

# C# 3.0 és LINQ

**Albert István**

[ialbert@aut.bme.hu](mailto:ialbert@aut.bme.hu)

**Q.B. 221, 1662**

Budapesti Műszaki és Gazdaságtudományi Egyetem

Automatizálási és Alkalmazott Informatikai Tanszék

# Tartalom

## Lambda kifejezések

- Névtelen metódusok még rövidebb szintaktikával

```
Func<string, bool> t =  
    delegate (string s) { return s.StartsWith("S"); };
```

> Ugyanaz mint:

```
Func<string, bool> t = s => s.StartsWith("S");
```

> 's' típusát kitalálja a környezetből !

- Több paraméter esetén a szintaktika

```
(x, y) => Math.Sqrt(x * x + y * y)
```

- Paraméter nélkül is írható

```
() => 42
```



## Objektum inicializálók

### Object initializers

- Objektumok inicializálása ha nincs megfelelő konstruktor

> Mezők és tulajdonságok beállítása

```
var b = new Music { Title = "Soul Bop", Length = 591 };
```

```
var b0 = new Music()  
{ b0.Title = "Soul Bop";  
  b0.Length = 591;  
  var b = b0;
```

- Kifejezésben is használható

```
new Music { Title = "Soul Bop", Length = 591 }.Dump();
```

```
var list = new List<Music>();  
list.Add(new Music { Title = "Soul Bop", Length = 591 });
```



## Standard lekérdezések

### Standard Query Operators

- Sorozatokon (sequence, IEnumerable<T>) működnek

> A sorozat egyszerűen átalakítható: ToList(), ToArray(), ...

- A System.Linq.Enumerable statikus osztály bővítő metódusaiból állnak

- Jellemzőik

> Tipikusan késői kiértékeléssel dolgoznak (yield)  
– Kivételek, például Reverse(), OrderBy(), stb.  
> Összefűzhetőek, mégis hatékonyak  
> Megmarad a jó olvashatóság, komponálhatóság



## LINQ to ...

```
from m in list where m.Title.StartsWith("S") select m.Title;
```

```
SELECT Title FROM Music WHERE Title LIKE 'S%'
```

- Mi kell ahhoz, hogy elvégezzük az átalakítást?

> Leképezés: osztály – tábla, tag – oszlop, stb.

> A szűrési feltétel eredeti, értelmezhető formában

– Nem lefordítva IL kódra

- Expression<T>: a kód adatként látszik

> ahol T a kifejezésre jellemző metódusreferencia



# Lambda kifejezések

- Névtelen metódusok még rövidebb szintaktikával

```
Func<string, bool> t =  
    delegate (string s) { return s.StartsWith("S"); };
```

> Ugyanaz mint:

```
Func<string, bool> t = s => s.StartsWith("S");
```

> 's' típusát kitalálja a környezetből !

- Több paraméter esetén a szintaktika

```
(x, y) => Math.Sqrt(x * x + y * y)
```

- Paraméter nélkül is írható

```
() => 42
```

# Lambda kifejezések

- A paraméterlistában a paraméter típusok megadása opcionális
- Lehet kifejezés:

```
x => x + 1
```

- Vagy statement:

```
{ Console.WriteLine("x:{0}", x); return x + 1; }
```

- Lehet benne több metódushívás

# Automatikus tulajdonságok

Automatically implemented properties

- A hozzá tartozó mező nem érhető el
  - > Get ÉS set is kell egyszerre !
  - > A láthatóság különbözhet

0 references

```
public class List<T>
```

```
{
```

1 reference

```
public int Count { get; private set; }
```

0 references

```
public void Add(T item)
```

```
{ this.array[Count++] = item; }
```

```
private int ◇field0;
```

```
public int Count {  
    get { return ◇field0; }  
    private set  
        { ◇field0 = value; } }
```

# Bővítő metódus készítése

## Extension methods

- Meglévő osztály kiterjesztése

```
namespace linqSample.Utills
```

```
{
```

0 references

```
public static class Helper
```

```
{
```

0 references

```
static public IEnumerable<T> Filter<T>
```

```
(this IEnumerable<T> seq, Predicate<T> predicate)
```

```
{
```

```
foreach (var t in seq)
```

```
if (predicate(t))
```

```
yield return t;
```

Statikus bővítő  
metódus

A bővítendő típus a 'this'  
kulcsszó után

```
}
```

# Bővítő metódus felhasználása

## Extension methods

- T lehet generikus osztály vagy interfész is
  - > Mostantól minden sorozat szűrhető 😊
  - > Használni kell a bővítő osztály névterét

```
using linqSample.Utils;
```

