

# C# e .NET (for Java dummies)

Maura Cerioli

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris

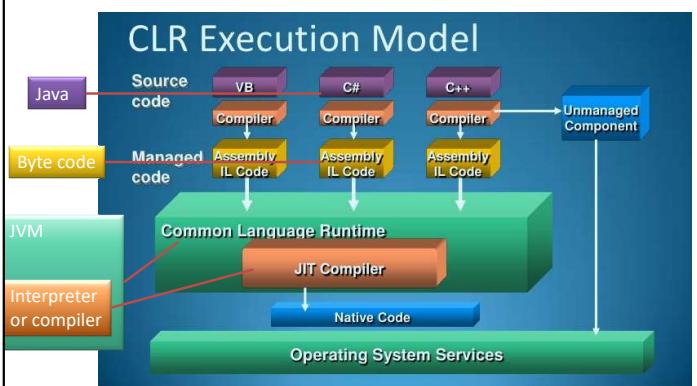


Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## .NET e C# - Preistoria

- C# gioco di parole “see sharp”
  - introdotto nel 2002
  - sintassi scopiazzata da C++ ma più simile a Java (che avrebbero voluto usare)



Dibris



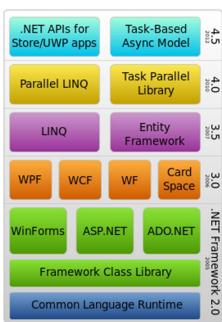
Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

2

## Evoluzione

Librerie/framework per ogni esigenza

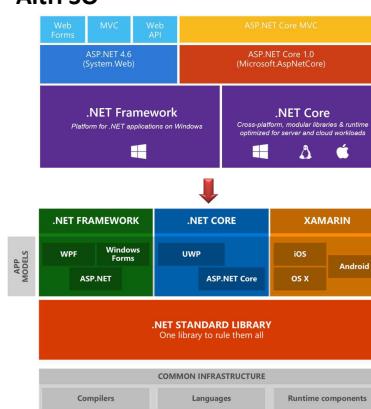


Dibris



Dibris

Altri SO



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Stato attuale

- .NET Framework 4.8 ultima release di .NET Framework
- .NET Core 3.1 ultima release di .NET Core
- .NET 5 prima versione unificata
  - evoluzione di .NET Core
  - open source (grasso gruppo di sviluppatori pagati da MS)



Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

4

## Oggi...

- Goal: rendervi “operativi” velocemente
- Panoramica dei costrutti di base
  - conoscete già i concetti da Java
  - alcuni costrutti sembrano (quasi) uguali e lo sono
    - se confondete la sintassi, il compilatore ve lo dice
  - alcuni costrutti sembrano uguali ma NON lo sono
    - se vi confondete, sono <BIP> ☺
- Vediamo direttamente demo
  - lascio le slide come riferimento per chi non c'era/dormiva

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

5

## DEMO

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

6

## Analogie

- Object-oriented, sintassi “alla C” (graffe, int, if, for, ...)
- Ereditarietà singola per le classi, ma multipla per le interfacce. Supporta i generici (metodi, classi e interfacce)
- Costruttori per inizializzare gli oggetti
- Oggetti creati con **new**, garbage collected
- Gestione degli errori tramite meccanismo delle eccezioni
- Compilato per un'unica architettura (virtuale)

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

7

## Statement

- if, while, do while, for, {}, expression-stmt (espressione seguita da “;”), dichiarazioni di tipo sono (quasi) uguali
- **foreach(T v in exp)** che in Java è **for(T v : exp)**
- **goto** (permesso in certi casi), sostanzialmente sostituisce break/continue etichettati
- **synchronized** è diventato **lock**
- Lo switch
  - supporta il “fall-through” SOLO se il caso è completamente vuoto
    - aka più casi con lo stesso body
    - se non vuoto dove esserci esplicito statement per uscire (break, return, throw, goto...)
  - ha **default** case
  - i casi possono essere molto più complicati che semplici costanti (pattern matching e case guard)
  - esiste anche una **espressione** switch
  - parliamo più a fondo dello switch in seguito

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

8

## L'immancabile Ciao mondo...

```
namespace ConsoleApplication {
    using System;
    using static System.Console;
    public class Program {
        public static void Main(/* string[] args */) {
            WriteLine("ciao mondo");
            ReadLine();
        }
    }
}
```

Permette di usare le classi dichiarate in System senza qualificarle pienamente  
Console invece di System.Console

meccanismo analogo per le classi  
permette di accedere ai membri statici  
senza esplicitare il nome della classe  
WriteLine invece di Console.WriteLine

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

9

## Modificatori di accesso

public

internal protected

protected

internal

Java

C#

- public
- protected
- (friendly)
- private

private protected

private

public

protected – classe e sottoclassi

---

internal – stesso assembly (dll o exe)

internal protected – protected o internal

private protected – protected e internal (sic)

private

Assembly

=

unità di compilazione e distribuzione

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

10

## var

- Solo per dichiarare variabili locali
- Il tipo (inferito) è quello dell'inizializzatore
- Esempio:  
`int i = 0;`  
`var i = 0; // equivalenti, finché non cambia inizializz.`
- È chiaramente molto comoda con classi con nome lungo...  
`List<List<string>> lls = new List<List<string>>();`  
`var lls = new List<List<string>>();`
- Non solo per i pigri, essenziale in alcuni casi

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

11

Potrebbe esserci  
private, internal,  
protected...

## Proprietà (Properties)

Potrebbe essere  
espressione più  
complicata

- Evoluzione di campo privato + getter e/o setter
  - che non hanno (quindi) ragione di esistere nei vostri programmi C#
- Metodi get/set che invocati attraverso la sintassi di accesso a campo, per esempio, `exp.P = 3;`
  - Usate per caching, lazy loading, ecc.
  - È possibile specificare solo il getter o il setter
  - Per il setter, `value` indica il valore destro dell'assegnazione
  - Si può imporre che il setter sia invocabile solo durante la creazione dell'oggetto (inizializzazione)
- Al client "sembrano" accessi a campo: è ragionevole usarle solo se getter/setter efficienti
- In quanto metodi, possono essere virtual, abstract, comparire nelle interfacce
- Uno dei due
  - può avere un modificatore di accesso più restrittivo (e.g. private set)
  - mancare del tutto (e.g. solo get per property di sola lettura)

```
private double _a;
public double A {
    get => _a; //get{ return _a; }
    set => _a = value; //set{ _a = value; }
}
public double A1 { get; set; }

var c = new C();
c.A = 42.0;// _a=42.0
Console.WriteLine(++c.A); // stampa 43 e _a==43
```

```
private string s;
public string S1 {
    get => s;// get{ return s; }
    init => s = value;//init{ s = value; }
}
c.S = "x";// ERRORE
var c = new C{S="x",A=5};

public C(string z) { s = z; }
```

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

12

## Uso ovvio

```

private double _a;
public double A {
    get => _a; //get { return _a; }
    set => _a = value; //set { _a = value; }
}
public double A1 { get; set; }

public class MyTime { /* */
    private double _seconds;
    public double Hours {
        get { return _seconds / 3600; }
        set { if (value < 0 || value > 24)
                throw new ArgumentOutOfRangeException();
              _seconds = value * 3600; }
    }
    MyTime t = new MyTime();
    t.Hours = 3;
    ++t.Hours; // !!!
    Console.WriteLine(t.Hours); // stampa 4
}

```

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

13

## Auto-Properties

```
public string S { get; init; } = "puffo";
```

- Come una property, ma il campo nascosto è generato automaticamente dal compilatore
  - non se ne conosce il nome
 

```
public class C{
    public int Z { get; private set; }
}
```
- Con ReSharper la trasformazione fra una versione e l'altra richiede solo un ALT Return ...
- Le Auto-Properties si possono inizializzare
  - sintassi analoga ai campi
 

```
public int I { get; set; } = 37;
```
  - come per i campi
    - inizializzazioni eseguite nell'ordine in cui sono scritte
    - nell'espressione di inizializzazione non si può far riferimento al this

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

14

## Properties senza campo - Esempio

```

public class Person {
    public int YearOfBirth { get; set; }
    public int Age {
        get {
            return DateTime.Now.Year - this.YearOfBirth;
        }
    }
    public int Age =>
        DateTime.Now.Year - this.YearOfBirth;
// ....
var p = new Person { YearOfBirth = 1973 };
Console.WriteLine(p.Age);

```

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

15

## Scorciatoie varie...

```

var array = new[] { "ciao", "mondo" };
var anA = new A(3) { G=3 };
var anotherA = new A { F=5, G=3 };
var ls = new List<string>()
    { "qui", "quo", "qua" };
var d = new Dictionary<int, string>()
    { { 1, "pippo" }, { 2, "pluto" } };
var d1 = new Dictionary<int, string>()
    { [1] = "pippo", [2] = "pluto" };

```

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

16

## Ereditarietà e Interfacce

### Sintassi Java

```
class C1 extends C2 implements I1, I2, I3 {
    ...
    C1() { super(...) ... }
    C1(int x) { this(...) ... }
```

**interface** I **extends** I1, I2, I3 ...

### Sintassi C#

```
class C1 : C2, I1, I2, I3 {
    ...
    C1() : base(...) { ... }
    C1(int x) : this(...) { ... }
```

**interface** I : I1, I2, I3 ...

- Le interfacce contengono solo metodi (astratti e pubblici, come in Java)
  - property (vedremo) sembrano campi ma sono coppie di metodi
  - i metodi possono avere implementazione di default
- Per convenzione, iniziano tutte con I: IPippo, IPluto, ecc.

## Metodi virtuali o no?

```
public class C0 { public void M(){Console.WriteLine("C0");} }
public class C1: C0 { public new void M() {Console.WriteLine("C1");} }
```

C0  
C1

Java

C#

C0  
C1

Default = virtual  
Ridichiarare in sottoclassi = overriding  
Dispatch dinamico

Default = final/sealed  
Ridichiarare in sottoclassi = hiding  
Dispatch statico

Per avere comportamento *alla Java* in C#

- virtual** al momento della (prima dichiarazione)
- override** quando si ridichiara

```
public class C0 { public virtual void M(){Console.WriteLine("C0");} }
public class C1: C0 { public override void M() {Console.WriteLine("C1");} }
```

C0 c0 = new C0();
c0.M();
c0 = new C1();
c0.M();

C#

C0  
C1

## Implementazione esplicita di interfacce

```
public interface I1 { void M(); }
public interface I2 { void M(); }
public class C: I1, I2 { public void M(){Console.WriteLine("M");} }
    C c = new C();
    c.M();
```

- Legale sia in Java che in C#
- M() implementa sia I1.M() che I2.M()

- In C# possiamo avere implementazioni diverse per le due interfacce (**implementazione esplicita** di interfacce)

```
public class C: I1, I2 {
    void I1.M() { Console.WriteLine("I1"); }
    void I2.M() { Console.WriteLine("I2"); }
}
```

- Non ci va nessun modificatore
- il metodo risulta *inaccessibile attraverso il tipo C*
- Si usa il cast per decidere quale versione

c.M();  
((I1) c).M();  
((I2) c).M();

## Metodi locali

Se è naturale per implementare un metodo M introdurre metodi ausiliari che servono solo in M, dove è ragionevole dichiararli?

### Java

- privati, dentro la stessa classe di M

### C#

- privati, dentro a M stesso
  - come convenzione di stile si mettono in fondo
- vantaggio 1: non sono visibili agli altri metodi di M
- vantaggio 2: chi deve capire la classe sa che non ha bisogno di leggerli se non vuole entrare nei dettagli di M

## RunTime Type Identification/Cast

### Java

- exp instanceof T
- (T)exp
- Object t = exp;  
t instanceof T ? (T)exp : (T)null

### C#

- exp is T
- (T)exp
- exp as T

## Costanti

### Java

- Campi final static
- Semantica "buffa" dei campi final static: se tipo primitivo e costante a tempo di compilazione, sull'accesso viene fatto inlining

### C#

- Modificatore readonly analogo a final
  - ok per istanza o static
  - non viene fatto inlining
- Modificatore const, usabile con tipi numerici ed enumerazioni, indica una costante a tempo di compilazione
  - implicitamente static
  - viene fatto inlining

## Eccezioni

- Entrambi hanno try/catch/finally e una gerarchia di eccezioni con radice (java.lang/System).Exception
  - membri analoghi StackTrace/InnerException
- Ma C# non ha le **eccezioni controllate**, quindi non ha la clausola throws nelle intestazioni dei metodi (e non vi costringe a dichiarare/catturare)
  - in teoria, è un grosso passo indietro
  - in pratica, rende le cose più gestibili

## Gestione Eccezioni Come ben sapete...



- le eccezioni si catturano per gestirle
    - catch (Exception e)  
{Console.WriteLine(e.Message);}
  - eventualmente si risolvono
    - catch (Exception e)  
{throw new Exception(e.Message);}
  - magari cambiamo tipo
    - catch (Exception e)  
{throw new MyException(e.Message);}
- ~~catch (Exception e)  
{Console.WriteLine(e.Message);}~~
- ~~catch (Exception e)  
{throw new Exception(e.Message);}~~
- ~~catch (Exception e)  
{throw new MyException(e.Message);}~~
- 
- OK solo per console applications Altrimenti
    - catch (Exception e)  
{  
Debug.WriteLine(e.Message);  
throw;  
}
  - Così si conserva history Oppure usando gli exception filter:
    - private static bool NotifyAndContinue(Exception e){  
/\*do logging/ console writing\*/  
return false;  
}
    - ...
    - catch (Exception e) when (NotifyAndContinue(e))  
{  
}
  - Volendo cambiare tipo
    - catch (Exception e)  
{throw new MyException(e);}

## Enum

### Java

- Permette usi "acrobatici"
- Un enum è un tipo reference

### C#

- Più simili a quelli di C/C++
  - Un modo per definire delle costanti di tipo integrale
- ```
internal enum Color {
    Red,
    Green,
    Blue
}
...
Color c = Color.Green;
```

## Rilascio automatico delle risorse: statement using (in Java 7, try)

L'argomento deve essere un IDisposable (esp. o dich.):

```
using (var pippo = new StreamReader(@"c:\bla"))
    // usa pippo...
```

È come scrivere:

```
{
    var pippo = new StreamReader(@"c:\bla");
    try {
        // usa pippo...
    } finally {
        if (pippo!=null)
            pippo.Dispose();
    }
}
```

Da C# 8 anche variable declaration using:

```
using var pippo = new StreamReader(@"c:\bla");
    // usa pippo...
    // quando pippo esce dallo scope viene invocata la dispose
```

## Altre similitudini

- Non ci sono funzioni/variabili globali (al più, membri statici)
  - In C# gli inizializzatori statici vanno dichiarati come "costruttori" statici senza parametri
  - statement globali..lasciamo perdere
- Numero variabile di parametri: **params T []** invece di **T...**
- Gli array possono essere "jagged", per es:  

```
int[][] myArray = new int[2][];
myArray[0] = new int[3];
myArray[1] = new int[9];

```

  - Ma ci sono anche gli array bidimensionali  

```
int[,] myArray = new int[,]{{1,2,3},{4,5,6}}
```

## Altre caratteristiche

- Il nome del file sorgente è irrilevante
- Annotazioni/metadati/threading, ci sono anche in C#, ma ne parleremo in seguito
- Inner/nested classes: C# supporta solo le nested
  - grazie ad altre caratteristiche non si sente la mancanza delle inner

## C#: tutto qui?

- Sinteticamente...**NO**
- Molti altri costrutti interessanti
  - a ogni versione qualcosa di nuovo/migliorato
  - possono o meno avere controparte in Java...dipende anche da quali versioni confrontate
- Ne vediamo alcuni ma restano esclusi molti
  - ad esempio tutti quelli legati alla concorrenza o alla manipolazione diretta della memoria
- Per TAP basta quello che vediamo, ma per diventare programmatore professionista dovete studiare tutte le feature
  - aka: RTFM

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

29

## C#: nameof

Giovanni Lagorio & Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## nameof

- `nameof(...)` restituisce il nome (non qualificato) di ...

– ... deve essere una variabile, un membro, un parametro...

```
public class C{
    public void Log(string f, string msg){}
    public int I { get; } = 42;
    private C anotherC =null;
    public void SomeMethod(int n){
        Console.WriteLine(nameof(n));// n
        var x = "puffo";
        Console.WriteLine(nameof(x));// x
        Console.WriteLine("{0} == {1}",nameof(I),I); // I == 42
        Console.WriteLine(nameof(this.anotherC.I)); // I
        Console.WriteLine("{0}.{1}", nameof(anotherC),
            nameof(this.anotherC.I)); // anotherC.I
        Log(nameof(SomeMethod), "exiting method");
    }
}
```

- differenza fondamentale: Refactoring!

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

2

## nameof – Uso motivante

- Problema comune: verifica della correttezza dei parametri
  - uniformità di gestione
  - centralità di evoluzione → **incapsulare in metodi**
  - leggibilità del codice

```
internal static void CheckString(string argument, string argumentName, int min, int max){
    CheckNotNull(argument, argumentName);
    var length = argument.Length;
    if (length < min || length > max) throw new ArgumentException(argumentName, "...");
}
```

```
public void CreateUser(string username, string password){
    CheckString(username, "username", /*...*/);
    CheckString(password, "password", /*...*/);
}
```

- Ridenominare i parametri di `CreateUser` richiede di cambiare anche la stringa nella chiamata di `CheckString`

```
public void CreateUser(string username, string password){
    CheckString(username, nameof(username), /*...*/);
    CheckString(password, nameof(password), /*...*/);
}
```

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

3

# C# Type system and Generics

Maura Cerioli

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Tipi primitivi/valore

### Java

- boolean
- char
- int, long, float, double, ...

### C#

- bool
- char
- idem, più le versioni unsigned:  
uint, ulong, ...
- **ma:**
  - int è un alias di Int32, ulong di UInt64, ecc.
  - object è un alias di Object
  - string è un alias di String

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

2

variabili memorizzano  
riferimenti a oggetti  
assegnazione crea aliasing

## C# Type System

### C# (in realtà: .NET)

- Unica radice del type system  
`System.Object`
- Con metodi
  - `ToString`
  - `Equals`
  - `GetHashCode`
  - `(Finalize)`
- `System` è un **namespace** (in Java è una classe di `java.lang`)
  - si dichiarano con **namespace** e si usano, con **using**

⚠ Alcune linee rappresentano inheritance (relazione is-a, e.g. `String` con `Object`)  
Alcune linee rappresentano sottotipo con conversione (e.g. `Int32` con `Object`)

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/>

Dibris



Dibris

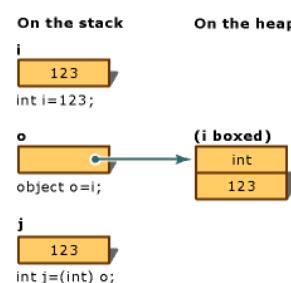
Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

3

## Boxing/unboxing

### • Conversioni fra tipi reference e valori

- Inutili (e “pericolose”) avendo tipi generici decenti
  - ... ma all’inizio non c’erano



Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

4

## Generici

- Simili a quelli Java (ma molto più efficienti)
  - Istanziabili anche su tipi primitivi
- Si possono mettere dei vincoli sugli argomenti tramite la clausola **where**

```
public class C<T> where T : new() {  
    public T Foo() {  
        return new T();  
    }  
}
```

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

5

## Generici, vincoli:

- **new()** : l'argomento deve essere un tipo istanziabile, col costruttore di default
- **AType** : l'argomento deve essere sottotipo di AType
  - AType può essere il nome di un tipo del sistema o di un altro parametro del generico
- **struct**: l'argomento deve essere un value type
- **class**: l'argomento deve essere un tipo reference (classe, interfaccia, delegate o array)

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

6

## Generici, perché vincolare gli argomenti?

- Per scrivere del codice che sfrutta quei vincoli
  - Per esempio, senza (il vincolo) new() su un argomento T, non è possibile fare **new T()**
- Il valore di default di un tipo si indica **default(T)** (equivale a null per i reference, 0 per i numerici, ecc.)
- Se si devono vincolare più argomenti, servono più clausole where:  
`class Test<T, U>  
where U : struct  
where T : E, new() { }`

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

7

## Co/Contro-varianza

- Un parametro di tipo di un'interfaccia (o delegate) generica può essere:
  - **out / covariante**, ovvero può essere usato solo come tipo di ritorno. Questo implica che  $I<Pere> \leq I<Frutta>$  se **Pere**  $\leq_{ref}$  **Frutta**
  - **in / controvariante**, ovvero può essere usato come tipo per i parametri (non out o ref) e nei vincoli. Questo implica che  $I<Frutta> \leq I<Pere>$
  - **invariante**, quando non ha modificatori
- Permette delle conversioni molto naturali (e safe)
- Parecchi dettagli tecnici su cui sorvoliamo

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

8

## Esempi

```
interface IEnumerator<out T> {  
    bool MoveNext();  
    T Current { get; }  
}  
  
interface IComparer<in T> {  
    int Compare(T left, T right);  
}  
  
delegate TResult Func<in TArg, out TResult>(TArg arg);
```

