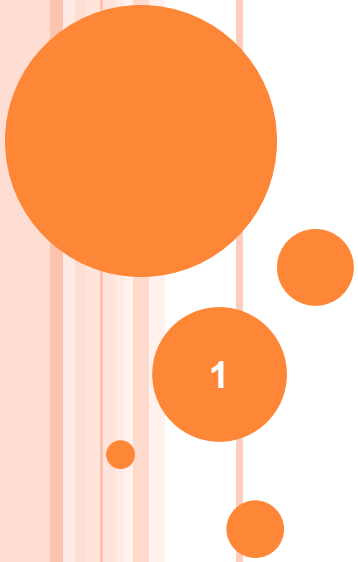


LINQ

1

Lambda izrazi, anonimni tipovi,
Eng. Language INtegrated Query

ŠTA JE LINQ?

- *Eng. 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.

ZAŠTO LINQ?

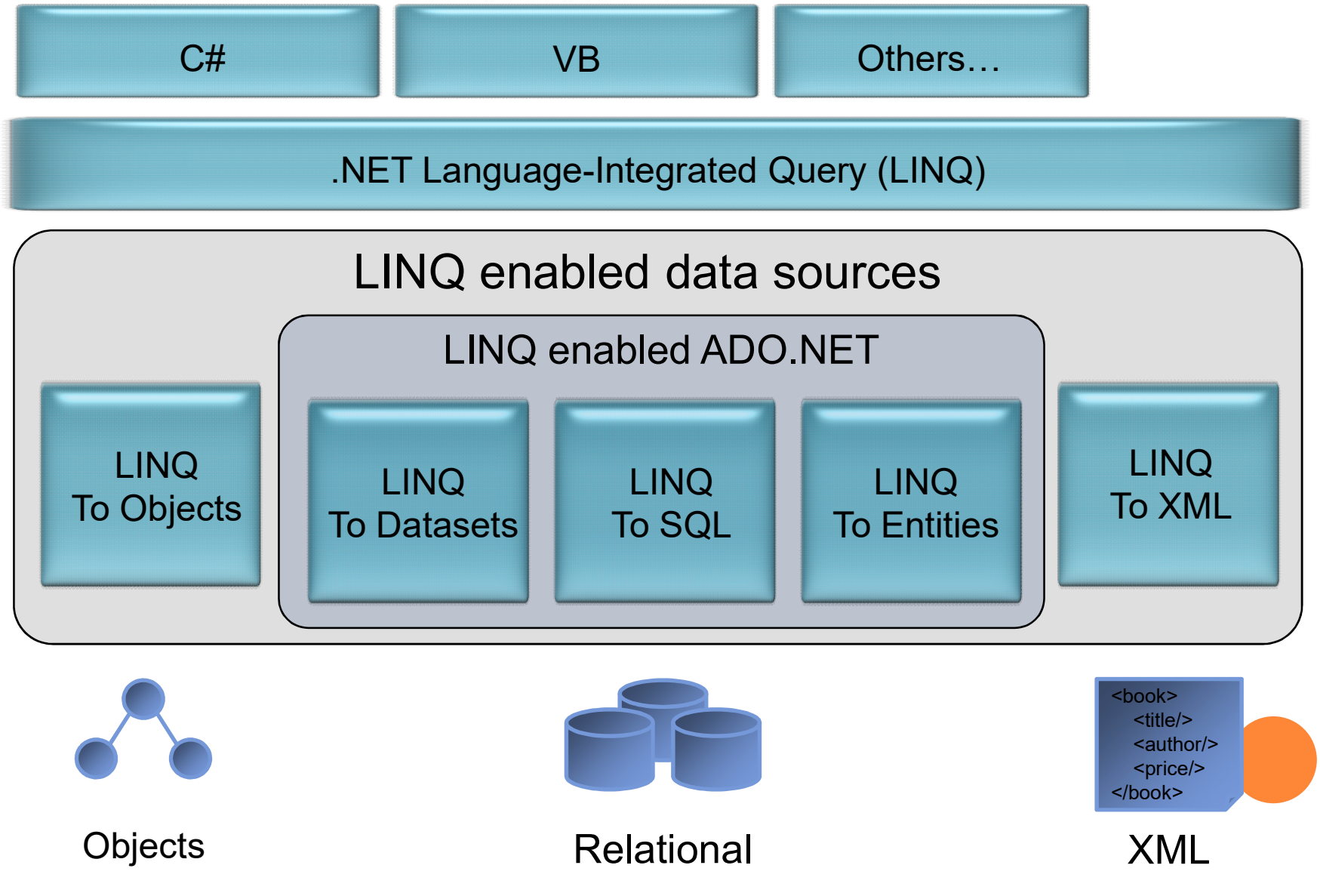
- Koristeći istu sintaksu pristupa se podacima bilo kog izvora:
- ***var query = from e in employees***

