {
            // Smjer?.Naziv -> ako je Smjer null neće se pozvati Naziv
            Console.WriteLine("Odrađujem posao na grupi, " +
                "npr ispisujem naziv smjera: {0}", Smjer?.Naziv);
        }
    }
}

```

```

namespace Ekstenzije
{
    public static class Ekstenzije
    {
        public static void OdradiPosao(this ISucelje s)
        {
            s.Posao();
        }
    }
}

```

```

using Ekstenzije;

Smjer smjer = new();

smjer.OdradiPosao();

var gr = new Grupa();
gr.OdradiPosao();

```

 Terminal – Ekstenzije

```

Odrađujem posao na smjeru, npr ispisujem trajanje smjera: 0
Odrađujem posao na grupi, npr ispisujem naziv smjera:

```