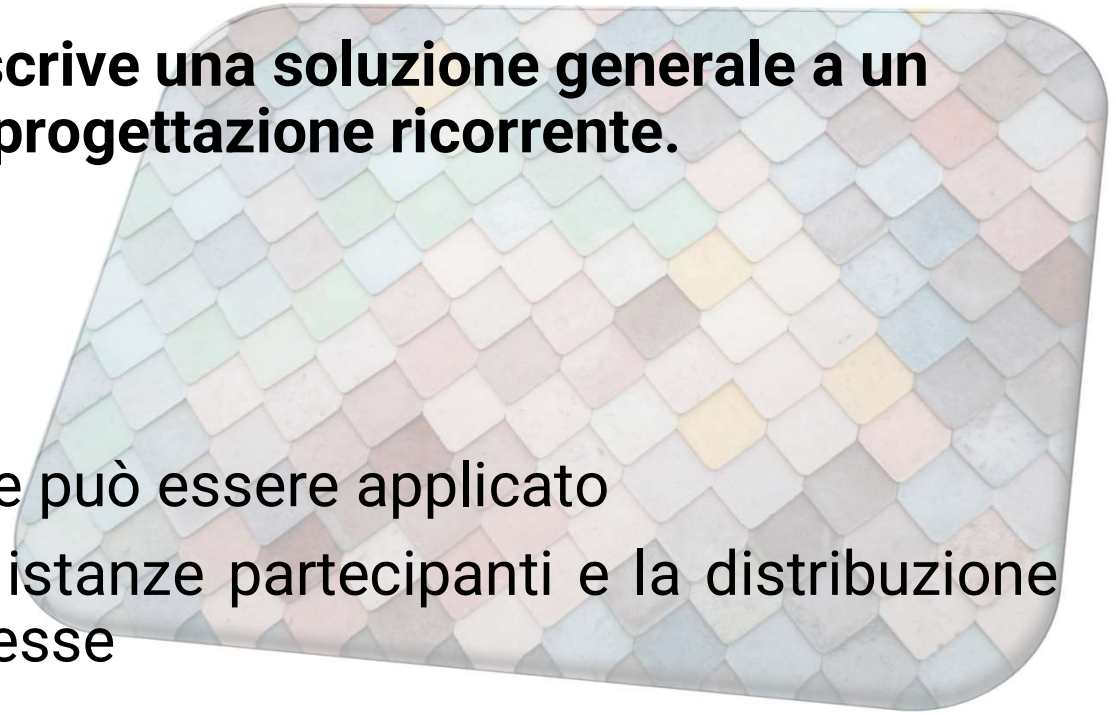
Design Patterns

Un design pattern descrive una soluzione generale a un problema di progettazione ricorrente.



Ogni pattern

- Possiede un nome
- Descrive quando e come può essere applicato
- Identifica le classi e le istanze partecipanti e la distribuzione delle responsabilità tra esse

Design Patterns

Un design pattern descrive una soluzione generale a un problema di progettazione ricorrente.

La principale raccolta di pattern è il volume

"Design Patterns: Elements of Reusable Object-Oriented Software"

del 1994, scritto da un gruppo di 4 sviluppatori noti col nome collettivo di "Gang of Four" (GoF).

Design Patterns

La "Gang of Four", ha identificato e descritto 23 design pattern classificati in tre gruppi principali:

- Creazionali (Creational)
  - **Factory**
  - Abstract Factory
  - Builder
  - Prototype
  - Singleton

Design Patterns

La "Gang of Four", ha identificato e descritto 23 design pattern classificati in tre gruppi principali:

- Strutturali (Structural)
  - Adapter
  - Bridge
  - Composite
  - **Decorator**
  - Façade
  - Flyweight
  - Proxy

