LINQ

Klasa Enumerable

```
public static class Enumerable
```

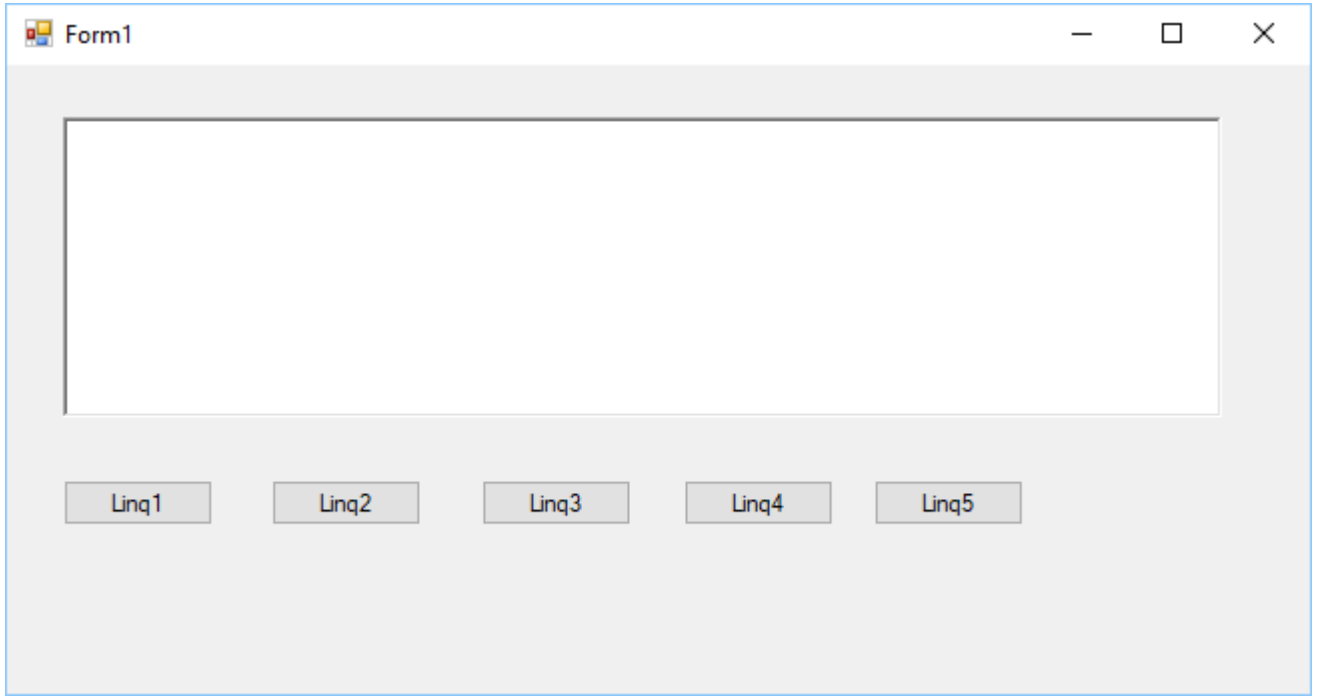
- Nalazi se u prostoru imena **System.Linq**
- **TSource** je generički tip elemenata izvora podataka
- Statičke metode klase **Enumerable** su ekstenzione metode koje proširuju izvor podataka koji implementira **IEnumerable<TSource>** intefejs
- Ove metode omogućavaju pisanje **LINQ** upita nad bilo kojim tipom koji primenjuje interfejs IEnumerable<T>
- **LINQ** (Language Integrated Query) omogućava pisanje upita nad kolekcijama objekata korišćenjem standardne C# sintakse

Ekstenziona metoda Select klase Enumerable

```
 public static IEnumerable<TResult> Select<TSource, TResult>(
    this IEnumerable<TSource> source,
    Func<TSource, TResult> selector
)
```

- Ekstenziona metoda **Select** proširuje tip **IEnumerable<TSource>**
- **source** predstavlja izvor podataka koji implementira interfejs IEnumerable<TSource>
- **selector** je funkcija transformacije koja za svaki element tipa TSource proizvodi vrednost tipa **TResult**
- Povratna vrednost je nova sekvenca tipa **IEnumerable<TResult>**, dobijena primenom funkcije selector nad elementima izvora