  where e.id == 1

  select e.name
- Ovo **nije pseudokod**; ovo je LINQ sintaksa i slična je SQL sintaksi.
- LINQ poseduje bogatu kolekciju naredbi za implementaciju kompleksnih upita koje sadrže agregantne funkcije, združivanje, sortiranje i puno toga još.

19:38

ARHITEKTURA



LINQ FUNKCIONALNOST

- ***LINQ to Objects*** je aplikacioni interfejs koji prikazuje standardne operatore upita (SQOs) za dobijanje podataka iz nekog objekta čija klasa implementira interfejs **IEnumerable<T> interface.** Ovi upiti se izvršavaju na podcima u memoriji.

- ***LINQ to ADO.NET*** proširuje SQOs na rad sa relacionim podacima. Sastoji se iz 3 dela koja su prikazana u donjem delu slike:
  - **LINQ to SQL** se koristi za upite ka relacionim bazama kao MSSQL.
  - **LINQ to DataSet** se koristi za upite koji koriste ADO.NET skupove podataka.
  - **LINQ to Entities** je Microsoft ORM rešenje za upotrebu Entities.
  - ***LINQ to XML***

LINQ TO OBJECTS

- 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 = "Anderson",
    FirstName = "Brad“
},
.........  ....... ....... ....... .......
.........  ....... ....... ....... .......
```

KAKO LINQ RADI?

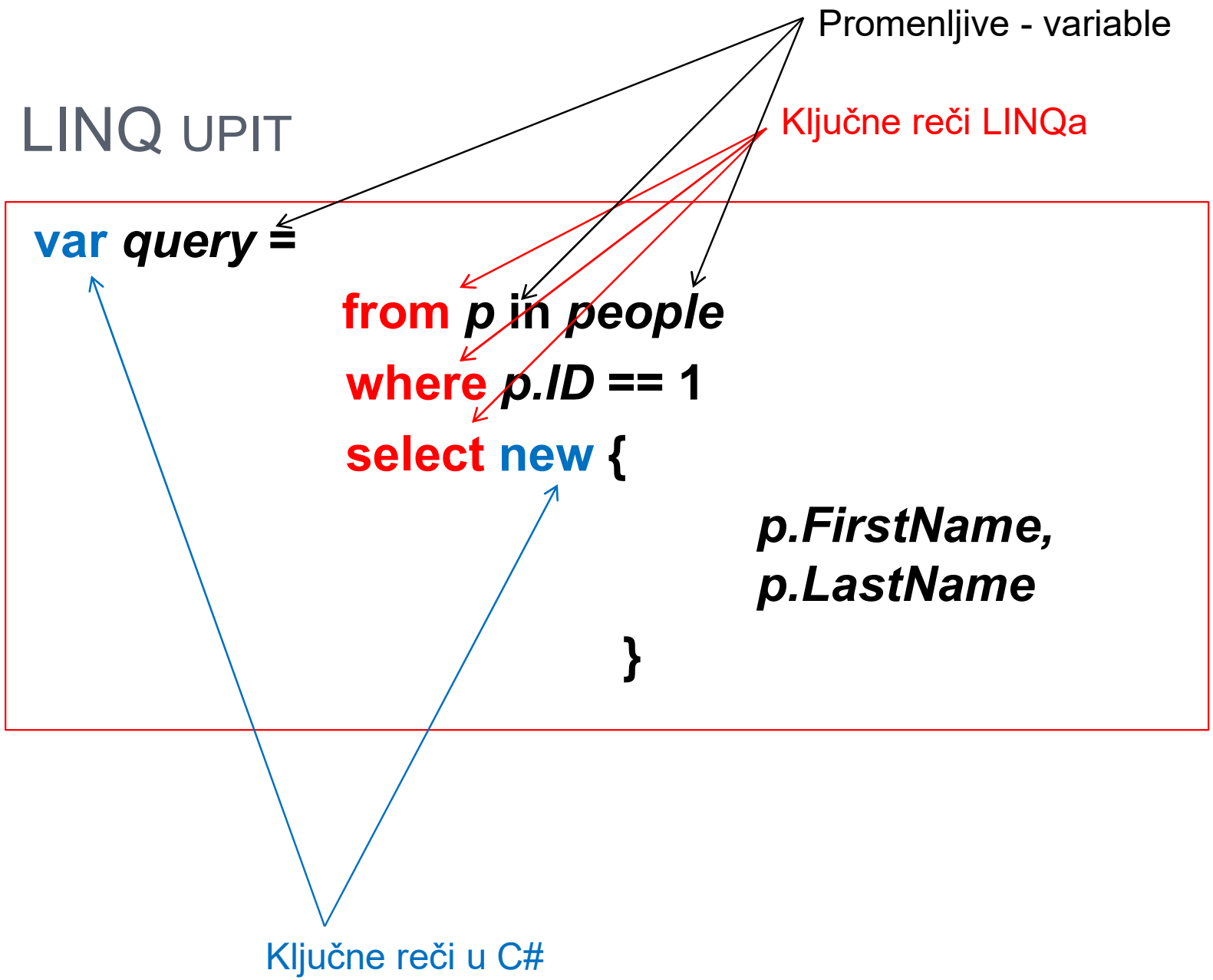
- Kada kompajler pronađe *query* upit u kodu, radi transformaciju u C# pozive metoda.

```
var query = people
.Where(p => p.ID == 1)
.Select(p => new { p.FirstName, p.LastName } );

