LINQ

Lambda izrazi, anonimni tipovi,

Language INtegrated Query

Šta je LINQ?

- *Language Integrated Query (LINQ)*
- Novi pristup koji unificira način pristupa i pretrage podataka. Primenljiv je za sve objekte čije klase implementiraju ***IEnumerable<T>*** *interface*.
  - ***Nizovi***,
  - ***kolekcije***,
  - ***relacioni podaci*** i
  - ***XML***
- su potencijalni izvori podataka za LINQ.

- Primer:

1. Kreirati Win C# aplikaciju. Dodati klasu *Person*, sa poljima

   public int ID;

   public int IDRole;

   public string LastName;

   public string FirstName;

2. Napraviti listu objekata tipa *Person*.
   - List<T> je generička klasa koja implementira *IEnumerable<T>*, tako da je pogodna za LINQ upite.

```
/*obratite paznju na inicijalizaciju */
List<Person> people = new List<Person>
{
    new Person() { ID = 1,
    IDRole = 1,
    LastName = "Ilic",
    FirstName = "Aca"
},
........  ....... ....... ....... .......
........  ....... ....... ....... .......
```

Kako LINQ radi?

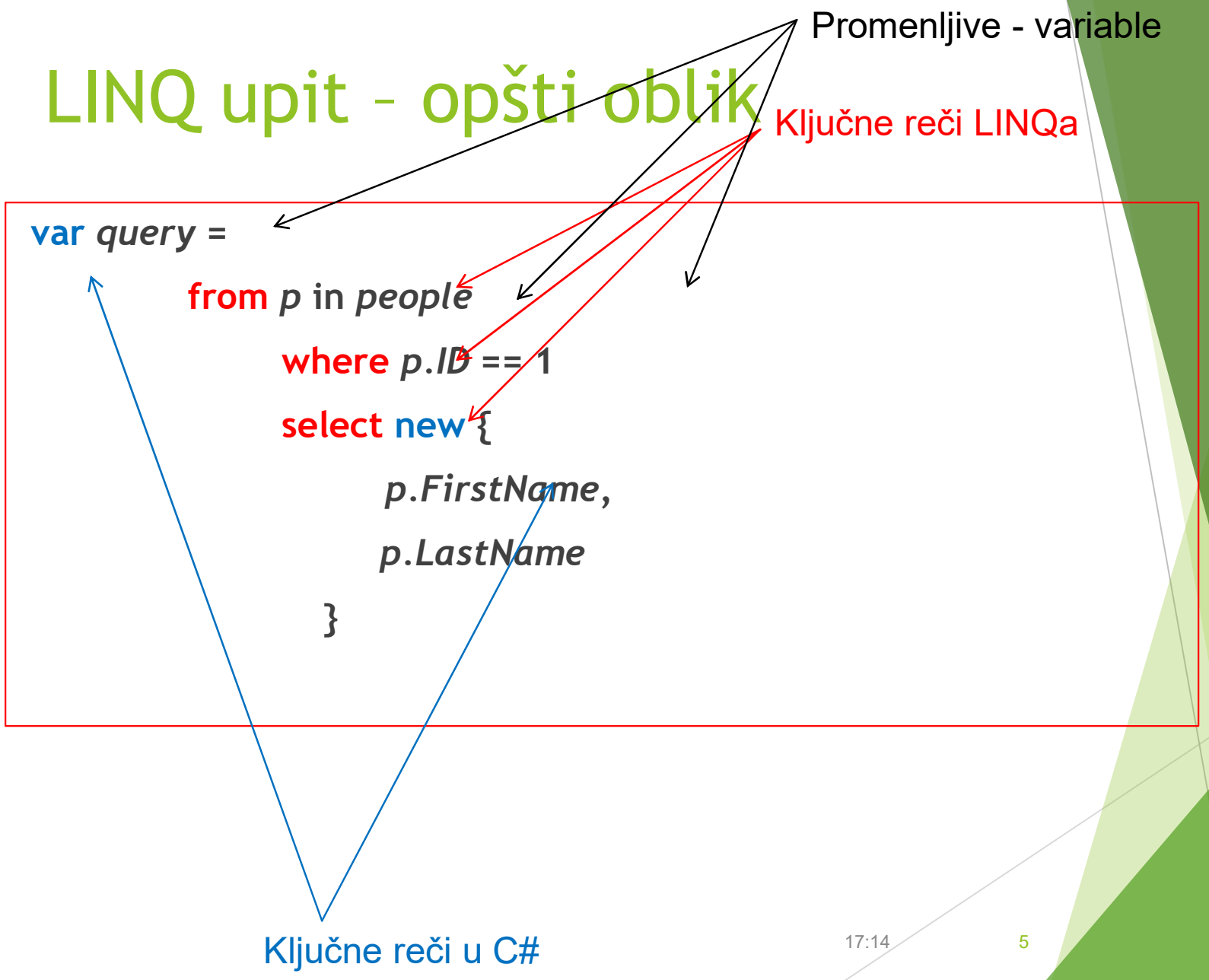
- Kada kompajler pronađe *query* upit u kodu, radi transformaciju u C# pozive metoda.

```
var q1 = from p in people
         where p.ID == 1
         select p;

var q2 = from p in people
         where p.IDRole==1
         select p;

var q3 = from p in people
         where (p.IDRole == 1 && p.ID > 0)
               || p.LastName.StartsWith("P")
         select p;
```