Korisnički interfejs



Primer upotrebe ekstenzione metode Select()

```
private void button1_Click(object sender, EventArgs e)
{
    int[] brojevi = { 1, 3, 5, 15, 9, 24 };

    IEnumerable<int> kvadrati = brojevi.Select(x => x * x);
    StringBuilder sb = new StringBuilder();
    foreach (int i in kvadrati)
    {
        sb.AppendLine(i.ToString());
    }
    richTextBox1.Text = sb.ToString();
}
```

Ekstenziona metoda Where klase Enumerable

```
public static IEnumerable<TSource> Where<TSource>(
     this IEnumerable<TSource> source,
     Func<TSource, bool> predicate
 )
```

- Ekstenziona metoda **Where** proširuje tip **IEnumerable<TSource>**
- source je izvor podataka koji implementira interfejs IEnumerable<TSource>
- Parametar **predicate** je funkcija (najčešće lambda izraz) tipa Func<TSource, bool> koja za dati element vraća true ili false
- Metoda Where vraća novu sekvencu **IEnumerable<TSource>** koja sadrži samo one elemente izvora za koje predicate vraća **true**

Primer upotrebe Where ekstenzione metode

```
private void button2_Click(object sender, EventArgs e)
{
    int[] brojevi = { 0, 1, 2, 3, 4, 5, 6 };

    // definisanje LINQ upita (filtriranje)
    IEnumerable<int> upit = brojevi.Where(n => n % 2 == 0);

    StringBuilder sb = new StringBuilder();

    foreach (int i in upit)
    {
        sb.AppendLine(i.ToString());
    }
     richTextBox1.Text = sb.ToString();
}
```

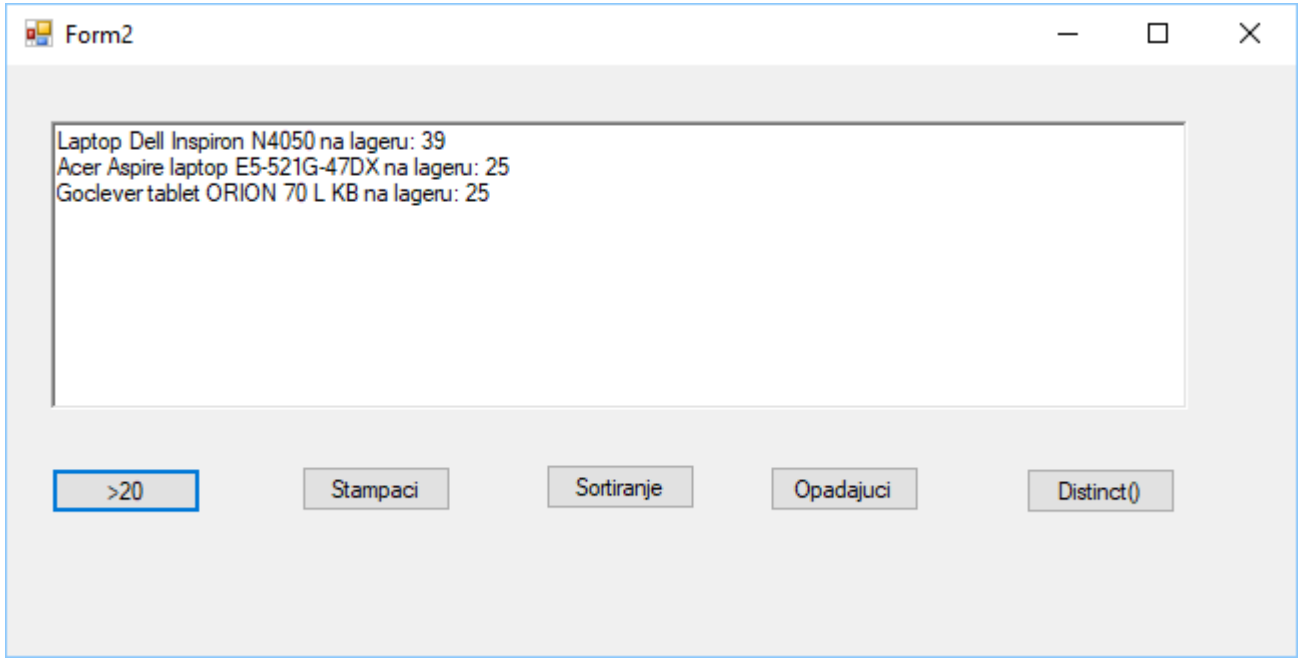
Primer upotrebe Where ekstenzione metode

```
private void button3_Click(object sender, EventArgs e)
{
    string[] imena = { "Marko", "Lazar", "Ivan", "Marija", "Jovana", "Jovan", "Desimir" };

    IEnumerable<string> upit = imena
        .Where(s => s.StartsWith("m",StringComparison.OrdinalIgnoreCase));


    StringBuilder sb = new StringBuilder();

    foreach (string n in upit)
    {
        sb.AppendLine(n);
    }

    richTextBox1.Text = sb.ToString();
}
```

