Tipi e variabili

In C# esistono due generi di tipi: *tipi valore* e *tipi riferimento*. Le variabili dei tipi valore contengono direttamente i propri dati, mentre le variabili dei tipi riferimento archiviano i riferimenti ai propri dati, noti come oggetti. Con i tipi di riferimento, è possibile che due variabili facciano riferimento allo stesso oggetto e pertanto le operazioni su una variabile influiscano sull'oggetto a cui fa riferimento l'altra variabile. Con i tipi valore, ogni variabile ha una propria copia dei dati e non è possibile che le operazioni su uno influiscano sull'altro (ad eccezione delle variabili di parametro ref e out).

I **tipi valore** di C# sono ulteriormente suddivisi in *tipi semplici*, *tipi enum*, *tipi struct* e *tipi valore nullable*. I **tipi riferimento** di C# sono ulteriormente suddivisi in *tipi classe*, *tipi interfaccia*, *tipi matrice* e *tipi delegato*.

Nella struttura seguente viene fornita una panoramica C# del sistema di tipi.

- **Tipi valore**
  - Tipi **semplici**
    - Signed Integer: ***sbyte***, ***short***, ***int***, ***long***
    - Unsigned Integer: ***byte***, ***ushort***, ***uint***, ***ulong***
    - Caratteri Unicode: ***char***
    - File binario IEEE a virgola mobile: ***float***, ***double***
    - Decimale ad alta precisione a virgola mobile: ***decimal***
    - Booleano: ***bool***
  - Tipi **enum**
    - Tipi definiti dall'utente nel formato enum E {...}
  - Tipi **struct**
    - Tipi definiti dall'utente nel formato struct S {...}
  - Tipi valore **nullable**
    - Estensioni di tutti gli altri tipi valore con un valore null
- **Tipi riferimento**
  - Tipi **classe**
    - Classe di base principale di tutti gli altri tipi: ***object***
    - Stringhe Unicode: ***string***
    - Tipi definiti dall'utente nel formato class C {...}
  - Tipi **interfaccia**
    - Tipi definiti dall'utente nel formato interface I {...}
  - Tipi di **matrice** (**array**)
    - Unidimensionale e multidimensionale, ad esempio int**[]** e int**[,]**
  - Tipi **delegato**
    - Tipi definiti dall'utente nel formato delegate int D(...)

Per altre informazioni sui tipi numerici, vedere la tabella dei tipi integrali e la tabella dei tipi a virgola mobile.

Tipo/parola chiave C#	Range	Dimensione	Tipo .NET
sbyte	Da -128 a 127	Valore intero con segno a 8 bit	System.SByte
byte	da 0 a 255	Intero senza segno a 8 bit	System.Byte
short	Da -32.768 a 32.767	Valore intero a 16 bit con segno	System.Int16
ushort	Da 0 a 65.535	Intero senza segno a 16 bit	System.UInt16
int	Da -2.147.483.648 a 2.147.483.647	Valore intero a 32 bit con segno	System.Int32
uint	Da 0 a 4.294.967.295	Intero senza segno a 32 bit	System.UInt32
long	Da -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Valore intero a 64 bit con segno	System.Int64
ulong	Da 0 a 18.446.744.073.709.551.615	Intero senza segno a 64 bit	System.UInt64

Il tipo **bool** di C# viene usato per rappresentare valori booleani, ovvero valori che sono *true* o *false*.

Per l'elaborazione di caratteri e stringhe, in C# viene usata la codifica Unicode. Il tipo **char** rappresenta un'unità di codice UTF-16, mentre il tipo **string** rappresenta una sequenza di unità di codice UTF-16.

I programmi C# usano le ***dichiarazioni di tipo*** per creare nuovi tipi. Una dichiarazione di tipo consente di specificare il nome e i membri del nuovo tipo. Cinque delle categorie di tipi di C# possono essere definite dall'utente: tipi *classe*, tipi *struct*, tipi *interfaccia*, tipi *enum* e tipi *delegato*.

