Programmazione C#

Lezione 5

Lambda e LINQ

Gestione Errori

Input e Output di File

e-Lios
e-Linking online Systems

Sommario della Lezione

1. Lambda
2. LINQ
3. Gestione Errori
4. System.IO.File

Lambda

- Le espressioni Lambda sono funzioni anonime
- Sono ampiamente utilizzate per creare query LINQ
- Vengono utilizzate per creare funzioni/delegati (i delegati sono riferimenti a specifiche funzioni, utilizzabili come variabili)

```
(a, b) => a + b;          // Lambda expression con due parametri
() => someMethod();       // Lambda expression priva di parametri
(a) => someMethod(a);     // Parentesi sono opzionali
a => someMethod(a);
```

Lambda

- Es. Creo una funzione che prende in input un intero e ritorna un altro intero (l'ultimo parametro della funzione è l'output – i primi N sono gli input): **Func<int, int> func1 = x => x + 1;** (richiamo la funzione con il «comando» int risultato = func1.invoke(int par)

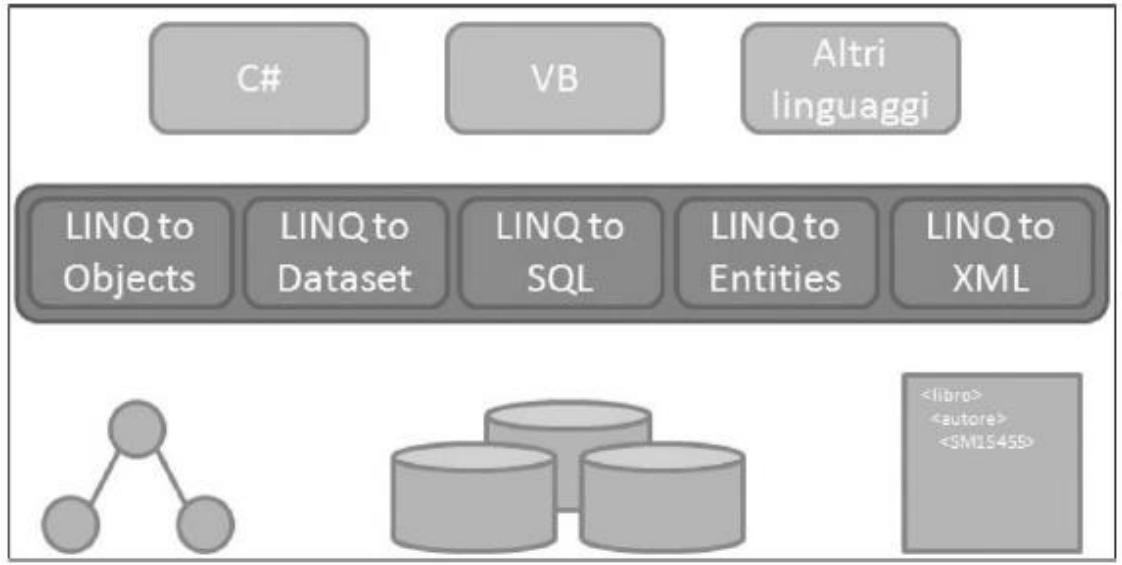
- Es. Creo una procedure che fa qualcosa (senza input e output): **Action func6 = () => Console.WriteLine();**

- Es. Creo un delegate (posso mettere I/O a mio piacimenti):

```
delegate int del(int i);
static void Main(string[] args)
{
    del myDelegate = x => x * x;
    int j = myDelegate(5); //j = 25
}
```

LINQ

- «Linguaggio» per effettuare ricerche/query all'interno di una qualsiasi struttura dati, direttamente dal linguaggio C#
- Alla base di LINQ c'è la classe Enumerable (quindi possiamo utilizzare query in tutte le strutture dati in C# che abbiamo visot List, Array ecc…
- Potenzialmente è possibile interrogare qualsiasi tipo di sorgente dati (Excel, JSON, ecc…)



LINQ

Aggregazione	`Aggregate, Average, Count, LongCount, Max, Min, Sum`
Concatenamento	`Concat, Zip`
Conversione	`Cast, ToArray, ToDictionary, ToList, ToLookup`
Elemento	`DefaultIfEmpty, ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault`
Generazione	`Empty, Range, Repeat,`
Intersezione	`GroupJoin, Join`
Ordinamento	`OrderBy, ThenBy, OrderByDescending, ThenByDescending, Reverse`
Partizionamento	`Skip, SkipWhile, Take, TakeWhile`
Proiezione	`Select, SelectMany`
Quantificazione	`All, Any, Contains`
Uguaglianza	`SequenceEqual`
Raggruppamento	`GroupBy`
Restrizione	`OfType, Where`
Insieme	`Distinct, Except, Union, Intersect`

```
var o = Ordini
   .Where(w => w.Id == 1)
   .Select(s => new { s.Data, s.Id });
```