from se izbacuje, ostaje samo kolekcija, osobe
var query = from p in people
                    where p.ID == 1
                    select new {p.FirstName, p.LastName}
```

19:38

LINQ UPIT


Promenljive - variable
Ključne reči LINQa
var query =
from p in people
where p.ID == 1
select new {
p.FirstName,
p.LastName
}
Ključne reči u C#

PRIKAZ REZULTATA

```
IEnumerable eElement = (IEnumerable) query;
if (eElement != null)
{
    foreach (object item in eElement)
    {
        listBox1.Items.Add(item);
    }
}
```

19:38

EXTENSION METHODS (1)

- Ključne reči ***where*** i ***select*** se transformišu u dve metode: ***Where<T>()*** i ***Select<T>()***.

  Metode se nadovezuju tako da se rezultati *Where* metode izdvajaju pomoću *Select* metode.
- Ovo se postiže proširenim metodama - *extension methods.*

Extension Methods (2)

- Proširuju postojeće .NET klase (types) sa novim metodama.
- Na primer, primenom proširenja metoda, postojećoj klasi *string* možemo dodati novu metodu za zamenu beline sa podcrtom.
- Kako?

PRIMER: EKSTENZIJA KLASE "*STRING*"

```
public static string SpaceToUnderscore(this string
    source)
{
    char[] cArray = source.ToCharArray();
    string result = null;
    foreach (char c in cArray)
    {
        if (Char.IsWhiteSpace(c))
        result += "_";
        else
        result += c;
    }
    return result;
}
```

Primer 2 Metoda za pov

Extension method