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

9

## Conversioni con varianza

- Un tipo  $T<A_1, \dots, A_n>$  è convertibile a  $T<B_1, \dots, B_n>$  se  $T$  è un'interfaccia o un delegate dichiarato con parametri  $X_1, \dots, X_n$  e, per ciascuno:
  - $X_i$  è covariante e  $A_i$  si converte (come identità o conversione reference implicita a)  $B_i$
  - $X_i$  è contravariante e  $B_i$  si converte (come identità o conversione reference implicita a)  $A_i$
  - $X_i$  è invariante e  $A_i = B_i$
- Esempio:  
`delegate T Func<in T1, in T2, out T>(T1 x, T2 y);  
Pere, Mele ≤ Frutta, Vegetali  
– Func<Frutta, Vegetali, Pere> ≤ Func<Pere, Mele, Frutta>`

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

10

## C#: varie ed eventuali

Giovanni Lagorio & Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2020-21

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

Memento  
default by value

## Passaggio per riferimento

- Usando `in` o `ref` o `out` nella signature

|                   | <code>in</code> | <code>ref</code> | <code>out</code> |
|-------------------|-----------------|------------------|------------------|
| pre-inizializzata | obbligatorio    | obbligatorio     | facoltativo      |
| modificata        | vietato         | facoltativo      | obbligatorio     |

verificata dal  
compilatore

- Overloading di metodi distinti solo dal tipo di passaggio
  - permesso fra valore/riferimento (in dubbio prevale per valore)
  - vietato fra diversi tipi di riferimento (a livello di virtual machine esiste solo ref)
- `ref` e `out` devono essere usate anche al momento della chiamata
  - il client deve essere consapevole delle potenziali modifiche ai parametri
- il parametro attuale può essere dichiarato direttamente nella clausola `out`
- `in` ha senso solo quando il passaggio per valore andrebbe bene, ma per efficienza è meglio non fare copia
  - cioè il tipo di dato occupa significativamente più spazio di un puntatore
  - la keyword può essere usata al momento della chiamata per risolvere overloading

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

2

## Passaggio per riferimento: esempi

```
static void Increment(ref int i) { ++i; }           int i = 1;
static void Read(out int i) {                         Increment(ref i);
    Console.WriteLine("Inserisci...");               Read(out i);
    i = Console.Read();                            Read(out var x);
static void Read(ref int i) /*errore              var a = new A();
    static*/{}                                     M(a); // by value
struct A /*molti campi*/{}                      M(in a); // in
static void M(in A z)
    {Console.WriteLine ("in"); }
static void M(A z)
    {Console.WriteLine("by value");}

```

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

3

## Argomenti opzionali e *con nome*

Si possono definire parametri opzionali fornendo un valore di default

public void M(int x, int y = 5, int z = 7) {...}

- Solo dopo quelli obbligatori

Esempi d'uso

M(1, 2, 3); // ok

M(1, 2); // equivalente a M(1, 2, 7)

M(1); // equivalente a M(1, 5, 7)

M(1, 7); // ERRORE

- Evita di definire tanti metodi in overloading in cui mancano i parametri opzionali

public void M(int x, int y, int z) {...}

public void M(int x, int y) {int z = 7;...}

public void M(int x) {int y = 5; int z = 7;...}

- Si possono indicare esplicitamente solo alcuni parametri opzionali usando la notazione con nome

- Esempi d'uso

M(1, z: 3); // z per nome indispensabile

M(x: 1, z: 3); // x e z per nome anche se x non è indispensabile

M(z: 3, x: 1); // rovesciando l'ordine...

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

4

## Higher-Order Types

Giovanni Lagorio & Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2023-24

Dibris

## Tipi Delegate

- Esempio di uso: definire un metodo che filtra un array di elementi di tipo T
  - argomenti: l'array da filtrare e il filtro  tipo del filtro T->bool
  - risultato un nuovo array con solo gli elementi che soddisfano il filtro
- Dichiarazione di un tipo delegate **custom**  
public delegate int D(int a, int b); // int x int -> int
  - sintassi richiama dichiarazione di metodo
  - nessun modo di aggiungere membri a D
  - i membri del tipo sono definiti e implementati dal framework
    - costruttori
    - metodi per invocazione sincrona e asincrona
- Dichiarazione di un tipo delegate definisce una classe che estende **System.Delegate**
- Equivalenza di tipi delegate per nome
  - public delegate int Foo(int a, int b);  
oggetti di tipo D e Foo incompatibili 

Dibris

## Tipi Delegate predefiniti

- **Azioni**, metodi void che si aspettano T1...Tn: **Action**, **Action<T>**, **Action<T1, T2>**, ...
- **Funzioni** da T1...Tn in TResult: **Func<TResult>**, **Func<T, TResult>**, **Func<T1, T2, TResult>**, ...
- **Predicati**: **Predicate<T>**
  - logicamente equivalente a **Func<T, bool>**
  - incompatibili come tipi
  - esiste solo unario
- Difficilmente si ha bisogno di definire altri tipi delegate
  - indispensabile se il tipo funzionale ha parametri non solo per valore standard (ref, in, out, params)
  - utile per veicolare intento dando un nome (es. delegate Filter)

Dibris

## Uso di delegate

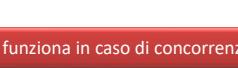
- Istanziazione come tutti gli oggetti con new
  - Argomento del costruttore il metodo da incapsulare

```
var f0 = new Func<int, int, int> ( Math.Max);  
var g0 = new Func<int, int> (42.CompareTo);  
Func<int, int, int> f1 = Math.Max;  
Func<int, int > g1 = 42.CompareTo;
```


- Invocazione come se fossero metodi

```
var x1 = f0(42,42);  
var x2 = g1(42);
```
- **Warning**: i delegate essendo oggetti possono essere nulli

```
if (f != null){var w = f(2, 3);}
```



Soluzione: null-conditional operator

Dibris

## Null-conditional operators

- Se exp è un'espressione che può valutarsi in null  
`exp.BlaBla();`  
rischia di sollevare **NullReferenceException**
- Soluzione ovvia: premetto controllo se exp è null
- Problemi
  - Non funziona in caso di concorrenza
  - Diminuisce leggibilità del codice, ad esempio  
`var x = M().S; ⇔ if (null != M()) x = M().S; else x = null;`  
(anche in questi casi semplici)
- Soluzione: Null-conditional operator `exp?.BlaBla()`
  - equivale a `IsNullOrEmpty(exp) ? null : exp.BlaBla()` in assenza di concorrenza
  - valuta exp una sola volta ⇒ corretto in caso di concorrenza
- Non si può usare se exp definisce una funzione da invocare, ovvero se dopo il ? andrebbero gli argomenti della chiamata
  - Bisogna usare esplicitamente il metodo che rappresenta l'invocazione di funzione
  - Ad esempio se d è un delegate si usa `d?.Invoke(/*...*/);`  
Invece di `d?/*...*/;`

Dibris

## Metodi anonimi

- In molti casi il metodo usato per creare il delegate non fa parte dell'interfaccia naturale
  - non rappresenta un concetto significativo
  - non ha un nome sensate
  - *sporcherrebbe* l'interfaccia
- Serve un meccanismo per creare al volo metodi anonimi
  - Nelle versioni vecchie *anonymous delegate*  
`Func<int, int, int> f =  
delegate(int a, int b) { return a + b; };`
  - Versione moderna: lambda

Dibris

## Lambda Expressions (Closures)

- Assimilabili a  $\lambda$ -astrazioni del  $\lambda$ -calcolo, ovvero definizioni di funzioni “inline”
- Esempi:
  - $(int x, int y) \Rightarrow x + y$   
rappresenta la funzione che somma x e y
  - $a \Rightarrow a * a$   
rappresenta la funzione quadrato
  - $(x, y) \Rightarrow \{ \text{if } (x < y) \text{return } x; \text{else return } y; \}$   
rappresenta la funzione che restituisce il minimo
- Il tipo dei parametri **può** essere inferito
  - se il parametro è uno solo, non servono le parentesi
- $\lambda$ -expression assegnate (o passate come argomento) a un tipo delegate si comportano come metodi anonimi
  - Le  $\lambda$  **NON** sono delegate, ma sono convertibili in delegate

Dibris

## Esempi di closure

- In un delegate si possono usare tutte le cose (variabili, metodi...) visibili dove viene dichiarato
  - quindi anche in una lambda
- Esempi
 

```
public static void Foo(Action<int> bar) {bar(21);}

public static void Main(string[] args) {
    int k = 0;
    Action<int> f = delegate(int a) { k += a; };
    Action<int> g = a => k += a;
    Foo(f);
    Foo(g);
    Console.WriteLine(k); // 42
    Console.ReadLine();
}
```
- Utile ad esempio per side-effect in call-back

Dibris

7

8

## Closure: comportamento inatteso(?)

```
public static void Main(string[] args){
    var arg = 0;
    const int max = 10;

    Separiamo le chiamate a M
    dalle asserzioni sul risultato

    for (int i = 0; i < max; i++) {
        Debug.Assert(M()(i));
        ++arg;
    }

    Func<int, bool> M() {
        return x => x == arg;
    }
}

public static void Main(string[] args){
    var arg = 0;
    const int max = 10;
    var res = new Func<int, bool>[10];
    for (int i = 0; i < max; i++) {
        res[i] = M();
        ++arg;
    }
    //arg = 0;

    for (int i = 0; i < max; i++) {
        Debug.Assert(!res[i](i));
        ++arg;
    }
}

Func<int, bool> M() {
    return x => x == arg;
}
```

Dibris

## Lambda-like syntax

- Sintassi molto simile alle lambda per
  - dichiarazione di proprietà
 

```
public int Z {get { return I+K; }}
```
  - dichiarazione di metodi
 

```
public int Pippo(int x){return Z + x;}
public int Pippo(int x) => Z + x;
```
- Grosse limitazioni
  - leggibilità
  - dopo la freccia ci può essere solo un’expression statement
    - NO concatenazione
    - NO {...}

Dibris

9

10

## Combinare delegate

- In molti casi serve un ulteriore passo di astrazione
  - invocare in un solo passaggio un numero arbitrario di metodi ignoti al momento di compilazione
  - esempio applicare un filtro a un'immagine
    - filtro ottenuto come combinazione di filtri base scelti dall'utente
- Ogni (oggetto) delegate ha una **invocation-list** associata
  - se creato con costruttore un unico elemento: il parametro della **new**
  - se creato da operazioni per combinare delegate può averne molti
- Invocazione di un (oggetto) delegate = sequenza di invocazione dei metodi della sua invocation-list
  - in ordine di inserimento
  - se un metodo solleva un'eccezione, i metodi successivi non verranno invocati
  - il risultato è il risultato dell'ultimo metodo chiamato

Dibris

## Operazioni sull'Invocation-List

I delegate possono essere

combinati

- **D operator +(D x, D y)**
  - $x + y$  nuovo delegate che invoca tutti i metodi di  $x$ , seguiti da quelli di  $y$
  - se  $x [y]$  è null il risultato è  $y [x]$  (non una copia)
  - $x+m$  assegna a  $x$  un nuovo delegate che invoca tutti i metodi di  $x$  e poi  $m$
- **D operator -(D x, D y)**
  - $x - y$  nuovo delegate che invoca tutti i metodi di  $x$  che non compaiono in  $y$
  - $x-m$  assegna a  $x$  un nuovo delegate che invoca tutti i metodi di  $x$  tranne  $m$

confrontati

- **bool operator==(System.Delegate x, System.Delegate y)**
  - $x == y$  se se "puntano" la stessa lista di metodi
- **bool operator!=(System.Delegate x, System.Delegate y)**

Attenzione quando si toglie l'ultimo elemento dall'invocation list il delegate diventa null

Dibris

12

Richiami/Anticipazioni da FIS  
Approfondimenti operativi

## Unit Testing

Giovanni Lagorio & Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris

## Definizione

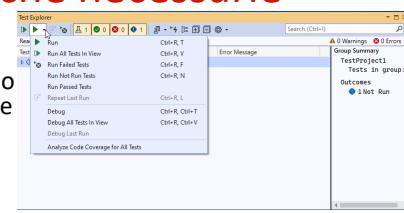
- Uno **unit test** è un frammento di codice (tipicamente, **un metodo**) che **invoca un altro** frammento di codice e **poi verifica** se delle condizioni sono soddisfatte
  - Se non lo sono, lo unit-test fallisce
- Una "unità" (unit) è un metodo o una funzione, spesso chiamata **System Under Test (SUT)**
  - E' molto importante che la unit sia piccola: quando un test fallisce è immediato sapere *dov'è* il problema

Dibris

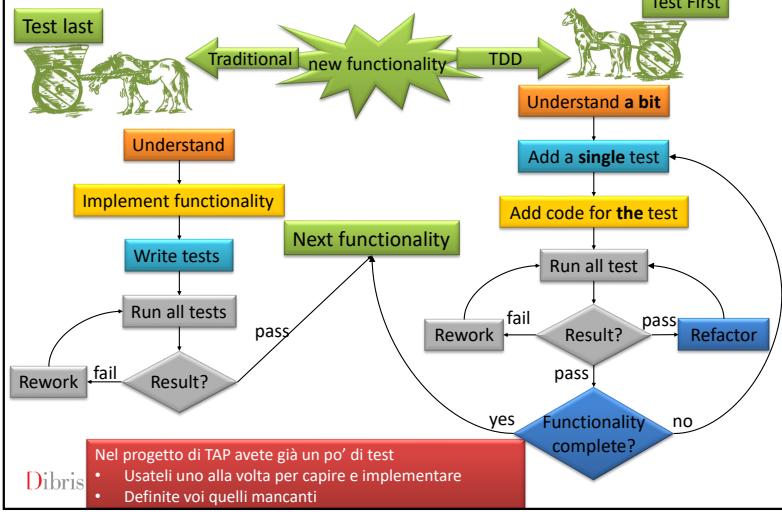
## Caratteristiche necessarie

- Automatico e ripetibile
  - Deve bastare un “click” (o anche meno!) per testare tutto il sistema
  - Si usano framework e test-runner
- Indipendente
  - Non deve richiedere particolari configurazioni
- Veloce (non è un integration-test!)
  - Da implementare
  - Da eseguire
- Usabile da chiunque
  - È documentazione in forma di codice

Dibris



## Test Driven Development



## Test Driven Development

- Classicamente, prima si scrive il codice e poi lo si testa
  - anche se i test li scrive in parallelo il test team
- Nel TDD prima si scrivono gli unit-test, che ovviamente falliranno, poi si scriverà il codice per farli passare
- Il livello di granularità nel TDD è più fine
  - in TDD più iterazioni per una singola funzionalità
  - classicamente una funzionalità/gruppo di funzionalità tutto assieme
- I test vengono poi usati per il regression testing e facilitano (=danno fiducia per) il refactoring
- Sistemato il codice (corretto e pulito), si ricomincia scrivendo nuovi test...

Dibris

## Refactoring

- E' la modifica di un frammento di codice che *non* ne modifica la funzionalità
- Serve a ottenere codice più “pulito”, facile da leggere, modificare e debuggare
- Un esempio classico di refactoring è rinominare un metodo
  - Gli IDE supportano sempre di più il refactoring
  - Per VS, ReSharper potenzia questo e altri aspetti

Dibris

## Testing Framework

- Ne esistono tanti
  - VS supporta MSTest (nativo), NUnit, xUnit.Net
- Useremo NUnit, [www.nunit.org](http://www.nunit.org)
  - Fa parte della famiglia xUnit (JUnit per Java, CppUnit per C++ e così via)
  - Free
  - Ben documentato
  - Supportato da
    - ReSharper
    - VS via NUnit Test Adapter
  - Stile di asserzioni più usabile (gusto personale)

Dibris

## Come si usa NUnit?

### Add a new project

Recent project templates



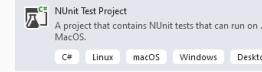
Search for templates (Alt+S) ▾ Clear all

C# Windows Test



MSTest Test Project  
A project that contains MSTest unit tests that can run on .NET Core on Windows, Linux and Mac OS.

C# Linux macOS Windows Test



Unit Test Project (.NET Framework)  
A project that contains MSTest unit tests.

C# Windows Test



xUnit Test Project  
A project that contains xUnit.net tests that can run on .NET Core on Windows, Linux and Mac OS.

C# Linux macOS Windows Test

[TestFixture] e [Test] sono attributi  
nel namespace NUnit.Framework

### Aggiungendo un progetto di test alla solution vengono

- importati i pacchetti  
necessari
- introdotte le reference
- inseriti gli using
  - in un file global using

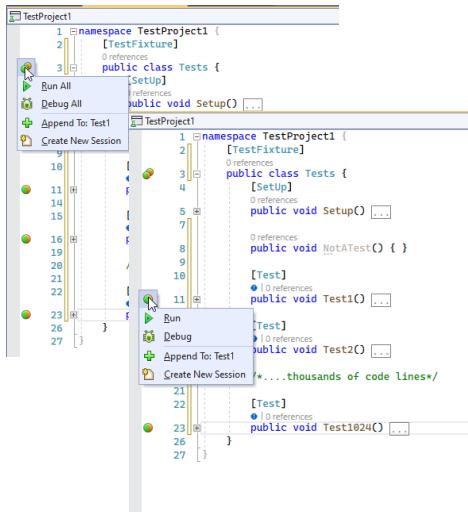
### Per definire test in questo progetto annotare

- le classi (di Test) con **[TestFixture]**  
(opzionale)
- i metodi con **[Test]**

Dibris

## ReShaper e VS riconoscono gli Unit Test

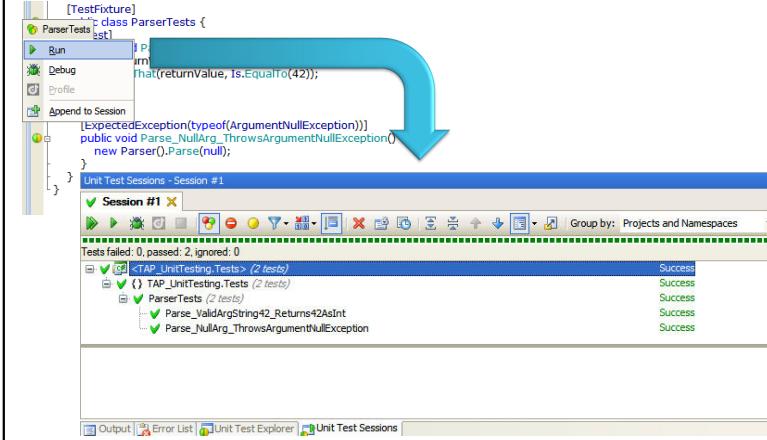
- Le icone colorate  
sul bordo indicano  
gli unit-test, che  
possono essere  
eseguiti da VS
- Per il momento non  
preoccupiamoci del  
corpo dei metodi...