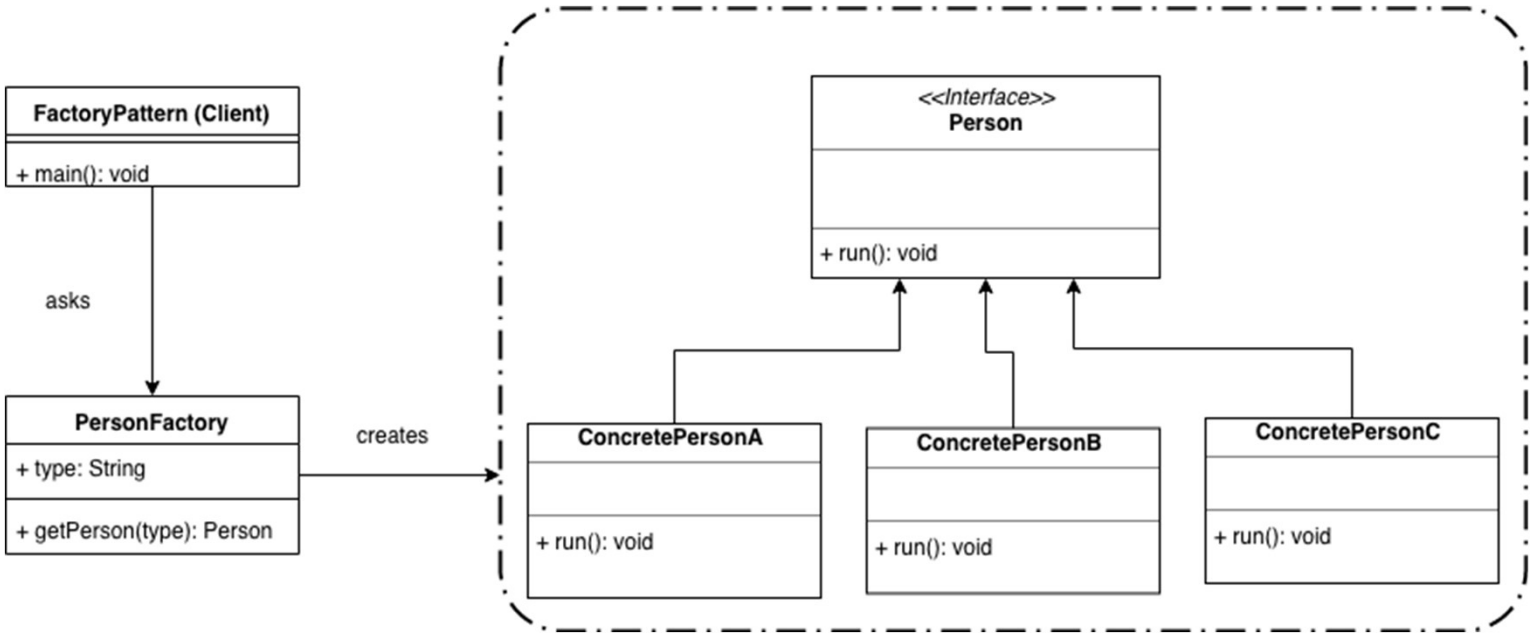
Design Patterns

La "Gang of Four", ha identificato e descritto 23 design pattern classificati in tre gruppi principali:

- Comportamentali (Behavioral)
  - **Chain of responsibility**
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template
  - Visitor

