

Sovelto



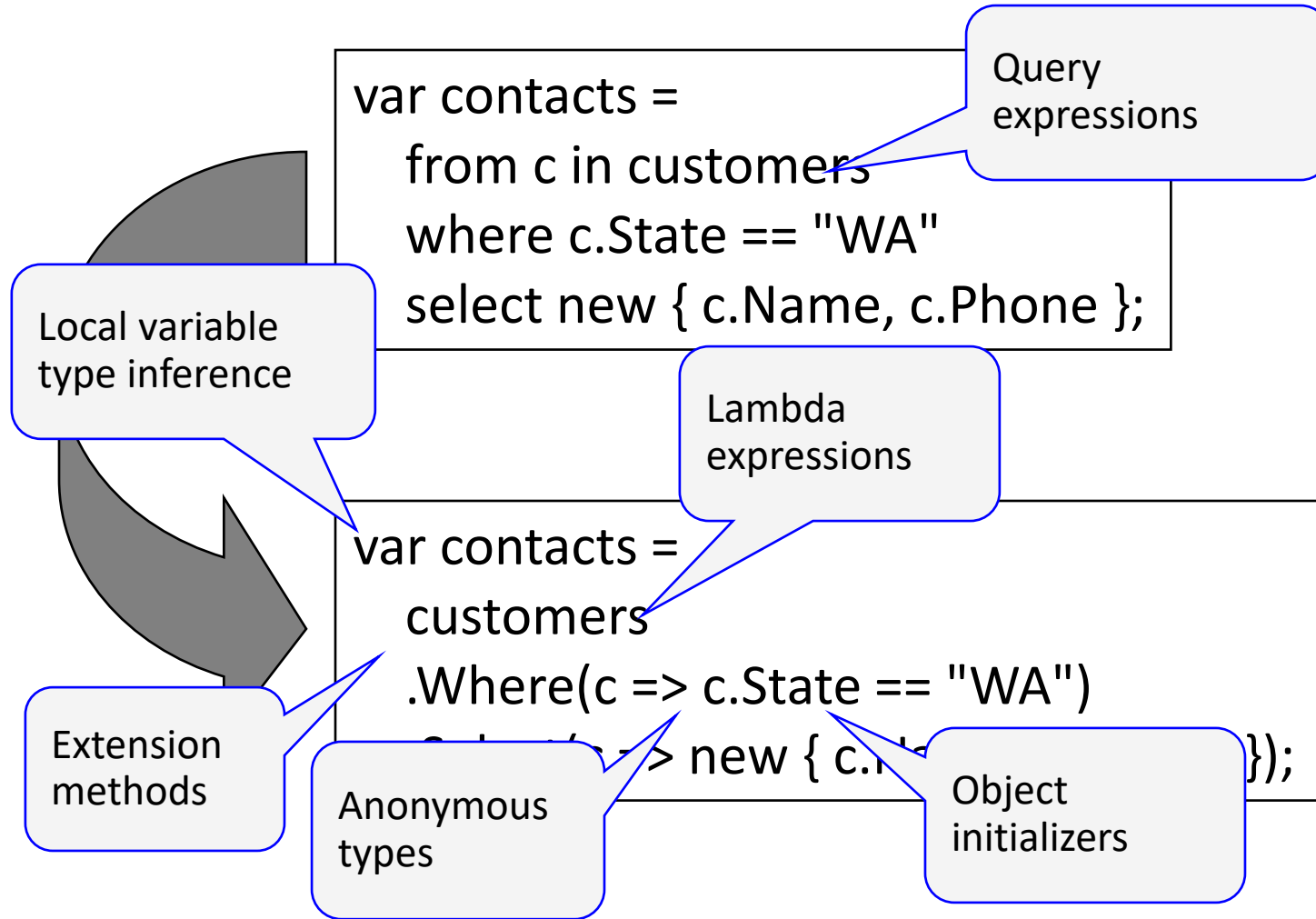
LINQ

Language Integrated Query

LINQ tavoitteita

- Sama ohjelmointimalli "kaikkeen" dataan riippumatta siitä missä data sijaitsee ja millaisessa formaatissa
 - XML → Linq to XML
 - SQL → Linq to SQL
 - Entity Framework → Linq to Entities
 - .NET Collections → Linq to Objects
 - ... (oma LINQ provider)
- Kyselykieli on ohjelmointikielen eli C#:n ominaisuus
- Vahvasti tyypitetty
 - mahdollistaa käännösaikaisen tarkistamisen
 - ja sen, että VS:n intellisense auttaa koodaamista
- Perustuu C# 3:n piirteille Extension methods, Lambda expressions, Anonymous types ja Object Initializers
- Syntaksi osin SQL-kielen kaltainen, muuten näillä ei ole mitään tekemistä keskenään

