Linguaggio c# Approfondimenti

Arianna Bolzoni

.Net Developer

Arianna.Bolzoni@icubed.it

Codice Elegante

"I bravi programmatori sanno cosa scrivere.
I migliori sanno cosa riscrivere."

ERIC STEVEN RAYMOND

1. Usare nomi descrittivi

Ad esempio: Le variabili sono i soggetti

I metodi sono le azioni

2. Dare a ogni classe uno scopo

Divisione in blocchi

3. Non ha bisogno di commenti per essere capito

Spiegare il come e il perché

Non il cosa

4. Deve essere leggibile

Codice pulito != Codice intelligente

Anzi

Codice pulito > Codice intelligente

5. E’ riutilizzabile

Eliminare parti duplicate

Non mantenere troppo a lungo parti commentate

6. E’ correttamente indentato

In termini prettamente visivi

Visual studio aiuta

7. Scegliere l’architettura giusta

Eliminare parti duplicate

Non mantenere troppo a lungo parti commentate

Consigli

Spesso è utile leggere e debuggare il codice “dei maestri”

Demo

Codice Elegante

Iban, carte di credito

icubed

Ricorsione e Iterazione

Iterazione

- Blocco di codice continua ad essere eseguita finchè una condizione non viene soddisfatta

Ricorsione

- Funzione richiamata all'interno della stessa funzione.

	Ricorsione	Iterazione
Memoria	Consumo alto	Consumo basso
Velocità	Lenta	Veloce
Pila	Stack	Non utilizzata
Leggibilità	Più facile	Più difficile

Quindi entrambi risolvono problemi di programmazione

MA

L'iterazione è sempre da preferire rispetto alla ricorsione

Demo

Ricorsione VS Iterazione

Fibonacci, fattoriale, interessi

Memory Leak

Consumo della memoria causato dalla mancata deallocazione di variabili/risorse non più utilizzati dai processi.

Un programma rimanendo attivo, continua ad allocare memoria finchè la memoria del sistema non viene completamente consumata.

- Rallentamento delle funzionalità
- Problemi di memoria su altri programmi
- Riavvio del sistema

Garbage Collector

Il Garbage Collector è un componente il cui ruolo è di liberare la memoria dagli oggetti non più utilizzati.

Gestisce gli oggetti allocate nel managed heap.

Agisce quando si ha necessità di avere maggiori risorse a disposizione:un oggetto può essere rimosso in una fase successive rispetto al suo inutilizzo.

Garbage Collector

Il funzionamento del Garbage Collector è il seguente:

1. Segna tutta la memoria allocata (heap) come “garbage”
2. Cerca i blocchi di memoria in uso e li marca come validi
3. Dealloca le celle non utilizzate
4. Compatta il managed heap

Value Type

VS

Reference Type

- Il sistema di **tipizzazione** di C# fa riferimento a 3 categorie:
  - Tipi **value**
  - Tipi **reference**
  - Tipi **pointer** (non trattati)
- I tipi **value** archiviano **dati**, i tipi **reference** archiviano dei **riferimenti** ai dati effettivi.

Value Type

- I tipi value possono essere suddivisi in due macro categorie:
  - **Struct**
  - **Enum**

- Le variabili **value** contengono valori e l’**assegnazione** di una variabile value ad un’altra variabile value effettua **la copia dei dati** inseriti.

- Tutte le variabili value sono derivate in modo implicito da **System.ValueType**.

- Una variabile di tipo value non potrà contenere **null**, a meno di non renderla di tipo **nullable**. (Lo vedremo dopo)

• **sbyte**	System.SByte	8 bit con segno
• **byte**	System.Byte	8 bit positivo
• **short**	System.Int16	16 bit con segno
• **ushort**	System.UInt16	16 bit positivo
• **int**	System.Int32	32 bit con segno
• **uint**	System.UInt32	32 bit positivo
• **long**	System.Int64	64 bit con segno
• **ulong**	System.uInt64	64 bit positivo
• **char**	System.Char	16 bit
• **float**	System.Single	32 bit precisione 7 cifre
• **double**	System.Double	64 bit precisione 15-16 cifre
• **decimal**	System.Decimal	128 bit precis 28-29 cifre
• **bool**	System.Boolean	true o false

… ovviamente

anche **struct** e **enum** sono tipi Value

e…

non dimentichiamoci del **DateTime**

Reference Type

- Con i tipi **reference**, due variabili possono far riferimento allo stesso oggetto presente nell'heap.

- Di conseguenza le operazioni su una variabile **influenzeranno** l'oggetto a cui farà riferimento l'altra variabile.

- Quando si assegna il contenuto di una variabile reference ad un altra variabile reference si **copierà l'indirizzo** e non il valore a cui si fa riferimento.

- Gli operatori **==** e **!=** applicati a tipi reference **non confronteranno i valori** presenti nell'heap **ma solo gli indirizzi**.

- Ad una variabile **reference** potrà essere assegnato il valore **null**.