A névtérben lévő  
összes  
bővítőmetódus  
engedélyezése

```
var a = new[] { 1, 2, 3 };  
IEnumerable<int> ints = a;  
foreach( var i in ints.Filter(i => i <= 2) )  
    Console.WriteLine(i);
```

# Bővítő metódus előnyei

- Rövidebb, átláthatóbb kód
- Bátran készíts sajátot!
- De ne szennyezd be az általános osztályokat speciális funkciókkal

```
var t1 = Enumerable.Where(a, i => i > 2);  
var t2 = Enumerable.OrderBy(t1, i => i);  
var t3 = Enumerable.ToList(t2);  
  
var t4 = a.Where(i => i > 2).OrderBy(i => i).ToList();
```



# Implicit típusú lokális változók


## Local Variable Type Inference

- Nem kell kiírni a lokális változó típusát
  - > A változó típusa a jobb oldali kifejezés típusa lesz

```
List<string> list = new List<string>();  
foreach (string x in list.Filter(s => s.StartsWith("S"))) ;
```

- var == „a jobb oldal típusa”

```
var list = new List<string>();  
foreach (var x in list.Filter(s => s.StartsWith("S"))) ;
```

 class System.String

Represents text as a sequence of UTF-16 code units.To browse the .NET Framework source

# Objektum inicializálók

## Object initializers

- Objektumok inicializálása ha nincs megfelelő konstruktor
  - > Mezők és tulajdonságok beállítása

```
var b = new Music { Title = "Soul Bop", Length = 591 };
```

```
var b0=new Music();  
b0.Title="Soul Bop";  
b0.Length=591;  
var b = b0;
```

- Kifejezésben is használható

```
new Music { Title = "Soul Bop", Length = 591 }.Dump();
```

```
var list = new List<Music>();  
list.Add(new Music { Title = "Soul Bop", Length = 591 });
```

# Beágyazott tagokra is

- Összetett típusú tagok inicializálása
  - > Csak a tulajdonságok beállítása

```
new Music {  
    Title = "Soul Bop", Length = 591,  
    Album = { Artist = "Bill Evans", Year = 2000 }  
};
```

- > Új objektum létrehozása értékadással

```
new Music {  
    Title = "Soul Bop", Length = 591,  
    Album = new Album { Artist = "Bill Evans", Year = 2000 }  
};
```

# Gyűjtemények inicializálása

- Hasonló a tömb inicializáláshoz

```
var list2 = new List<Music> {  
    new Music { Title = "Soul Bop", Length = 591 },  
    new Music { Title = "Hangin' in the City", Length = 397 } };
```

- Publikus Add metódust hív
  - > Több paraméteres Add metódust is támogat, pl:

```
var szamok = new Dictionary<int, string>  
{ { 0, "nulla" }, { 1, "egy" }, { 2, "kettő" } };
```

- Új implicit tömb inicializálás

```
var v = new[] { 1, 1.5, 2.3 };    // double [ ]
```

# Gyűjtemények inicializálása

- IEnumerable-t kell implementálnia
- Az Add metódust hívja meg
  - > normal overload resolution

# Névtelen osztályok

## Anonymous object creation expressions

- Az osztály az inicializáláskor kapott tagokat kapja
  - > Csak olvasható tulajdonságok
  - > ToString(), Equals( object ), GetHashCode()
- Azonos típus- és névsorrend ugyanazt a típust hivatkozza

```
var list3 = new[] {  
    new { Title = "Soul Bop", Length = 591 },  
    new { song.Title, song.Length } };  
  
foreach (var x in list.Filter(s => s.Length < 300))  
    Console.WriteLine(x);
```

# Standard lekérdezések

## Standard Query Operators

- Sorozatokon (sequence, `IEnumerable<T>`) működnek
  - > A sorozat egyszerűen átalakítható: `ToList()`, `ToArray()`, ...
- A `System.Linq.Enumerable` statikus osztály bővítő metódusaiból állnak
- Jellemzőik
  - > Tipikusan késői kiértékeléssel dolgoznak (`yield`)
    - Kivételek, például `Reverse()`, `OrderBy()`, stb.
  - > Összefűzhetőek, mégis hatékonyak
  - > Megmarad a jó olvashatóság, komponálhatóság