Design Patterns - Factory

Tipo: Creazionale



icubed

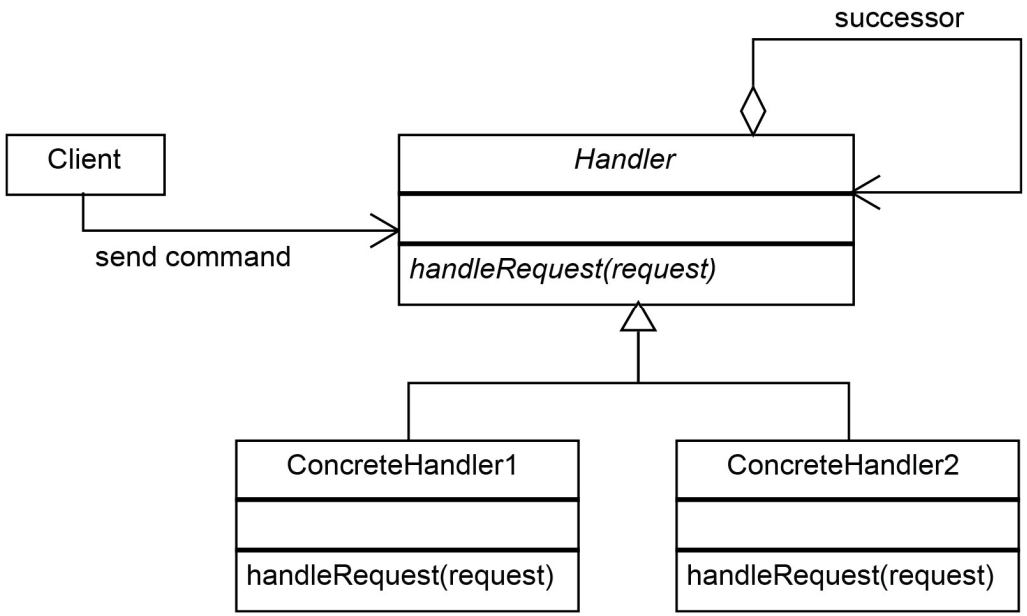
Demo

Design Pattern – Factory

icubed

Design Patterns - Chain of Responsibility

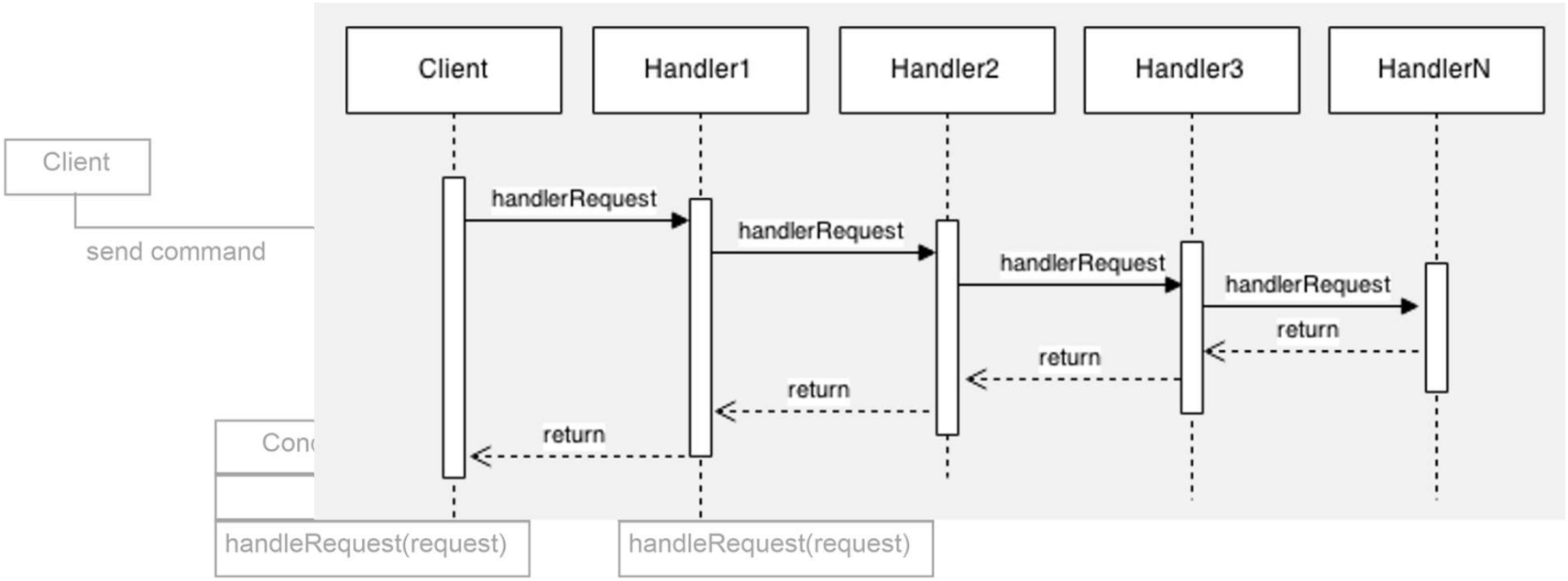
Tipo: Comportamentale



icubed

Design Patterns - Chain of Responsibility

Tipo: Comportamentale



icubed

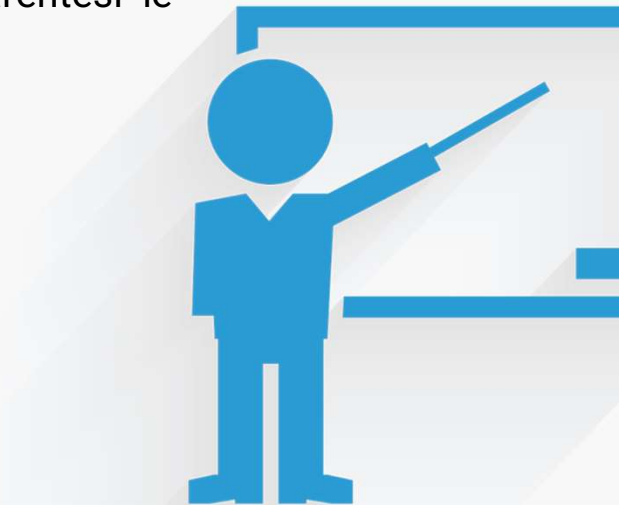
Demo

Design Pattern – Chain of Responsibility

icubed

Esercitazione 7

- Realizzare una classe **Dipendente** con i seguenti membri
  - (P) *Codice Dipendente* (string)
  - (P) *Nome* (stringa)
  - (P) *Cognome* (stringa)
  - (P) *Data di Assunzione* (DateTime)
  - (M) *StampaDettagli()* – stampa tutte le informazioni relative al dipendente (benefit inclusi)
- Implementare poi tramite il design pattern Decorator la possibilità per un Dipendente di avere uno o più dei seguenti benefit assegnati (tra parentesi le proprietà extra da assegnare):
  - **Posto Auto** (*Codice Posto Auto* (string))
  - **Assicurazione Sanitaria** (*Codice Cliente* (string))
  - **Ticket restaurant** (*Nr Tessera* (string), *Nr Ticket Mensili* (int))
  - **Auto Aziendale** (*Nr di Targa* (string), *Modello* (string))



icubed

Esercitazione 3