- Con tale valore si indica che **non esiste alcuna allocazione** nell'heap.

- Alle variabili reference possono essere associati:
  - **oggetti (Classi)**
  - **delegati**
  - **stringhe**
  - **array**

Heap Memory VS Stack Memory

- Esistono due tecniche di allocazione:
  - l’allocazione **stack frame**, dove si utilizza effettivamente un blocco di memoria
  - L’allocazione **heap** dove viene sempre fornito un puntatore (indirizzo) ad un blocco di memoria

Stack Memory

- Lo **stack frame** è un’area della memoria di tipo temporaneo che viene utilizzata per memorizzare gli argomenti dei metodi e le variabili locali.

- Tali variabili sono denominate «**auto**» in quanto il compilatore allocherà automaticamente lo spazio necessario per contenerle.

- Le variabili presenti nello **stack frame** vengono *automaticamente eliminate* quando si esce dallo **scope**.

- In pratica le variabili associate ai parametri di un metodo e quelle definite all'interno del metodo saranno cancellate non appena il metodo termina.

- L'eliminazione automatica delle variabili presenti nello **stack frame** è importante in quando consente di recuperare la memoria in modo semplice ed efficiente.

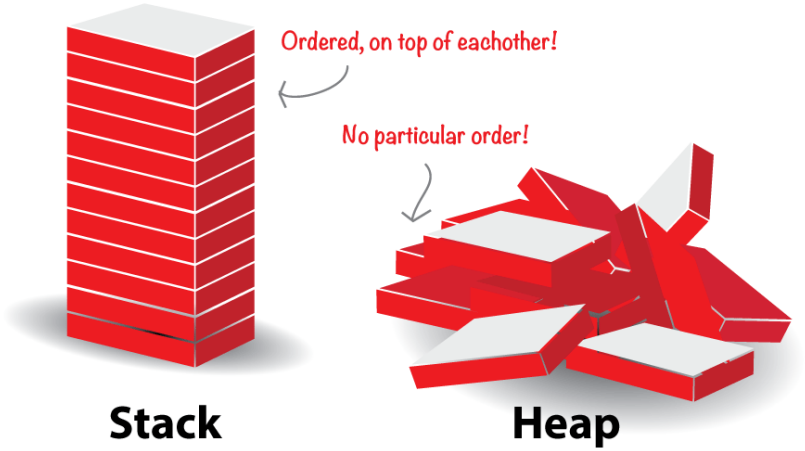
- Lo svantaggio è che le variabili nello stack frame *non possono essere utilizzate all'esterno dell'ambito dove sono definite*.

- Oltre a ciò lo stack ha *dimensioni limitate* e può portare ad errori di tipo **stack overflow**.

Heap Memory

- L'**heap** è invece riservato per le necessità di allocazione della memoria del programma e ha dimensione rilevanti rispetto allo **stack frame**.

- I dati memorizzati nell'**heap** vengono referenziati utilizzando un *puntatore* (indirizzo) e poiché sono memorizzati senza un particolare ordine è possibile che siano frammentati.



- Una variabile **value** occuperà solo lo **stack**, mentre una variabile reference utilizzerà lo **stack** per inserire l'indirizzo della zona dell'**heap** che conterrà i dati.

- Con i tipi **reference**, due variabili possono far riferimento allo stesso oggetto presente nell'heap.

- Di conseguenza le operazioni su una variabile **influenzeranno** l'oggetto a cui farà riferimento l'altra variabile.

Demo

ValueType Vs ReferenceType

Null e Nullable

Abbiamo detto che i Value Type non possono assumere il “valore” null.

Ma se volessimo invece non avere alcun valore in fase di inizializzazione?

C# matte a disposizione la classe **Nullable<T>**

che con un semplice **?** permette di aggiungere ai

Value Type anche il valore **null**

tra i suoi possibili valori

- Una volta definito un valore nullable gli si vorrebbe assegnare un valore non nullable, oppure si vorrebbe assegnare un valore nullable ad una variabile non nullable.
- Esistono delle conversioni **implicite** ed **esplicite**.
- Ad esempio, se abbiamo una variabili di tipo nullable, gli possiamo assegnare un valore non nullable, e il compilatore effettuerà un **cast implicito**.

```
int? n = null;
int k = 12;

n = k;      // cast implicito
k = n;      // assegnazione non valida
k = (int)n; // cast esplciito
```

- Viceversa, se abbiamo una valore nullable non lo potremo assegnare ad una variabile non nullable a meno di non effettuare un **cast esplicito**.