```
namespace MyStuff
{
    public static class Extensions
    {
        public static string Concatenate(this IEnumerable<string> strings,
                                          string separator) {…}
    }
}
```

19:38

Uključivanje imenskog prostora

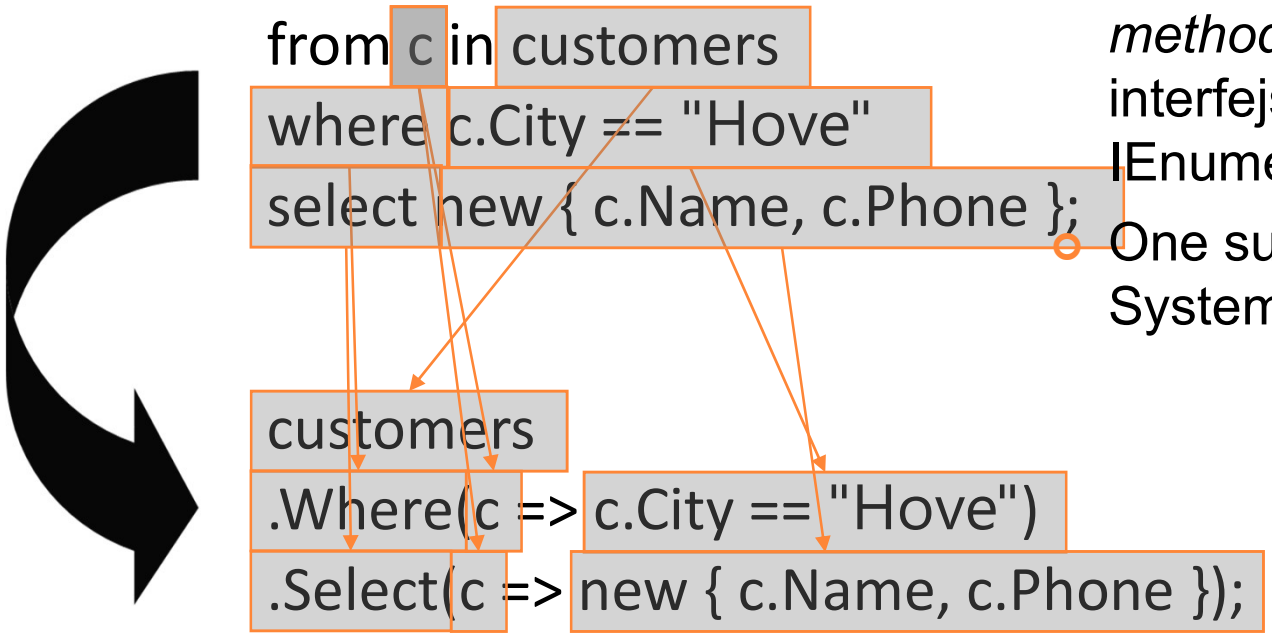
```
using MyStuff;
```

