Programmazione C#

Lezione 2

Sintassi di Base del Linguaggio C#

e-Linking online Systems

Sommario della Lezione

1. Parole Chiave
2. Tipi del Linguaggio, Conversioni Tipi e Array
3. Funzioni e Procedure
4. Istruzioni Condizionali
5. Cicli e Iterazioni
6. Test su Visual Studio!

Parole Chiave C#

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	int	interface	internal
is	lock	long	namespace
new	null	object	operator
out	override	params	private
protected	public	readonly	ref
return	sbyte	sealed	short
sizeof	stackalloc	static	string
struct	switch	this	throw
true	try	typeof	uint
ulong	unchecked	unsafe	ushort
using	virtual	void	volatile
while			

Contestuali:

add	alias	ascending	async
await	descending	dynamic	from
get	global	group	into
join	let	orderby	partial
remove	select	set	value
var	where	yield	

Commenti

Due tipi di commento:

- Singola Riga
- Righe Multiple

```
// Testo di commento su singola riga
Console.Write("Hello, World!"); // Commento dopo il codice
/* Testo di commento
    ripartito su più righe */
```

Tipi Base del Linguaggio

bool	System.Boolean	Tipo di valore
byte	System.Byte	Tipo di valore
char	System.Char	Tipo di valore
decimal	System.Decimal	Tipo di valore
double	System.Double	Tipo di valore
float	System.Single	Tipo di valore
int	System.Int32	Tipo di valore
long	System.Int64	Tipo di valore
object	System.Object	Tipo di riferimento
sbyte	System.SByte	Tipo di valore
short	System.Int16	Tipo di valore
string	System.String	Tipo di riferimento
uint	System.UInt32	Tipo di valore
ulong	System.UInt64	Tipo di valore
ushort	System.UInt16	Tipo di valore

Attenzione!! Alcuni tipi non sono CLS-Compliant!

(es uint, ulong…)

Che differenza c'è tra riferimento e valore?

```
int x = 2;                  // Intero che contiene 2
bool y = true;              // Valore booleano che vale true
string s1 = "Hello";        // Punta ad un'area di memoria
string s2 = null;           // Non punta a nulla
object obj1 = new           // Crea un nuovo oggetto
object();
object obj2 = null;         // Non punta a nulla
```

Namespaces

Servono per:

- «Garantire» l'univocità dei nomi
- «Dividere» il Software in «parti»
- Organizzano Gerarchicamente il Codice

```
namespace ASPItalia.Books.Chapter3
{
  // Dichiarazione di un tipo appartenente al
  // namespace ASPItalia.Books.Chapter3
  class HelloWorld
  {
    // …
  }
}
```

Variabili

- Le Variabili sono entità che contengono un valore (o il riferimento al valore)
- Ogni variabile ha uno «scope» (ambito di attività)
- Le costanti sono varibili che non possono essere modificate
- C# supporta il ***type inference*** (capacità di ricavare il tipo della variabile dalla sua assegnazione iniziale)

```
int i = 1, j = 2, k = 3;
string x, y, z;
```

```
object x = new object();     // Punta ad una nuova istanza
object y = null;             // Punta a null
object z;                    // Punta a null
object obj = x;              // obj si riferisce alla stessa istanza
                             di x
```

```
const double PI = 3.1416;
const string HELLO = "Hello World";
```

```
// Dichiarazione di una variabile intera senza e con type inference
int i = 1;                   // i è di tipo intero
var j = 1;                   // j è di tipo intero (type inference)

// Dichiarazione di una stringa senza e con type inference
string x = "Hello World";    // x è di tipo string
var y = "Hello World";       // y è di tipo string (type inference)
```

Espressioni e Operatori

```
bool x, y;                  // Variabili booleane
x = true;                   // Assegnazione
x = !y;                     // Negazione booleana (x vale true)
x = (18 > 8);               // A x viene assegnato il valore
                            booleano
                            // risultante dalla valutazione
                            // dell'espressione di confronto (true)


int i, j, k;                // Variabili intere
i = 2;                      // Assegnazione
j = i + 1;                  // Somma e assegnazione
k = i * 2;                  // Prodotto e assegnazione

int num;                    // Variabile intera
num = i * j + k;            // Prodotto, somma e assegnazione

bool isOdd;                 // Variabile booleana
isOdd = (num % 2) != 0;     // Espressione che valuta se num è
                            dispari


string hello;               // Variabile di tipo string
hello = "Hello " +          // Concatenazione di stringhe
"World";
```

Categoria	Operatori
Aritmetici	`+ - * / %`
Logici (booleani e bit per bit)	`& | ^ ! ~ && || true false`
Concatenamento di stringhe	`+`
Incremento e decremento	`++ --`
Spostamento	`<< >>`
Relazionali	`== != < > <= >=`
Assegnazione	`= += -= *= /= %=`
	`&= |= ^= <<= >>= ??`
Accesso a membri	`.`
Indicizzazione (array)	`[]`
Casting	`()`
Condizionale	`?:`
Concatenazione e rimozione di delegati	`+ -`
Creazione di oggetti (classi)	`new`
Informazioni sui tipi	`as is sizeof typeof`
Controllo delle eccezioni di overflow	`checked unchecked`
Riferimento indiretto e indirizzo	`* -> [] &`
Lambda	`=>`