- Implementare una soluzione che consenta di creare diverse tipologie aziende a partire dal numero di dipendenti che dichiara di possedere.

Potremo avere ad esempio

- una piccola azienda fino a 20 dipendenti,
- media fino a 100 dipendenti,
- grande fino a 500 dipendenti,
- multinazionale da 500 in su.

icubed

Esercitazione 4

- Aggiungere all'esercizio precedente la possibilità di inserire una collezione di dipendenti.
- Ciascun dipendente sarà caratterizzato da: Nome (string), Cognome (string), Data di nascita (DateTime), Data di assunzione (DateTime), tasso di produttività (int), tasso di assenza (int)
- Dopo aver creato opportunamente un'azienda stabilire per ogni dipendente presente in essa un modulo in grado di attribuire premi ad impiegati sulla base di opportune opzioni di selezione.

Ciascuna opzione è mutualmente esclusiva (non tutte assieme). Sia possibile individuare le seguenti possibilità:

- Opzione **PRODUTTIVITA'** - Impiegati con età < Y e produttività > W%;
- Opzione **PRESENZA** - Impiegati con età < Y e tasso di assenza annuo < Z%;
- Opzione **ANZIANITA' DI SERVIZIO** - Impiegati con un'anzianità di servizio > 43 anni
- Opzione **BENESSERE COLLETTIVO** - Impiegati con una produttività >= W;