```
string[] names = new string[] { "Jenny", "Daniel",
"Rita" };
string s = names.Concatenate(", ");
```

IntelliSense!

EXTENZIJA U QUERY IZRAZIMA

- Upit se transformiše u pozive metoda
  - Where, Join, OrderBy, Select, GroupBy, …

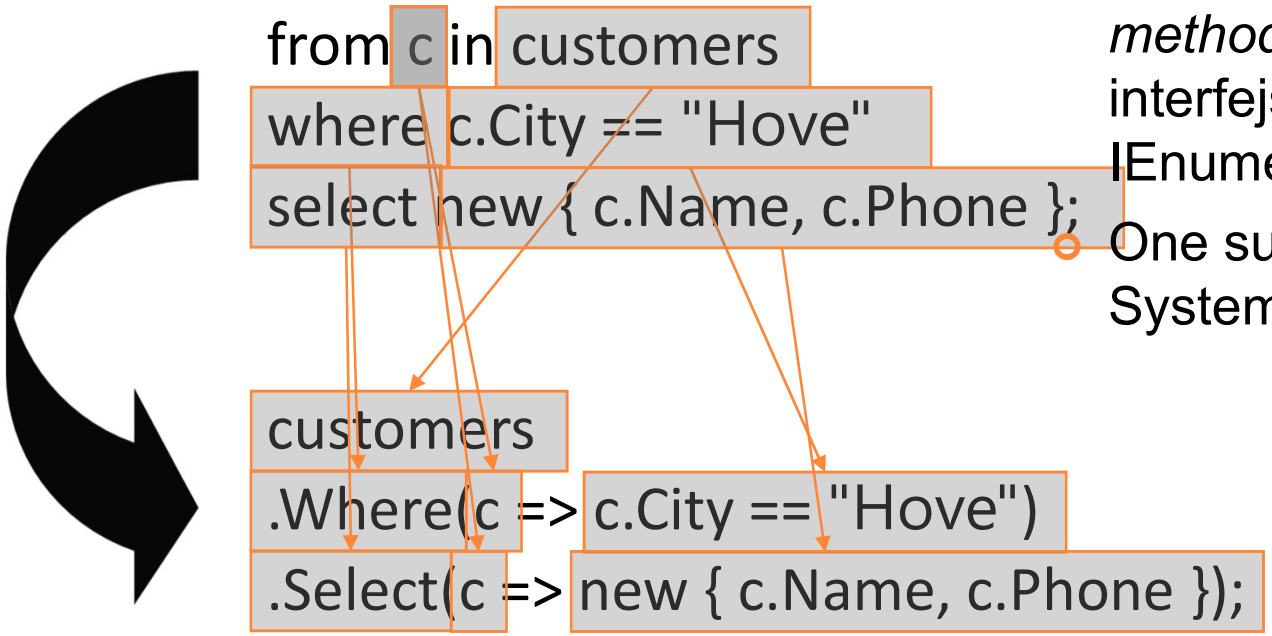


- **Where<T>,Select<T>** metode, koje odgovaraju *where* odnosno *select* rečima u upitu se transformišu u *extension methods* koje su definisane interfejsom IEnumerable<T>.
- One su u prostoru imena System.Linq namespace.

19:38

EXTENZIJA U QUERY IZRAZIMA

- Upit se transformiše u pozive metoda
  - Where, Join, OrderBy, Select, GroupBy, …



- **Where<T>,Select<T>** metode, koje odgovaraju *where* odnosno *select* rečima u upitu se transformišu u *extension methods* koje su definisane interfejsom IEnumerable<T>.
- One su u prostoru imena System.Linq namespace.

19:38

NAPOMENA

- Ako neka ekstenzija ima isti naziv tačnije potpis kao i unutrašnja metoda, prioritet ima, svakako, unutrašnja metoda.
- **Svojstva, događaji i operatori se ne mogu proširivati**.

LAMBDA EXPRESSIONS – PRIMER (1)

```
public delegate bool Predicate<T>(T obj);

public class List<T>
{
    public List<T> FindAll(Predicate<T> test)
    {
        List<T> result = new List<T>();
        foreach (T item in this)
            if (test(item)) result.Add(item);
        return result;
    }
    …
}
```

Metoda u klasi *List*

LAMBDA EXPRESSIONS – PRIMER (2)

```
public class MyClass
{
    public static void Main() {
        List<Customer> customers = GetCustomerList();
        List<Customer> locals =
            customers.FindAll(
                new Predicate<Customer>(CityEqualsHove)
            );
    }

    static bool CityEqualsHove(Customer c) {
        return c.City == "Hove";
    }
}
```

19:38

LAMBDA EXPRESSIONS – PRIMER (3)

19:38