Conversione dei Tipi

- ***Conversioni Implicite***: Avvengono senza il rischio di perdita di informazione
- ***Conversioni Esplicite***: Avvengono con perdita di informazioni

```
int x = 0;              // Variabile intera (4 byte)
byte y = 100;           // Variabile di tipo byte (1 byte)
x = y;                  // Conversione implicita
y = (byte) x;           // Conversione esplicita
y = x;                  // Errore di compilazione
x = 1000;               // Valore non incluso nel range del tipo byte
y = (byte) x;           // Errore di overflow a runtime
```

Array

- Un Array è una variabile composta da un gruppo di oggetti dello stesso tipo
- Gli Array possono essere multidimensionali
- Ogni elemento è contrassegnato da un indice numerico
- Gli indici sono sequenziali

```
string[] x = new string[3];          // Vettore composto da tre stringhe
x[0] = "Hello ";                     // Contiene "Hello "
x[1] = "World";                      // Contiene "World"
x[2] = a[0] + a[1];                  // Contiene "Hello World"

int[] y = { 1, 2, 3 };               // Vettore composto da tre interi
int z = y[1];                        // La variabile z vale 2
```

Enumerazioni

- Definiscono insiemi chiusi di valori
- Il tipo di Default è intero

```
enum Gender    // Enumerazione di tipo int (default) - Sesso
{
  Male,        // Vale 0 (int) - Maschio
  Female       // Vale 1 (int) - Femmina
}
```

```
enum DayOfWeek : byte        // Giorno della settimana
{
  Monday = 1,                // Vale 1 (byte) - Lunedì
  Tuesday = 2,               // Vale 2 (byte) - Martedì
  Wednesday = 3,             // Vale 3 (byte) - Mercoledì
  Thursday = 4,              // Vale 4 (byte) - Giovedì
  Friday = 5,                // Vale 5 (byte) - Venerdì
  Saturday = 6,              // Vale 6 (byte) - Sabato
  Sunday = 7                 // Vale 7 (byte) - Domenica
}
```

Funzioni e Procedure (void)

- Le funzioni ritornano un «qualcosa»
- Le Procedure (void) eseguono una serie di istruzioni
- Passaggio dei Parametri per valore, riferimento (ref) e riferimento in uscita (out)
- E' possibile inserire parametri «opzionali» e «nominali»

```
// Passaggio per valore
int Sum(int x, int y)
{
  return x + y;
}
// Passaggio per valore e per riferimento
void Sum(int x, int y, out int result)
{
  result = x + y;
}
int num = Sum(11, 22);  // num vale 33
Sum(111, 222, out num); // num vale 333
```

```
// Funzione che calcola la somma di due interi
int Sum(int x, int y)
{
  return x + y;
}
// Procedura che scrive un messaggio a video
void Write(string text)
{
  Console.Write(text);
}
```

```
// Funzione che somma di tre numeri con due parametri opzionali
int Sum(int x, int y = 0, int z = 0)
{
  return x + y + z;
}
// Numeri interi
int i = 18, j = 8, k = 5;
// Invocazione classica: il risultato è 31
Sum(i, j, k);
// Uso dei parametri opzionali
Sum(i, j);                // Equivale a: Sum(i, j, 0) e il risultato è 26
Sum(i);                   // Equivale a: Sum(i, 0, 0) e il risultato è 18
// Uso dei parametri nominali
Sum(z: i, y: j, x: k);  // Equivale a: Sum(k, j, i) e il risultato è 31
Sum(z: k, i);           // Errore di compilazione
// Uso di un parametro nominale per assegnare un valore diverso da
// quello di default all'ultimo parametro opzionale
Sum(i, z: k);           // Equivale a: Sum(i, 0, k) e il risultato è 23
Sum(i, ,k);             // Errore di compilazione
```

Istruzioni Condizionali (IF e SWITCH)

```
if (name == "C#")
{
  // Blocco principale (obbligatorio)
}
else
{
  // Blocco alternativo (facoltativo)
}
```

```
switch (name)
{
  case "Alessio":  // Valore singolo
    // …
    break;
  case "Cristian":  // Viene eseguito lo stesso blocco di
  case "Daniele":   // codice nel caso in cui il valore di
  case "Marco":     // name sia uno dei tre indicati
    // …
    break;
  default:          // Blocco di default opzionale
    // …
    break;
}
```

Operatore Binario e Null Condition

- Sono degli «IF-inline»
- Con l'operatore «??» è possibile riassegnare un valore nullo

```
// Se i è dispari, isOdd vale true
bool isOdd = (i % 2 == 1) ? true : false;
// Se x è null, ritorna y, altrimenti ritorna x
string x = null;
string y = "Hello World";
string z = x ?? y; // z vale "Hello World"
```

- L'operatore null condition permette di valutare del codice «nullo»

```
// codice precedente
string name = "";
if (customer != null)
  name = customer.Name;
// nuova sintassi
string name = customer?.Name; // null se customer è null
```

Cicli e Iterazioni

- WHILE
- DO…WHILE
- FOR
- FOREACH

- ***Break***: Termina esecuzione di un ciclo
- ***Continue***: va all'iterazione successiva
- ***Goto***: va ad una istruzione specifica

Formattazioni di Stringhe

```
string name = "Daniele";
string city = "Rionero in Vulture";
string fullName = string.Format("Ti chiami {0} e vieni da {1}", name, city);
```

```
string name = "Daniele";
string city = "Rionero in Vulture";
string fullName = $"Ti chiami {name} e vieni da {city}";
```

Fine Lezione 2

DOMANDE?