# *A LINQ To Objects műveletei*

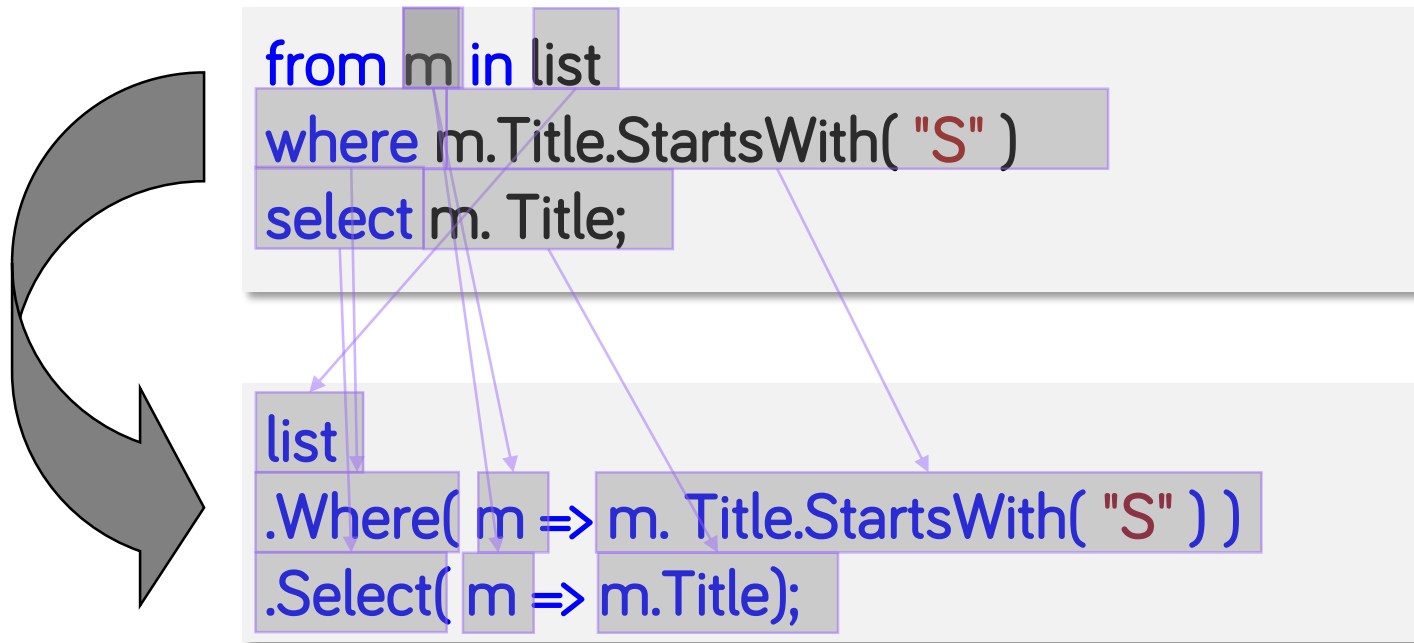
<b>Szűrés</b>	<b>Where</b>
<b>Projekció</b>	<b>Select, SelectMany</b>
<b>Rendezés</b>	<b>OrderBy, ThenBy</b>
<b>Csoportosítás</b>	<b>GroupBy</b>
<b>Joinok</b>	<b>Join, GroupJoin</b>
<b>Quantifiers</b>	<b>Any, All</b>
<b>Particionálás</b>	<b>Take, Skip, TakeWhile, SkipWhile</b>
<b>Halmazműveletek</b>	<b>Distinct, Union, Intersect, Except</b>
<b>Elemek</b>	<b>First, Last, Single, ElementAt</b>
<b>Aggregáció</b>	<b>Count, Sum, Min, Max, Average</b>
<b>Konverzió</b>	<b>ToArray, ToList, ToDictionary</b>
<b>Kasztolás</b>	<b>OfType&lt;T&gt;, Cast&lt;T&gt;</b>



# Lekérdezések

## Query Expressions

- Lekérdezések nyelvi szinten (C# 3.0, VB 9.0)
- A nyelvbe ágyazott lekérdezéseket metódushívásokká alakítja a fordító



# A lekérdezések mintája

from-mal  
kezdődik

További from, join, let,  
where, vagy orderby

```
from id in source
{ from id in source /
  join id in source on expr equals expr [ into id ] |
  let id = expr |
  where condition |
  orderby ordering, ordering, ... }
select expr | group expr by key
[ into id query ]
```