C# 3 piirteet ja LINQ



LINQ-kyselyn suoritus

- Query –muuttuja sisältää kyselyn määrittelyn
- Kysely suoritetaan vasta sitten kun tulosjoukkoa käsitellään
 - Deferred Execution
- Tulosjoukon käsittelyä ovat:
 - foreach
 - funktiot, jotka palauttavat singletonin, kuten Sum(), Count()
 - tulosten siirtäminen listaan (cache) (esim. ToList(), ToArray())

Query -muuttujat

- IEnumerable<T> tai siitä johdettu tyyppi
- Tai annetaan kääntäjän päättää tyyppi (var)

```
IEnumerable<Henkilö> q =  
    from h in hlöt  
    where h.Ikä > 20  
    select h;
```

```
List<Henkilö> hlöt = new List<Henkilö>() {  
    new Henkilö() { Nimi = "Matti", Ikä = 20 },  
    new Henkilö() { Nimi = "Teppo", Ikä = 25 },  
    new Henkilö() { Nimi = "Maija", Ikä = 22 }  
};
```

```
var q =  
    from h in hlöt  
    where h.Ikä > 20  
    select h;  
// q.GetType().Name == "WhereListIterator<T>"
```

Query - muuttujatyypit

Lähde: Rainer Stropek, Basta!

```
List<string> names =  
    new List<string>{"John", "Rick", "Maggie", "Mary"};  
  
IEnumerable<string> nameQuery = from name in names  
                                where name[0] == 'M'  
                                select name;  
  
foreach (string str in nameQuery)  
{  
    Console.WriteLine(str);  
}
```

Diagram illustrating the flow of variables in the first example:

- 1: Variable `names` is used in the `from` clause of the query.
- 2: Variable `name` is used in the `select` clause of the query.
- 3: Variable `nameQuery` is used in the `foreach` loop.

```
Table<Customer> Customers = db.GetTable<Customers>();  
  
IQueryable<string> custNameQuery =  
    from cust in Customers  
    where cust.City == "London"  
    select cust.Name;  
  
foreach (string str in custNameQuery)  
{  
    Console.WriteLine(str);  
}
```

Diagram illustrating the flow of variables in the second example:

- 1: Variable `Customers` is used in the `from` clause of the query.
- 2: Variable `cust` is used in the `select` clause of the query.
- 3: Variable `custNameQuery` is used in the `foreach` loop.

```
Table<Customer> Customers = db.GetTable<Customers>();  
  
var namePhoneQuery =  
    from cust in Customers  
    where cust.City == "London"  
    select new { name = cust.Name,  
                phone = cust.Phone };  
  
foreach (var item in namePhoneQuery)  
{  
    Console.WriteLine(item);  
}
```

Diagram illustrating the flow of variables in the third example:

- 1: Variable `Customers` is used in the `from` clause of the query.
- 2: Variable `cust` is used in the `select` clause of the query.
- 3: Variable `namePhoneQuery` is used in the `foreach` loop.

```
var Customers = db.GetTable<Customers>();  
  
var custQuery = from cust in Customers  
                where cust.City == "London"  
                select cust;  
  
foreach (var item in custQuery)  
{  
    Console.WriteLine(item);  
}
```

Diagram illustrating the flow of variables in the fourth example:

- 1: Variable `Customers` is used in the `from` clause of the query.
- 2: Variable `cust` is used in the `select` clause of the query.
- 3: Variable `custQuery` is used in the `foreach` loop.