- Siano Y, W, Z parametri di input

icubed

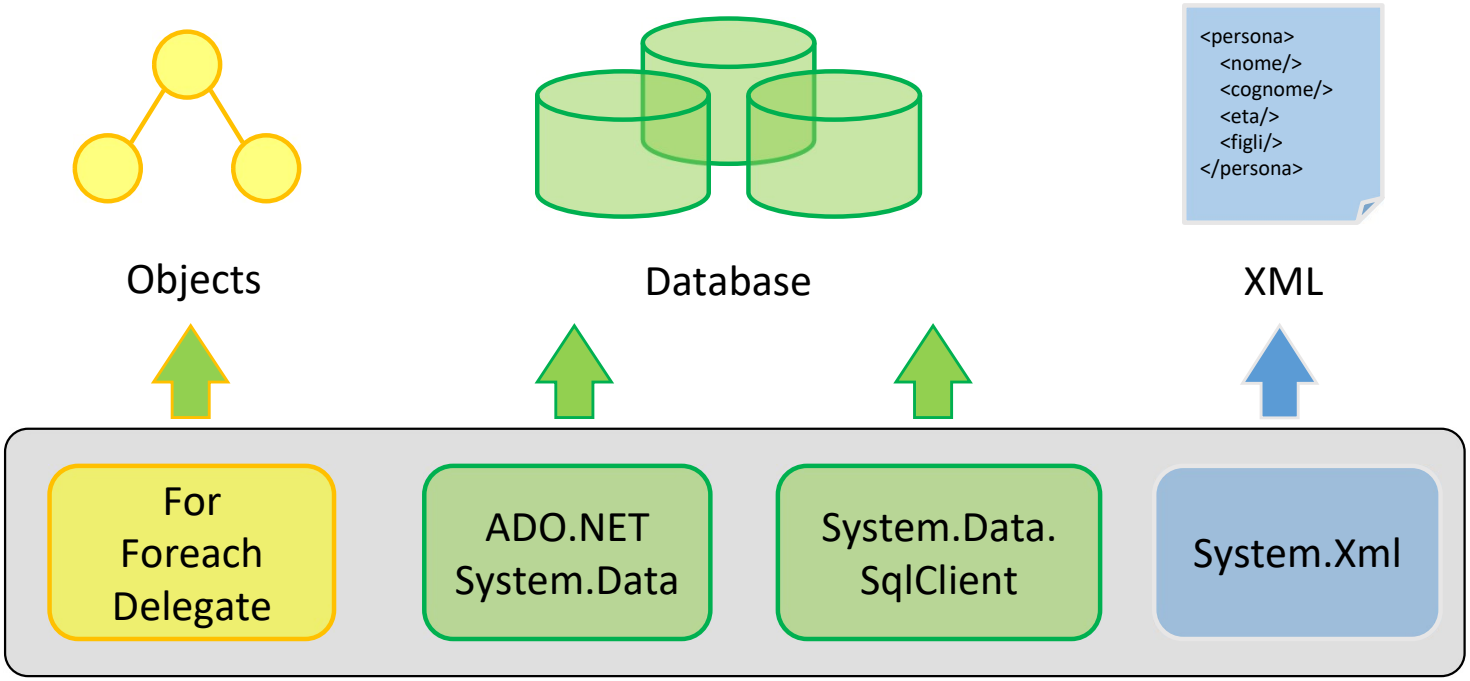
Cosa è LINQ

LINQ sta per **L**anguage **IN**tegrated **Q**uery

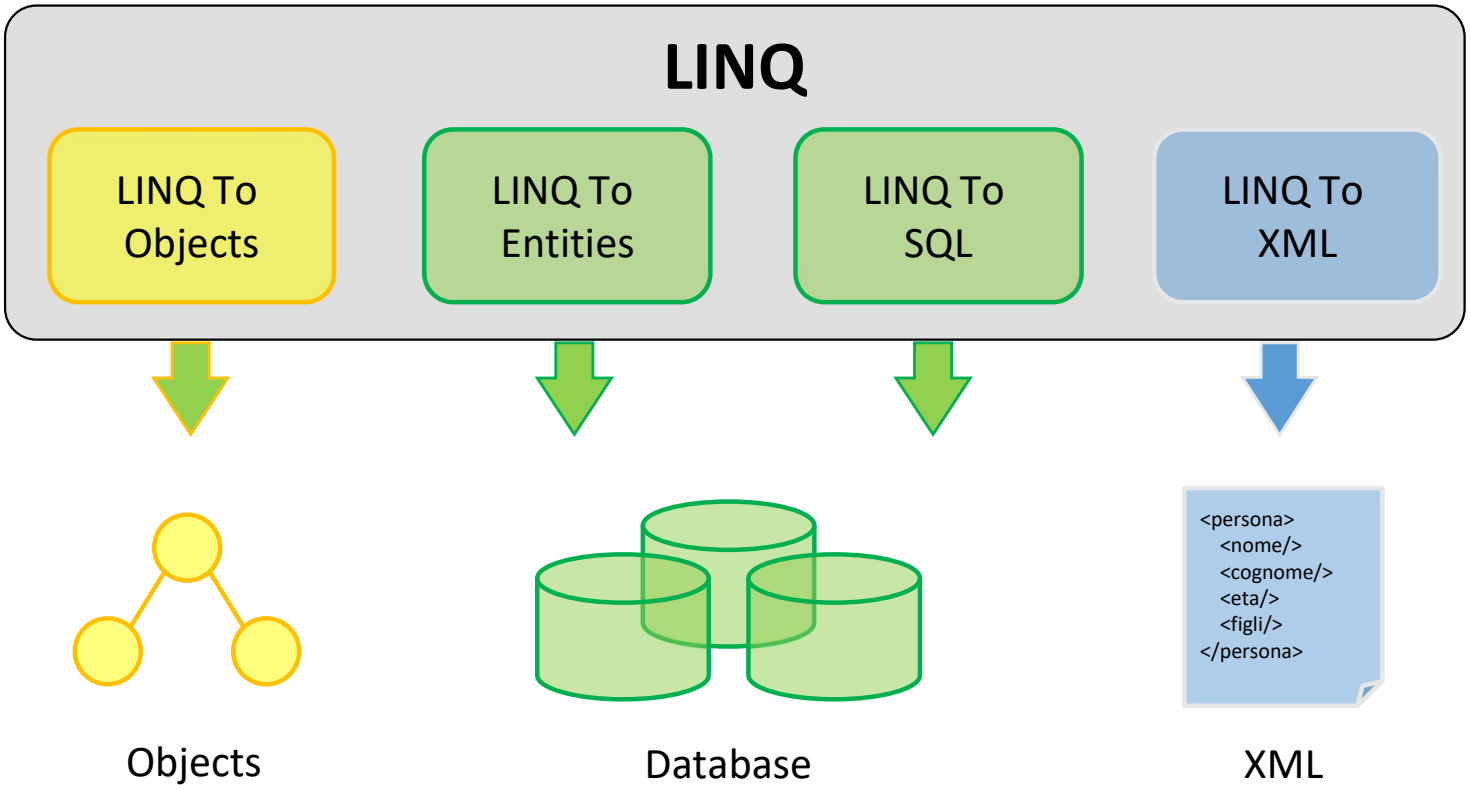
LINQ è un framework per eseguire interrogazioni su sorgenti dati all'interno del linguaggio.