Opcionálisan into-  
val folytatható

Végül select by  
group by

**A Query  
Expression Pattern**  
metódusai

Where

Select, SelectMany

OrderBy,  
OrderByDescending,  
ThenBy,  
ThenByDescending

GroupBy

Join, GroupJoin

Cast<T>

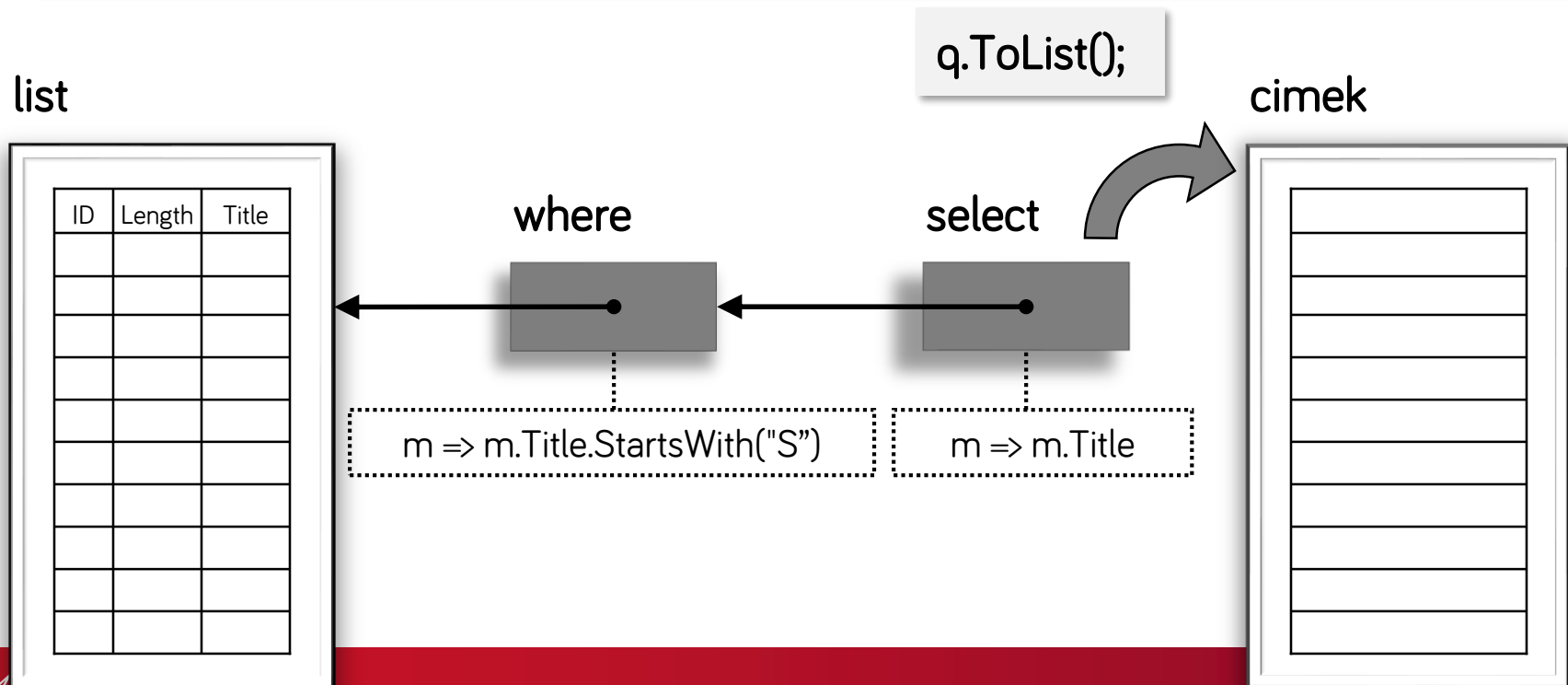
# Késleltetett kiértékelés mégegyszer

## Deferred Query Execution

```
var list = new List<Music> { ... };
```

```
var q = from m in list where m.Title.StartsWith( "S" ) select m.Title;
```

```
var q = list.Where( m => m.Title.StartsWith( "S" ) ) . Select( m => m.Title);
```



# ToArray

- A felsorolást tömbbé alakítja

```
double[] doubles = { 1.7, 2.3, 1.9, 4.1, 2.9 };  
var sortedDoubles = from d in doubles  
                    orderby d descending  
                    select d;  
double[] doublesArray = sortedDoubles.ToArray();
```

# ToDictionary

- A felsorolást kulcs-érték alapú szótárrá alakítja

```
var scoreRecords = new[] {  
    new {Name = "Alice", Score = 50},  
    new {Name = "Bob" , Score = 40},  
    new {Name = "Cathy", Score = 45}};  
  
var scoreRecordsDict =  
    scoreRecords.ToDictionary(sr => sr.Name);
```