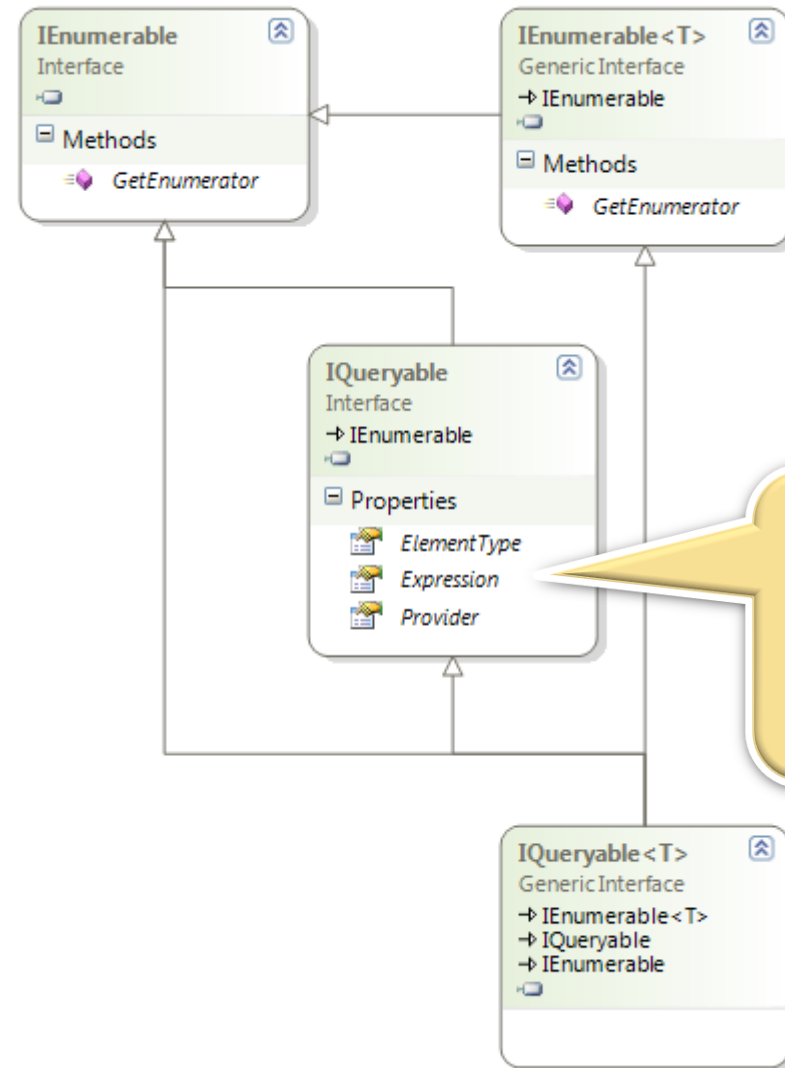
Perusoperaatiot

Operaatiot	LINQ varattu sana (C#)
DataSource-määrittely	from, let
Valinta (Filtering)	where
Järjesty	orderby
Ryhmittely (Grouping)	group
Liittäminen (Joining)	from, join
Projektio	select

from

- from range_variable in data_source
- data_source
 - IEnumerable tai IEnumerable<T>
 - Tai näistä johdettu tyyppi (esim. IQueryable<T>)
- range_variable
 - Kuten foreach -lauseen iteraattori-muuttuja

from: datasource -rajapinnat



IQueryable sisältää Expression-propertyn. Koodia, joka onkin dataa, ja voidaan tulkata ajon aikana.

from: Range-variable

- Vahvasti tyypitetty
 - Tyyppi määräytyy data-sourcen tyyppin perusteella

```
string[] nimet = { "Matti", "MattiPekka", "Pekka" };  
  
var matit = from n in nimet  
            where n.Contains("Matti")  
            select n;
```

```
// kääntäjä tietää tyyppin, joten "oikeasti" syntaksi on  
var matit = from string n in nimet  
            ...
```

from

- Yhdistetty from-lause
 - voi olla navigointi oliohierarkiassa tai
 - muodostaa cross-join:in

```
NorthwindDataContext dc = new  
NorthwindDataContext();
```

```
// tuotteet, joiden kategoriassa on yli 10 tuotetta  
var q =  
    from c in dc.Categories  
    from p in c.Products  
    where c.Products.Count() > 10  
    select new { Ryhmä = c.CategoryName, Tuote =  
p.ProductName };
```