icubed

Accesso ai dati senza LINQ



Accesso ai dati con LINQ



icubed

LINQ – Query Expression

Query standard per accedere a:

- Oggetti
- Dati relazionali
- Dati XML

Più di **50 operatori predefiniti**

- Aggregazione, Proiezione, Join, Partizionamento, Ordinamento

Sintassi e operatori **simile a SQL**

icubed

LINQ – Anatomia di una Query

- Due modelli di sintassi
  - Query
  - Lambda Expression
- Possibilità di utilizzare combinate
- Non modificano la sequenza originale

Query Lambda

- Più controllo e flessibilità
- Gli operatori sono applicati in sequenza
- ***Select*** può essere opzionale

icubed

LINQ – Operatori

- Utilizzo di **operatori Standard**
- Libreria di riferimento **System.Linq**
- Utilizzo con tipi **IEnumerable<T>**
- Pieno supporto ed integrazione con Intellisense

icubed

Operatori

Tipologia	Operatore
Projection	Select, SelectMany, (From)
Ricerca	Where
Ordinamento	OrderBy, OrderByDescending, Reverse, ThenBy, ThenByDescending
Raggruppamento	GroupBy
Aggregazione	Count, LongCount, Sum, Min, Max, Average, Aggregate,
Paginazione	Take, TakeWhile, Skip, SkipWhile
Insiemistica	Distinct, Union, Intersect, Except
Generazione	Range, Repeat, Empty
Condizionali	Any, All, Contains
Altri	Last, LastOrDefault, ElementAt, ElementAtOrDefault, First, FirstOrDefault, Single, SingleOrDefault, SequenceEqual, DefaultIfEmpty

LINQ - Operatori

- Reference: **System.Linq**
- Estende le funzionalità di **IEnumerable<T>** e **IQueryable<T>**

```
public static class Enumerable
{
  static public Ienumerable<Tsource> Where(this IEnumerable<TSource> source, Func<TSource, bool> predicate)
    …
    …
    …
}
```

```
public static class Enumerable
{
    public static TSource Aggregate<TSource>(this IEnumerable<TSource> source, Func<TS
    public static TAccumulate Aggregate<TSource, TAccumulate>(this IEnumerable<TSource
    public static TResult Aggregate<TSource, TAccumulate, TResult>(this IEnumerable<TS
    public static bool All<TSource>(this IEnumerable<TSource> source, Func<TSource, bo
    public static bool Any<TSource>(this IEnumerable<TSource> source);
    public static bool Any<TSource>(this IEnumerable<TSource> source, Func<TSource, bo
    public static IEnumerable<TSource> AsEnumerable<TSource>(this IEnumerable<TSource>
    public static decimal? Average(this IEnumerable<decimal?> source);
    public static decimal Average(this IEnumerable<decimal> source);
    public static double? Average(this IEnumerable<double?> source);
    public static double Average(this IEnumerable<double> source);
    public static float? Average(this IEnumerable<float?> source);
    public static float Average(this IEnumerable<float> source);
    public static double? Average(this IEnumerable<int?> source);
    public static double Average(this IEnumerable<int> source);
    public static double? Average(this IEnumerable<long?> source);
    public static double Average(this IEnumerable<long> source);
```