# OfType

- Szűri a sorozatot típus alapján

```
object[] numbers =  
|   { null, 1.0, "two", 3, "four", 5, "six", 7.0 };  
  
var doubles = numbers.OfType<double>();
```

# Distinct

- Szűri a sorozatot: csak a különböző elemeket adja vissza

```
int[] factorsOf300 = { 2, 2, 3, 5, 5 };  
var uniqueFactors = factorsOf300.Distinct();
```

# FirstOrDefault

```
double?[] doubles = { 1.7, 2.3, 4.1, 1.9, 2.9 };  
double? num = doubles.FirstOrDefault( n => n <= 0.5 );  
if( num == null )  
    Console.WriteLine( "There are no small numbers." );  
else  
    Console.WriteLine("The first small number: {0}", num.Value);
```



# SelectMany 1

- Hierarchikus (fésűs) sorozatokból egyszintű sorozatot készít

```
List<Customer> customers = GetCustomerList();  
var orders = from c in customers  
              from o in c.Orders where o.Total < 500.00M  
              select new { c.CustomerID, o.OrderID, o.Total };
```

# SelectMany 2

- ,A' minden eleméhez hozzáveszi ,B' sorozat tagjait

```
int[] numbersA = { 0, 2, 4, 5, 6, 8, 9 };  
int[] numbersB = { 1, 3, 5, 7, 8 };  
var pairs = from a in numbersA  
            from b in numbersB where a < b  
            select new { a, b };  
Console.WriteLine("Pairs where a < b:");  
foreach (var pair in pairs)  
    Console.WriteLine("{0} is less than {1}", pair.a, pair.b);
```

# TakeWhile

- Addig fut, amíg a sorozat elemeire igaz a feltétel

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
var firstNumbersLessThan6 = numbers.TakeWhile(n => n < 6);  
Console.WriteLine("First numbers less than 6:");  
foreach (var n in firstNumbersLessThan6)  
    Console.WriteLine(n);
```

# SkipWhile

- Kihagyja azokat az elemeket amelyekre igaz a feltétel

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
var allButFirst3Numbers =  
    numbers.SkipWhile(n => n % 3 != 0);  
Console.WriteLine(  
    "All elements starting from " +  
    "first element divisible by 3:");  
foreach (var n in allButFirst3Numbers)  
    Console.WriteLine(n);
```

# Intersect (metszet)

- Metszetet készít és sorozatból

```
int[] numbersA = { 0, 2, 4, 5, 6, 8, 9 };  
int[] numbersB = { 1, 3, 5, 7, 8 };  
var commonNumbers = numbersA.Intersect(numbersB);  
Console.WriteLine("Common numbers shared by both arrays:");  
foreach (var n in commonNumbers)  
    Console.WriteLine(n);
```

# bool Any

- Igaz, ha a sorozat legalább egy elemére igaz a feltétel

```
int[] numbers = { -9, -4, -8, -3, -5, -2, -1, -6, -7 };  
bool zeroMatch = numbers.Any(n => n == 0);  
Console.WriteLine("There is a zero in the array: " + zeroMatch);
```

# Enumerable.Range, Repeat, ...

- Sorozatot generál

```
var numbers = from n in Enumerable.Range(100, 50)
               select new { Number = n,
                           OddEven = n % 2 == 1 ? "odd" : "even" };
foreach (var n in numbers)
    Console.WriteLine("The number {0} is {1}.",
        n.Number, n.OddEven);
```

# Average

- Átlagot számol a sorozat elemeire

```
string[] words = { "cherry", "apple", "blueberry" };  
double averageLength = words.Average(w => w.Length);  
Console.WriteLine(  
    "The average word length is {0} characters.", averageLength);
```



# Aggregate

- Agregál saját függvény szerint

```
int[] ia = { 1, 5, 2, 6, 7 };  
Console.WriteLine(  
    ia.Aggregate((seed, i) => seed += i));
```

```
string[] sa = { "aaa", "bbb", "ccc" };  
Console.WriteLine(  
    sa.Aggregate(  
        new StringBuilder(),  
        (sb, i) => sb.Append(i),  
        sb => sb.ToString()  
    ) );
```

# LINQ to ...



```
from m in list where m.Title.StartsWith( "S" ) select m.Title;
```

```
SELECT Title FROM Music WHERE Title LIKE 'S%'
```