Kombinovanje metoda Where() i Select()

```
private void button4_Click(object sender, EventArgs e)
{
    string[] imena = { "Aca", "Pera", "Mika", "Lazar", "Ivana", "Jovan", "Jovana", "Djordje" };
    IEnumerable<string> upit = imena
        .Where(s => s.Length == 5)
        .Select(s => s.ToUpper());
    StringBuilder sb = new StringBuilder();

    foreach (string clan in upit)
    {
        sb.AppendLine(clan);
    }
    richTextBox1.Text = sb.ToString();
}
```

Linq upiti nad kolekcijama objekata



Klasa Kategorija

```
class Kategorija
 {
     public int KategorijaId { get; set; }
     public string NazivKategorije { get; set; }
     public string OpisKategorije { get; set; }
 }
```

Klasa Proizvod

```
class Proizvod
    {
        public int ProizvodId { get; set; }
        public int KategorijaId { get; set; }
        public string NazivProizvoda { get; set; }
        public decimal Cena { get; set; }
        public int KolicinaNaLageru { get; set; }
    }
```

Metoda VratiKategorije(), statička klasa Podaci

```
public static List<Kategorija> VratiKategorije()
{
    List<Kategorija> listaKategorija = new List<Kategorija>() {
        new Kategorija{KategorijaId=1,NazivKategorije="Laptopovi", OpisKategorije="Laptopovi razlicitih proizvodjaca" },
        new Kategorija{KategorijaId=2,NazivKategorije="Stampaci", OpisKategorije="Stampaci razlicitih proizvodjaca" },
        new Kategorija{KategorijaId=3,NazivKategorije="Tableti", OpisKategorije="Tablet racunari razlicitih proizvodjaca }"

    };
    return listaKategorija;
}
```

Metoda VratiProizvode() statička klasa Podaci

```
public static List<Proizvod> VratiProizvode()
{
    List<Proizvod> listaProizvoda = new List<Proizvod>() {
        new Proizvod{ProizvodId=1, KategorijaId=1,NazivProizvoda="Laptop Dell Inspiron N4050",Cena=30999.25M,KolicinaNaLageru=39 },
        new Proizvod{ProizvodId=2, KategorijaId=1,NazivProizvoda="Laptop Asus X55U-SX009D",Cena=32990.12M,KolicinaNaLageru=17 },
        new Proizvod{ProizvodId=3, KategorijaId=1,NazivProizvoda="Acer Aspire laptop E5-521G-47DX",Cena=41989,KolicinaNaLageru=25 },

        new Proizvod{ProizvodId=4, KategorijaId=2,NazivProizvoda="Stampač Laser A4 Lexmark E260",Cena=8549.1M,KolicinaNaLageru=13 },
        new Proizvod{ProizvodId=5, KategorijaId=2,NazivProizvoda="Canon laserski stampac LBP-6670DN",Cena=31989.1M,KolicinaNaLageru=18 },
        new Proizvod{ProizvodId=6, KategorijaId=2,NazivProizvoda="Canon stampač imageCLASS LBP6030W",Cena=13589.12M,KolicinaNaLageru=11 },

        new Proizvod{ProizvodId=7, KategorijaId=3,NazivProizvoda="Acer tablet B1-730HD 8GB",Cena=11999.3M,KolicinaNaLageru=12 },
        new Proizvod{ProizvodId=8, KategorijaId=3,NazivProizvoda="Asus tablet MeMO Pad 7 ME70C-1A003A",Cena=12999.9M,KolicinaNaLageru=14 },
        new Proizvod{ProizvodId=9, KategorijaId=3,NazivProizvoda="Goclever tablet ORION 70 L KB",Cena=5699.45M,KolicinaNaLageru=25 }
    };

    return listaProizvoda;
}
```

Inicijalizacija izvora podataka

```
private List<Kategorija> listaKategorija = Podaci.VratiKategorije();
private List<Proizvod> listaProizvoda = Podaci.VratiProizvode();
```