Navigointi
kategoriasta
tuotteisiin

```
var q = from p1 in dc.Products  
        from p2 in dc.Products  
        select new { P1=p1.ProductName, P2=p2.ProductName};
```

Cross Join

where

- Kullekin elementille (range variable) suoritettava boolean-ehto
- Päättää, otetaanko elementti tulosjoukkoon
- Ehtoja voi olla 0...n kappaletta
- where lause voi olla missä tahansa kohtaa kyselyä lukuunottamatta ensimmäinen/viimeinen positio
- Huom: OfType metodia voidaan käyttää valitsemaan tietyn tyyppiset elementit

where

```
var q = from p in dc.Products
        where p.UnitsInStock < 10
        where p.UnitsOnOrder == 0
        select new { p.ProductName, p.UnitsInStock, p.UnitsOnOrder };
```

Useita where-lauseita,
Sama kuin

```
where p.UnitsInStock < 10
    && p.UnitsOnOrder == 0
```

```
var q1 =
    from p in dc.Products
    orderby p.UnitsInStock
    where p.UnitsInStock < 10
    select new { p.ProductName, p.UnitsInStock, p.UnitsOnOrder };
```

OrderBy/Where -järjestyksellä
ei ole väliä

```
var q2 = from p in dc.Products
        where p.UnitsInStock < 10
        orderby p.UnitsInStock
        select new { p.ProductName, p.UnitsInStock, p.UnitsOnOrder };
```

select

- Projektio: minkä tyyppisiä olioita query palauttaa
- Vaihtoehdot joista select voi koostua
 - Range-variable
 - saman tyyppisiä olioita kuin data-source
 - property tai metodin paluuarvo
 - Tyyppi on propertyn tai metodin paluuarvon tyyppi
 - luo anonyymin tyypin
 - luodaan tunnetun tyypin instansseja

select

```
//kunkin kategorian ensimmäinen tuote  
var q1 = from c in dc.Categories  
        select c.Products.First();
```

Metodin paluuarvo

```
var q2 = from c in dc.Categories  
        select c.CategoryID;
```

Property

```
var q3 = from c in dc.Categories  
        select new { c.CategoryID, c.CategoryName };
```

Anonyymi tyyppi

```
var q4 = from c in dc.Categories  
        select new Category {  
            CategoryID=c.CategoryID,  
            CategoryName=c.CategoryName  
            Description="eräs kategoria",  
        };
```

Tunnetun tyypin instanssi

group

- group range_variable
by expression
[into continuation_variable]
- Palauttaa joukon (sequence) elementtejä, joiden tyyppi on:
IGrouping<Tkey, TElement>
 - Siis hierarkkinen tulosjoukko
 - SQL:ssä ei ole vastaavaa käsitettä
- group määreen jälkeinen where valitsee grouped –tuloksien arvoja

Group

- Tuloksena on lista `Group<TKey, TValue>` -oliota, joilla on
 - Property Key, joka sisältää ryhmittelyarvon (tässä `c.City`-arvon, eli string)
 - `GetEnumerator()` –metodi, jolla voidaan loopata ryhmään kuuluvat oliot
 - Ja ryhmän tulosjoukkoon voidaan kohdistaa muitakin LINQ-hakuja

```
//asiakkaiden lukumäärä kaupungeittain  
var q1 = from c in dc.Customers  
        group c by c.City;
```

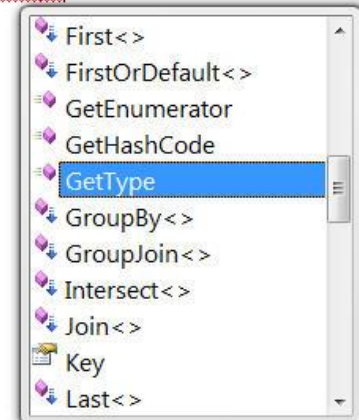
```
foreach (var customerGroup in q1) {  
    listBox1.Items.Add("kaupunki:" + customerGroup.Key);  
    foreach (var cc in customerGroup) {  
        listBox1.Items.Add(" " + cc.CompanyName);  
    }  
}
```

Mahdollisimman simppelellä
group, ei selectiä



kaupunki:Buenos Aires
Cactus Comidas para llevar
Océano Atlántico Ltda.
Rancho grande
kaupunki:Butte
The Cracker Box
kaupunki:Campinas
Gourmet Lanchonetes
kaupunki:Caracas
GROSELLA-Restaurante
kaupunki:Charleroi
Suprêmes délices

customerGroup.