Dibris

## Test Runner di ReSharper

Anche Visual Studio e NUnit hanno un Test Runner, se preferite usate uno di quelli



Dibris

## Requisiti per usare un test runner

Poiché il test runner deve poter invocare i test in modo totalmente automatico:

- Le classi e i metodi di test devono essere **public**
- Le classi di test devono avere un costruttore senza parametri ed essere annotate **[TestFixture]**
- I metodi di test devono essere **void**, senza parametri e annotati **[Test]**
  - (vedremo in seguito che non è vero al 100%)

Attributi: ne parliamo in seguito

Generano metadati usabili per manipolare ed eseguire codice

Dibris

## Struttura di un (metodo di) unit test

1. Creare e inizializzare gli oggetti coinvolti **Setup**
2. Invocare il metodo da testare **Call under test**
3. Asserire che qualcosa sia come ci si aspetta **Assert**
4. (se serve rilasciare risorse) **(Tear-down)**

Per esempio:

```
[Test] public void  
Parser_Parse_ValidArgString42Returns42AsInt() {  
    _parser = new Parser(); Setup  
    var returnValue = p.Parse("42"); Call under test  
    Assert.That(returnValue, Is.EqualTo(42)); Assert  
    _parser = null; (Tear-down)  
}
```

Dibris

## Stili di Assert

- Modello *classico*
  - `Assert.AreEqual`, `Assert.IsTrue`, `CollectionAssert.AreEquivalent...`
  - un metodo per ciascuna asserzione, raggruppati in classi
  - difficile da usare (senza limitarsi ai banali)
  - prima tipologia introdotta
- Modello *a vincoli*
  - `Assert.That(..., Is.EqualTo(...))`, `Assert.That(..., Is.True)`, `Assert.That(..., Is.EquivalentTo(...))`
  - una sola classe `Assert`, un solo metodo `That`
  - un parametro in più per definire il vincolo
    - vincoli creati da metodi in classi factory per rendere più leggibile
- Modello *Fluent*
  - `actual.Should().BeEqualTo(...).And....`
  - si possono concatenare i metodi di asserzione
  - molto potente
  - non sempre leggibile
  - facilita asserzioni multiple sullo stesso elemento

MSTest JUnit

NUnit Hamcrest

Assert FluentAssertions

Dibris

## Stili di Assert in NUnit

- Classic Model - Obsoleto:  
`Assert.AreEqual(returnValue, 42);`  
invocando un metodo statico diverso per ogni asserzione  
~~Mantenuto per backward compatibility~~ **⇒ non aggiornato**
- Constraint Model - da NUnit 2.4 (2007):  
`Assert.That(returnValue, Is.EqualTo(42));`  
unico metodo statico `That`
  - primo parametro == risultato della chiamata sotto test
  - secondo parametro == **constraint** che deve essere soddisfatto
  - terzo parametro opzionale == messaggio da visualizzare in caso di fallimento
    - anche i metodi obsoleti ce l'hanno

Dibris

## Constraint

- Oggetti di una qualsiasi classe che implementa `IResolveConstraint`
  - se ne possono definire di custom
  - 99% esiste già quello che serve  
<https://github.com/nunit/docs/wiki/Constraints>
- Spesso ottenuto invocando un metodo statico di una *helper class*
  - `Is`, `Has`, `Does`, `Contains`, `Throws...`
  - basta ricordarsi il nome dell'helper class e poi con intellisense si ha la lista dei constraint
  - stile discorsivo, si legge come frase  
`Assert.That(actual, Is.Ordered.Ascending.And.Contains(3));`

Dibris

## Quante Asserzioni in un Test? (Punto di visto logico)

- **Un** test == verifica che in **una** esecuzione **una** proprietà è vera
  - valore di ritorno
  - modifica allo stato del sistema
- **Quindi una** asserzione
  - da cui l'implementazione: `Assert`.That se fallisce solleva eccezione e esce dal test

Dibris

## Esempio Sbagliato di Assert multiple 1

```
[Test] public void Parser_Parse_ValidateStringReturnsInts() {
    int res = _parser.Parse("42");
    Assert.That(res, Is.EqualTo(42));
    res = _parser.Parse("0");
    Assert.That(res, Is.EqualTo(0));
    res = _parser.Parse("1");
    Assert.That(res, Is.EqualTo(1));
}
```

```
[Test] public void Parser_Parse_String42ToInt42() {
    int res = _parser.Parse("42");
    Assert.That(res, Is.EqualTo(42));
}
```

```
[Test] public void Parser_Parse_String1ToInt1() {
    int res = _parser.Parse("1");
    Assert.That(res, Is.EqualTo(1));
}
```

```
[Test] public void Parser_Parse_String0ToInt0() {
    int res = _parser.Parse("0");
    Assert.That(res, Is.EqualTo(0));
}
```

Errore indotto da pigrizia: molti test simili collassati in uno

Dibris

## Test parametrici

- Permettono di generare diversi test usando un solo metodo, per esempio:

```
[TestCase("42", 42)]
[TestCase("0", 0)]
[TestCase("1", 1)]
public void Parser_Parse_ValidArg(string a, int r) {
    int res = _parser.Parse(a);
    Assert.That(res, Is.EqualTo(r));
}
```
- I test vengono generati a load-time da NUnit

Dibris

## Test parametrici

- L'attributo **[TestCase]** identifica un metodo come test e gli fornisce i valori dei parametri
- E' anche possibile specificare i valori sui singoli parametri (attributi **[Random]**, **[Range]** e **[Values]**) e combinarli in vari modi (attributi **[Combinatorial]**, che è il default, e **[Sequential]**)

```
[Test]  
public void MyTest([Values(1,2,3)] int x,  
    [Random(-1.0, 1.0, 5)] double d) {  
    // eseguito 15 volte, per ogni x (1,2,3)  
    // cinque valori casuali fra -1 e 1  
}
```

Valori random aiutano a trovare casi di errori inattesi  
Non sono molto *debug-friendly*

```
[TestCase, sequential]  
public void Parser_Parse_ValidArg ([Values("42", "0", "1")] string a,  
    [Values(42, 0, 1)]int r) {  
    Assert.That(_parser.Parse(a), Is.EqualTo(r));  
    // equivalente alla versione sulla slide precedente  
}
```

Dibris

## Esempio Sbagliato di Assert multiple 2

```
[Test] public void CInCondition...EverythingWorks() {  
    /*  
     * Long and painful setup  
     */  
    var res1 = o.M1(...);  
    Assert.That(res1, Is.EqualTo(...));  
    var res2 = o.M2(...);  
    Assert.That(res2, Is.EqualTo(...));  
    Assert.That(o.Xyz, Is.EqualTo(...));  
}  
[Test] public void CInCondition...M1Returns...() {  
    SetCondition...();  
    var res1 = o.M1(...);  
    Assert.That(res1, Is.EqualTo(...));  
}  
[Test] public void CInCondition... M2Returns...() {  
    SetCondition...();  
    var res2 = o.M2(...);  
    Assert.That(res2, Is.EqualTo(...));  
}  
[Test] public void CInCondition...M2SetXyz() {  
    SetCondition...();  
    var res2 = o.M2(...);  
    Assert.That(o.Xyz, Is.EqualTo(...));  
}
```

Errore indotto da pigrità: molti test con stesso setup collassati in uno  
Bisogna introdurre metodo ausiliario

Dibris

## Esempio Corretto di Assert multiple

- Da usare quando per una singola asserzione logica servono (sono più pratiche/leggibili) più chiamate ad **Assert.That**
- Esempio tipico: asserzione su metodo di inizializzazione
  - asserzione logica  
l'oggetto è correttamente inizializzato in tutte le sue parti
  - implementazione dell'asserzione  
per ciascuna property X, **Assert.That(o.X, Is.EqualTo(...))**
- Soluzione tecnica

```
Assert.Multiple(() => {Assert.That(...); Assert.That(...);});
```

anche se una asserzione fallisce le esegue **tutte** e passa al test runner l'**elenco di tutti** i fallimenti
- N.B. **Tecnicamente** risolve anche il problema degli esempi logicamente sbagliati, ma sarebbe una pessima scelta di design
  - scarsa leggibilità del risultato
  - difficile manutenibilità

Dibris

## Se ci aspettiamo un'eccezione?

I constraint corrispondenti sono accessibili grazie alla classe statica **ThrowsMatch** esatto (il test passa se l'eccezione sollevata ha tipo **BlaBlaException**)  
**Assert.That( SomeMethodCall, Throws.TypeOf<BlaBlaException>());**  
**Assert.That( SomeMethodCall, Throws.TypeOf<BlaBlaException>()**  
.With.Property("Parameter").EqualTo("myParam"));

Accetta anche sottotipo (il test passa se l'eccezione sollevata ha tipo **BlaBlaException** o sua estensione)  
**Assert.That( SomeMethodCall, Throws.InstanceOf<BlaBlaException>());**  
**Assert.That( SomeMethodCall, Throws.InstanceOf<BlaBlaException>()**  
.With.Property("Parameter").EqualTo("myParam"));

Per esempio,

```
[Test]  
public void Parser_Parse_NullArgThrows () {  
    Assert.That(()=>new Parser().Parse(null), Throws.TypeOf<ArgumentNullException>());  
}
```

((=>new Parser().Parse(null)

trasforma una chiamata in una action da usare come parametro **SomeMethodCall**

Dibris

## SetUp e TearDown

- Poiché ogni test deve essere indipendente e l'ordine di esecuzione non deve influenzarne l'esito, ogni test deve allocare e rilasciare risorse
- Le parti comuni (a *tutti* i test) possono essere inserite in metodi di SetUp/TearDown
- I metodi annotati con **[SetUp]** vengono eseguiti **prima di ogni test**
- I metodi annotati con **[TearDown]** vengono eseguiti **dopo ogni test**

Per esempio...

Dibris

## Esempio SetUp/TearDown

```
private Parser _parser;  
  
[SetUp] public void Init() {  
    _parser = new Parser();  
}  
  
[TearDown] public void CleanUp() {  
    _parser = null;  
}  
  
[Test] public void Parser_Parse_ValidString42Returns42AsInt() {  
    var returnValue = _parser.Parse("42");  
    Assert.That(returnValue, Is.EqualTo(42));  
}  
// ...
```

Dibris

## Altro su SetUp/TearDown

- In caso di ereditarietà (della classe di Test):
  - Gli attributi di Setup/TearDown vengono ereditati
  - I metodi SetUp/TearDown della classe base vengono invocati prima di quelli della derivata (che, chiaramente, non devono invocare quelli base)
  - In caso un SetUp sollevi un'eccezione, vengono invocati (solo) i TearDown delle classi per cui sono stati completati i SetUp prima dell'errore
- **[OneTimeSetUp]** e **[OneTimeTearDown]** permettono di annotare metodi che vengono eseguiti prima dell'esecuzione del primo test (risp. dopo l'esecuzione dell'ultimo)

Dibris

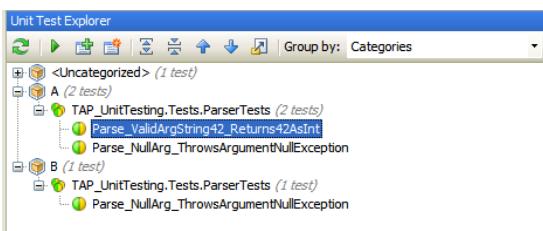
## Annotazione [Ignore] vs Assert.Inconclusive

- L'annotazione **[Ignore]** indica una class o dei test da ignorare e la motivazione.
  - Per esempio,  
[Ignore("Incomplete test")][Test] public void Foo() {}
  - I test così annotati non vengono eseguiti
    - si risparmia tempo
    - si evitano feedback confusi
- L'asserzione **Assert.Inconclusive()** si usa per indicare flussi di esecuzione del test che non hanno senso.
  - Per esempio i parametri del test non sono corretti  
[TestCase(42.42)]  
public void PriceUpTo100NoDiscount(double price) {  
 if (price < 0)  
 Assert.Inconclusive("Price cannot be negative");  
 /\*...\*/
  - Il test viene eseguito ma se quella specifica esecuzione passa da questa asserzione il risultato non è significativo (non è né giusto né sbagliato)

Dibris

## Categorie

- E' possibile associare delle categorie ai test, tramite l'annotazione **[Category]**
- In questo modo diventa possibile eseguire solo i test di alcune categorie



Dibris

## Come testare le classi internal?

Poiché le classi di test sono in un assembly diverso da quello delle classi testate, dobbiamo ricorrere a un attributo a livello di Assembly:

`[assembly:InternalsVisibleTo(`«nome del progetto di test»`)]`

Va inserito subito dopo gli using

Dibris

## Code Coverage

- È una misura che indica quale percentuale di sorgenti viene coperta dall'esecuzione dei test
  - nel caso ideale, il 100%
- VS misura e fa report di quale parte del SUT è esercitata dai test
- Attenzione:  
codice coperto/esercitato != esistono test specifici per quel pezzo di codice
  - se un metodo M1 è usato da un metodo M2 i test di M2 esercitano anche M1

Dibris

## Operator overloading

Giovanni Lagorio & Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2021-22

Dibris

## Operator overloading

- In molti linguaggi alcuni operatori sono overloaded (per es. quelli aritmetici)
  - Esempio:  
`DateTime x = DateTime.Now;  
DateTime y = new DateTime(1982, 1, 1);  
TimeSpan t = x - y;  
DateTime k = x + t;`
- C# permette di definire il comportamento di molti operatori, quando (almeno) un operando è di un tipo definito dall'utente
  - Non si possono aggiungere operatori o variarne la precedenza
  - Non si può ridefinire l'assegnazione (nemmeno struct)
- Discussione su quando è opportuno ridefinire un operatore  
<https://learn.microsoft.com/en-us/dotnet/standard/design-guidelines/operator-overloads>

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

2

## Operator overloading – esempio di definizione

- Vanno definiti **public static** e hanno come nome **operator op**
- ```
public class C {  
    public static C operator +(C x, int y) {/*...*/}  
}
```

  - esempio di uso  
`var c = new C();  
C foo = c + 3;  
c += 27; // (!!) come scrivere c = c+27;`
- N.B. negli operatori binari servono due parametri
- Si possono definire anche conversioni implicite o esplicite
  - esempio di sistema da **int** a **double**
  - esempio user defined (nella classe **C**)  
`public static implicit operator C(int i) {/*...*/}  
public static implicit operator int(C from) {/*...*/}  
public static explicit operator string(C from) {/*...*/}`
  - uso  
`C c = 42;  
int i = c;  
string s = (string)c;`

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

3

## Operator overloading – uguaglianza e confronto

- Alcuni operatori (== e !=, > e <, >= e <=) vanno a coppie, se si definisce uno, si deve definire anche l'altro
  - Se si definisce == è buona norma ridefinire anche Equals e GetHashCode
  - Ulteriori dettagli <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/operator-overloading>
    - contiene un esempio simile al laboratorio sulle frazioni
  - Per semplificarvi la vita...alt+ins e inserite lo snippet per l'uguaglianza
- Alcuni operatori suggeriscono simmetria, non obbligatorio ma potrebbe essere meglio rispettare l'intuizione
  - se definisco **public static C operator +(C x, int y) {/\*...\*/}** allora definisco anche **public static C operator +(int y, C x) {/\*...\*/}** in modo che  $c + 42$  sia uguale a  $42 + c$

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

4

## Parentesi su Equals

- Equals(c0, c1) e Equals(c1, c0) sono equivalenti?
  - se C0 e C1 sono lo stesso riferimento (valido o entrambi null) vale vero
  - altrimenti se uno (solo) dei due è null vale falso
  - altrimenti Equals(first, second) si traduce in first.Equals(second)
- c0.Equals(c1) e c1.Equals(c0) sono equivalenti?
  - se c0 == null e c1 != null
    - c0.Equals(c1) solleva **NullReferenceException**
    - **da contratto** c1.Equals(c0) restituisce false
  - se c0 != null e c1 == null
    - c0.Equals(c1) è l'overriding (se esiste) di Equals nella classe di c0
    - c1.Equals(c0) è l'overriding (se esiste) di Equals nella classe di c1
    - se c0 è un punto colorato e c1 è un punto...
      - pensate bene se veramente volete che l'uguaglianza sia vera fra tipi diversi

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

5

# Metadati & Reflection

Giovanni Lagorio & Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris

## Metadati – Custom attributes

- Annotazioni per assembly, tipi, membri e parametri
  - In Java @Blablabla
  - Un po' tipo i modificatori public, private, ...
  - Ma definite dall'utente
    - Sono classi che estendono **System.Attribute**
      - Convenzionalmente, il nome termina in Attribute ma questo non si scrive quando si applica l'annotazione
    - Per esempio, **Serializable** (**System.SerializableAttribute**) su una classe indica che i suoi oggetti sono serializzabili (e non ha senso su, per esempio, un parametro o un singolo campo)
- Alcune influiscono sul comportamento del compilatore
  - Per esempio, **Conditional**
  - Tutte sono salvate nell'assembly e possono essere recuperate a runtime tramite reflection (introspection)

Dibris

2

## Esempio (scemo)

```
public class TapAttribute : Attribute { }

[Tap] // classe annotata con Tap
public class Program {

    [Tap] // metodo annotato con Tap
    public static void Main(string[] args) {

    }
}
```

Dibris

## Esempio (meno scemo)

```
[AttributeUsage(AttributeTargets.Method|AttributeTargets.Class,
    AllowMultiple = true, Inherited = false)]
public class AuthorAttribute : Attribute {
    private readonly string _authorName;
    public AuthorAttribute(string authorName) {
        this._authorName = authorName;
    }
    public string AuthorEmail { get; set; }
    public string GetAuthorName() {
        return this._authorName;
    }
}

[Author("arthur")][[Author("ford")]
public class Program {

    [Author("marvin", AuthorEmail = "marvin@lifesucks.org")]
    public static void Main(string[] args) { }
}
```

Dibris

4

## Parentesi: reflection/introspection

- Con **Reflection** si intende la capacità di un programma di osservare/modificare la sua struttura e comportamento
  - Per ovvie ragioni, è relativamente
    - comune nei linguaggi tipati *dinamicamente*
    - rara nei linguaggi tipati *staticamente*
- Java/C# chiamano Reflection quella che sarebbe più giusto chiamare **introspection**, ovvero Reflection «read-only»
- Lo «stargate» (termine assolutamente inventato! Non lo troverete nella documentazione ufficiale ☺) fra il mondo compile-time e quello run-time è la classe Type (e, da .NET 4.5, TypeInfo)

Dibris

## Esecuzione in .Net e reflection

- Application domains
  - gestiti dal common language runtime
  - contenitori per esecuzione di pezzi di una stessa applicazione
- Pezzi organizzati gerarchicamente
  - Assemblies
  - Modules
  - Types
  - Members
- System.Reflection namespace per rappresentare questi elementi
- Per eseguire
  - l'assembly deve essere in memoria
  - si usa TypeInfo
    - rappresenta il tipo
- Per vedere che cosa è contenuto
  - l'assembly viene letto ma non caricato nell'AppDomain
  - si usa Type
    - in System (non System.Reflection)
    - rappresenta una vista read-only del tipo

Dibris <https://learn.microsoft.com/en-us/dotnet/api/system.reflection?view=net-6.0>

## Tornando ai custom-attributes...

**Non generic** ⇒ restituisce array di object  
Potenzialmente con custom attribute di vari tipi

```
public static void Main(string[] args) {
    Type type = typeof(Program);
    object[] attributes = type.GetCustomAttributes(false);
    foreach (AuthorAttribute aa in attributes)
        Console.WriteLine(aa.GetAuthorName());
    MethodInfo methodInfo = type.GetMethod("Main");
    AuthorAttribute[] authorAttributes =
        methodInfo.GetCustomAttributes<AuthorAttribute>(false);
    foreach (AuthorAttribute aa in attributes)
        Console.WriteLine("Name = {0}, Email = {1}",
            aa.GetAuthorName(), aa.AuthorEmail);
    Console.ReadLine();
}
```

Quando possibile usate le versioni generiche dei metodi

Dibris

## Alcune considerazioni

- Tutto questo ha senso solo per classi che non si conoscono staticamente
- Ci sono metodi per caricare exe/dll e ottenere l'oggetto Assembly corrispondente
- Pensate, per esempio, a un programma che carica una DLL ed esegue tutti i metodi annotati con **[Test]**

Dibris

## C#: Tipi Nullabili e Non

Giovanni Lagorio & Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris

 Dibris Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Tipi valore nullabili

- Introdotti per interfacciarsi facilmente ai DB

```
public class Artist {  
    public int ArtistId { get; set; }  
    public string ArtistName { get; set; }  
    public DateTime? ActiveFrom { get; set; }  
}
```
- Dato un tipo valore T, T? è il corrispondente nullabile (è sempre un tipo valore, ma ...)
  - Per esempio,

```
int? i = null;  
if (i.HasValue) {  
    int j = i.Value;  
    i = j;  
}
```

int y = i.GetValueOrDefault(); //y==0  
int z = i ?? 27; //z==27  
int wErr = i; //errore di compilazione  
int w = (int) i; //errore dinamico  
i = 1024;  
w = (int) i; //w==1024
  - T è sottotipo di T?
    - nota: non è una domanda (dalla risposta ovvia ☺)

Dibris

 Dibris Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

2

## Tipi valore nullabili (sorprese indesiderate)

- Solo per i maniaci dei linguaggi ... se per dei tipi non-nullabili sono definiti degli operatori relazionali, allora esistono anche delle versioni "liftate" per i corrispondenti tipi nullabili
  - Fanno la cosa ovvia quando entrambi gli operandi sono non-null
  - Restituiscono false altrimenti
- Semantica (a dir poco) "buffa":  
`int? i = null;`  
`Console.WriteLine(i == i); // true`  
`Console.WriteLine(i <= i); // false`

Dibris

 Dibris Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

3

## Tipi reference nullabili - Razionale

Interessante spiegazione del come/perché è stato introdotto null fatto dal suo creatore  
<https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/>

- Tipi reference ~ astrazione di indirizzo di memoria  
⇒ inerentemente nullabili  
⇒ rischio di NullReferenceException
- Nei programmi "scritti bene" servono valori nulli in pochi casi ben delimitati
  - var a = new string[42]; // le singole celle
  - valori *mancanti* in tipi strutturati  
es. relazioni 0-1 (SignificantOther in Person per i single)
- Si sarebbe potuto impostare le cose diversamente
  - bit "inizializzato"
  - controlli prima di ogni accesso
  - troppo inefficiente
- Analisi statica può aiutare a individuare molti dei casi *pericolosi*

Dibris

 Dibris Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Tipi reference nullabili - Soluzione

- Introduzione dei tipi reference nullabili in C# 8  
es. string?
  - string **cambia semantica** e rappresenta le stringhe non nullabili  
⇒ concettualmente introduzione dei tipi reference **non** nullabili
  - sintassi scelta per analogia con i tipi valore nullabili
- Usare C? vs C permette al programmatore di esprimere il suo intento
- Modificare la semanticà dei tipi reference senza "?" è un cambiamento retroattivo
  - incoraggia miglioramento del codice legacy
  - va fatto per gradi per evitare insurrezioni
- Impatto della scelta limitato
  - warning del compilatore, non errori
  - opt-in per i progetti esistenti (e opt-out per i nuovi)
- Qui riassunto dei punti principali, ma leggete (mandatory!!! in ordine inverso)
  - <https://docs.microsoft.com/en-us/dotnet/csharp nullable-references>  
per i dettagli
  - <https://devblogs.microsoft.com/dotnet/nullable-reference-types-in-csharp/>  
per la logica del design (che aiuta a capire e ricordare i dettagli)

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Tipi reference nullabili – Implementazione