Proizvodi kojih na lageru ima više od 20

```
private void button5_Click(object sender, EventArgs e)
{
    IEnumerable<Proizvod> upit = listaProizvoda
        .Where(p => p.KolicinaNaLageru > 20);

    StringBuilder sb = new StringBuilder();

    foreach (Proizvod p in upit)
    {
        sb.AppendLine("Naziv: " + p.NazivProizvoda);
        sb.AppendLine("Cena: " + p.Cena);
        sb.AppendLine("Količina: " + p.KolicinaNaLageru);
        sb.AppendLine(new string('-', 40));
    }

    richTextBox1.Text = sb.ToString();
}
```

Ekstenziona metoda SingleOrDefault() klase Enumerable

```
public static TSource SingleOrDefault<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, bool> predicate
)
```

- Ekstenziona metoda SingleOrDefault proširuje tip IEnumerable<TSource>
- Vraća jedini element sekvence koji zadovoljava dati uslov
- Ako ne postoji nijedan takav element, vraća podrazumevanu vrednost tipa TSource
  - za referentne tipove: null
  - za vrednosne tipove: 0, false, DateTime.MinValue, itd
- Ako postoji više od jednog elementa koji zadovoljava uslov, metoda generiše izuzetak

Metoda SingleOrDefault() upotreba

```
private void button6_Click(object sender, EventArgs e)
{
    Proizvod p1 = listaProizvoda.SingleOrDefault(p => p.ProizvodId == 1);

    if (p1 != null)
    {
        label1.Text = p1.NazivProizvoda;
    }
    else
    {
        MessageBox.Show("Ne postoji proizvod");
    }

}
```

Metoda FirstOrDefault()

```
public static TSource FirstOrDefault<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, bool> predicate
)
```

- Ekstenziona metoda FirstOrDefault proširuje tip IEnumerable<TSource>
- Vraća prvi element sekvence koji zadovoljava dati uslov
- Ako ne postoji nijedan element koji zadovoljava uslov, vraća podrazumevanu vrednost tipa TSource
- Ako postoji više elemenata koji zadovoljavaju uslov, metoda vraća samo prvi pronađeni element
- Metoda ne generiše izuzetak ako postoji više odgovarajućih elemenata

Metoda FirstOrDefault upotreba

```
private void button7_Click(object sender, EventArgs e)
{
    Proizvod p1 = listaProizvoda.FirstOrDefault(p => p.KategorijaId == 1);
    if (p1 != null)
    {
        label1.Text = p1.NazivProizvoda;
    }
    else
    {
        MessageBox.Show("Ne postoji proizvod");
    }

}
```

Metoda Find() klase List<T>

```
public T Find(Predicate<T> match)
```

- Metoda Find je deo klase List<T>
- Vraća prvi element liste koji zadovoljava uslov definisan parametrima
- Parametar **match** je tipa Predicate<T> – anonimna funkcija koja vraća true ili false
- Ako ne postoji element koji zadovoljava uslov, vraća se podrazumevana vrednost tipa T
- Ne generiše izuzetak zbog više pronađenih elemenata – uvek vraća prvi koji odgovara
- Metoda Find() radi samo sa listama dok FirstOrDefault() radi sa bilo kojim tipom koji implementira IEnumerable<T>

Metoda Find() klase List<T>

```
private void button8_Click(object sender, EventArgs e)
{
    Proizvod p1 = listaProizvoda.Find(p => p.KategorijaId == 1);

    if (p1 != null)
    {
        label1.Text = p1.NazivProizvoda;
    }
    else
    {
        MessageBox.Show("Proizvod nije pronađen");
    }

}
```

Metoda OrderBy

```
public static IOrderedEnumerable<TSource> OrderBy<TSource, TKey>(
    this IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector
)
```

Sortira elemente sekvence u rastućem poretku prema ključu

keySelektor - anonimna funkcija koja izdvaja ključ iz svakog elementa sekvence

Sortiranje rezultata

```
private void button9_Click(object sender, RoutedEventArgs e)
{
    IEnumerable<Proizvod> sortiraniProizvodi = listaProizvoda
    .Where(p => p.KategorijaId == 1)
    .OrderBy(p => p.Cena);

    StringBuilder sb = new StringBuilder();
    foreach (Proizvod p in sortiraniProizvodi)
    {
        sb.AppendLine(p.NazivProizvoda + " " + p.Cena);
    }
    richTextBox1.Text = sb.ToString();
}
```