LINQ upit – opšti oblik



Prikaz rezultata

```
for (int i = 0; i < q2.Count(); i++)
{
    Person p = q2.ElementAt<Person>(i);
}
foreach (Person p in q2)
{
    Console.WriteLine(p.LastName + " " + p.FirstName);
}

//OPŠTI OBLIK -
IEnumerable eElement = (IEnumerable) q1;
if (eElement != null)
{
    foreach (object item in eElement)
    {
        Console.WriteLine(item);
    }
}
```

Lambda izrazi *(Lambda Expressions – LE)*

Anonimne metode (ili klase) su metode (ili klase) koje se koriste u izrazima a nemaju eksplicitno ime i ne mogu se koristiti samostalno.

LE je anonimna metoda (zato mora postojati delegat)

```
delegate int del(int i);

static void Main(string[] args)
{
    del myDelegate = x => x * x;

    int j = myDelegate(5);  //j = 25
}
```

Lambda izrazi (2)

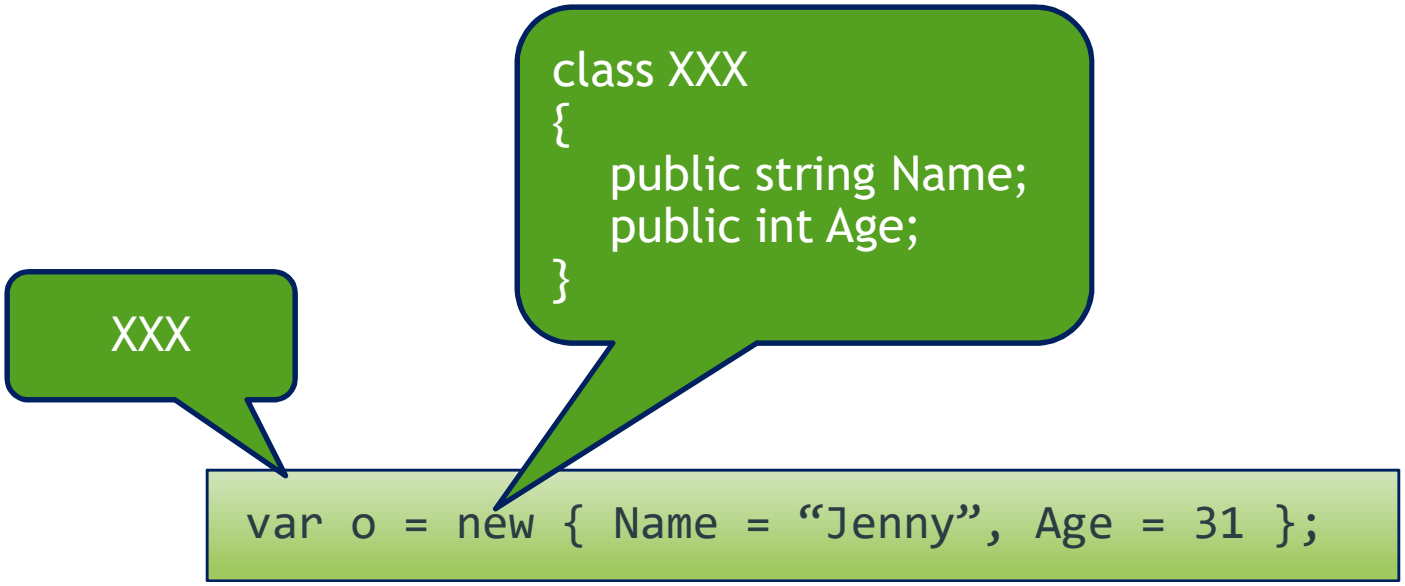
Izrazi:

```
(input parameters) => expression
        (x, y) => x == y
        (int x, string s) => s.Length > x
```