- La stessa classe implementa sia C? che C
- Nel codice compilato non c'è nessuna differenza fra usare C? o C
- Durante la compilazione variabili, campi etc di tipo reference [nullable] vengono tracciati per capire se sono usati in accordo con l'intento del programmatore
  - assegnazioni
    - x = new...//qui x è non nulla
    - x=null ...//qui x è nullabile
  - test: if(x!=null){/\*qui x è non nulla\*/}
  - annotazioni (vedere <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/attributes/nullable-analysis>)
    - public static bool IsNullOrEmptyWhiteSpace([NotNullWhen(false)] string message);
  - null-forgiving operator !
    - name!.Length;

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Tipi reference nullabili – Effetti

- Ogni espressione di tipo [nullable] reference può essere not-null o maybe-null
- Usare un maybe-null per assegnare una variabile/campo o come ritorno di metodo di tipo reference non nullable dà warning
- Dereferenziare un maybe-null dà warning
- Analisi statica limitata per
  - calcolabilità
  - complessità
  - convenienza

costrutto che aiuta a trovare bug  
ma NON garantisce di trovare  
tutti gli accessi a null



Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

```
public class C{
    private string _x;
    private readonly string? _xMaybe;
    2 references | 0 changes | 0 authors, 0 changes
    public string ReturnNull(){
        return null;
    }
    2 references | 0 changes | 0 authors, 0 changes
    public string? SafeReturnNull(){
        return null;
    }
    1 reference | 0 changes | 0 authors, 0 changes
    public C(){
        _x = "hello";
        _x = null;
        _xMaybe = null;
    }
    2 references | 0 changes | 0 authors, 0 changes
    C Involve(string s){
        var c = new C();
        c._x = s;
        return c;
    }
}
```

```
public void M(){
    string s = _x;
    string s1 = ReturnNull();
    string s2 = SafeReturnNull();
    string s3 = Involve("ggg").ReturnNull();
    string s4 = Involve("egg").SafeReturnNull();
    string? w = null;
    string k = _xMaybe;
    var a = new string[42];
    a[0] = w;
    a[1] = s;
}
```

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Tipi reference nullabili – Contesto

- A seconda del *contesto* la gestione è diversa

Context	Dereference warnings	Assignment warnings	Reference types	? suffix	! operator
<code>disabled</code>	Disabled	Disabled	All are nullable	Can't be used	Has no effect
<code>enabled</code>	Enabled	Enabled	Non-nullable unless declared with ?	Declares nullable type	Suppresses warnings for possible <code>null</code> assignment
<code>warnings</code>	Enabled	Not applicable	All are nullable, but members are considered <i>not null</i> at opening brace of methods	Produces a warning	Suppresses warnings for possible <code>null</code> assignment
<code>annotations</code>	Disabled	Disabled	Non-nullable unless declared with ?	Declares nullable type	Has no effect

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Tipi reference nullabili – Attivazione

- A livello di progetto nel file di configurazione `<PropertyGroup>`
  - .....
  - `<Nullable>enable</Nullable>`
  - `</PropertyGroup>`
- A livello di specifiche parti di codice usando direttive
  - `#nullable enable`: Sets the nullable annotation context and nullable warning context to enabled.
  - `#nullable disable`: Sets the nullable annotation context and nullable warning context to disabled.
  - `#nullable restore`: Restores the nullable annotation context and nullable warning context to the project settings.
  - Analoghi per warning e annotations

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## null-coalescing operator

- Se `exp1` ed `exp2` sono espressione di un tipo che ammette null (reference nullabili e no, nullable value types)

`exp1??exp2`

corrisponde a

`IsNull(exp1)?exp2:exp1`

- Interazione con eccezioni

In C# throw può generare un'espressione (non solo uno statement)  
 ⇒ sono legali

– `var x = exp?? throw new ArgumentNullException`  
 – `var x = exp?.BlaBla?? throw new XYZException`

- null-coalescing assignment

`x??=exp`

corrisponde a

`if(null==x)x=exp`

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

11

## Dependency Injection

Giovanni Lagorio & Maura Cerioli  
 Dipartimento di Informatica, Bioingegneria, Robotica e  
 Ingegneria dei Sistemi  
 University of Genova  
 TAP 2022-23

Dibris

## Esempio: EmailSender vs BulkEmailServer

Goal: spedire mail a liste (spam!)

```
public class BulkEmailSender {  
    private EmailSender MyEmailSender {get;}  
    public BulkEmailSender() {MyEmailSender = new EmailSender();}  
    public void SendEmail(List<string> tos, string body) {  
        foreach (var a in tos)  
            if (!MyEmailSender.SendEmail(a, body)) throw new Exception("...");  
    }  
}
```

Basandosi su Mailer che spedisce a un singolo indirizzo (componente)

```
public class EmailSender {  
    public bool SendEmail(string to, string body) {  
        // TODO: send mail and return true if everything ok  
        return false;  
    }  
}
```

Dibris

2

## Dipendenze

- Di cosa abbiamo bisogno per distribuire BulkEmailSender?
- Il cliente può usare un'altra componente per spedire le singole email?
- Che impatto ha bug-fixing in EmailSender su BulkEmailSender?
- Vincoli e dipendenze di EmailSender hanno impatto su BulkEmailSender?
- Come possiamo testare BulkEmailSender?
  - se siamo offline?
  - se non vogliamo “spammare” durante il testing?

Dibris

3

## Dipendenze - Risposte

Nell'esempio il designer di BulkEmailSender ha deciso di usare EmailSender ⇒

- BulkEmailSender richiede EmailSender per funzionare
- il cliente non può cambiare
- ogni nuova release di EmailSender richiede di fare una nuova release di BulkEmailSender (ricompilare)
- vincoli e dipendenze di EmailSender vengono ereditate
- testing richiede notevoli acrobazie

Dibris

Brutto Design!!!

## Due livelli di dipendenza

- Come tipo: usare la classe EmailSender è troppo restrittivo
  - basta introdurre un'interfaccia
  - da qui in poi assumiamo di averlo fatto
- Come oggetto per inizializzare la property: facendo new fisso la classe
  - no, non si può fare new di un'interfaccia
  - vari pattern per risolvere questo problema
    - Factory (tipicamente un Singleton, statico)
    - Service Locator
    - **Dependency Injection**

Dibris

5

## Classe Factory (singleton “alla GoF”)

E’ già qualcosa,

- Minimizza i cambiamenti (una sola new, invece di tante sparse qua e là)

ma non risolve:

- Le classi dipendono dalla Factory invece che dalle implementazioni, ma lei dipende da loro... per transitività siamo al punto di partenza ☺
- E’ statica (se non lo fosse, chi la istanzierebbe e come? Si sposterebbe solo il problema...)
  - il singleton non static va benissimo... ne parliamo dopo

Dibris

6

## Service Locator (Anti-pattern!)

- Scarica la responsabilità di istanziare gli oggetti su un oggetto, il *Service Locator* (o Registry, Context, Manager, Environment, ...), che si passa alle classi che devono “istanziare interfacce”
- Le vere dipendenze sono nascoste dal fatto che ogni classe dipende dal Service Locator
  - Possibile, ma difficile il testing: cosa ridefinisco/modifico nel Service Locator per creare gli stub?
- Se SL è una classe, praticamente ogni classe dipende da lei, che sa istanziare tutte le altre classi, quindi ogni classe dipende da tutte le altre...

Dibris

7

## Dependency Injection

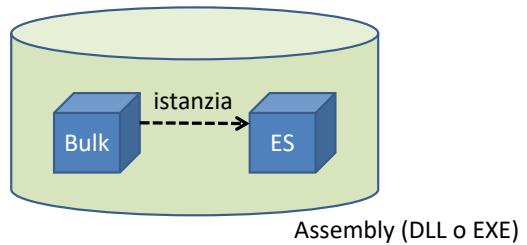
- Esistono vari tipi di DI, consideriamo la constructor injection (è quella da usare “di default”) che “inietta” tramite i costruttori
  - Altre sono la method, property, field, ...
- L’idea è estremamente semplice:  
**i costruttori richiedono gli oggetti che servono direttamente, invece di crearli**

Analizziamo il nostro esempio...

Dibris

8

## Situazione iniziale

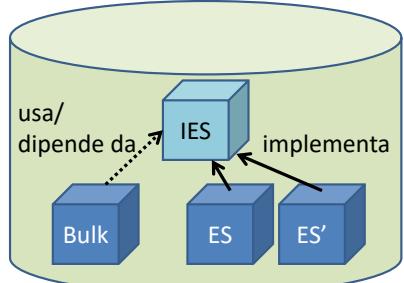


- Assumiamo che siano nello stesso assembly
  - Metterle in assembly diversi cambierebbe qualcosa per Bulk?

Dibris

9

## Astraiamo introducendo IES

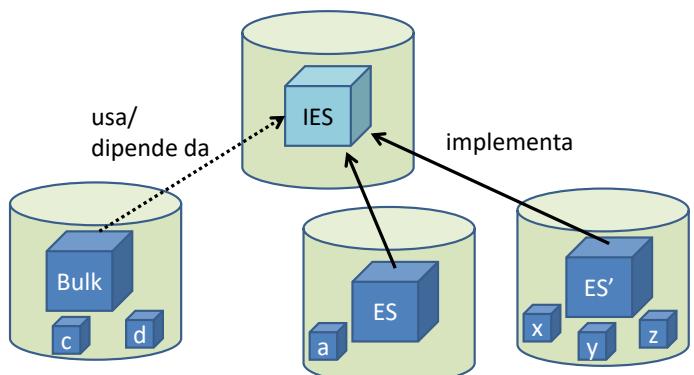


- Nota: Bulk non ha bisogno di creare degli ES, ma gliene serve uno (rimuoveremo dopo questa semplificazione, che per Bulk è assolutamente ragionevole ma in altri scenari potrebbe non esserlo)
- Adesso «spezzare» l'assembly cambierebbe qualcosa?

Dibris

10

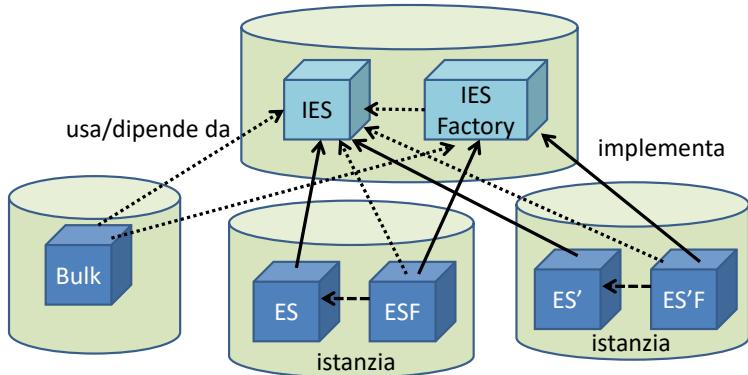
## Ci siamo quasi...



Dibris

11

## Ma se Bulk dovesse creare degli IES?

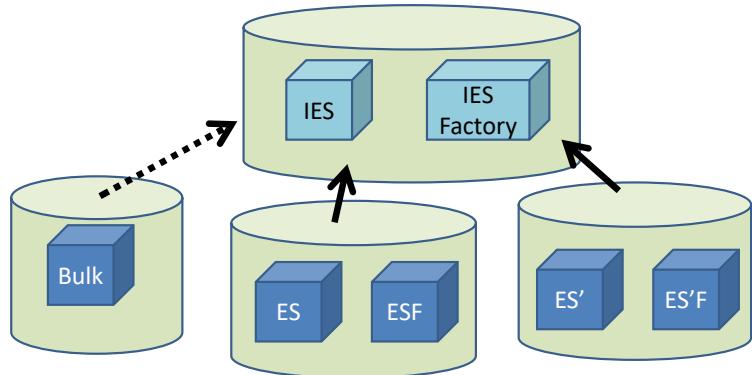


- Dimentichiamoci le altre "classettine" per semplicità

Dibris

12

## ...e consideriamo solo le dipendenze fra assembly



Dibris

13

## In codice (senza factory):

```
public BulkEmailSender() {  
    MyEmailSender = new EmailSender();  
}  
  
public BulkEmailSender(IEmailSender  
emailSender) {  
    MyEmailSender = emailSender;  
}
```

Dibris



14

## In codice (con factory):

```
public BulkEmailSender() {  
    MyEmailSender = new EmailSender();  
}  
  
public BulkEmailSender(IEmailSenderFactory  
factory) {  
    MyEmailSenderFactory = factory;  
    MyEmailSender = factory.CreateNew();  
}
```

Dibris

Ha senso solo quando la classe client, BulkEMailSender ha bisogno di creare un numero (ignoto a priori o molto grande) di oggetti

15

## Dove...

```
interface IEmailSender {  
    bool SendEmail(string to, string body);  
}  
  
interface IEmailSenderFactory {  
    IEmailSender CreateNew();  
}  
  
// e, per esempio, una factory è:  
class EmailSenderFactory : IEmailSenderFactory {  
    public IEmailSender CreateNew() {  
        return new EmailSender();  
    }  
}
```

Dibris

16

## Altro esempio di (non static) Factory

```
public class C {  
    private IPoointFactory PointFactory {get;}  
    private ILineFactory LineFactory {get;}  
    public C(IPoointFactory pointFactory, ILineFactory lineFactory) {  
        this.PointFactory = pointFactory;  
        this.LineFactory = lineFactory;  
    }  
    public void DoSomething(/* ... */) {  
        // ...  
        ILine newLine = CreateLine(/* ... */);  
        // ...  
    }  
    private ILine CreateLine(int x0, int y0, int x1, int y1) {  
        IPooint p0 = this.PointFactory.Create(x0, y0);  
        IPooint p1 = this.PointFactory.Create(x1, y1);  
        return this.LineFactory.Create(p0, p1);  
    }  
}
```

Dibris

17

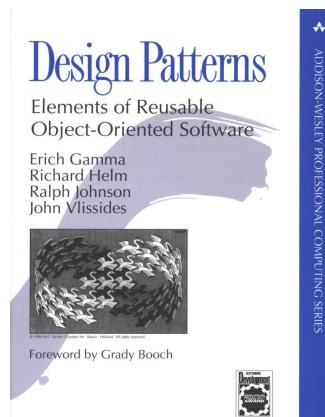
Ok, ma cosa ci guadagniamo?

- Diventa facile sostituire un'implementazione con un'altra
  - Di conseguenza
    - diventa (più) facile il testing
    - estendere le funzionalità, per esempio, tramite *Decorator* (alla GoF)

Dibris

18

## Gang of Four (GoF)



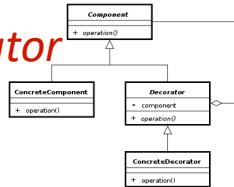
## La “bibbia” dei Design Pattern

## *Catalogo di 23 pattern divisi in: creazione, struttura e comportamento*

Dibris

19

## Esempio di *Decorator*

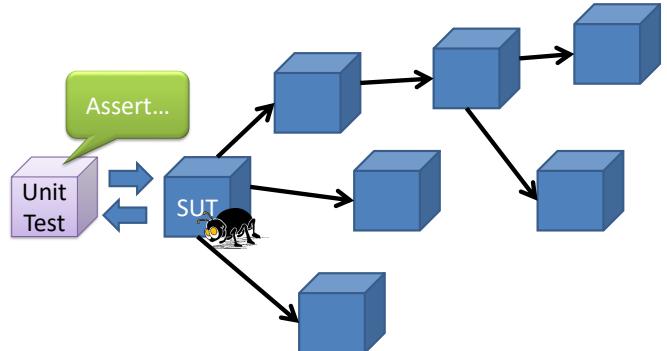


```
class LoggingEmailSender : IEmailSender {
    private IEmailSender OriginalSender {get;}
    public LoggingEmailSender(IEmailSender originalSender) {
        this.OriginalSender = originalSender;
    }
    public bool SendEmail(string to, string body) {
        Console.WriteLine("Sending mail to {0}", to);
        return this.OriginalSender.SendEmail(to, body);
    }
}
```

Dibris

20

## Un altro esempio: il Testing

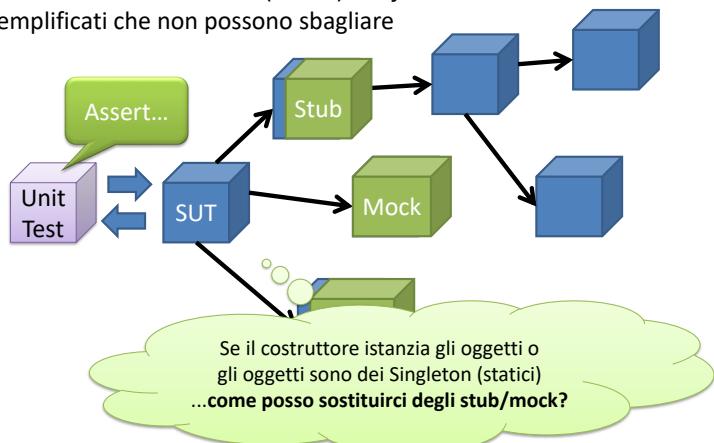


Dibris

21

## Soluzione: Stub e Mock

Sostituiamo i sottosistemi (diretti) con *fake* semplificati che non possono sbagliare



Dibris

22

## Cose ovvie

- Per testare un metodo (d'istanza) la classe che lo contiene va istanziata
- Se il costruttore "fa troppo" può diventare difficile o troppo lento (connessione di rete, DB...)
- La presenza di inizializzatori/campi statici peggiora le cose
  - Globali alla VM (in realtà all'AD), non all'applicazione
    - Ogni unit-test deve, logicamente, istanziare una (porzione) di nuova app
  - Devono fare/contenere cose diverse durante il testing
  - L'ordine di esecuzione diventa importante
    - I test non possono essere lanciati in parallelo!

Dibris

23

## Classi (facilmente) testabili

La responsabilità di creare degli oggetti va separata!

Nei costruttori,

- invece di *istanziare* gli oggetti che servono, si *richiedono* tali oggetti
  - Disinteressandosi sul come vengono istanziati
- si richiedono solo gli oggetti che servono *direttamente* (quelli che vengono copiati nei campi)
  - Idealmente, il corpo di un costruttore dovrebbe essere una sequenza di assegnazioni
  - Ricevere *x* e poi fare *this.f = x.qualcosa...* è molto sospetto

Dibris

24

## Principio di minima conoscenza (anche: Legge di Demetra)



Per pagare una birra al pub si danno i soldi al barista.

Non lo si lascia cercarci in tasca il portafoglio per pagarsi da solo

Equivalenti (di passare il portafoglio) in codice...

```
public void Purchase(Customer c) {  
    Money m = c.GetWallet().GetMoney();  
    this.RecordSale(..., m);  
}
```

Nel testing:

```
Money m = new Money(5);  
Wallet w = new Wallet(m);  
Customer c = new Customer(w);  
Goods g = ...  
g.Purchase(c);  
// ...assert...
```

Dibris

25

## Torniamo al nostro esempio:

```
public BulkEmailSender() {  
    MyEmailSender = new EmailSender();  
}  
  
public BulkEmailSender(IEmailSender  
emailSender) {  
    MyEmailSender = emailSender;  
}
```

Dibris



## Il testing diventa facile!

```
[TestFixture]  
public class TestBulkEmailSender {  
    [Test] public void TestSendMailEmailSenderSucceeds() {  
        ...  
        var bulk = new BulkEmailSender(mock_ok);  
        bulk.SendEmail(new List<string> { "a@a.com", "b@b.com" }, "hi!");  
    }  
  
    [Test]  
    public void TestSendMailEmailSenderFails() {  
        ...  
        var bulk = new BulkEmailSender(stub_fails);  
        Assert.Throws<Exception>(() =>  
            bulk.SendEmail(new List<string> { "a@a.it", "b@b.it" }, "hi!"));  
    }  
}
```

Dibris

27

## Note che:

- Nei test è ragionevole usare new del SUT
  - Il codice di test dipende da ciò che sta testando
  - Analogamente la root di un sistema farà tutte le new dei pezzetti da passare ai vari costruttori per creare i sottosistemi
- Per il progetto i test saranno gli stessi per tutte le diverse implementazioni ⇒
  - peculiarità ulteriori, che non dipendono da DI
  - il progetto di test non può dipendere da quello dell'implementazione
  - le interfacce vanno associate a implementazioni non note a tempo di design/compilazione
- DI ⇒ codice più facilmente testabile perché è facile iniettare stub/mock
- DI ⇒ codice con low coupling  
⇒ codice con maggiore qualità (interna)

Dibris

28

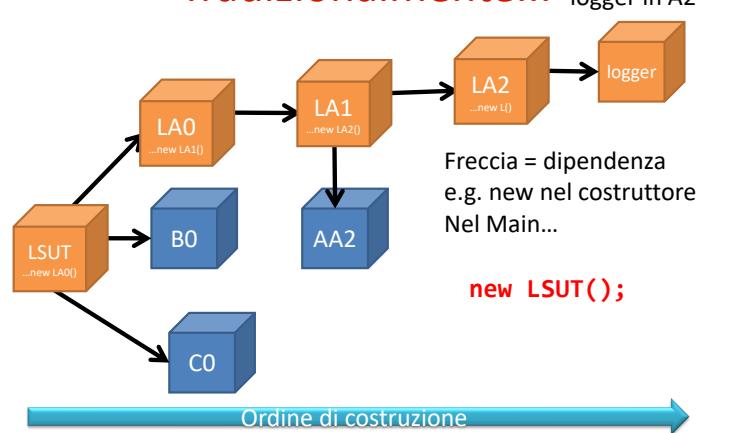
## Dubbi (classici)

- Ma se ho 10 campi il costruttore deve richiedere 10 oggetti? Non sono troppi?
  - Salvo eccezioni, sì deve richiedere 10 parametri
  - Possibili eccezioni: le “classi di sistema” (come le collection), si possono istanziare senza problemi (ci giochiamo la possibilità di sostituirle... solitamente non è un problema)
  - Se i parametri sono troppi, è probabile che la classe abbia troppe responsabilità, ma ce le aveva già prima di usare la DI!
  - La DI rende esplicite le dipendenze, non le diminuisce e non le aumenta
- Quindi, se una classe di basso livello ha bisogno, diciamo, di un logger, lo devo passare lungo tutta la catena?
  - NO! Tipico fraintendimento... vediamo un esempio

Dibris

29

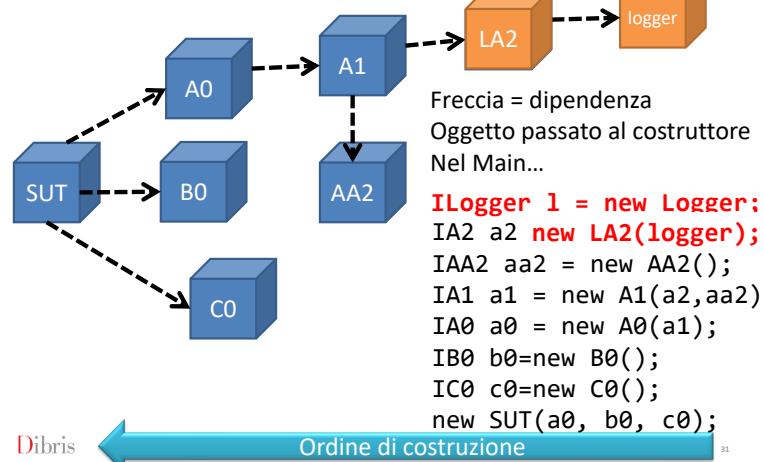
## Tradizionalmente...