Query Syntax (alternativa a LINQ)

Query Syntax Operator	Extension Method
From	
group by	Groupby
join	Join
let	
order by	Orderby
select	Select
where	Where

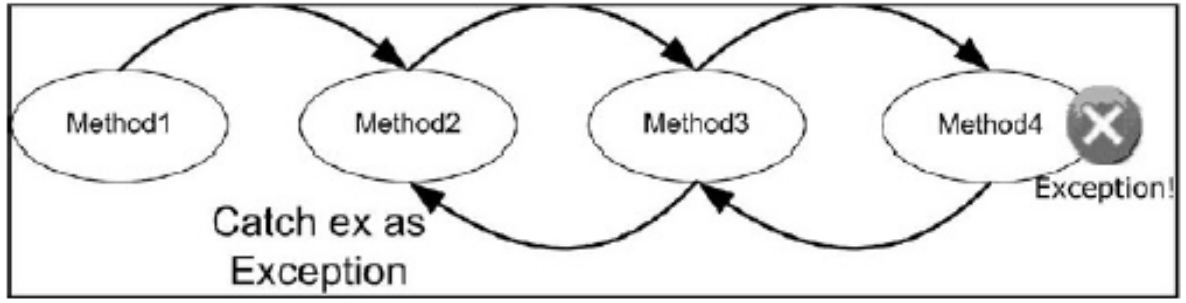
```
var result = from o in ordini
             where o.Id == 1
             select o;
```

Gestione Errori

```
try
{
  var result = Division(5, 0);
  // Questo codice non viene mai eseguito
  Console.WriteLine("Il risultato è {0}", result);
}
catch (Exception ex)
{
  Console.WriteLine("Si è verificato un errore");
}
```

Nome	Significato
`Message`	Contiene un messaggio di errore descrittivo dell'eccezione.
`Source`	Se non impostato diversamente, contiene il nome dell'assembly che ha sollevato l'eccezione.
`StackTrace`	Lo stack di chiamate al momento in cui si verifica l'eccezione.
`Data`	Una dictionary che consente di specificare ulteriori informazioni sull'errore verificatosi.
`HelpLink`	È una proprietà in cui è possibile memorizzare un URL per rimandare a una pagina di help.
`InnerException`	Un oggetto `Exception` può fungere da contenitore per un'ulteriore eccezione più specifica, che viene esposta tramite questa proprietà. Questo concetto sarà spiegato meglio nel corso del capitolo.
`TargetSite`	La definizione del metodo che ha sollevato l'eccezione.
`ToString()`	Questo metodo è ridefinito nella classe `System.Exception` e produce una stringa contenente la natura dell'eccezione, il messaggio e lo stack trace.

```
try
{
  var result = Division(5, 0);
  // Questo codice non viene mai eseguito
  Console.WriteLine("Il risultato è {0}", result);
}
catch (DivideByZeroException ex)
{
  Console.WriteLine("Errore: non si può dividere per zero");
}
catch (OutOfMemoryException ex)
{
  Console.WriteLine("Memoria terminata");
}
catch (Exception ex)
{
  Console.WriteLine("Si è verificato un errore generico");
}
```



Gestione Errori

```
StreamReader sr = null;
try
{
  sr = new StreamReader(@"c:\test.txt");
  string content = sr.ReadToEnd();
}
finally
{
  if (sr != null)
     sr.Close();
}
```

Esempio 7.9

```
public static void Main()
{
  var myObject = new DisposableObject();
  try
  {
    myObject.SomeMethod();
  }
  finally
  {
    myObject.Dispose();
  }
}
```

C# mette a disposizione una sintassi particolare, di fatto equivalente a quella precedente, utilizzando il blocco `using` come nell'esempio 7.10.

Esempio 7.10

```
using (var myObject = new DisposableObject())
{
  myObject.SomeMethod();
}
```

Gestione Errori

```
public static void SomeMethod(List<int> items)
{
  if (items == null)
  {
    // Si solleva l'eccezione indicando il nome dell'argomento non
    // correttamente valorizzato
    throw new ArgumentNullException("items");
  }
  // Codice del metodo qui
}
```

Input e Output su File

https://msdn.microsoft.com/it-it/library/system.io.file(v=vs.110).aspx

https://www.completecsharptutorial.com/basic/c-filestream-tutorial-with-programming-example/

Fine Lezione 5

DOMANDE?