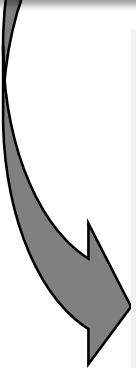
- Mi kell ahhoz, hogy elvégezzük az átalakítást?
  - > Leképezés: osztály – tábla, tag – oszlop, stb.
  - > A szűrési feltétel eredeti, értelmezhető formában
    - Nem lefordítva IL kódra
- **Expression<T>** : a kód adatként látszik
  - > ahol T a kifejezésre jellemző metódusreferencia

# Kifejezés fák – kódból adat

## Expression Trees

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

```
Expression<Predicate<Music>> expr = m => m.Title.StartsWith( "S" );
```



```
ParameterExpression p=  
    Expression.Parameter(typeof(Music),"m");  
expr=Expression.Lambda<Predicate<Music>>(  
    Expression.Call(  
        Expression.Property(p, GetMethod(Music.get_Title)),  
  
        GetMethod(string.StartsWith),  
        new Expression [ ] {Expression.Constant("S", typeof(string)) } ),  
  
    new ParameterExpression [ ] { p } );
```

# Expression

- Egy lambda kifejezés reprezentációja
- `System.Linq.Expressions.Expression<D>`, ahol D egy delegate típus
- Nem tartalmazhat statementet és értékadást
- A `Compile` metódus IL kódot készít

```
Func<int, int> del = x => x + 1;           // kód  
Expression<Func<int, int>> exp = x => x + 1; // adat
```

# Expression, Queryable, IQueryable

- A Queryable hasonló az Enumerable-höz, de:

```
public static IQueryable<TSource> Where<TSource>(
    this IQueryable<TSource> source,
    Expression<Func<TSource, bool>> predicate);
```

- IQueryable<T> interfész : IEnumerable<T>, és:
  - > Lehetővé teszi a kifejezések összefűzését is, például:

```
list . Where(...) . Take(...) . Select (...); // végül egy Expression lesz
```

- > A *Queryable* osztály bővítő metódusai segítségével a teljes kifejezés eljut az adatforrásig (list, DataTable, stb.)
  - > Végül a forrás alakítja át a kifejezést, például T-SQL-lé

# LINQ to Desktop Search

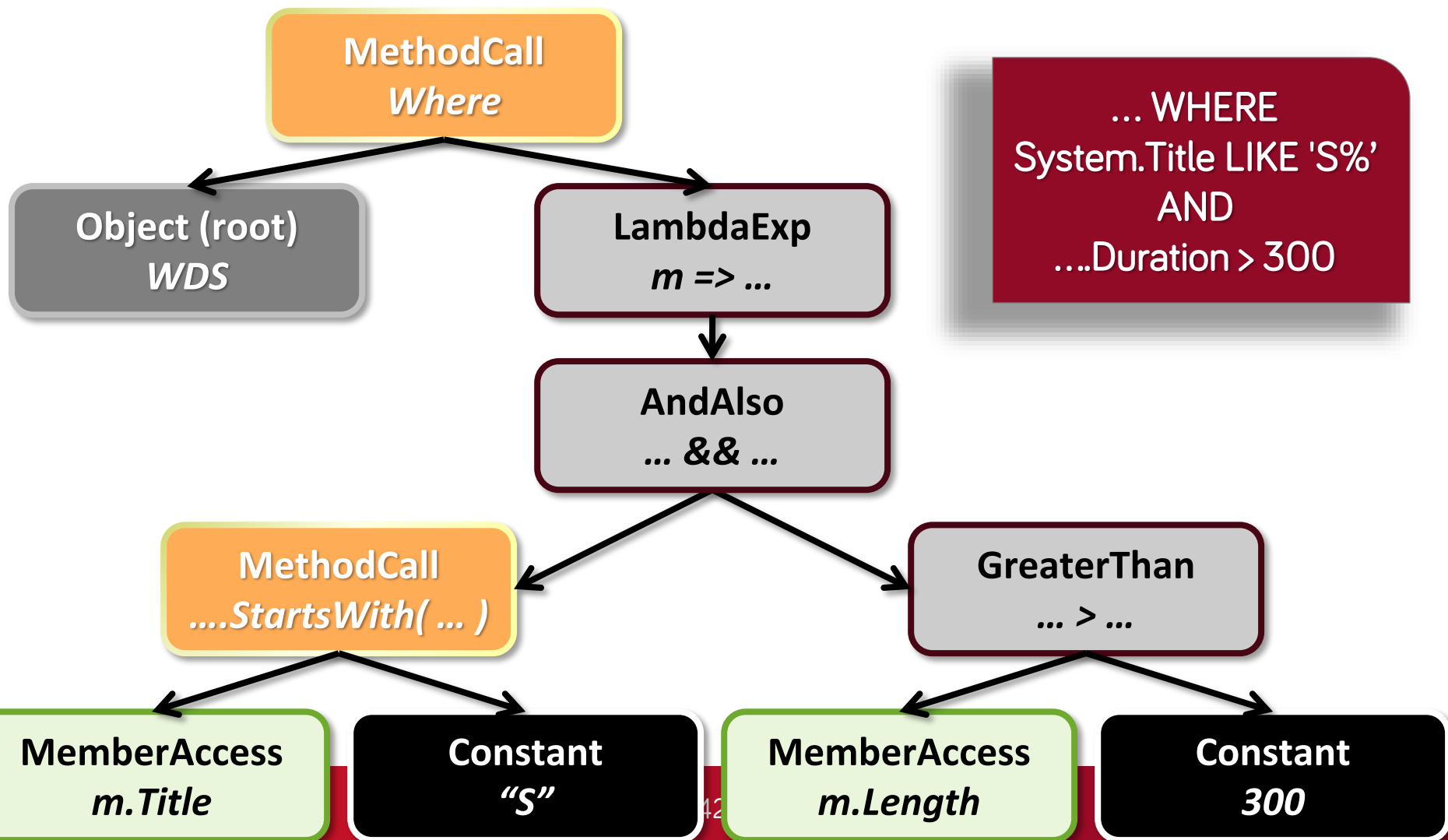
- OLEDB provideren keresztül, tipikus lekérdezés:

```
SELECT System.Title, System.Media.Duration FROM SystemIndex  
WHERE System.Title = 'Soul Bop'
```