Dibris

## Con la DI...

Aggiungendo un logger in A2



31

## In codice...

(versione tradizionale con singleton)

```
DB.Init(...); // inizializza il singleton
Logger.Init(); // idem, l'ordine è importante!
var bulk = new LoggingBulkEmailSender();
    // apparentemente indipendente da sopra
    // ma se il Logger non è stato inizializzato
    // LoggingEmailSender (istanziato da
    // LoggingBulkEmailSender) si schianta
```

Dibris

## Con la DI...

```
var db = new DB(...);
var logger = new Logger(db);
var emailSender = new LoggingEmailSender(logger);
var bulk = new BulkEmailSender(emailSender, "");
    // sbagliando l'ordine, la compilazione
    // fallisce! :-)
```

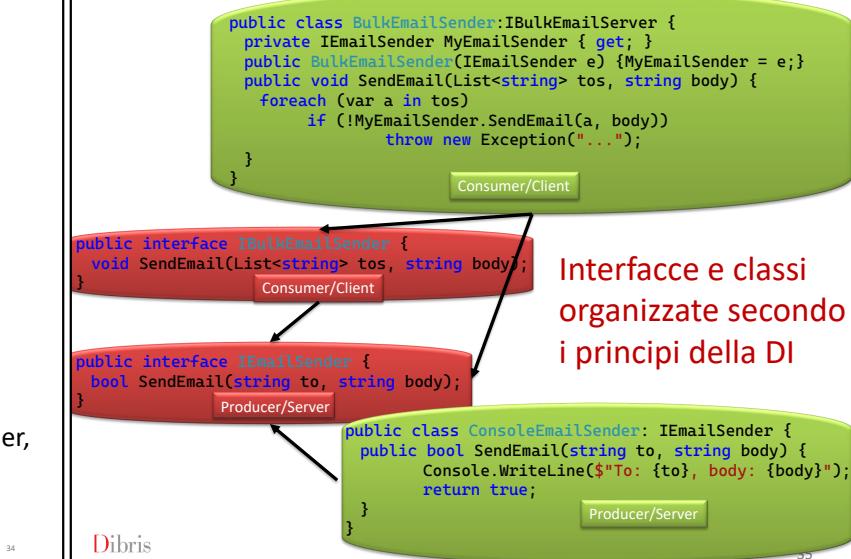
Dibris

33

## DI container

- DI container  $\cong$  framework per composizione automatica
  - non indispensabile per usare DI ma comodo
- Si programmano associando implementazioni a interfacce
- Gestiscono la creazione e lo scope di oggetti di interfacce note
- Molte varianti:
  - `Microsoft.Extensions.DependencyInjection` (built-in)
  - Ninject, Autofac, Castle of Windsor, Guice, NanoContainer, PicoContainer, Spring.NET, Unity, Winter4net...

Dibris



Dibris

35

## Uso di DI container

Nel codice che usa BulkEmailSender

- Si scelgono le implementazioni delle interfacce
- Dipendenza da tutti
 

```
using EmailSenderInterface;
using EmailSenderImplementation;
using BulkEmailSenderInterface;
using BulkEmailSenderImplementation;
```
- Si usa il container
  - installare con nuget
  - `using Microsoft.Extensions.DependencyInjection;`
- Si programma il container
 

```
ServiceProvider serviceProvider = new ServiceCollection()
    .AddTransient<IEmailSender, ConsoleEmailSender>()
    .AddTransient<IBulkEmailSender, BulkEmailSender>()
    .BuildServiceProvider();
```

  - di solito nella aggregation root ~ main
- Si usa il container per ottenere gli oggetti
 

```
- var es = serviceProvider.GetService<IEmailSender>();
- var bes = serviceProvider.GetService<IBulkEmailSender>();
```

AddTransient restituisce un nuovo oggetto a ogni chiamata di GetService  
AddSingleton restituisce sempre lo stesso oggetto  
...ci sono altre varianti...

usa costruttore senza parametri

Dibris usa costruttore di BulkEmailServer e se stesso per creare il parametro

## Risorse

- Inversion of Control Containers and the Dependency Injection pattern by Martin Fowler  
<http://martinfowler.com/articles/injection.html>
- Documentazione Microsoft
  - <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>
  - <https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.serviceprovider?view=dotnet-plat-ext-7.0>
  - <https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.servicecollection?view=dotnet-plat-ext-7.0>
- The clean code talks .... (ce ne sono diversi)  
by Miško Hevery <http://www.youtube.com/watch?v=RlfLCWKxHJ0>

Dibris

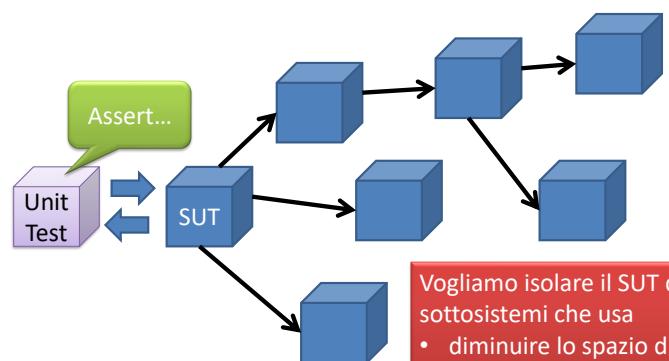
37

## Unit Testing Avanzato

Giovanni Lagorio & Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris

## Unit Testing nel mondo reale...



Dibris

Vogliamo isolare il SUT dai sottosistemi che usa

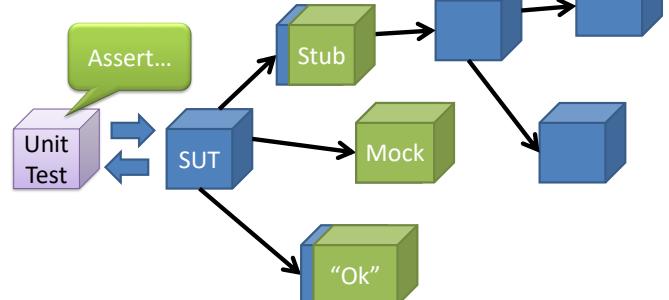
- diminuire lo spazio di ricerca dei bug
- rendere il testing più economico

## Microsoft Fakes

- Nomenclatura Microsoft: Fakes = tutti gli oggetti che servono a isolare SUT da sottosistemi
  - <https://learn.microsoft.com/en-us/visualstudio/test/isolating-code-under-test-with-microsoft-fakes?view=vs-2022&tabs=csharp>
  - shim da usare in assenza di DI
  - stub da usare in presenza di DI
- Usare shim vuol dire modificare il codice sotto test
  - per ogni assembly che contiene oggetti da sostituire si crea un Fake
  - nel Fake si definiscono gli oggetti da usare nel test con nome che inizia con Shims
  - il compilatore rimpiazza il codice vero con gli Shims
- Gli shim hanno senso solo
  - codice legacy scritto male
  - isolare SUT da chiamate di sistema
  - li ignoriamo in TAP (=scriviamo il codice bene e lo testiamo in modo sensato ☺)
- Nomenclatura nel resto del mondo...ci stiamo lavorando  
<https://stackoverflow.com/questions/346372/whats-the-difference-between-faking-mocking-and-stubbing>

Dibris

## Stub e Mock



Dibris

## Stub e Mock

- Stub e Mock sono oggetti che, durante il testing, sostituiscono gli oggetti "veri" (le dipendenze del SUT) facendo il minimo indispensabile
- I mock registrano le interazioni per una successiva verifica (quante volte è stato chiamato il metodo m?)
- Usiamo:
  - gli stub per testare lo stato finale (per esempio, "la proprietà counter è maggiore di zero?")
  - i mock per testare le interazioni fra oggetti (per esempio, "il metodo chiama Close sul parametro di tipo IConnection?")

Dibris

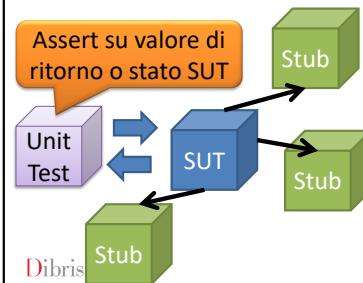
## Stub vs Mock

Concetti simili (e spesso confusi) poiché mock ≈ "stub che registra le interazioni"

Cloud: solo un mock può far fallire un test

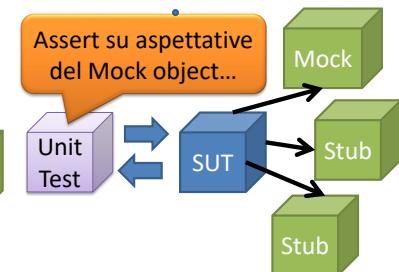
### Stub

#### State based test



### Mock

#### Behavior based test



## Cosa usare? Stub o Mock?

- Se appena possibile fare verifiche state-based  
⇒ usare solo stub
  - in questo modo i test dipendono molto meno dall'implementazione
    - più astratti (più information hiding)
    - meno *brittle* (rigidi/fragili)
- Anche quando indispensabile verifica behavior-based limitare il numero dei mock
  - Un mock, tanti stub
    - se servono tanti mock probabilmente il test fa troppo

Dibris

## Come testare BulkEmailSender?

- Lo state-based testing è piuttosto difficile: come verificare che una mail è davvero stata spedita?
  - E, come già detto, siamo proprio sicuri di volerle spedire durante il testing?!?

Dibris

## Ripasso: BulkEmailSender

```
public class BulkEmailSender {
    private readonly IEmailSender _emailSender;
    private readonly string _footer;

    public BulkEmailSender(IEmailSender emailSender, string footer) {
        this._emailSender = emailSender;
        this._footer = footer;
    }

    public void SendEmail(List<string> addresses, string body) {
        if (addresses == null)
            throw new ArgumentNullException("addresses");
        if (body == null) throw new ArgumentNullException("body");
        foreach (var a in addresses) {
            if (!this._emailSender.SendEmail(a, body + this._footer))
                throw new Exception("Cannot send email");
        }
    }
}
```

Dibris

## Testiamo (senza utilizzare Moq...)

```
[TestFixture] public class BulkEmailSenderTests {
    [Test]
    public void SendEmail_WhenEmailSenderFails_ThrowsException() {
        var bulk = new BulkEmailSender(* ??? *, string.Empty);
        var addresses = new List<string> { "a@a.com" };
        Assert.That(()=>bulk.SendEmail(addresses, string.Empty),
                   Throws.TypeOf<Exception>());
    }

    [Test]
    public void SendEmail_Passing3Addresses_Sends3Mails() {
        var bulk = new BulkEmailSender(* ??? *, string.Empty);
        var addresses = new List<string> { "a@a.com", "b@b.org", "c@c.info" };
        bulk.SendEmail(addresses, string.Empty);

        /* verifico che SendEmail su IEmailSender sia stata invocata tre volte */
    }

    [Test]
    public void SendEmail_PassingThreeAddresses_SendsOnceEach() {
        var bulk = new BulkEmailSender(* ??? *, string.Empty);
        var addresses = new List<string> { "a@a.com", "b@b.org", "c@c.info" };
        bulk.SendEmail(addresses, string.Empty);

        /* verifico che SendEmail su IEmailSender con to=b@b.org
           sia stata invocata una sola volta */
    }
}
```

Dibris

## Mock framework

- Scrivere “a mano” stub e mock è facile, ma noioso e una grossa perdita di tempo
- I mock framework sono librerie che permettono di creare mock/stub (dinamici) con pochissime linee di codice
  - Ne esistono diversi, noi useremo Moq:  
<http://code.google.com/p/moq/>  
si installa tramite NuGet

Dibris

## Testiamo usando Moq...

```
[Test]
public void SendEmail_WhenEmailSenderFails_ThrowsException() {
    var mb = new Mock<IEmailSender>();
    var bulk = new BulkEmailSender(mb.Object, string.Empty);
    var addresses = new List<string> { "a@a.com" };
    Assert.That(()=>bulk.SendEmail(addresses, string.Empty),
               Throws.TypeOf<Exception>());
}

[Test]
public void SendEmail_Passing3Addresses_Sends3Mails() {
    var mb = new Mock<IEmailSender>();
    var bulk = new BulkEmailSender(mb.Object, string.Empty);
    var addresses = new List<string> { "a@a.com", "b@b.org", "c@c.info" };
    mb.Setup(es => es.SendEmail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);
    bulk.SendEmail(addresses, string.Empty);
    mb.Verify(es => es.SendEmail(It.IsAny<string>(), It.IsAny<string>()), Times.Exactly(3));
}

[Test]
public void SendEmail_PassingThreeAddresses_SendsOnceEach() {
    var mb = new Mock<IEmailSender>();
    var bulk = new BulkEmailSender(mb.Object, string.Empty);
    var addresses = new List<string> { "a@a.com", "b@b.org", "c@c.info" };
    mb.Setup(es => es.SendEmail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);
    bulk.SendEmail(addresses, string.Empty);
    mb.Verify(es => es.SendEmail("b@b.org", It.IsAny<string>()), Times.Once());
}
```

Dibris

## Attenzione!

```
var mb = new Mock<IEmailSender>();
```

Quello che Moq chiama **Mock** è un mock-builder, il vero stub/mock (che si passa alla classe da testare) è contenuto nella proprietà **Object** del “**Mock**”

```
var bulk =  
    new BulkEmailSender(mb.Object, ...)
```



Dibris

## Setup e Verify

- **Setup** indica al mock cosa rispondere
  - Altrimenti risponde false per bool, 0 per int, ecc
    - A meno che venga costruito con MockBehavior.Strict
- **Verify** verifica che le invocazioni/accessi a proprietà/ecc siano avvenuti
- Entrambe usano le  $\lambda$  per descrivere lo scenario; nei parametri di tipo T:
  - Un valore di tipo T rappresenta se stesso
  - **It.IsAny<T>()** rappresenta qualsiasi valore
  - **It.Is<T>(predicato)** rappresenta i valori che rispettano il predicato; esempio: **It.Is<string>(s => s.Length > 3)**
    - Inoltre: **It.IsNotNull** e **It.IsRegex**

Dibris

## Returns e Throws

- **Returns** specifica il valore di ritorno, per esempio:  
`mb.Setup(...).Returns(true);`
- **Throws** specifica l'eccezione da sollevare, per esempio:  
`mb.Setup(...).Throws<Exception>();`  
`mb.Setup(...).Throws(new Exception("bla bla"));`

Dibris

## Verifica di proprietà

Non ha molto senso, ma consideriamo come esempio:

```
public interface IEmailSender {  
    bool SendEmail(string to, string body);  
    string Subject { get; set; }  
}
```

Se vogliamo verificare che la proprietà **Subject** venga letta o scritta possiamo usare...

Dibris

## Verificare che una proprietà venga...

- ...letta  
`mb.VerifyGet(es=>es.Subject);`
- ...scritta  
`mb.VerifySet(es=>es.Subject);`
- ...scritta con un certo valore  
`mb.VerifySet(es=>es.Subject = "test");`
- ...scritta con un valore che rispetta un predicato  
`mb.VerifySet(es=>es.Subject = It.Is<string>(s=>s.Length>0));`

Dibris

## Settare il valore “letto”

```
mb.Setup(foo => foo.Name).Returns("bar");

// auto-mocking hierarchies (a.k.a. recursive mocks)
mb.Setup(foo => foo.Bar.Baz.Name).Returns("baz");

// verify the setter
mb.VerifySet(foo => foo.Name = "foo");
```

Dibris

## Implementare più interfacce

- Se il mock deve implementare più interfacce, basta usare il metodo **As**:

```
var mb = new Mock<IEmailSender>();
// mb.Setup... (per il tipo IEmailSender)
var mbAsIFoo = mb.As<IFoo>();
// mbAsIFoo.Setup... (per il tipo IFoo)
Debug.Assert(mb.Object==mbAsIFoo.Object);
```

Dibris

## Callback

- Per eseguire del codice quando “qualcosa” accade
  - (per esempio, l’invocazione di un metodo)  
`int counter = 0;
mb.Setup(es => es.SendEmail(*...*))
 .Returns(true)
 .Callback(() => ++counter);`
- Si può sfruttare ad esempio per ritornare valori che dipendono dalle chiamate ricevute  
`var calls = 0;
mb.Setup(foo => foo.Bar(It.IsAny<string>()))
 .Returns(() => calls)
 .Callback(() => ++calls);`
- Le  $\lambda$  dei metodi in cascata a un Setup possono anche ricevere gli argomenti dell’invocazione corrispondente  
`mb.Setup(foo => foo.Bar(It.IsAny<string>()))
 .Returns((string s) => s.Length)
 .Callback((string s) => calls += s.Length);`

Dibris

## Sequenze (Iteratori)

Giovanni Lagorio & Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris

## Sequenze

- Tipo di dato per
  - leggere un elemento alla volta
  - passare al successivo (finché ce ne sono)
- Astrazione di liste
  - algoritmi su liste/array/dizionari agnostici sul tipo
    - es. sorting
  - efficienza per sequenze con elementi generati
    - es. potenze di b (basta memorizzare l'ultimo elemento letto)
  - gestione sequenze infinite/illimitate
    - elementi generati (o letti da sorgente esterna)
- Inadeguato per
  - accesso diretto a elemento i-esimo
  - muoversi avanti e indietro
    - solo un unico scorriamento da testa verso coda è ragionevole
      - elementi generati/provenienti da sorgente esterna potrebbero cambiare in iterazioni successive

Dibris

## IEnumerable<T>

Partiamo da come dovrebbe essere...