```
public class MyClass
{
    public static void Main() {
        List<Customer> customers = GetCustomerList();
        List<Customer> locals =
            customers.FindAll(
                delegate(Customer c) { return c.City == "Hove"; }
            );
    }
}
```

LAMBDA EXPRESSIONS – PRIMER (4)

```
public class MyClass
{
    public static void Main() {
        List<Customer> customers = GetCustomerList();
        List<Customer> locals =
            customers.FindAll(
                (Customer c) => {return c.City == "Hove";}
            );
    }
}
```

Lambda expression

19:38

LAMBDA EXPRESSIONS – PRIMER (5)

```
public class MyClass
{
    public static void Main() {
        List<Customer> customers = GetCustomerList();
        List<Customer> locals =
            customers.FindAll(
                c => c.City == "Hove"
            );
    }
}
```

Lambda expression

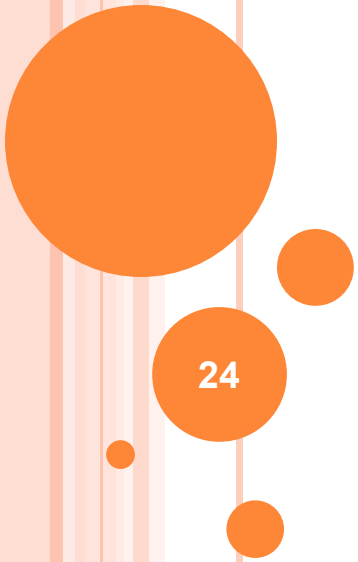
19:38

PRIMER

```
public Form1()
{
    InitializeComponent();
    List<string> a = new List<string>();
    a.Add("pera"); a.Add("test"); a.Add("mika");
    a.Add("test"); a.Add("test");

    List<string> rez1 = a.FindAll(x=>x=="test");
    List<string> rez2 = a.FindAll(delegate(string x) { return x == "test"; });
    List<string> rez3 = a.FindAll(new Predicate<string>(testiranje));
}
public bool testiranje(string x)
{
    return x == "test";
}
```

19:38



LAMBDA IZRAZI – NA DRUGI NAČIN

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");
```

19:38

LAMBDA IZRAZI (3)

- LE pojednostavljuje kodovanje delegata i anonimnih metoda. Argument koji se prosljeđuje metodi **Where<T>** je jedan primer *lambda expression*:

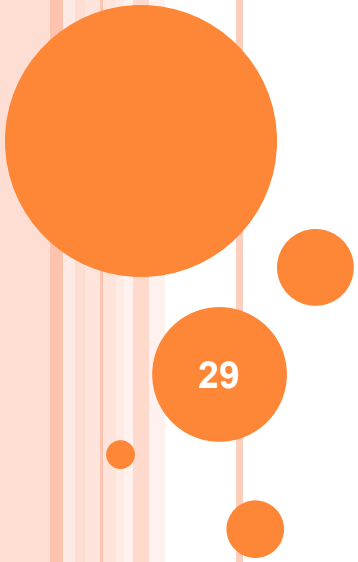
  *Where(**p => p.ID == 1**)*

- LE nam dozvoljavaju da pišemo funkcije koje se mogu proslediti kao argumenti drugim metodama.
- Moguće je koristiti LINQ i bez LE tj. kodovati na standaradan način, ali je korišćenjem LE jednostavnije.

```
Func< Osoba, bool> filter = delegate(Osoba p)   {
   return p.ID == 1; };

var query = osobe
.Where( filter(p) )
.Select(p => new { p.FirstName, p.LastName } );
```