icubed

LINQ

Sostituzione di **foreach** con query Linq

Query Deferred

- Query expression come se dati
- Composizione di query

Definizione

```
IEnumerable<Employee> employee =
  from p in employees
  where p.Name == "Scott"
  select p.Name;
```

Esecuzione

```
foreach (var emp in employee)
{
  …
  …
}
```

icubed

LINQ – Lambda Expression

```
IEnumerable<string> filteredList = cities.Where(StartsWithL);


public bool StartsWithL(string name)
{
  return name.StartWith("L");
}
```

```
  IEnumerable<string> filteredList = cities.Where(name => name.StartWith("L"));
```

icubed

LINQ – Lambda Expression

- Rappresentazione sintetica
- (input-parameters) => expression

- Utilizzo dell'operatore **=>**
  - **A sinistra:** firma della funzione
  - **A destra:** statement della funzioni

icubed

LINQ – Lambda Expression

Parametri ed i tipi opzionali

- Non sono richieste parametri, quando sono impliciti

Logica negli statement

- Utilizzo di variabili locali
- Attenzione: le lambda expression dovrebbero essere tenute più semplici possibile

```
IEnumerable<string> filteredList =
cities.Where((string s) =>
  {
    string temp = s.ToLower();
    return temp.StartsWith("L");
  }
);
```

icubed

LINQ – Lambda Expression

Lambda Expression usano particolari **delegate:**

- **Action<T>**
  - Non ritornano un valore
- **Func<T>** e **Expression<T>**
  - Ritornano un valore

```
Func<int, int> square = x => x * x;
Func<int, int, int> mult = (x, y) => x * y;
Action<int> print = x => Console.WriteLine(x);
print(square(mult(3, 5)));
```

icubed

LINQ – Query Expression

- **Extension Methods**
- **Lambda expressions**
  - Delegati
  - Expression Trees
- **Query Expression**

icubed

LINQ – Query Expression

- Extension Methods
- Lambda expressions
- **Query Expression**

```
IEnumerable<string> filterCities =
  from city in cities
  where city.StartsWith("L") && city.Lenght <15
  orderby city
  select city;
```

icubed

Demo

LinQ – Lambda Expression VS Query Expression

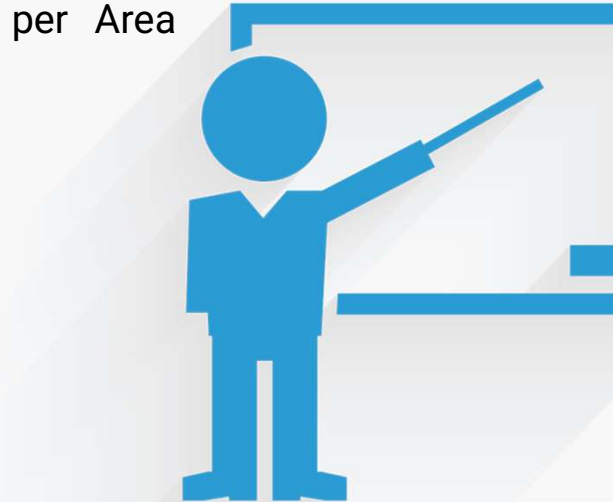
icubed

Esercitazione 5

Riprendere l'esercizio precedente:

- Creare una List di almeno 10 Shape
- Scrivere le query LINQ per
  - Elencare tutte le Shape con un'Area superiore a 20
  - Elencare tutte le Shape con il Nome che inizi per 'A'
  - Elencate solo i Nomi delle Shape
  - Elencare tutte le Shape in ordine Alfabetico per Nome e poi per Area decrescente

Le query devono essere scritte in entrambe le sintassi (Extension Methods e Query Expression)



icubed