1. A Queryable . Where ( ... ) átadja a szűrőt
  - Építjük és eltároljuk az egészet kifejezést
2. Enumerálásnál átalakítjuk WDS lekérdezéssé
3. Végrehajtjuk a lekérdezést OLEDB-n keresztül
  - Lezárjuk a kapcsolatot

# Fgv kifejezés fa. beírása

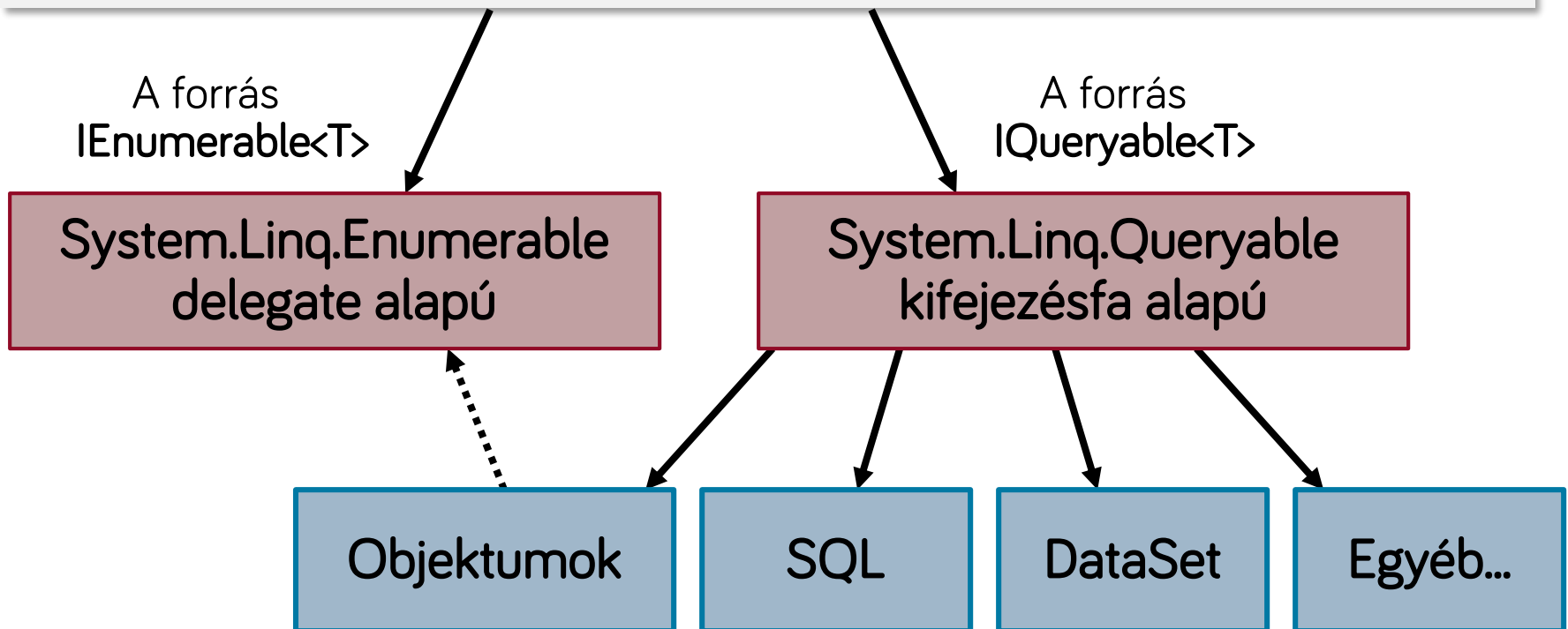
```
new WDS(). Where(  
    m => m.Title.StartsWith( "S" ) && m.Length > 300 );
```