Demo

Nullable

Iscrizione utente

Struct

Un tipo di struttura è un value type che incapsula dati e funzionalità correlate

Vengono utilizzati per progettare tipi di piccolo dimensioni che offrono un comportamento ridotto o un non comportamento

Per crearle si usa la parola struct

Demo

Cerchio

Le stringhe

- È un oggetto di tipo String il cui valore è un testo
- È un array di char – caratteri
- Vediamo insieme alcuni metodi e proprietà utili

- String.Empty
- String.IsNullOrEmpty()
- Lenght
- Trim
- -> La stringa è un reference type ma si comporta come un value type

Demo

Stringhe

Le collections

- Quando si vogliono raggruppare oggetti dello stesso tipo si hanno a disposizione 2 strade

    - Creare matrici di oggetti - Array

    - Creare racconlte di oggetti

- Le collections fanno parte del namespace **System.Collections**

- Le collection più comuni sono

- ArrayList
- Hashtable
- Queue
- Stack
- List
- Dictionary

Ma

Le collection possono far parte anche del namespace **System.Collections.Generics**

Fanno parte di questa raccolta

- Dictionary <Tkey, Tvalue>
- List <T>
- Queue <T>
- Stack <T>

Demo

Collections

Generics

- In C# tutto deriva da Object

quindi

È possibile memorizzare un valore di qualsiasi tipo e definire parametri dei metodi in modo da accettare oggetti di diverso tipo

- C# fornisce i **generics** con i quali è possibile rimuovere la necessità di utilizzo dei cast, aumentare la sicurezza e ridurre il numero di boxing richiesti rendendo semplice creare classi e metodi **generalizzati**.

- E’ possibile indicare che una classe è di tipo **generic** fornendo un parametro tra i simboli di **<** e **>**

  class Queue <T> { … };

- La lettera **T** nell’esempio è un **segnaposto** che verrà utilizzato per assegnare il tipo reale durante la compilazione.

- I generic possono essere utilizzate con le classi ma anche con le **interfacce** e con le **struct**.

- La sintassi da usare è **identica** a quella delle classi.

Demo

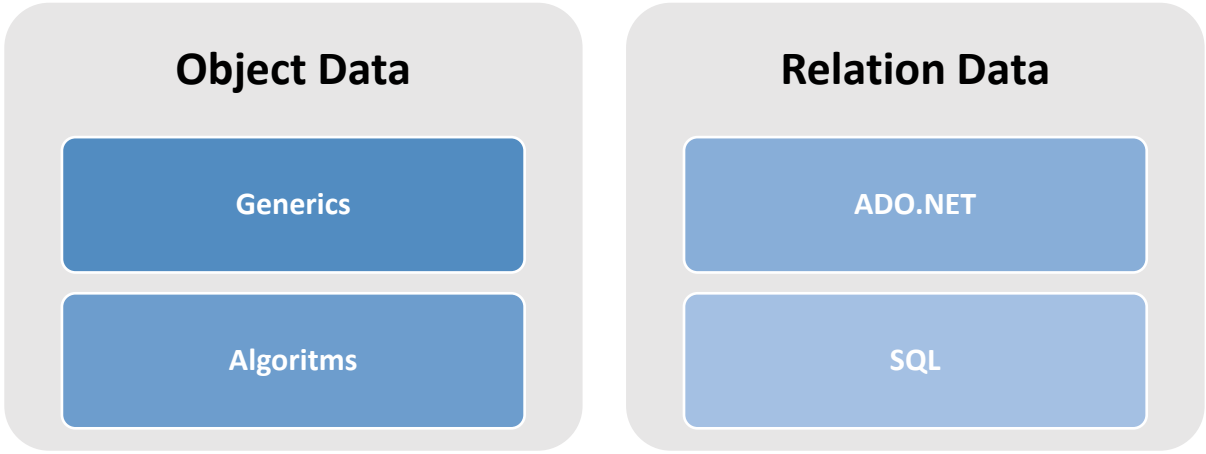
Generics

Linq

- Uno degli aspetti importanti del linguaggio C# è il supporto che è in grado di fornire per l'**interrogazione dei dati**.
- Spossono utilizzare strutture e classi per la modellazione dei dati e che gli array e le collezioni consentono di memorizzare temporaneamente i dati in memoria.
- Tuttavia, è in genere complesso cercare in una collezione di oggetti tutti gli elementi che corrispondono ad una serie specifica di criteri.

- Poiché è possibile avere a che fare con diverse tipologie di dati, per ognuna di esse nel tempo sono state create apposite **API**.

- Saranno gli sviluppatori a capire quali API utilizzare in funzione della sorgente di dati utilizzata.



Entra quindi in gioco **LINQ**

Language Integrated Query

Utilizzando un **unico set di operatori** sarà possibile **operare su diversi tipi di dati**: potremo scrivere un codice per accedere ad un database relazionale, e senza cambiarlo accedere anche a dati XML, ecc

- **LINQ** è una **API uniforme** che consente di lavorare con più fonti di dati sia di tipo **relazionale** che di tipo **gerarchico**.

- L'API di **LINQ** è composto da una serie di **operatori** di query standard presenti nello spazio dei nomi **System.Linq**

- **LINQ** è in grado di operare su **qualsiasi tipo di dati** che implementi l'interfaccia **IEnumerable**.

- **LINQ** è simile alla scrittura sql

- Usa **lambda expression**

Demo

Generics