Naredbe:

```
(input parameters) => {statement;}
        delegate void TestDelegate(string s);
        TestDelegate myDel = n =>
        {
            string s = n + " " + "World";
       Console.WriteLine(s);
      };
        myDel("Hello");
```

Anonimni tipovi (1)



Anonimni tipovi (2)

- Predstavljaju sastavni deo Linq-a:

  ```
  select new { p.FirstName, p.LastName };
  ```

  nije definisan tip iza operatora new!? Kompajler kreira lokalni tip za nas.

- **Anonimni tipovi dozvoljavaju da se radi sa rezultatima upita bez eksplicitne definicije klase koja ih predstavlja.**

Standardni operatori za upite

Operatori za filtriranje

1. ***Where***
2. ***OfType***

Primer:

```
ArrayList list = new ArrayList();
list.Add("Dash");
list.Add(new object());
list.Add("Skitty");
list.Add(new object());
var query = from name in list.OfType<string>()
    where name == "Dash" select name;
```

- *OfType* operator se može koristiti u slučaju ne-generičkih kolekcija (kao na primer *ArrayList*) u LINQ upitima.
- Pošto *ArrayList* ne implementira *IEnumerable<T>*, *OfType* operator je jedini LINQ operator koji možemo primeniti na listu. *OfType* je takođe koristan ako radite više nasleđenih klasa i ako želite da selektujete samo objekte određenog tipa.

Operatori sortiranja

- **OrderBy, OrderByDescending, ThenBy, ThenByDescending, *Reverse*.**

- **Primer:**
- var query = from name in names **orderby** name, name.Length select name;

- Povratna vrednost *OrderBy* operatora je *IOrderedEnumerable<T>*.

- Ovaj specijalni interfejs je nasleđen od *IEnumreable<T>* i dozvoljava

ThenBy , ***ThenByDescending*** operatore.

Skupovni operatori

1. **`Distinct`** (za izbacivanje dupliranih vrednosti),
2. **`Except`** (vraća razliku dve sekvence),
3. **`Intersect`** (vraća presek dve sekvence),
4. **`Union`** (vraća uniju elemenata dve sekvence).

- **Primer**

```
int[] twos = { 2, 4, 6, 8, 10 };
int[] threes = { 3, 6, 9, 12, 15 };

//6
var intersection = twos.Intersect(threes);
// 2, 4, 8, 10
var except = twos.Except(threes);
// 2, 4, 6, 8, 10, 3, 9, 12, 15
var union = twos.Union(threes);
```

Operatori provere po broju/količini

1. **All,**
2. **Any,**
3. **Contains**

- **Primeri**
- `int[] twos = { 2, 4, 6, 8 };`
- `// true`
- `bool sviParni = twos.All(i => i % 2 == 0);`
- `// true`
- `bool barNekiDeljivSaTri = twos.Any(i => i % 3 == 0);`
- `// false`
- `bool sadrziSedam = twos.Contains(7);`