# LINQ architektúra

from m in list where m.Title.StartsWith( "S" ) select m.Title;

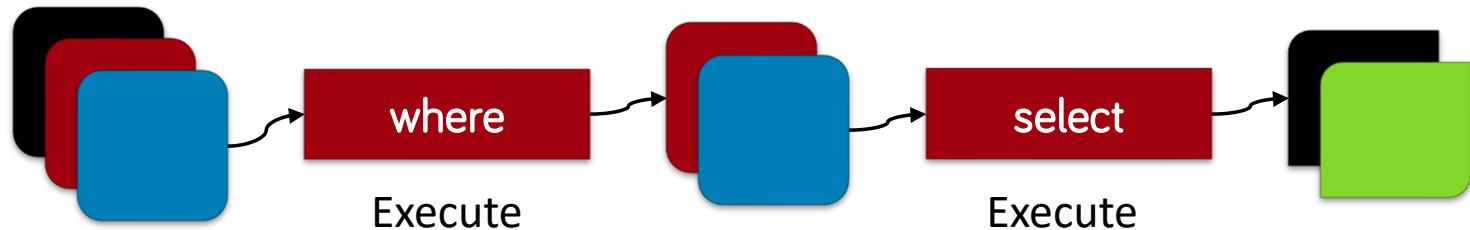
SELECT Title FROM Music WHERE Title LIKE 'S%'



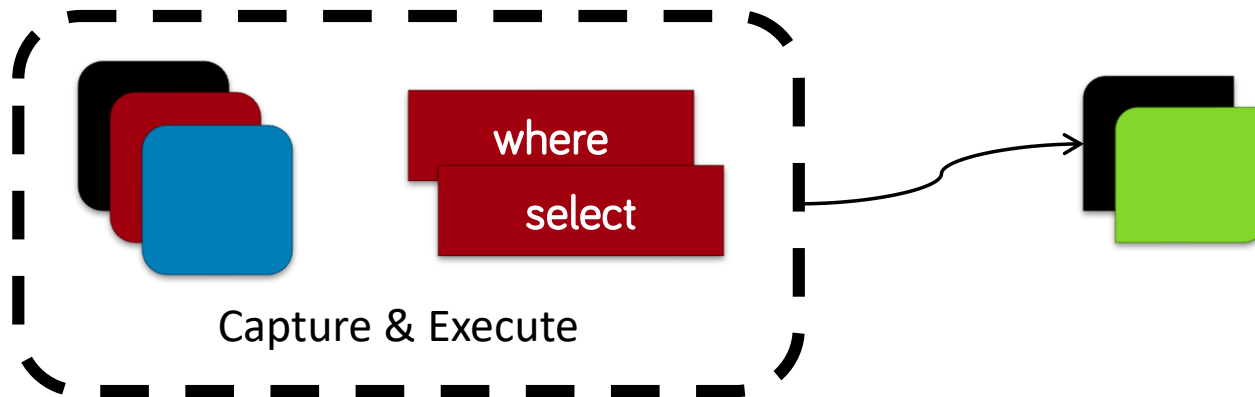


# IEnumerable<T> & IQueryable<T>

- IEnumerable – query executed piece by piece



- IQueryable – query executed in one go



# C# 3.0

Lekérdezések

```
var q = from m in list  
where m.Title.StartsWith( "S" )  
select new { m.Title, m.Length };
```

Implicit típusú  
lokális változó

Lambda  
kifejezések

```
var q = list  
.Where( m => m.Length.StartsWith( "S" ) )  
.Select( m => new { m.Title, m.Length } );
```

Bővítő  
metódusok

Névtelen  
típusok

Objektum  
inicializálók