- `IEnumerable<T>`
  - `IEnumerator<T> GetEnumerator()`
- `IEnumerator<T>`
  - `bool MoveNext()`
  - `T Current { get; }`
  - `void Dispose()`

**Covariante**  
T1 sottotipo di T2  
implica  
`IEnumerable<T1>`  
sottotipo di  
`IEnumerable<T2>`

Nota: in Java le  
classi Iterable e  
Iterator svolgono  
lo stesso ruolo



Dibris

## In realtà, per backward compatibility ...

- `IEnumerable<T> : IEnumerable`
  - `IEnumerator<T> GetEnumerator()`
  - `IEnumerator GetEnumerator()`
- `IEnumerator<T> : IEnumerator`
  - `bool MoveNext()`
  - `T Current { get; }`
  - `void Dispose()`
  - `object Current { get; }`
  - `void Reset()` Reset riporta il cursore all'inizio
    - esiste per compatibilità con COM
    - collezione modificata ⇒ comportamento indefinito
    - da **NON** usare

Versioni non  
generiche ereditate  
da C# antidiluviano  
senza generici

estende IDisposable  
rilascia eventuali risorse  
(non in memoria)

Dibris

4

## Un esempio

- Modelliamo gli intervalli (chiusi) di interi in modo tale che siano “foreach-abili”
  - Per semplicità, ignoriamo Equals (& company)
- Per esempio,

```
var oneToFive = new IntInterval(1, 5);
foreach (var i in oneToFive)
    Console.WriteLine(i);
```
- Come li modellereste?
- Cosa deve implementare IntInterval?
- In quale categoria ricade?
  - algoritmi su liste/array/dizionari agnostici sul tipo
  - efficienza per sequenze con elementi generati
  - gestione sequenze infinite-illimitate
  - altro

Dibris

## Classe IntInterval

```
public class IntInterval : IEnumerable<int> {
    public int LowerBound { get; }
    public int UpperBound { get; }

    public IntInterval(int lowerBound, int upperBound) {
        this.LowerBound = lowerBound;
        this.UpperBound = upperBound;
    }

    public IEnumerator<int> GetEnumerator() {
        return new Enumerator(this);
    }

    IEnumerator IEnumerable.GetEnumerator() {
        return this.GetEnumerator();
    }

    private class Enumerator : IEnumerator<int> { ... }
```

l'IEnumerator deve sapere su quale collezione lavora

sempre scaricare sulla versione generica

Dibris

5

6

## Classe IntInterval.Enumerator

```
private class Enumerator : IEnumerator<int> {
    private readonly IntInterval _interval;

    private int _current;

    public Enumerator(IntInterval interval) {
        this._interval = interval;
        this.Reset();
    }

    public void Reset() { this._current = this._interval.LowerBound - 1; }

    public bool MoveNext() {
        return ++this._current <= this._interval.UpperBound;
    }

    public int Current { get { return this._current; } }

    object IEnumerator.Current { get { return this.Current; } }

    public void Dispose() { }

}
```

Dibris

## Enumerator

- La classe IntInterval.Enumerator è piuttosto semplice da scrivere, ma non particolarmente interessante
  - Nota: è come si implementerebbe in Java
- Da C# 2, il compilatore crea “automagicamente” degli Enumerator a partire da un blocco di codice che restituisce un valore dopo l’altro usando **yield return**

Dibris

7

8

## Classe IntInterval (con yield return)

- Tutto come prima tranne
  - non dobbiamo definire esplicitamente la classe `Enumerator`
  - la definizione di `GetEnumerator` diventa

```
public IEnumerator<int> GetEnumerator() {
    for (var a = LowerBound; a <= UpperBound; ++a)
        yield return a;
}
```
- Il compilatore usa questo codice per generare automaticamente la classe che implementa `IEnumerable<int>`

Dibris

## Iteratori

- Un membro si chiama **iteratore (iterator)** se
  - ha tipo di ritorno sia `IEnumerator<T>` o `IEnumerable<T>` (o varianti non generiche 
  - contiene **yield return**
- Il suo corpo si chiama **iteration-block**
- Un iteratore
  - non può contenere **return**
  - non può avere parametri passati per riferimento
  - può contenere **yield break**, che termina la sequenza
- L'invocazione di un iteratore **non** provoca l'immediata esecuzione del codice, ma la costruzione di un oggetto enumerator (o enumerable) per la sequenza corrispondente
  - Una possibile implementazione è tramite macchina a stati

Dibris

9

10

## Esempio di iteratore `IEnumerable`

```
public class Program {
    public static IEnumerable<int>
        AnInterval(int lowerBound, int upperBound) {
        for (var a = lowerBound; a <= upperBound; ++a)
            yield return a;
    }
}
Il compilatore genera entrambe le classi: l'IEnumerable (ex IntInterval) e il relativo IEnumerator (ex IntInterval.Enumerator)
```

```
public static void Main(string[] args) {
    foreach (var i in AnInterval(1, 5))
        Console.WriteLine(i);
    // ...
}
```

Esercizio: modificare `AnInterval` per rappresentare gli intervalli  $[n, \infty)$  (modulo limiti di rappresentazione degli interi, ovviamente)

Dibris

## Esempio di uso per generalizzare tipo

- Vogliamo generalizzare la verifica che un array di interi è ordinato...

```
public static bool IsOrdered(int[] a) {
    for(var i=1; i<a.Length; ++i)
        if (a[i-1]>a[i]) return false;
    return true;
}
```
- Cambiando tipo da array a `IEnumerable<int>` si perde l'accesso diretto  

```
public static bool IsOrdered(IEnumerable<int> sequence) {
    var enumerator = sequence.GetEnumerator();
    if (!enumerator.MoveNext()) return true;
    int previous = enumerator.Current;
    while (enumerator.MoveNext()) {
        int current = enumerator.Current;
        if (previous>current) return false;
        previous = current;
    }
    return true;
}
```
- Esercizio (facile): generalizzare ulteriormente rispetto alla relazione di ordine e al tipo base degli elementi della sequenza
  - ad esempio deve funzionare (anche) con un `IEnumerable<string>` e l'ordine lessicografico o un `IEnumerable<double>` e l'ordine decrescente

Dibris

11

12

# LINQ

## Language INtegrated Query

Giovanni Lagorio & Maura Cerioli

Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris

 Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Accesso a DB via codice - un esempio

```
// ...
string firstName = Console.ReadLine();
using ( IDbConnection conn =
new SqlConnection("...")) {
conn.Open();
var cmd = conn.CreateCommand();
cmd.CommandText =
"select * from Students where FirstName=@"
+ firstName + " order by City";
using (IDataReader reader = cmd.ExecuteReader()) {
while (reader.Read()) {
Console.WriteLine("Surname:" + reader.GetString(1));
Console.WriteLine("Birth:" + reader.GetDateTime(3));
}
}
// ...
}
```



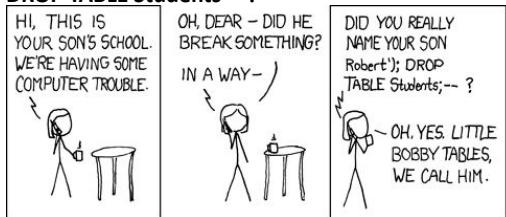
 Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

2

## xkcd insegna...

Cosa succede se l'utente inserisce, per esempio,  
'; DROP TABLE Students -- ?



2021: [GabLeaks](#) – ...SQL injection that allowed the attackers to obtain 70 GB of data.

2020: [Sophos XG Firewall](#) – ...SQL injection in their product ...

2019: [Bulgarian VAT service](#) – ...SQL injection attack, records ...for 5 million citizens ...became accessible via underground forums.

<https://www.invicti.com/learn/sql-injection-sqli/>

Dibris

 Dibris

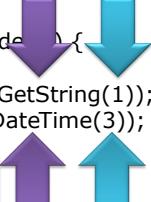
Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Ma ci sono anche altri problemi...

```
// ...
string firstName = Console.ReadLine();
using ( IDbConnection conn = new SqlConnection("...")) {
conn.Open();
IDbCommand cmd = conn.CreateCommand();
cmd.CommandText =
"select * from Students where FirstName=@"
+ firstName + " order by City";
using (IDataReader reader = cmd.ExecuteReader()) {
while (reader.Read()) {
Console.WriteLine("Surname:" + reader.GetString(1));
Console.WriteLine("Birth:" + reader.GetDateTime(3));
}
}
// ...
}
```

Il problema è che l'interazione codice-DB  
non è veramente tipato

- si usano stringhe al posto di SQL-query
- si usano tabelle di struttura ignota per il risultato



 Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

4

## LINQ

- LINQ: Language INtegrated Query

– porta il concetto di query a livello di linguaggio

- Vantaggi

– Uniformità ed Estensibilità

– Query a livello di linguaggio

- type checking

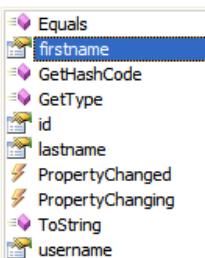
- nell'IDE:

- Intellisense

- Refactoring

- non serve escaping esplicito → niente più (SQL) injection

```
var q = from user in context.Users  
        where user.lastname  
        select user.
```



## Cos'è LINQ?

- Uno strumento per interrogare in modo uniforme sorgenti di dati arbitrari

Oggetti in memoria    DB relazionali    Modelli E/R

Documenti XML    Spreadsheet

...altro...

- LINQ *non è* ORM, ma è un'ottima interfaccia per interrogarli

- LINQ è tutta "compiler magic"

– codice compilato compatibile con versione precedenti alla sua introduzione

- Analogo agli stream (introdotti in Java 8)

## LINQ to Objects

```
string?[] strings = { "g", null, "disi", "gENoVa"};  
  
var query = from s in strings  
            where s.Length>2  
            orderby s descending  
            select s.ToUpper();  
  
var query2 = from s in query  
            where s.StartsWith("G")  
            select s.Length;  
  
strings[1] = "a";  
  
int sum = query2.Sum();  
  
Console.WriteLine("Sum={0}\n", sum);  
foreach (string s in query) Console.WriteLine(s);
```



Sum=6  
GENOVA  
DISI

## ToList/ToArray

```
var q = from user in context.Users  
        orderby user.lastname  
        select user;  
// esegue 10 volte la query q sul db  
for (int a = 0; a < 10; ++a)  
    foreach (var x in q) // ...  
  
List<User> l = q.ToList();  
// esegue una sola volta sul DB  
for (int a = 0; a < 10; ++a)  
    foreach (var x in l) // ...
```

## Proiezioni e Tipi anonimi

```
var q = from user in context.Users
    orderby user.lastname
    select new { user.firstname, user.lastname,
        fullName = user.firstname + " " + user.lastname } ;
foreach (var x in q)
    Console.WriteLine("nome={0}, cognome={1}, nome
        completo={2}", x.firstname, x.lastname, x.fullName);
```

Qui **var** diventa obbligatorio! Che tipo ha x?

Un **tipo anonimo**, sappiamo solo che ha un campo `firstname` di tipo string, ecc.

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

9

## Esempio con groupby

```
string[] strings = {"Gio", "DISI", "Genova", "Dilbert", "Zorro"};
var query = from s in strings
    group s by s[0] into sameFirstLGroup
    orderby sameFirstLGroup.Count() ascending
    select new { FirstLetter = sameFirstLGroup.Key,
        Words = sameFirstLGroup };
foreach (var g in query) {
    Console.WriteLine("FirstLetter = {0}",
        g.FirstLetter);
    Console.WriteLine("Words = ");
    foreach (var w in g.Words)
        Console.WriteLine("\t{0}", w);
}
```

FirstLetter = Z  
Words =  
Zorro  
FirstLetter = G  
Words =  
Gio  
Genova  
FirstLetter = D  
Words =  
DISI  
Dilbert

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

10

## Cos'è davvero una query?

- Qualcosa che “trasforma” una sequenza in un'altra
  - `string[] strings = { "qui", "quo", "qua" };`
  - `var q = from s in strings select s.Length;`
  - sequenza = “roba `foreach`-abili”: `IEnumerable<T>`
    - `IEnumerable<int> q1 = from s in strings select s.Length;`
    - `IEnumerable<bool> q4 = from i in (from s in strings select s.Length)
 select i > 0;`
- Compiler magic trasforma “keyword sql” in chiamate di metodo
  - `var q = from user in /* un certo IEnumerable<User> */
 where user.firstname == "Giovanni"
 select user;`
  - `var q = /* ... un IEnumerable... */
 .Where(user => user.firstname == "Giovanni")
 .Select(user => user);`
- Ma ci sono Where e Select in `IEnumerable`????

Dibris



Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

11

## EXTENSION METHODS

Dibris

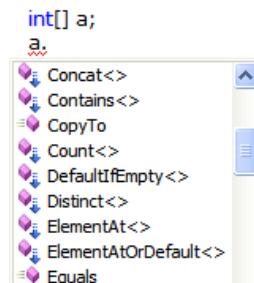


Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

## Extension methods

- E' possibile aggiungere metodi (implementati) a una qualunque classe/**interfaccia**
- Visti dall'intellisense (quasi) come gli altri
- Meccanismo molto potente, ricorda l'Aspect Oriented Programming
  - sembra fin troppo facile abusarne



## Extension methods: come?

- Si usa **this** come modificatore del primo parametro di un metodo statico  
**public static int m(this T x, int k, ...)**
- Il compilatore trasforma le chiamate **expr.m(1, 2, 3)** dove **expr** ha tipo **T** (e non c'è nessun metodo migliore) in:  
**MiaClasseStatica.MioExtMethod(expr, 1, 2, 3)**

## Esempio di estensione (e di abuso ☺)

```
public static class MyStringExtension {  
    public static int TwiceLength(this string s) {  
        return s.Length * 2;  
    }  
    ...  
    Console.WriteLine("ciao".TwiceLength());  
    // stampa 8, trasformata dal compilatore in:  
    Console.WriteLine(MyStringExtension.  
        TwiceLength("ciao"));
```

## EXTENSION METHODS



## Linq to Objects

- `var q = /* ... un IEnumerable... */  
.Where(user => user.firstname == "Giovanni")  
.Select(user => user);`
- Where e Select sono extension method con argomenti di tipo IEnumerable (parametro this) e delegate
- Può funzionare per interrogare un database?
  - lavorerebbe in memoria
  - sempre inefficiente
  - spesso impossibile (capienza della memoria)
- Serve un modo di far eseguire sul DB (più in generale su chi gestisce la sorgente di dati)



Dibris

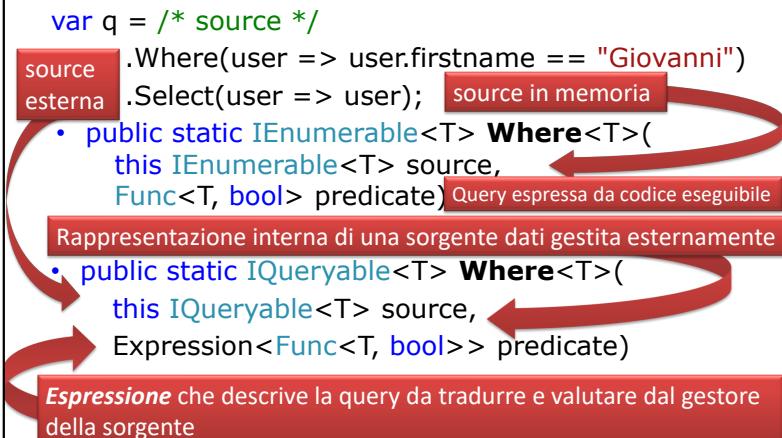


Dibris

Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

17

## IQueryable<T>



Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

18

## Chi è IQueryable<T>?

- Estensione di IEnumerable
- Segue la stessa logica di IEnumerable per la variante generica



Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

19

da "04 Higher-Order Types"

## Lambda Expressions (Closures)

- Assimilabili a λ-astrazioni del λ-calcolo, ovvero definizioni di funzioni “inline”
- Esempi:
  - $(int x, int y) => x + y$   
rappresenta la funzione che somma x e y
  - $a => a * a$   
rappresenta la funzione quadrato
  - $(x, y) => \{ if (x < y) return x; else return y; \}$   
rappresenta la funzione che restituisce il minimo
- Il tipo dei parametri può essere inferito
  - Se il parametro è uno solo, non servono le parentesi
- λ-expression assegnate (o passate come argomento) a un tipo delegate si comportano come metodi anonimi
  - Le λ **NON** sono delegate, ma sono convertibili in delegate

Dibris



Dipartimento di Informatica,  
Bioingegneria, Robotica e Ingegneria dei Sistemi

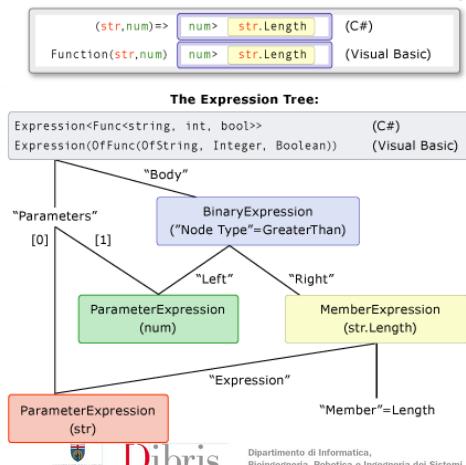
20

## Lambda Expression (cont.)

Una lambda-expression, di per sé, non è niente (!) ma può essere convertita in:

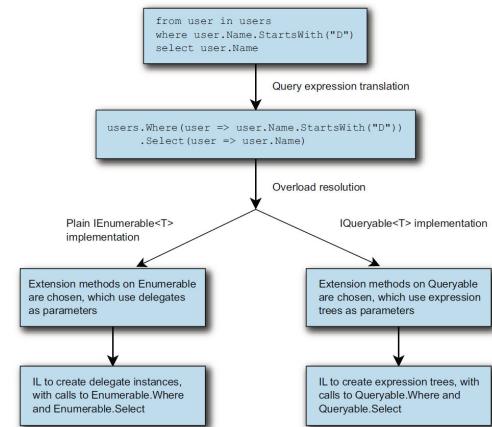
- un delegate, in sostanza, **in codice .NET che valuta la funzione**, oppure in
- un Expression tree, (**codice .NET che costruisce una struttura dati** che rappresenta l'espressione, che il provider trasformerà in una qualche forma più furba per l'esecuzione (per esempio, una query SQL per un DB relazionale)

## Esempio: expression tree per $(\text{str}, \text{num}) \Rightarrow \text{num} > \text{str.Length}$

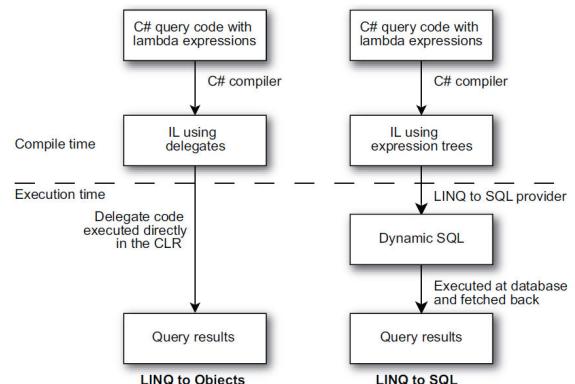


## Risolvendo l'overloading...

Dal libro  
«C# in Depth»  
di Jon Skeet

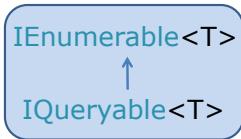


## Di conseguenza



Dal libro «C# in Depth» di Jon Skeet

## IEnumerable<T> o IQueryable<T>?!?



Quindi? Dipende, comunque un `IEnumerable` si converte facilmente con `AsQueryable`

Non è (sempre) un cast:

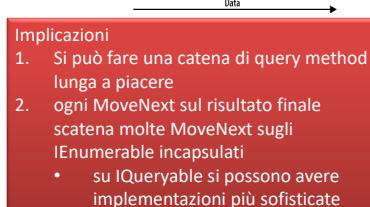
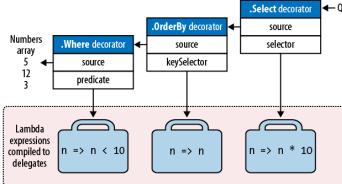
```
int[] ints = { 1, 2, 3 };
IQueryable<int> iq = ints.AsQueryable();
```

## Limiti dell'approccio

- Alcune espressioni possibili solo con LINQ to Objects, per esempio in LINQ to SQL:  
`var q = from user in context.Users  
 where user.MyMethod() > 0 /* boom */  
 select user.MyMethod() /* ok */;`
  - Comunque, molti (tutti?) metodi standard di numeri/stringhe/date vengono tradotti
  - Con EF, `SqlFunctions` (e con LINQ to SQL, `SqlMethods`) fornisce funzioni SQL chiamabili via CLR
- Non mi sembra furbo, ma limitandosi a LINQ to Objects è possibile *modificare* gli oggetti visitati

## Query Method applica *Decorator*

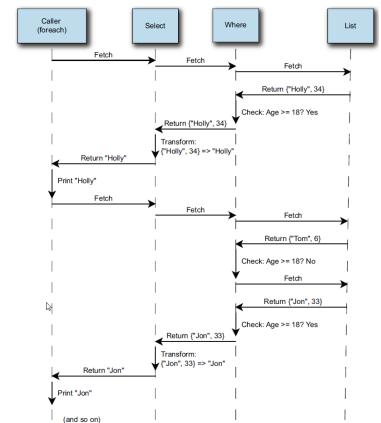
```
var query = new int[] { 5, 12, 3 }
    .Where (n => n < 10)
    .OrderBy (n => n)
    .Select (n => n * 10);
```



Dal libro: «C# 5 in a Nutshell» di Joseph Albahari e Ben Albahari

## Deferred execution and streaming

```
var adultNames =
    from p in people
    where p.Age >= 18
    select p.Name;
foreach(var n in adultNames) {
    //...
```



Dal libro «C# in Depth» di Jon Skeet

## «Query syntax» vs Query Methods

- De gustibus...
  - La sintassi «alla SQL» viene comunque tradotta in (catene di invocazioni di) Query Method

```
int[] a = {1, 2, -10, 0};var pos = from i in a    where i > 0    select i;var squares = from i in a    select i*i;
```
  - Molti preferiscono usare direttamente i Query Method

```
int[] a = {1, 2, -10, 0};var pos = a.Where(i => i > 0);var squares = a.Select(i => i*i);
```

Dibris



29

## Entity Framework Core

Maura Cerioli  
Dipartimento di Informatica, Bioingegneria, Robotica e  
Ingegneria dei Sistemi  
University of Genova  
TAP 2022-23