Implementacija pravila za validaciju

```
class Program{
    static void Main(string[] args) {
        Student student1 =  new Student { ID = 1, Index = "nrt-1/11" };
        Student student2 = new Student { ID = 2, Index = "" };

        Func<Student, bool>[] ispravanPodadatak = {
            e => e. ID > 0,
            e => !String.IsNullOrEmpty(e.Index)
        };

        bool isValidStudent1 = ispravanPodadatak.All(rule => rule(student1));
        bool isValidStudent21 = ispravanPodadatak.All(rule => rule(student2));
        bool isValidStudent22 = ispravanPodadatak.Any(rule => rule(student2));
    }
}
class Student{
    public int ID;
    public string Index;
}
```

Operatori koji vraćaju rezultat

- **`Select`**
  - vraća 1 izlaz za 1 ulaz. Može da da i novi tip podataka.
- **`SelectMany`**
  - kada radimo sa sekvencom od sekvenci.
  - *SelectMany* se koristi kada postoji višestruki ***from***.

Operatori izdvajanja

1. **Skip**
2. **Take**

- Da bi dobili treću stranicu rezultata, pri čemu ide 10 zapisa po stranici, treba da uradite Skip(20) a zatim Take(10).

- Postoje I operatori sa uslovom: **SkipWhile**, **TakeWhile**.
- `int[] numbers = { 1, 3, 5, 7, 9 };`
- `var query = numbers.SkipWhile(n => n < 5) .TakeWhile(n => n < 10); // daje 5, 7, 9`

Join operatori

- **`Join`** - sličan **SQL INNER JOIN.**
- **`GroupJoin`** - sličan **LEFT OUTER JOIN**

- **Primer:**

```
var Students = new List<Student> {
new Student { ID=1, Name="Ana", SmerID=1 },
new Student { ID=2, Name="Jova", SmerID=1 },
new Student { ID=2, Name="Jova2"}, //za zadatak
new Student { ID=3, Name="Mika", SmerID=2} };
var Smers = new List<Smer> {
new Smer { ID=1, Name="NRT" },
new Smer { ID=2, Name="RT" },
new Smer { ID=3, Name="IS" } };
```

```
var query =
    from Student in Students
    join Smer in Smers
    on Student.SmerID equals Smer.ID
select  new {
   StudentName = Student.Name,
   SmerName = Smer.Name
};
```

Operatori grupisanja

- **GroupBy ,ToLookup**
- Vraćaju sekvencu ***IGrouping<K,V>***. Ovaj interfejs specificira da objekat izlaže neko ***Key*** svojstvo koje omogućava grupisanje.
- Primer:

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var query = numbers.ToLookup(i => i % 2);
foreach (IGrouping<int, int> group in query)
{
        Console.WriteLine("Key: {0}", group.Key);
        foreach (int number in group) { Console.WriteLine(number); }
}
```

- Vraćaju se dve grupe sa **Key** vrednostima 0,1. U svakoj grupi je lista objekata originalnog niza koji joj pripadaju.
- *Osnovna razlika između GroupBy i ToLookup-a je da je GroupBy lazy i nudi odloženo izvršenje. ToLookup je greedy i odmah će se izvršiti.*

Primer grupisanja

```
var qg = people.GroupBy(x =>
x.IDRole).Select(x=> new { kljuc = x.Key, broj
= x.Count() });
var qg2 = from p in people
          group p by p.IDRole into g
          select new
          {
              kljuc = g.Key,
              broj = g.Count()
          };
```

Operatori generisanja

- ***Empty*** kreira praznu sekvencu *IEnumerable<T>*.
- `var empty = Enumerable.Empty<Student>();`
- ***Range*** operator generiše sekvencu brojeva
- ***Repeat*** generiše sekvencu bilo koje vrednosti.
- `var empty = Enumerable.Empty<Student>();`
- `int start = 1;`
- `int count = 10;`
- `IEnumerable<int> numbers = Enumerable.Range(start, count);`
- `var tenTerminators = Enumerable.Repeat(new Student { Name ="Perica" }, 10);`
- ***DefaultIfEmpty*** generiše praznu kolekciju sa podrazumevanom vrednošću koja pripada tipu kada se primeni.
- `string[] names = { }; //empty array`
- `IEnumerable<string> oneNullString = names.DefaultIfEmpty();`

Operator jednakosti

- ***SequenceEquals***
- Prolazi kroz dve sekvence i poredi objekte unutar obe da li su jednaki.

- **Primer:**