Sortiranje u opadajućem poretku

```
private void button10_Click(object sender, RoutedEventArgs e)
{
    IEnumerable<Proizvod> sortiraniProizvodi = listaProizvoda
    .Where(p => p.KategorijaId == 1)
    .OrderByDescending(p => p.Cena);

    StringBuilder sb = new StringBuilder();
    foreach (Proizvod p in sortiraniProizvodi)
    {
        sb.AppendLine(p.NazivProizvoda + " " + p.Cena);
    }
    richTextBox1.Text = sb.ToString();
}
```

Višestruko sortiranje

```
private void button11_Click(object sender, EventArgs e)
{
    IEnumerable<Proizvod> upit = listaProizvoda
    .OrderBy(p => p.KategorijaId)
    .ThenByDescending(p => p.Cena);

    StringBuilder sb = new StringBuilder();
    foreach (Proizvod p in upit)
    {
        sb.AppendLine(p.KategorijaId + " " +p.NazivProizvoda + " " + p.Cena);
    }
    richTextBox1.Text = sb.ToString();
}
```

Metoda Distinct()

- Uklanja duplikate
- Zadržava prvu pojavu svakog elementaKoristi podrazumevani komparator jednakosti

```
private void button12_Click(object sender, EventArgs e)
{
    List<int> brojevi = new List<int> { 21, 46, 46, 55, 17, 21, 55, 55 };
    IEnumerable<int> razliciti Brojevi = brojevi.Distinct();

    StringBuilder sb = new StringBuilder();
    foreach (int b in razliciti Brojevi)
    {
        sb.AppendLine(b.ToString());
    }
    richTextBox1.Text = sb.ToString();
}
```

Metoda Count()

```
public static int Count<TSource>(
    this IEnumerable<TSource> source
)
```

```
 public static int Count<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, bool> predicate
)
```

- Vraća **broj elemenata** u sekvenci
- U varijanti sa **predicate**-om vraća broj elemenata koji **zadovoljavaju uslov**
- Radi nad bilo kojim tipom koji implementira **IEnumerable<T>**
- Ne menja podatke u izvoru

Metoda Count()

```
private void button13_Click(object sender, EventArgs e)
{
    List<int> brojevi = new List<int>{ 5, 4, 1, 3, 9, 8, 6, 7, 1, 0 };
    int ukupno = brojevi.Count();
    int neparnih = brojevi.Count(b => b % 2 == 1);
    int parnih = brojevi.Count(b => b % 2 == 0);

    StringBuilder sb = new StringBuilder();
    sb.AppendLine("Ukupno: " + ukupno.ToString());
    sb.AppendLine("Neparnih: " + neparnih.ToString());
    sb.AppendLine("Parnih: " + parnih.ToString());
    richTextBox1.Text = sb.ToString();
}
```

Metode Min() i Max()

- Vraća najveću vrednost u sekvenci
- Ako se prosledi selector, metoda vraća najveću projekciju vrednosti
- Radi nad bilo kojim tipom koji implementira IEnumerable<T>, pod uslovom da tip podržava poređenje
- Generiše izuzetak ako je sekvenca prazna
- Za proste tipove (int, double, decimal, DateTime) koristi se prirodno poređenje
- Za složenije tipove potrebno je proslediti selector

Metoda Min()

```
public static int Min(
    this IEnumerable<int> source
)
```

```
 public static decimal Min<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, decimal> selector
)
```

selector - transformaciona funkcija koja svaku vrednost sekvence pretvara u decimal

Metoda Max()

```
public static int Max(
    this IEnumerable<int> source
)
```

```
public static decimal Max<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, decimal> selector
)
```

Primer Min()/Max()

```
private void button14_Click(object sender, EventArgs e)
{
    int[] brojevi = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    StringBuilder sb = new StringBuilder();

    int minBroj = brojevi.Min();
    sb.AppendLine(minBroj.ToString());

    decimal najJeftiniji = listaProizvoda.Min(p => p.Cena);
    sb.AppendLine(najJeftiniji.ToString());

    richTextBox1.Text = sb.ToString();
}
```

Metoda Average()

```
public static double Average(
    this IEnumerable<int> source
)
```

long, double, float, decimal

```
public static double Average<TSource>(
this IEnumerable<TSource> source,
Func<TSource, int> selector)
```

- Računa **srednju vrednost** elemenata u sekvenci
- Ako se koristi selektor, prvo primenjuje selektor pa računa srednju vrednost
- Rezultat je tipa **double** (osim za decimal → decimal)