Dibris

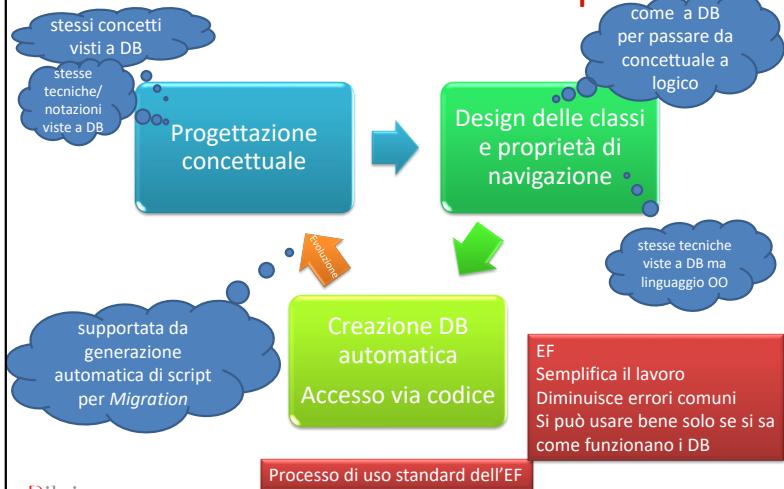
## Entities Framework

- È un ORM
  - per gestire i dati si pensa/lavora in termini di classi/oggetti/property, non tabelle/righe/colonne
  - traduzione fatta in automatico dal framework
- Permette di
  - lavorare a livello di progettazione concettuale (modello E/R , ereditarietà, relazioni molti-a-molti, ...)
  - descrivere il modello attraverso classi
    - generare automaticamente lo schema del DB corrispondente
    - creare/aggiornare il DB
  - fare operazioni CRUD sulla rappresentazione a oggetti
    - e.g. per interrogare i dati usare LINQ
  - salvare le modifiche sul/interrogare il DB via query autogenerate [all'interno di una transazione]

Dibris

2

## Studiare DB non serve più?

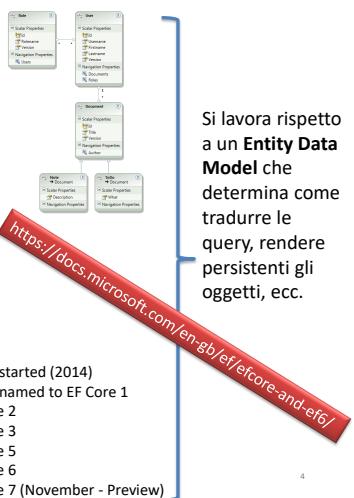


Dibris

3

## Storia - Panoramica

- .NET 3.5 (2008, EF o EF3.5)
  - Unica modalità di lavoro: **DataBase First**
    - Si mappa un modello E/R su un DB esistente
    - Descrizione del modello in XML (file EDMX)
      - Il designer in VS «nasconde» XML
    - Le entità devono estendere EntityObject
    - ObjectContext API
- .NET 4 (2010, EF4)
  - **Model First**
    - Si genera il DB a partire da un modello E/R
    - Supporto ai POCO (Plain Old CLR Object)
- Fra .NET 4 e .NET 4.5 (2011, EF4.1)
  - **Code First** (Code Based sarebbe più appropriato...)
    - Si basa tutto sul codice (no VS designer, no XML)
    - Convention over Configuration (AKA Coding by Convention)
  - DbContext API
  - Migration (EF 4.3, 2012)
- EF 5 (2012)
  - independent (from .NET) release
- EF 6 (2013)
  - open source project!
- EF 6.1 (2014)
  - EF 7 project started (2014)
  - 2016 EF 7 renamed to EF Core 1
  - 2017 EF Core 2
  - 2019 EF Core 3
  - 2020 EF Core 5
  - 2021 EF Core 6
  - 2022 EF Core 7 (November - Preview)



Si lavora rispetto a un **Entity Data Model** che determina come tradurre le query, rendere persistenti gli oggetti, ecc.



liberamente tratto da

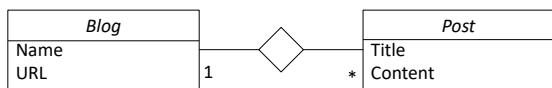
<https://docs.microsoft.com/en-us/ef/core/get-started/overview/first-app>

## UN ESEMPIO INIZIALE

## Installazione di EF

- Si usa Nuget
- Versione da scegliere EF6 Core
  - EF7 core non è ancora stabile
- Pacchetto da scegliere Microsoft.EntityFrameworkCore.XYZ dove XYZ è il tipo di DB che volete usare
  - ad esempio XYZ = SqlServer, o XYZ = Sqlite
  - più pacchetti possono coesistere se volete usare più tipi di DB
  - sono supportati anche Postgres e alcuni DB nosql
  - automaticamente viene installato anche Microsoft.EntityFrameworkCore

## Modello Concettuale



Ogni Blog ha un nome, si trova ad uno specifico URL e possiede una collezione di Post di cardinalità arbitraria  
Ogni Post appartiene a uno e un solo Blog e ha un titolo e un contenuto

## Entità del Modello

```
public class Blog {  
    public int BlogId { get; set; }  
    public string Name { get; set; }  
    public string URL { get; set; }  
    public List<Post> Posts { get; set; } = new();  
}  
  
public class Post {  
    public int PostId { get; set; }  
    public string Title { get; set; }  
    public string Content { get; set; }  
    public Blog Blog { get; set; }  
    public int BlogId { get; set; }  
}
```

chiavi primarie aggiunte per rendere efficienti i join  
relazione 1-n come proprietà di navigazione  
chiave esterna per semplificare implementazione

Dibris

## Contesto

### Gestore della connessione al DB

```
using System.Data.Entity;
```

```
// ...
```

```
public class BloggingContext : DbContext {  
    public DbSet<Blog> Blogs { get; set; }  
    public DbSet<Post> Posts { get; set; }  
}
```

```
protected override void
```

```
OnConfiguring(DbContextOptionsBuilder options) =>  
    options.UseXYZ(Connection string);
```

tipo del DB

indirizzo del DB+credenziali

rappresentazione delle entità del DB

collegamento con il DB

Dibris

9

## Creazione del DB

- Senza supportare evoluzione dello schema del DB
  - Ha senso solo se non si vogliono/possono mantenere i dati quando si cambia schema

```
using (var c = new BloggingContext()) {  
    c.Database.EnsureDeleted();  
    c.Database.EnsureCreated();  
}
```

cancella il DB (se c'è)  
qualsiasi schema abbia

SE non esiste un DB con qualsiasi schema lo crea

- Supportando evoluzione  $\Rightarrow$  Migration
  - Modalità corretta dalla prima release
  - Ne parliamo dopo

La property Database di un contesto fa da Facciata [Facade] verso il DB

Dibris

10

## Lettura/Scrittura

```
using (var db = new BloggingContext()) {  
    // Create and save a new Blog  
    var blog = new Blog {Name = "First Blog", Url = "http://x.com/y"};  
    db.Blogs.Add(blog);  
    db.SaveChanges();  
    // Read  
    var blog1 = db.Blogs.OrderBy(b => b.BlogId).First();  
    // Update  
    blog1.Url = "https://devblogs.microsoft.com/dotnet";  
    blog1.Posts.Add(new Post {Title = "Hello!", Content = "BlaBla" });  
    db.SaveChanges();  
    // Delete  
    db.Remove(blog1);  
    db.SaveChanges();  
}
```

Rende permanente i cambi sul DB

Dibris

11

## DIVING DEEPER

Dibris



12

## DbContext – Il Bandolo della Matassa

- Interfaccia verso lo strato di gestione dei dati
  - da ora innanzi un DB relazionale
  - (EF Core supporta altre modalità: DB-NoSQL, files, memory...)
- Incapsula
  - Modello dei dati
    - Entità come classi ≈ [Schema delle] Tabelle nel DB
    - Entità hanno una chiave primaria
      - si possono anche avere entità senza chiave che corrispondono a risultato di query/viste e non supportano scrittura, ma non ne parliamo
  - Gestione del DB
    - Creazione/aggiornamento/cancellazione
    - Connessioni
  - Gestione delle entità
    - Entità come istanze ≈ Righe nella tabella (della classe) nel DB
    - Realizza/supporta vari design pattern
      - Identity <https://martinfowler.com/eaCatalog/identityMap.html>
      - Repository <https://martinfowler.com/eaCatalog/repository.html>
      - Unit of work <https://martinfowler.com/eaCatalog/unitOfWork.html>

Dibris

## Modello dei Dati - Schema del DB

- Generato da code convention + indicazioni esplicite
- Code convention
  - Entity discovery
    - si parte dalle entità esplicitamente usate in una property del contesto

```
public DbSet<Blog> Blogs { get; set; }
```
    - si chiude per transitività aggiungendo le entità riferite
      - entità (classi con chiave primaria)
      - usate in proprietà di navigazione da una entità già scoperta
  - Alle entità vengono automaticamente aggiunte le tabelle per le relazioni n-n

Dibris

14

## Modello dei Dati Indicazioni Esplicite sullo Schema del DB

Due modi

- Validation Attribute
  - Custom attribute che espongono un metodo di validazione
  - Esprimono vincoli sulle entità
    - ⇒ evitate implementation leak
    - ⇒ da usare per esprimere vincoli sul modello del dominio
  - EF Core li usa per generare vincoli sul DB (quando possibile)
    - EF NON core li usa anche per validazione lato client
    - potete usarli per implementare facilmente la validazione dei parametri
- Fluent API
  - Più espressivi dei validation attribute
  - Usate nell'override del metodo OnModelCreating di DbContext
  - Alcuni preferiscono usare solo Fluent API

Dibris

15

## Validation Attribute - Esempi

- Required (su properties)
  - ha senso solo per tipi nullabili
  - si traduce in not null
- MaxLength(n) o StringLength(n) (su properties di tipo stringa)
  - il tipo sul DB è nvarchar(n)
- Index(prop1,...propn,IsUnique=...,Name=...) (su entità)
  - crea un indice usando le colonne indicate
  - se IsUnique=true introduce vincolo unique
- Molti attributi non sono esprimibili su DB
  - MinLength(n)
  - RegularExpression
  - CreditCard
- Alcuni attributi non è opportuno usarlo (mischiano DB e Dominio)
  - Table (stabilisce il nome della tabella associata a un'entità)
  - Column (stabilisce il nome della colonna associata a una property)

Dibris

## Fluent API - Esempi

```
protected override void OnModelCreating(ModelBuilder modelBuilder) {  
    //Write Fluent API configurations here  
    modelBuilder.Entity<Student>().ToTable("Studenti"); //cambia nome della tabella  
  
    //Property Configurations  
    modelBuilder.Entity<Student>()  
        .Property(s => s.StudentId)  
        .HasColumnName("Id") //cambia nome della colonna  
        .HasDefaultValue(0) //default clause  
        .IsRequired(); //not null  
}
```

Dibris

17

## Ereditarietà

<https://docs.microsoft.com/en-us/ef/core/performance/modeling-for-performance#inheritance-mapping>

- Concetto assente in DB relazionali
  - né notazione E/R standard
  - ma molto importante i OO
- A BD avete imparato a tradurla
  - una sola tabella per tipo radice e tutti i suoi discendenti
    - colonna in più per discriminare il tipo
    - molti valori nulli nelle colonne che non appartengono al particolare tipo
  - una tabella per ciascun tipo
    - nelle tabelle dei sottotipi
      - campi specifici
      - FK verso l'elemento del tipo padre con gli altri campi
    - query che usano sia campi specifici che campi del padre richiedono join
  - a seconda del carico di lavoro atteso una scelta può essere migliore dell'altra

EF non considera sottotipi  
nel trovare le entità

EF Table Per Hierarchy  
Il default

EF Table Per Type  
Basta indicare il nome  
delle tabelle per ogni tipo

Dibris

## Chiavi Primarie

- La chiave primaria è obbligatoria per essere un'entità
- Può essere definita
  - per convenzione
    - una property pubblica di nome Id o <nome dell'entità>Id
  - usando custom attributes [Key] (solo singola colonna)
  - usando fluent API (su modelBuilder.Entity<MyEntity>())  
HasKey(e => e.MyKeyProperty);
  - HasKey(e => new{e.col1, e.col2});
- Possono avere qualsiasi tipo primitivo
  - non tutti i DB supportano chiavi di tipo arbitrario
  - può essere necessario fornire metodo di conversione di tipo
- Chiavi primarie multicolonne sono possibili ma poco efficienti
  - prassi standard: aggiungere Id di tipo autogenerato (int)

Dibris

19

## Chiavi Esterne - Convenzioni

- Servono (nei DB) per rappresentare le relazioni
- Vengono implicitamente introdotte da EF quando ci sono proprietà di navigazione fra entità
  - property con tipo di ritorno una classe entità (relazioni 1-?)
  - property con tipo di ritorno collection di una classe entità (relazioni n-?)
- Situazione ideale
  - in A una proprietà di navigazione verso [collezione di] B
  - in B una proprietà di navigazione verso [collezione di] A
  - EF riconosce che sono una inversa dell'altra e determinano una relazione (1-1, 1-n, n-1, n-n a seconda dei casi)
- Se c'è solo la proprietà di navigazione in A verso [collezione di] B EF inventa l'altra metà
  - sceglie la variante più semplice: 1-n o 0-n a seconda della nullabilità
  - ovviamente non si può navigare dove non c'è la proprietà, ma per il resto è come se ci fosse

Dibris

20

## Chiavi Esterne Molteplicità Singola

Se l'entità A ha una proprietà di navigazione verso l'entità B

- è buona norma mettere anche una property con la chiave esterna es.  

```
public Blog Blog { get; set; }
public int BlogId { get; set; }
```
- Non è obbligatorio ma è consigliato
- Convenzione: chiave stesso nome della chiave primaria dell'entità riferita
- Si può scegliere nome arbitrario e annotare
  1. `[ForeignKey(nameof(BlogId))]`  
`public Blog Puffo { get; set; }`
  2. `[ForeignKey(nameof(Blog))]`  
`public int Puffo { get; set; }`
  3. si può anche annotare la chiave primaria nell'entità riferita
- Si può scegliere nome arbitrario e usare fluent API (nell'ipotesi 2)  
`.HasOne(p => p.Blog)
.WithMany(b => b.Posts)
.HasForeignKey(p => p.Puffo);`
- Se la chiave non è nullabile viene imposto il vincolo ondelete cascade
  - se ci sono cicli questo può introdurre errori

Dibris

21

## Chiavi Esterne - Personalizzazioni

- In caso di ambiguità (o quando si vuole avere controllo più fine) si possono specificare vari aspetti
  - ne vediamo un paio, per gli altri RTFM
- Metodi delle fluent API per configurare le inverse
  - `HasOne(property)` crea un oggetto per configurare proprietà di navigazione con cardinalità 0-1
  - `HasMany(property)` crea un oggetto per configurare proprietà di navigazione con cardinalità n
  - sugli oggetti per configurare proprietà di navigazione per definire le proprietà di navigazione inverse si possono invocare
    - `WithOne(inverse property)`
    - `WithMany(inverse property)`
  - si può configurare anche la propagazione della cancellazione
    - `OnDelete(DeleteBehavior.Xyz)`
    - discorso delicato ne parleremo dopo

Dibris

22

## Chiavi Alternative

- Si possono definire usando fluent API `HasAlternateKey` stesse tipologie di `HasKey`
- Sono `readonly`
- Si possono usare come foreign key (FK)
- Vengono introdotte automaticamente se si usano come FK  
es.  
`.HasForeignKey(p => p.BlogUrl)
.HasPrincipalKey(b => b.Url);`
- A livello di DB viene generato un indice unique

Dibris

23

## Indici e Vincoli Unique

- Per rendere più efficienti le query si usano gli indici
    - non necessariamente corrispondono a chiavi alternative
  - In EF Core si possono introdurre
    - usando attributo sulla classe che rappresenta l'entità  
[Index(nameof.FirstName), nameof.LastName)]

```
public class Person {  
    public int PersonId { get; set; }  
    public string FirstName { get; set; }  
    public string LastName { get; set; }  
}
```
    - usando Fluent API  
modelBuilder.Entity<Product>().HasIndex(x => new { x.Name, x.ProductId });
  - Per esprimere il vincolo di unicità per una collezione di property si introduce il vincolo richiedendo che sia unique
    - usando attributo  
[Index(nameof.FirstName), nameof.LastName], IsUnique = true]]
    - usando Fluent API  
modelBuilder.Entity<Product>().HasIndex(x => new { x.Name, x.ProductId }).IsUnique();
  - A livello di DB viene generato un indice [unique]
  - EF Core distingue logicamente fra
    - chiavi alternative
    - indici unique
- nel campo di DB non si usa questa distinzione

Dibris

## Interazione con il DB – Creazione – Migration

<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=vs>

- Usando il modello dei dati un DbContext può creare il DB
    - direttamente come visto prima
    - usando Migration
1. Installare Microsoft.EntityFrameworkCore.Tools 5.0.12
    - potete seguire le istruzioni o usare direttamente nuget
    - rende disponibile la Package Manager Console
  2. Nella PM Console Add-Migration <Nome>
    - la prima volta aggiunge al Progetto l'infrastruttura per Migration
      - vedi cartella Migration
    - crea la Migration con il nome che avete scelto
  3. Nella PM Console Update-Database
    - effettivamente crea/aggiorna il database

Dibris

cambio  
modello

25

## Interazione con il DB – Migration - Script

<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=vs>

- Usare la PM Console ha senso in sviluppo ma non è adeguato per deployment/system version update
- Si possono creare degli script  
Script-Migration [<from-migration>] [<to-migration>]
  - comandi da usare nella PM Console
  - genera script SQL per modificare il DB dallo stato descritto dalla from-migration allo stato descritto dalla to-migration
    - default per from-migration = DB vuoto
    - default per to-migration = migration più recente
    - se to-migration è più vecchia di from-migration equivale a un roll-back
  - lo script compare in una nuova finestra
- Gli script generati possono essere distribuiti e usati (dagli admin) per fare le migrazioni (nei momenti e modi opportuni)
  - sconsigliato eseguirli da codice perché  
poco controllo ⇒ alto rischio  
pensate ad esempio a instance multiple che eseguono concorrentemente
  - attenzione a potenziali perdite di dati
- Se non si è sicuri dello stato del DB si possono usare script *idempotenti* che non applicano le modifiche che non servono
  - Script-Migration -Idempotent

Dibris

## Interazione con il DB – Gestione dati

- Le property di tipo DbSet<E> rappresentano in memoria le tabelle del tipo E
- Le property di tipo DbSet<E> rappresentano in memoria gli elementi del tipo E che servono al developer per effettuare una specifica operazione
  - siccome sono di tipo E possono avere un corrispettivo sul DB
  - se li sto creando sul DB non ci sono ancora
  - se qualcuno li ha cancellati sul DB non ci sono più
  - sul DB di solito ci sono moooooooooolti più dati

Dibris

27

## 3 Design Pattern per Capire i DbContext

Dibris



28

## Problema Generale

- Identity, Unit of Work, Repository nel contesto di programmi per gestire dati con una controparte permanente
  - entità
- Esigenze
  - lavorare su entità in memoria *dimenticando* che esistono DB e duplicazioni (meglio dei due mondi)
  - quando un'operazione è conclusa rendere permanenti tutti i cambiamenti (o nessuno)
  - incapsulare gli accessi al DB per mantenibilità/flessibilità

Dibris

29

## Identity Map

- Se chiedo due volte un'entità al DB ottengo due oggetti che rappresentano la stessa entità
  - aggiornamenti degli oggetti indipendenti
    - potenziali update inconsistenti sul DB
    - comportamenti inattesi
- Soluzione
  - layer IM fra codice applicativo e DB
  - il codice applicativo chiede le entità all'IM
    - se IM non conosce l'entità la chiede al DB, crea l'oggetto corrispondente e se lo salva
    - se IM conosce l'entità restituisce l'oggetto che si era salvato
  - ogni entità è rappresentata da un solo oggetto

Dibris

IM → DbSet<E>

30

## Repository

- Spargere il codice di accesso del DB nel codice applicativo rende difficile aggiornamenti
  - es. cambio del DB, del suo schema...
- Soluzione
  - layer R fra codice applicativo e DB
  - R espone le operazioni CRUD, find assortite per i tipi OO che servono al codice applicativo
  - Il codice applicativo parla solo con R
  - R implementa le operazioni parlando con il DB
    - cambi sul DB sono incapsulati in R

quasi...Incapsula  
DB ma espone EF

R → DbSet<E>

Dibris

31

## Unit of Work

- Una singola operazione per l'utente finale può corrispondere a operazioni su varie entità
  - es matrimonio coinvolge entrambi i coniugi
  - bisogna mantenere la consistenza
- Soluzione a livello di DB transazione
  - non banale da implementare da codice
  - non tutti si ricordano di farla
- Soluzione
  - introdurre un oggetto UoW che gestisce le entità coinvolte
    - tiene traccia di modifiche, creazioni, cancellazioni
    - su richiesta esplicita rende permanenti le modifiche in una transazione
  - se una operazione sul DB fallisce falliscono tutte
    - molto importante avere in UoW SOLO entità indispensabili per l'operazione
      - se ne può fallire perché fallisce aggiornamento di qualcosa di non rilevante
      - se va a buon fine modifica lo stato di entità che magari non devono ancora essere aggiornate

UoW → DbContext

Dibris

32



Dibris

33

## DbContext ⇌ Pattern

- DbContext implementa i 3 pattern ⇒
- Ogni interazione con il DB deve passare da un DbContext
- Si usa un DbContext per ogni operazione utente
  - siccome è un IDisposable bisogna garantire il rilascio delle risorse anche in caso di eccezione
    - usate sempre using, è il modo più facile
- Si attaccano al contesto solo gli oggetti indispensabili all'operazione
  - se la stessa entità viene identificata in due modi, ad esempio attraverso diverse relazioni e FK, l'oggetto è sempre lo stesso
- All'interno dello scope di visibilità del contesto si inserisce la logica business
  - le operazioni CRUD richieste si fanno sulle entità attaccate al contesto
  - per rendere permanenti i cambiamenti si usa il metodo SaveChanges

Dibris

34

## Tracking di Entità

- Gli oggetti/entità "controllati" (tracked) da un context sono detti **Attached** (al db-context)
  - Un oggetto può essere controllato da un solo contesto
  - Un grafo di oggetti è sempre completamente **Attached** o completamente **Detached**
    - se e1 è in relazione con e2 o entrambe sono attached al contesto c o non lo è nessuna delle due
- Ognuno di questi è encapsulato in un EntityEntry< TEntity > che
  - si può recuperare tramite il metodo **Entry< TEntity >** del DbContext
  - espone proprietà (read-only) per la gestione dell'entità
    - DbPropertyValues CurrentValues
    - DbPropertyValues OriginalValues
    - EntityState State
    - ...