Un tipo **class** definisce una struttura dati contenente membri dati (campi o *variabili*) e membri funzione (*metodi*, *proprietà* e altro). I tipi classe supportano l'ereditarietà singola e il polimorfismo, meccanismi in base ai quali le classi derivate possono estendere e specializzare le classi di base.

Un tipo **struct** è simile a un tipo classe in quanto rappresenta una struttura con membri dati e membri funzione. Tuttavia, a differenza delle classi, gli struct sono tipi di valore. I tipi struct *non supportano l'ereditarietà* specificata dall'utente e tutti i tipi struct ereditano implicitamente dal tipo object.

Un tipo **interface** definisce un contratto come un set denominato di membri funzione pubblici. Un tipo class o struct che implementa un tipo interface deve fornire le implementazioni dei membri funzione dell'interfaccia. Un tipo interface può ereditare da più interfacce di base e un tipo class o struct può implementare più interfacce.

Le **enum** consentono di associare nomi simbolici a costanti numeriche. Richiamando il **metodo ToString()** di una variabile di tipo enum, è possibile ottenere la stringa contenente il nome simbolico del valore corrente.

Un tipo **delegate** rappresenta i riferimenti ai metodi, con un elenco di parametri e un tipo restituito particolari. I delegati consentono di trattare i metodi come entità che è possibile assegnare a variabili e passare come parametri. I delegati sono analoghi ai tipi funzione forniti dai linguaggi funzionali. Sono anche simili al concetto di puntatori a funzione disponibili in altri linguaggi. A differenza dei puntatori a funzione, i delegati sono orientati agli oggetti e indipendenti dai tipi.

C# supporta **matrici** unidimensionali e multidimensionali di qualsiasi tipo. A differenza dei tipi elencati in precedenza, i tipi di matrice non devono essere dichiarati prima di poter essere usati. Al contrario, i tipi matrice vengono costruiti facendo seguire a un nome di tipo delle parentesi quadre. Ad esempio, int[] è una matrice unidimensionale di int, int[,] è una matrice bidimensionale di int e int[][] è una matrice unidimensionale di matrici unidimensionali di int.

Anche i tipi di valore **nullable** non devono essere dichiarati prima di poter essere usati. Per ogni tipo di valore non nullable T, esiste un tipo di valore nullable corrispondente T?, che può avere un valore aggiuntivo, null. Ad esempio, int? è un tipo che può contenere qualsiasi Integer a 32 bit o il valore null.

Il sistema di tipi di C# è unificato in modo tale che un valore di qualsiasi tipo può essere trattato come un ***object***.
In C# ogni tipo deriva direttamente o indirettamente dal tipo **classe object** che è la classe di base principale di tutti i tipi.

I valori dei *tipi riferimento* vengono trattati come oggetti semplicemente visualizzando tali valori come tipi object.
I valori dei *tipi valore* vengono trattati come oggetti mediante l'esecuzione di operazioni di ***boxing*** e ***unboxing***. Nell'esempio seguente un valore int viene convertito in object e quindi convertito nuovamente in int.

```
using System;
class BoxingExample
{
    static void Main()
    {
        int i = 123;
        object o = i;    // Boxing
        int j = (int)o;  // Unboxing
    }
}
```

Quando un valore di un tipo valore viene convertito in un tipo object, un'istanza di object, denominata anche "box", viene allocata per contenere tale valore.

Al contrario, quando viene eseguito il cast di un riferimento object a un tipo valore, il sistema effettua un controllo per verificare che l'object a cui viene fatto riferimento sia un box del tipo valore corretto. Se il controllo ha esito positivo, il valore presente nel box viene copiato.

Con il sistema di tipi unificato di C#, i tipi valore possono diventare oggetti "su richiesta". Grazie all'unificazione, le librerie generiche che usano il tipo object possono essere usate con entrambi i tipi riferimento e valore.

La **conversione tra tipi** valore può essere implicita o esplicita attraverso l'operatore di casting()

```
// conversione implicita
int i = 123;
byte b = i;

// conversione esplicita
int i = 5;
long b = 123;
int a = i + (int)b;
```