```
Student e1 = new Student() { ID = 1 };
Student e2 = new Student() { ID = 2 };
Student e3 = new Student() { ID = 3 };
var Students1 = new List<Student>() { e1, e2, e3 };
var Students2 = new List<Student>() { e3, e2, e1 };
//false
bool result = Students1.SequenceEqual(Students2);
```

Element Operators

- **ElementAt, First, Last, Single**
- Za svaki operator postoji odgovarajući "**ordefault**" operator koji se može koristiti da se izbegne izuzetak kada element ne postoji (**ElementAtOrDefault, FirstOrDefault, LastOrDefault, SingleOrDefault**).
- `string[] empty = { };`
- `string[] notEmpty = { "Hello", "World" };`
- `var result = empty.FirstOrDefault(); // null`
- `result = notEmpty.Last(); // World`
- `result = notEmpty.ElementAt(1); // World`
- `result = empty.First(); // InvalidOperationException`
- `result = notEmpty.Single(); // InvalidOperationException`
- `result = notEmpty.First(s => s.StartsWith("W"));`
- Osnovna razlika između *First* i *Single* je što *Single* operator baca izuzetak ako sekvenca ne sadrži jedan element, dok *First* vraća rezultata samo ako je prvi element.

Conversions

- ***OfType ,Cast***
- *OfType* operator je i operator filtriranja – vraća samo objekte kojem može kastovati, dok Cast će pucati ako ne može da kastuje sve objekte u neki tip.
- `object[] data = { "Foo", 1, "Bar" };`
- `// vraća sekvencu od 2 stringa`
- `var query1 = data.OfType<string>();`
- `// vraća izuzetak pri izvršavanju`
- `var query2 = data.Cast<string>();`

Concatenation

- **Concat** operator spaja dve sekvence.
- Sličan je *Union* operatoru ali ne izbacuje duplikate.

```
string[] firstNames = { "Scott", "James", "Allen", "Greg" }; string[]
lastNames = { "James", "Allen", "Scott", "Smith" };

//“Allen”, “Allen”, “Greg”, “James”, “James”, “Scott”, “Scott”, “Smith”
var concatNames = firstNames.Concat(lastNames).OrderBy(s => s);

//“Allen”, “Greg”, “James”, “Scott”, “Smith”
var unionNames = firstNames.Union(lastNames).OrderBy(s => s);
```

Aggregation Operators

- ***Average***, ***Count***, ***LongCount*** **(for big results)**, ***Max***, ***Min***, ***Sum***.

- Statistika pokrenutih procesa:

```
Process[] runningProcesses = Process.GetProcesses();
var summary = new { ProcessCount =
runningProcesses.Count(), WorkerProcessCount =
runningProcesses.Count(p => p.ProcessName == “zcirovic"),
TotalThreads = runningProcesses.Sum(p => p.Threads.Count),
MinThreads = runningProcesses.Min(p => p.Threads.Count),
MaxThreads = runningProcesses.Max(p => p.Threads.Count),
AvgThreads = runningProcesses.Average(p => p.Threads.Count)
};
```

Zadatak

- Na slajdu 22 je uvesti podatke o još studenata koji pripadaju nekom smeru kao i o studentima koji ne pripadaju ni jednom od smerova.
- Uvesti odgovarajuće klase za Student i Smer.
- Potrebno je napraviti Linq upit za grupisanje studenata po smeru, a prikazati brojno stanje kao i nazive studenata po grupama.