Dibris

35

memorizza il tipo di query da fare (if any)

in caso di modifiche per creare le query di update

## Entity State

<https://docs.microsoft.com/en-gb/ef/core/change-tracking/>

Ogni entità è sempre in uno dei cinque stati definiti dall'enumeration  **EntityState**

- aggiornato automaticamente sulla base delle azioni fatte in memoria
- determina le query generate da EF ed eseguite durante SaveChanges

	Stato	Rispetto a ultima interazione con DB	Azione nel prossimo SaveChanges
Attached	Unchanged	Nessuna modifica	Nessuna azione
	Added	Creato nuovo oggetto	INSERT
	Deleted	Cancellato oggetto esistente	DELETE
	Modified	Modificato oggetto esistente	UPDATE
Detached = non gestito			

Dibris

36

## Debugging

<https://docs.microsoft.com/en-us/ef/core/change-tracking/debug-views>

- DbContext traccia gli oggetti attraverso la sua property ChangeTracker
- ChangeTracker.DebugView produce un output leggibile dagli umani con l'elenco di tutte le entità tracciate
  - ChangeTracker.DebugView.ShortView elenca solo entità, PK, FK, State
  - ChangeTracker.DebugView.LongView elenca anche tutte le proprietà (incluse quelle di navigazione)
- Il risultato è una stringa che si può stampare
  - Console.WriteLine(ChangeTracker.DebugView.ShortView)
  - Debug.WriteLine(ChangeTracker.DebugView.ShortView)

Dibris

37

## Logging

<https://docs.microsoft.com/en-us/ef/core/logging-events-diagnostics/simple-logging>

- EF Core offre diversi meccanismi per il logging
- Per il debug si usa Simple Logging
- Basta configurarlo al momento della configurazione del DbContext
  - nell'override di OnConfiguring  
optionsBuilder.LogTo(action);
  - action è una Action<T> che può accettare una stringa
    - ad esempio Console.WriteLine o Debug.WriteLine
  - EF Core invocherà action per ogni stringa da loggere
  - *di serie* i messaggi loggati non contengono informazioni potenzialmente sensibili, quindi sono poco informativi
    - per vedere i dati aggiungere .EnableSensitiveDataLogging()
    - per vedere i dettagli delle eccezioni aggiungere .EnableDetailedErrors()
    - ogni read viene eseguita in un try-catch per fornire questo dato
    - complessivamente optionsBuilder.LogTo(action).EnableDetailedErrors().EnableSensitiveDataLogging();
- Il risultato è molto verboso
  - si può filtrare rispetto al livello di logging, la categoria di oggetto loggato...

Dibris

38

## Popolare i DbSet<E>

### Da Oggetti in Memoria

- Si crea e inizializza l'oggetto  
var e= new E(){...}
- Si attacca al contesto (avendo DbSet<E> Es{get;set;})
  - Es.Add  
default: e ha stato Added
  - Es.Attach  
default: e ha stato Unchanged
  - Es.Update  
default: e ha stato Modified
- Attaccare l'oggetto al contesto scatena TrackChanges
  - oggetti relativi a e vengono aggiunti al contesto
    - stesse regole di e sullo stato
  - oggetti già controllati dal contesto e relativi a e vengono aggiornati

a meno che la chiave di e  
• abbia tipo generato dal DB e  
• sia NON inizializzata

Dibris

39

## Popolare i DbSet<E> Da Oggetti del DB

- DbSet implementa IQueryable
- Usando LINQ le query vengono eseguite sul DB
  - se nella Select si usano funzionalità non disponibili sul DB la query sul DB fa SELECT(\*) e la proiezione avviene in memoria
  - eccezione se funzionalità non disponibili sul DB vengono usate in altre parti
  - se veramente si vuole fare la query lato client usare DbSet<E>.AsEnumerable
- Il risultato della query (parte visitata) viene aggiunto al contesto
  - in stato Unchanged
  - collegandolo agli oggetti già nel contesto a cui è relato
- Attenzione: query su DbSet = query sul DB ⇒
  - oggetti Added nel contesto non fanno parte del risultato
  - fa la query anche se quello che serve è già in memoria
- Per cercare in memoria si fa la query sulla property local del DbSet
  - oggetti Deleted non vengono restituiti
  - in generale Local contiene tutte le entità controllate e non cancellate e si può usare per data binding
- Per cercare prima in memoria e poi sul Db se non si trova si usa il metodo Find di DbSet
 

```
Blogs.Find(5)
```

  - cerca nei Blog controllati dal contesto quello con chiave 5 e lo restituisce
  - se non lo trova lo cerca sul Db (lo attacca al contesto) e lo restituisce
  - se non lo trova restituisce null

Dibris

40

## Popolare i DbSet<E> Da Oggetti del DB - Problema

- Quando si fa una query sul DB quali/quante entità relate vanno caricate?
- Se carico un utente carico anche
  - i suoi ordini  
⇒ i prodotti ordinati
  - ⇒ i fornitori
  - ⇒ gli altri prodotti
  - ⇒..magari gli altri utenti che ne hanno ordinato  
????



Dibris

41

## Strategie di Loading (Qualsiasi Interazione con DB non solo EF Core)

- Assioma: non possiamo caricare sempre tutto
- Due possibili strategie
  1. Carico tutto quello che mi serve per la UoW
    - pro: una singola query
    - contro: spesso carico anche (alcuni) dati non necessari
  2. Carico solo l'entità e quando mi servirà accedere a entità relate le caricherò
    - pro: carico il minimo indispensabile
    - contro: potenzialmente molte query
- Quello che carico viene attaccato al contesto e si aggiunge a quello che c'è già
  - nessuna duplicazione (Identity Map)
  - possibili aggiornamenti delle proprietà di navigazione
  - nessuna sincronizzazione dei dati già esistenti col DB

Dibris

42

## Eager Loading

<https://docs.microsoft.com/en-us/ef/core/querying/related-data/eager>

- Nella query si indicano le proprietà di navigazione i cui risultati si vogliono caricare DbSet<E>.Include(e=>e.prop).il resto della query
- Si possono avere più Include sulla stessa query
- Si può iterare nella navigazione
  - dopo Include si mette ThenInclude(p=>p.prop\_della\_prop)
    - si possono avere più ThenInclude in cascata
  - nell'Include si inserisce direttamente la navigazione DbSet<E> .Include(e=>e.prop.prop\_della\_prop)
    - carica sia prop che prop\_della\_prop
- Si possono filtrare i risultati dell'include per caricarne solo una parte
  - per ciascuna proprietà al più un filtro in ogni query
    - stesso = a livello sintattico

si possono usare i costrutti LINQ nei parametri della Include  
.ThenInclude(p => p.Tags.OrderBy(pt => pt.TagId).Skip(3))  
.Include(b => b.Posts.FirstOrDefault(p=>p.Author.Name=="John")

Dibris

```

var b = context.Blogs
    .Include(b=>b.Host)
    .Include(b=>b.Sponsors)
    .ThenInclude(s=>s.SponsoredBlogs)
    .ThenInclude(bl => bl.Owner)
    .Include(b => b.Owner.AuthoredPosts)
    .ThenInclude(p => p.Blog.Owner.Photo)
    .Include(b => b.Posts.Where(p => p.BlogId == 1))
    .ThenInclude(p => p.Author)
    .Include(b => b.Posts.Where(p => p.BlogId == 1))
    .ThenInclude(p => p.Tags)
    .FirstOrDefault();
  
```

Ogni Include implica (almeno) un join  
Le query generate possono essere molto pesanti  
Da valutare se non sia meglio fare più query più semplici

43

## On Demand Loading

### Explicit Loading

<https://docs.microsoft.com/en-us/ef/core/querying/related-data/explicit>

- Quando ho bisogno, prima di navigare, faccio load esplicito
  - context.Entry(blog).Collection(b => b.Posts).Load();  
equivale a  
context.Posts.Where(p=>p.Blog==blog).ToList()
  - context.Entry(blog).Reference(b => b.Owner).Load();
- Usando Collection posso anche fare query più sofisticate
  - var postCount = context.Entry(blog).Collection(b => b.Posts).Query().Count();  
non carica le entità solo il count
  - var goodPosts = context.Entry(blog).Collection(b => b.Posts).Query().Where(p => p.Rating > 3).ToList();  
filtra

**Attenzione:** eager e on-demand loading non interagiscono perfettamente  
Se con eager una proprietà di navigazione è stata caricata (anche parzialmente a causa di un filtro), né explicit né lazy loading la ricaricano

Dibris

44

### Lazy Loading

<https://docs.microsoft.com/en-us/ef/core/querying/related-data/lazy>

- Configuro il modello in modo che EF Core possa *magicamente* intervenire quando serve navigare verso una property non caricata
- Non ci devo più pensare: se la property non c'è, EF interviene e fa il Load (come se lo avessi richiesto esplicitamente)
- Punto chiave della soluzione
  - usare istanze di classi proxy invece che delle classi entità
  - le proprietà di navigazione nei proxy fanno caching quando serve
  - per poter usare i proxy
    - importare package
    - enable i proxy
    - dichiarare virtual le proprietà di navigazione
    - mai usare new E ma sempre far creare le entità all'EF
      - metodo Create sul DbSet

## Strategia di Loading...Quale?

- A seconda dei casi una è più appropriata di un'altra
- Siccome mischiarle può causare comportamenti inattesi non sempre si può scegliere query per query
- IMHO
  - in un mondo ideale per una UoW servono pochi dati e si sa quali sono ⇒ eager loading è adeguata
    - una sola query, non troppo complessa
    - se alcune property servono sempre si può usare AutoInclude
      - <https://dotnetcoretutorials.com/2021/03/07/eager-load-navigation-properties-by-default-in-ef-core/>
  - eventualmente si può ottimizzare la strategia di caricamento usando anche explicit loading
    - facendo attenzione alle interazioni
    - sapendo che si paga con una query in più
  - lazy loading ottima alternativa se siete coscienti che funziona solo finché siete nel contesto
    - qualsiasi strategia adottate usare proprietà di navigazione non inizializzate su entità detached causa eccezione
    - ma se abitualmente usate eager/explicit avete l'automatismo di pensare se la proprietà è inizializzata prima di accedere

Dibris

45

## No-Tracking Queries

- Quando non si vogliono modificare le entità, è inutile attaccarle al contesto
  - non serve
  - può solo rendere più grosso il contesto e potenzialmente creare conflitti inutili
- Usando AsNoTracking() si evita che le entità vengano controllate dal contesto

```
using (var context = new BloggingContext()) {
    // Query for all blogs without tracking them
    var blogs1 = context.Blogs.AsNoTracking();

    // Query for some blogs without tracking them
    var blogs2 = context.Blogs
        .Where(b => b.Name.Contains(".NET"))
        .AsNoTracking()
        .ToList();
}
```

Dibris

## Change Tracking & Relationship Fix-Up

- Relazioni in DB memorizzate in un solo posto
  - chiave esterna in entità referente
  - link in tabella per molti-molti
- In OO ridondanza
  - proprietà di navigazione *dai due lati*
  - sul lato 1 (se c'è) anche chiave esterna
  - facilita accesso alla relazione ma introduce potenziali inconsistenze
- Cambiando una proprietà vorremmo che cambiassero di conseguenza le altre
- Per oggetti detached non c'è speranza di automatismo
- Per oggetti attached ci pensa ChangeTracker.DetectChanges()
  - per ogni entità controllata
    - aggiorna le property di navigazione delle entità attached relate
    - se navigando raggiunge oggetti detached li attacca
      - non cerca istanze detached con le chiavi usate come FK
      - se ci sono cambiamenti a qualche property mette in stato Modified (e se è stato cancellato in Deleted)
- DetectChanges viene chiamata
  - quando EF sa di stare potenzialmente cambiando le cose: Add, Attach, Update...
    - se usiamo l'EF per cambiare la property X dell'oggetto o con context.Entry(o).Property(e => e.X).CurrentValue = ...;
    - quando EF sta per rendere permanenti i cambiamenti e ha bisogno di avere tutti i dati
    - quando il programmatore lo invoca esplicitamente
- Se si usano i proxy ovunque è possibile configurarli in modo che i fix-up siano immediati



3 viste su una sola realtà

Dibris

47

## Cancellare Entità dal DB

- Un'entità in stato Deleted verrà cancellata dal DB al prossimo SaveChanges
- Per mettere o in stato Deleted si usa Remove(o) sul DbSet del tipo giusto
  - ma o deve già essere attached
    - possibilità 1: la carico dal DB (sembra scemo)
    - possibilità 2: creo un oggetto vuoto con la chiave di quello da cancellare (e lo stesso concurrency token se serve...vedi dopo) e faccio Attach
- Che succede alle entità dipendenti da quella cancellata?
  - dipende da come abbiamo configurato OnDelete  
modelBuilder.Entity<Blog>()  
.HasOne(e => e.Owner)  
.WithOne(e => e.OwnedBlog)  
.OnDelete(DeleteBehavior.Xyz);

Dibris

48

## On Delete

<https://docs.microsoft.com/en-us/ef/core/saving/cascade-delete>

- A seconda dal tipo di proprietà di navigazione (required o nullable) e del DB non tutte le 7 opzioni sono possibili
  - vediamo solo le più interessanti
  - Cascade: le entità referenti sono cancellate dal DB.
  - ClientCascade: le entità referenti sono cancellate da EF **ma solo quelle attached**
    - se ce ne sono altre e la chiave non è nullable il DB solleverà eccezione
  - SetNull: il DB mette a null la chiave esterna nelle entità referenti
  - ClientSetNull: EF mette a null la chiave esterna nelle entità referenti, **ma solo quelle attached**
  - NoAction ~ ClientSetNull
- In generale
  - quelle che iniziano con Client lavorano solo lato client sulle sole entità attached
    - il DB è configurato di default NoAction
    - se ci sono altre entità referenti nel DB genera eccezione
  - le altre corrispondono lato DB alle azioni con lo stesso nome
    - sul client vengono compiute le stesse azioni sulle sole entità attached
    - peculiarità: NoAction sul client corrisponde all'azione di default, che è SetNull...no comment

Dibris

49

## Concorrenza

- Cosa succede se
  - due (o più) programmi cercano di scrivere gli stessi dati?
    - "Vince" l'ultimo?
    - Si fa un "merge"?
    - ...
  - un programma cerca di aggiornare dati che non esistono più?
- Due approcci per gestire i conflitti:
  - Concorrenza pessimistica
    - Usa il locking, poco scalabile
  - Concorrenza ottimistica
    - Niente locking, l'applicazione deve gestire eventuali conflitti
    - In EF: <https://docs.microsoft.com/en-us/ef/core/saving/concurrency>

Dibris

50

## Concorrenza ottimistica: conflitti

- Come ci accorgiamo se ci sono dei conflitti?
  - UPDATE ...  
WHERE Id=... AND f1=old1 AND f2=old2 ...
    - E poi si guarda quante righe sono state modificate
      - 1 → ok
      - 0 → la riga originale è stata modificata o cancellata
- Verifica poco efficiente
  - molte egualianze, magari fra valori di tipi complicati
- Non garantisce che non ci siano state modifiche
  - solo che i valori sono [tornati a essere] quelli originali
- Soluzione migliore
  - un campo specifico che indica la *versione* dell' entità
  - aggiornato automaticamente a ogni modifica
  - tipo facile da confrontare
  - `[Timestamp]  
public byte[] Timestamp { get; set; } o usando API Property(p => p.Timestamp).IsRowVersion();`
- EF Core (attraverso DbProvider)
  - aggiunge la verifica che il Timestamp sia lo stesso al filtro nella query di ogni Update (e Delete)
    - nel caso di SQL-like aggiunge la verifica `Timestamp= old-Timestamp` a WHERE
    - se le righe modificate sono 0 solleva `DbUpdateConcurrencyException`

Dibris

51

## In caso di conflitto?

- SaveChanges solleva l'eccezione **DbUpdateConcurrencyException**, facendo rollback di eventuali modifiche già avvenute
  - Di default, SaveChanges usa una DbTransaction in automatico
- L'eccezione espone la proprietà **IEnumerable<DbEntityEntry> Entries** che permette di sapere quali entità hanno creato problemi
- A seconda delle specifiche del problema diverse strategie
  - client win = si forza la scrittura aggiornando il Timestamp e ripetendo la scrittura
  - Db win = si aggiorna l'entità in memoria con i valori del DB
  - lasciare la gestione dell'eccezione al chiamante
    - eventualmente cambiando il tipo dell'eccezione incapsulandola
- Esempi: <https://docs.microsoft.com/en-us/ef/core/saving/concurrency>

Dibris

52

## Validazione dei parametri nella classica architettura 3tier

### Si fa

- Lato client
  - per dare feedback immediato
  - per evitare connessioni inutili
- Lato server logica business
  - per evitare attacchi
- Lato server strato dei dati
  - perché non si può assumere che l'unico accesso al DB sia attraverso questa applicazione

### Si può evitare (solo) se

- Lato client
  - tempi e costi della connessione inutile sono accettabili
  - sono fatti **tutti** lato server
- Lato server logica business
  - sono fatti **tutti** dal DB
  - si gestiscono gli eventuali errori
- Lato server strato dei dati
  - sono **sicuramente** fatti **tutti** dalla logica business
  - **nessuna** altra applicazione userà **mai** lo stesso DB

Dibris

53

## Problemi ed eccezioni

<https://www.thereformedprogrammer.net/entity-framework-core-validating-data-and-catching-sql-errors/>

- Se c'è un problema, EF solleva un'eccezione
- Se riguarda la connessione a un DB SQL, l'eccezione ha tipo **Microsoft.Data.SqlClient.SqlException**
- Se riguarda l'esecuzione di SQL, l'eccezione ha tipo **Microsoft.EntityFrameworkCore.DbUpdateException**
  - con property **Entries** che memorizza le entità coinvolte
  - con inner di tipo **Microsoft.Data.SqlClient.SqlException**
- Molti errori non si possono prevenire con validazione lato client
  - per via di esecuzioni concorrenti
- Bisogna gestire le eccezioni del DB
  - hint: ridefinire SaveChanges

Dibris

54

## SqlException

<https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqlexception>

- Design meno user-friendly del desiderabile
  - limiti dovuti ai dati ricevuti da SQL
- Contiene una collection di **SqlError**
- Ogni **SqlError** ha property che espongono i codice SQL dell'errore rappresentato
  - quelli del primo vengono esposti come property a livello di **SqlException**
  - [Severity] Class più è alta più è grave <https://docs.microsoft.com/en-us/sql/relational-databases/errors-events/database-engine-error-severities>
    - 11 Indicates that the given object or entity does not exist.
    - 12 A special severity for queries that do not use locking because of special query hints. In some cases, read operations performed by these statements could result in inconsistent data, since locks are not taken to guarantee consistency.
    - 13 Indicates transaction deadlock errors.
    - 14 Indicates security-related errors, such as permission denied.
    - 15 Indicates syntax errors in the Transact-SQL command.
  - [Error] Number indica il tipo di errore
    - quindi come va gestito
    - elencati nelle tabelle di Sistema

```
USE master
GO
SELECT * FROM SYSMESSAGES
WHERE message_id=Number AND language_id IN (1040,1033)
```
- e online <https://docs.microsoft.com/en-us/sql/relational-databases/errors-events/database-engine-events-and-errors?view=sql-server-ver15>

Non fare parsing del messaggio di errore  
dipende dalla lingua ⇒ rende fragile il sistema

italiano

inglese

Dibris

55

## Esempio di violazioni

[Index(nameof(Login), IsUnique = true, Name = "UniqueLogin")]

```
public class User{
    public int UserId { get; set; }
    public string Login { get; set; }
    ...
}
```

Se proviamo ad aggiungere due utenti con la stessa login...

```
using (var db = new BTDbContext()) {
    db.Users.Add(new User() { Login = "Puffo" });
    db.Users.Add(new User() { Login = "Puffo" });
    db.SaveChanges();
}
```