First<>
FirstOrDefault<>
GetEnumerator
GetHashCode
GetType
GroupBy<>
GroupJoin<>
Intersect<>
Join<>
Key
Last<>

orderby

- orderby expression [ascending|descending]
[, expression [ascending|descending]...]
- Tuloslaidien järjestys
- Voi olla missä kohtaa tahansa kyselyä, paitsi ei ensimmäisenä tai viimeisenä

```
var q =  
    from c in dc.Categories  
    from p in c.Products  
    orderby c.CategoryName, p.UnitPrice descending  
    select new {c.CategoryName, p.ProductName,  
p.UnitPrice }  
    ;
```

Koostainen järjestelyehto

join

- join voidaan käyttää:
 - Inner join
 - Group join
 - Left Outer join
- Tarpeen ainoastaan silloin kun olioilla ei ole suoraa suhdetta itse oliomallissa
 - tai se on "väärin päin", from-lauseella voidaan ottaa parent-olioon viittaavat child-oliot
- Ainoastaan equity join (==). Jos join on tehtävä muilla ehdoilla (tai useammalla kentällä), on käytettävä where-ehtoa.

Left outer join

- Käytä DefaultIfEmpty –metodia

```
var q =  
    from em in dc.Employees  
    join o in dc.Orders  
        on em equals o.Employee into ords  
    from o in ords.DefaultIfEmpty()  
    select new { Employee = em, Order = o },  
  
foreach (var emp_Order in q) {  
    listBox1.Items.Add(emp_Order.Employee.LastName  
        + " order:" + emp_Order.Order.OrderID);  
}
```

Ehto voidaan tehdä oliolla,
(ei siis viiteavaimen arvolla)

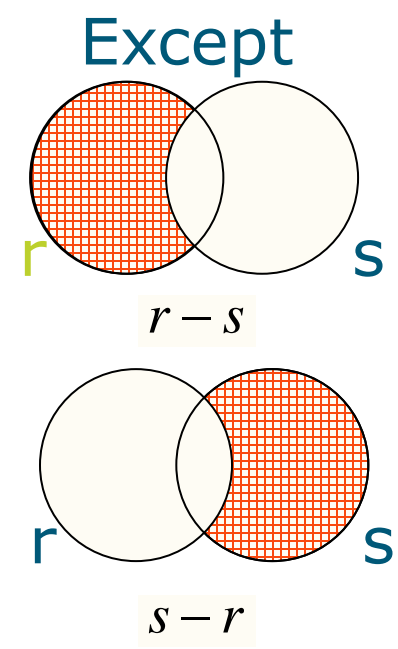
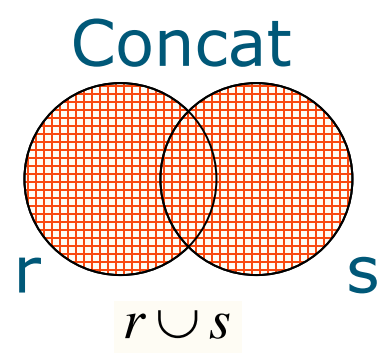
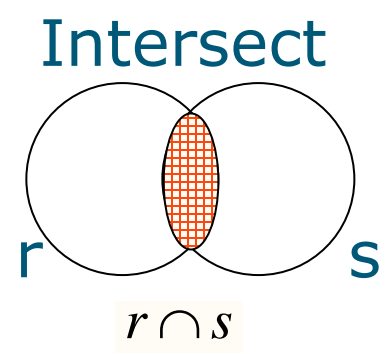
Jos on Employee, jolla ei
ole Ordereita,
niin o == null

Query-muuttujien operaatioita

Operaatio	IEnumerable<T> Method
Elementti tietystä indeksistä	ElementAt, ElementAtOrDefault
Ensimmäinen elementti	First, FirstOrDefault
Viimeinen elementti	Last, LastOrDefault
Tietyn tyyppinen elementti	OfType
Sivutus	Skip, SkipWhile, Take, TakeWhile
Yksilölliset elementit	Distinct