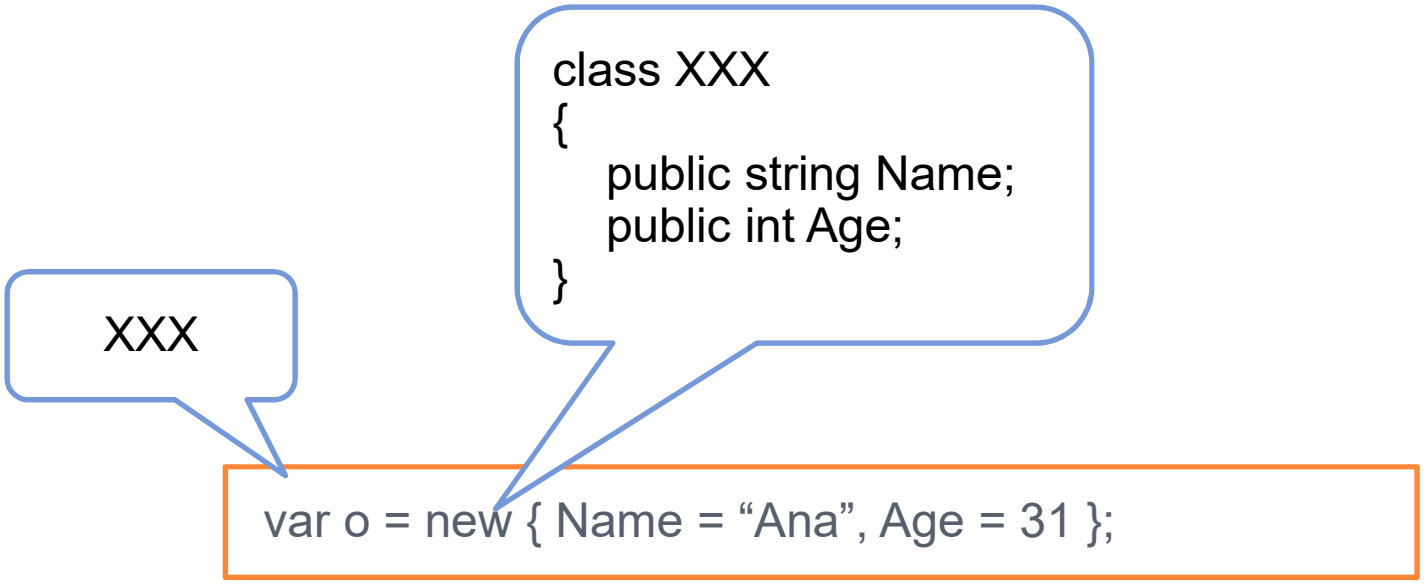
19:38



ANONIMNI TIPOVI

Anonymous Types

ANONIMNI TIPOVI (1)



ANONIMNI TIPOVI (2)

- Predstavljaju sastavni deo Linq-a:

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

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

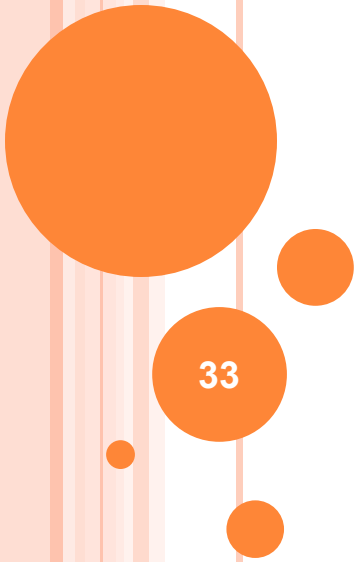
- **Anonimni tipovi dozvoljavaju da se radi sa rezultatima upita bez eksplicitne definicije klase koja ih predstavlja.**

- internal class ???
- {

```
    private string _firstName;
    private string _lastName;

    public string FirstName {
     get { return _firstName; }
     set { firstName = value; }
    }
    public string LastName {
     get { return _lastName; }
     set { lastName = value; }
    }
```

- }



STANDARD QUERY OPERATORS

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;
```

19:38

- *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;

19:38

- 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).

19:38

- **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);
```

QUANTIFICATION OPERATORI

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

- **Primeri**

- int[] twos = { 2, 4, 6, 8, 10 };

- // true
- bool areAllevenNumbers = **twos.All**(i => i % 2 == 0);