Metoda Average()

```
private void button15_Click(object sender, EventArgs e)
{
    List<int> brojevi = new List<int> { 78, 92, 100, 37, 81 };

    StringBuilder sb = new StringBuilder();

    double prosek = brojevi.Average();
    sb.AppendLine(prosek.ToString());

    decimal prosecnaCena = listaProizvoda.Average(p => p.Cena);
    sb.AppendLine(prosecnaCena.ToString());

    richTextBox1.Text = sb.ToString();
}
```

Metoda Sum()

Sum() računa zbir elemenata ili zbir vrednosti dobijenih selektorom

```
private void button16_Click(object sender, RoutedEventArgs e)
{
    StringBuilder sb = new StringBuilder();

    List<int> brojevi = new List<int> { 78, 92, 100, 37, 81 };
    int suma = brojevi.Sum();
    sb.AppendLine(suma.ToString());

    int ukupnoLaptopova = listaProizvoda
                .Where(p => p.KategorijaId == 1)
                .Sum(p => p.KolicinaNaLageru);

    sb.AppendLine(ukupnoLaptopova.ToString());

    richTextBox1.Text = sb.ToString();
}
```

Metoda Take()

```
public static IEnumerable<TSource> Take<TSource>(
     this IEnumerable<TSource> source,
     int count
 )
```

- Vraća specificirani broj uzastopnih elemenata od početka sekvence
- Vraća prvih count elemenata iz sekvence
- Elementi se uzimaju redosledom pojavljivanja u izvornoj sekvenci
- Metoda ne menja izvor podataka – vraća novu sekvencu
- Ako sekvenca sadrži manje elemenata od traženog broja, vraća se cela sekvenca

Metoda Take() - primer

```
private void button17_Click(object sender, EventArgs e)
{
    int[] poeni = { 59, 82, 70, 56, 92, 98, 85 };

    IEnumerable<int> najbolja3 =
        poeni.OrderByDescending(p => p).Take(3);

    StringBuilder sb = new StringBuilder();
    foreach (int p in najbolja3)
    {
        sb.AppendLine(p.ToString());
    }
    richTextBox1.Text = sb.ToString();

}
```

Metoda Skip()

```
private void button18_Click(object sender, EventArgs e)
{
    int[] poeni = { 59, 82, 70, 56, 92, 98, 85 };

    IEnumerable<int> preskociPrvaTri =
        poeni.OrderByDescending(p => p).Skip(3);

    StringBuilder sb = new StringBuilder();
    foreach (int p in preskociPrvaTri)
    {
        sb.AppendLine(p.ToString());
    }
    richTextBox1.Text = sb.ToString();
}
```

- Preskače prvih count elemenata u sekvenci
- Vraća ostatak sekvence nakon preskočenih elemenata
- Elementi se preskaču redosledom pojavljivanja u sekvenci
- Metoda ne menja izvor podataka – vraća novu sekvencu

Pitanje 1

Za pisanje LINQ upita korišćenjem lambda izraza koriste se:

a. Ekstenzione metode klase Enumerable
b. Metode interfejsa IEnumerable
c. Metode klase Linq

Odogovor: a

Pitanje 2

Ako je sa TSource označen tip podataka u izvoru podataka, ekstenzione metode statičke klase Enumerable proširuju:

a. Izvor podataka koji implementira **IEnumerable<TSource>** interfejs
b. Izvor podataka koji implementira **ISqlSource<TSource>** interfejs
c. Izvor podataka koji implementira **IComparable<TSource>** interfejs

Odogovor: a

Pitanje 3

Koja LINQ ekstenziona metoda prima lambda izraz kojim se definiše uslov filtriranja elemenata?

a. Select
b. Where
c. Take

Odogovor: b

Pitanje 4

Koja LINQ ekstenziona metoda prima lambda izraz kao transformacionu funkciju i menja vrednost svakog elementa sekvence?

a. Select
b. Where
c. Take

Odogovor: a

Pitanje 5

Koja metoda ne generiše izuzetak ako više elemenata zadovoljava uslov?

a. SingleOrDefault
b. FirstOrDefault
c. Single

Odogovor: b

Pitanje 6

Koja metoda nije LINQ ekstenziona metoda?

a. SingleOrDefault
b. FirstOrDefault
c. Find

Odogovor: c

Pitanje 7

Lambda izraz prosleđen metodi Where predstavlja?

a. Transformacionu funkciju
b. Funkciju koja vraća true ili false (predikat)
c. Agregacionu funkciju

Odogovor: b