Joukko-operaatiot

Operaatio	IEnumerable<T> Method
Unioni	Concat
Leikkaus	Intersect
Joukon erotus	Except



Joukko-operaatiot

```
var q =  
    (  
        from c in dc.Customers  
        select new {  
            Nimi = c.CompanyName,  
            Puhelin = c.Phone }  
    ).Concat(  
        from s in dc.Suppliers  
        select new {  
            Nimi = s.CompanyName,  
            Puhelin = s.Phone }  
    );
```

Aggregaatti-operaatiot

Operaatio	Metodi
Oma aggregaatti	Aggregate
Keskiarvo	Average
Lkm	Count, LongCount
Maksimi	Max
Minimi	Min
Summa	Sum

Aggregaatit: count

```
MessageBox.Show($"tuotteita on {dc.Products.Count()} kappaletta");
```

```
MessageBox.Show(  
    $"Tuotteita {dc.Products.Count(p => p.UnitsInStock > 0)} kpl", );
```

```
var q =  
    from c in dc.Categories  
    select new {  
        c.CategoryID,  
        c.CategoryName,  
        ProductCount = c.Products.Count()  
    };
```

Aggregaatit – lisää esimerkkejä

```
MessageBox.Show(  
    $"tuotteita yht {dc.Products.Sum(p => p.UnitsInStock)} kpl"());  
  
//tuotteiden lukumäärä kategorioittain  
var q = from p in dc.Products  
        group p by p.Category into g  
        select new {  
            g.Key.CategoryName,  
            TotalInStock = g.Sum(p => p.UnitsInStock)  
        };  
  
//tuotteet, joilla on lyhimmät nimet  
int shortest = dc.Products.Min(p => p.ProductName.Length);  
var shortestNameProducts =  
    from p in dc.Products  
    where p.ProductName.Length == shortest  
    select new { p.ProductName };
```

Query vs. Method Syntax

- LINQ on kielen ominaisuus, ei .NET Frameworkin
- Kääntäjä muuttaa query:t metodikutsuiksi
 - jotka saavat pääosin Expression –tyyppisiä argumentteja
- Voit käyttää kumpaa tahansa: Query tai Metodi –syntaksia
 - joka tapauksessa on viivästetty suoritus, eli LINQ suoritetaan vasta sitten kun dataa luetaan (ei määrittelyvaiheessa)
- Query-syntaksi on helpompi, sillä kannattaa aloittaa
 - luettavampaa
 - lyhyempää
- Jotkut kyselyt edellyttävät kuitenkin metodi-syntaksin käyttämistä, esim:
 - ehdon täyttävien elementtien lkm
.Count(lambda_expression)
 - maksimiarvon hakeminen tietystä propertystä
.Max(lambda_expression)
 - Skip, Take

LINQ-metodeja: Skip, Take

```
var q = (  
    from c in dc.Customers  
    orderby c.ContactName  
    select new { c.ContactName, c.CustomerID }  
)  
    .Skip(50)  
    .Take(10)  
;
```

Sivun pituus on 10
elementtiä ja mennään 6.
sivulle

Query vrs. Method Syntax

```
var q =
```

```
    from c in dc.Customers  
    where c.Country == "Finland"  
    select new { c.CompanyName, c.Phone };
```

Query-syntaksi

```
var qMethod =
```

```
    dc.Customers  
    .Where(c => c.Country == "Finland")  
    .Select(c => new {c.CompanyName, c.Phone });
```

Metodi-syntaksi

Query vs. Method Syntax

- LINQ:ä voi käyttää kaikkien kokoelmien kanssa – myös tiedostolistausten

```
var di = new DirectoryInfo(  
    Environment.GetFolderPath(Environment.SpecialFolder.Desktop));
```

```
from fi in di.GetFiles()  
where fi.Extension == ".txt"  
orderby fi.Name  
select fi.Name
```

Query-syntaksi

```
di.GetFiles()  
.Where(fi => fi.Extension == ".txt")  
.OrderBy(fi => fi.Name)  
.Select (fi => fi.Name )
```

Metodi-syntaksi