- // true
- bool containsMultipleOfThree = **twos.Any**(i => i % 3 == 0);
-
- // false
- bool hasSeven = **twos.Contains**(7);

19:38

Implementacija sopstvenog pravila

```
Employee employee =
            new Employee { ID = 1, Name = "Poonam" };

 Func<Employee, bool>[]  validEmployeeRules =
{
        e => e. ID > 0,
        e => !String.IsNullOrEmpty(e.Name)
};

bool isValidEmployee =
        validEmployeeRules.All(rule => rule( employee ));
```

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.***

19:38

PARTITION OPERATORS

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 ***SkipUntil*** , ***TakeUntil***.
- int[] numbers = { 1, 3, 5, 7, 9 };
- var query = numbers.SkipWhile(n => n < 5)
  .TakeWhile(n => n < 10); // daje 5, 7, 9

19:38

JOIN OPERATORI

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

- **Primer:**

```
var employees = new List<Employee> {
new Employee { ID=1, Name="Scott", DepartmentID=1 },
new Employee { ID=2, Name="Poonam", DepartmentID=1 },
new Employee { ID=3, Name="Andy", DepartmentID=2} };
var departments = new List<Department> {
new Department { ID=1, Name="Engineering" },
new Department { ID=2, Name="Sales" },
new Department { ID=3, Name="Skunkworks" } };
```

19:38

```
var query =
from employee in employees
join department in departments
on employee.DepartmentID equals department.ID
select  new {
   EmployeeName = employee.Name,
   DepartmentName = department.Name
};
```

OPERATORI GRUPISANJA

- **GroupBy ,ToLookup**

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

- **Primeri:**
- 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* operatora je što operator *GroupBy* obavlja izvršavanje sa kašnjenjem (engl. *lazy*). *ToLookup* obavlja izvršavanje odmah.

19:38

GENERATIONAL OPERATORS

- ***Empty*** kreira praznu sekvencu *IEnumerable<T>*.
- var empty = Enumerable.**Empty**<Employee>();
- ***Range*** operator generiše sekvencu brojeva
- ***Repeat*** generiše sekvencu bilokojih vrednosti.
- var empty = Enumerable.Empty<Employee>();
- int start = 1;
- int count = 10;
- IEnumerable<int> numbers = Enumerable.Range(start, count);
- var tenTerminators = Enumerable.Repeat(new Employee { Name ="Arnold" }, 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:**
- Employee e1 = new Employee() { ID = 1 };
- Employee e2 = new Employee() { ID = 2 };
- Employee e3 = new Employee() { ID = 3 };
- var employees1 = new List<Employee>() { e1, e2, e3 };
- var employees2 = new List<Employee>() { e3, e2, e1 };
- //false
- bool result = employees1.SequenceEqual(employees2);

ELEMENT OPERATORS

- **ElementAt, First, Last**, **Single**
- Za svaki operator postoji odgovarajući "**or default**" 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 operacija *First* i *Single* je što *Single* operator šalje izuzetak ako sekvenca ne sadrži jedan element, dok *First* vraća rezultat koji je prvi element. *First* šalje izuzetak samo ako ne postoji ni jedan 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" };
// will return a sequence of 2 strings
var query1 = data.OfType<string>();
// will create an exception when executed
var query2 = data.Cast<string>();
```

19:38

CONCATENATION

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

19:38

```
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) };
```

19:38

LINQ TO DATASET

primer 1.

var q = from role in ds.Role

select role;

DataTable dtRole = q.**CopyToDataTable**();

primer 2.

```
var oct2006 = from d in ds.Doctors join c in ds.Calls on d.Initials
   equals c.Initials
where c.DateOfCall >= new DateTime(2006, 10, 1) && c.DateOfCall <=
   new DateTime(2006, 10, 31)
group c by d.Initials into g
orderby g.Count() descending
select new { Initials = g.Key, Count = g.Count(), Dates = from c in
   g select c.DateOfCall };
```

19:38