**Metodi**

http://msdn.microsoft.com/it-it/library/ms173114%28v=vs.80%29.aspx

Un metodo è essenzialmente un blocco di codice contenente una serie di istruzioni. In C# ogni istruzione viene eseguita nel contesto di un metodo.

I metodi vengono dichiarati all'interno di una classe o di una struttura specificando il livello di accesso, il valore restituito, nonché il nome e gli eventuali parametri del metodo. I parametri sono racchiusi tra parentesi e separati da virgole. Le parentesi vuote indicano che il metodo non richiede parametri. Questa classe contiene tre metodi:

class Motorcycle

{

static void StartEngine() { }

static void AddGas(int gallons) { }

static int Drive(int miles, int speed) { return 0; }

…

static void Main(string[] args)

}

**Parametri dei metodi**

Per passare argomenti a un metodo è sufficiente indicarli tra parentesi quando si effettua la chiamata al metodo. Per il metodo chiamato, gli argomenti in arrivo sono denominati parametri.

Anche i parametri ricevuti da un metodo vengono forniti in un insieme di parentesi, ma è necessario specificare il tipo e un nome per ciascuno di essi. Il nome non deve necessariamente essere uguale all'argomento. Esempio:

public static void PassesInteger()

{

int fortyFour = 44;

TakesInteger(fortyFour);

}

static void TakesInteger(int i)

{

i = 33;

}

In questo esempio un metodo denominato PassesInteger passa un argomento a un metodo denominato TakesInteger. All'interno di PassesInteger l'argomento è denominato fortyFour, mentre in TakeInteger corrisponde a un parametro denominato i. Questo parametro esiste solo all'interno del metodo TakesInteger. È possibile denominare un numero qualsiasi di variabili, che possono essere di qualsiasi tipo, purché non si tratti di parametri o variabili dichiarati all'interno di questo metodo.

Si noti che TakesInteger assegna un nuovo valore all'argomento fornito. Contrariamente a quanto si potrebbe prevedere, il metodo PassesInteger non viene aggiornato in base a questa modifica dopo che TakeInteger viene restituito, ma il valore della variabile fortyFour rimane invariato, perché int è un tipo di valore. Per impostazione predefinita, quando un tipo di valore viene passato a un metodo, viene passata una copia anziché l'oggetto stesso. Trattandosi di copie, le eventuali modifiche apportate al parametro non hanno effetto all'interno del metodo chiamante. I tipi di valore sono definiti in questo modo perché viene passata una copia dell'oggetto anziché l'oggetto stesso. Viene passato il valore, ma non l'oggetto.

Per ulteriori informazioni sul passaggio dei tipi di valore, vedere le pagine: **Passaggio di parametri**

**Valori restituiti**

I metodi possono restituire un valore al chiamante. Se il tipo restituito, ossia il tipo indicato prima del nome del metodo, non è void, il metodo può restituire il valore tramite la parola chiave return. Un'istruzione con la parola chiave return seguita da un valore che corrisponde al tipo restituito restituirà tale valore al chiamante del metodo. La parola chiave return determina anche l'interruzione dell'esecuzione del metodo. Se il tipo restituito è void, un'istruzione return senza alcun valore è comunque utile per interrompere l'esecuzione del metodo. Senza la parola chiave return, l'esecuzione del metodo viene interrotta quando viene raggiunta la fine del blocco di codice. Nei metodi con un tipo restituito non void deve essere utilizzata la parola chiave return per consentire la restituzione di un valore. Questi due metodi utilizzano ad esempio la parola chiave return per restituire valori integer:

class SimpleMath

{

public int AddTwoNumbers(int number1, int number2)

{

return number1 + number2;

}

public int SquareANumber(int number)

{

return number \* number;

}

}

**return**

L'istruzione return termina l'esecuzione del metodo in cui è contenuta e restituisce il controllo al metodo di chiamata. Può anche restituire un valore facoltativo. Se il metodo è un tipo void, è possibile omettere l'istruzione return.

**Esempio**

Nell'esempio seguente, il metodo A() restituisce la variabile Area sotto forma di valore double.

using System;

class ReturnTest

{

static double CalculateArea(int r)

{

double area = r \* r \* Math.PI;

return area;

}

static void Main()

{

int radius = 5;

Console.WriteLine("The area is {0:0.00}", CalculateArea(radius));

}

}

Output

The area is 78.54

**Passaggio di parametri**

http://msdn.microsoft.com/it-it/library/0f66670z%28VS.80%29.aspx

In C# è possibile passare parametri per valore o per riferimento. Il passaggio di parametri per riferimento consente ai membri delle funzioni, ad esempio metodi, proprietà, indicizzatori, operatori e costruttori, di modificare il valore dei parametri e rendere permanenti le modifiche. Per passare un parametro per riferimento, è possibile utilizzare la parola chiave **ref** o **out**.

// Passing by value

static void Square(int x)

{

// code...

}

// Passing by reference

static void Square(ref int x)

{

// code...

}

# Passaggio di parametri di tipi di valore

Una variabile associata a un tipo che rappresenta un valore contiene direttamente i dati, mentre una variabile associata a un tipo che rappresenta un riferimento contiene un riferimento ai dati. Il passaggio a un metodo di una variabile associata a un tipo che rappresenta un valore consiste pertanto nel passaggio al metodo di una copia della variabile. Eventuali modifiche del parametro apportate nel metodo non avranno alcun effetto sui dati originali archiviati nella variabile. Per fare in modo che il metodo chiamato modifichi il valore del parametro, è necessario passarlo per riferimento mediante la parola chiave **ref** o **out**. Per semplicità, negli esempi che seguono viene utilizzata solo la parola chiave **ref**.

Esempio: passaggio di tipi di valore per valore

Nell'esempio che segue viene illustrato il passaggio di parametri associati a tipi che rappresentano un valore per valore. La variabile n viene passata per valore al metodo SquareIt. Eventuali modifiche apportate nel metodo non avranno alcun effetto sul valore originale della variabile.

class PassingValByVal

{

static void SquareIt(int x)

// The parameter x is passed by value.

// Changes to x will not affect the original value of x.

{

x \*= x;

System.Console.WriteLine("The value inside the method: {0}", x);

}

static void Main()

{

int n = 5;

System.Console.WriteLine("The value before calling the method: {0}", n);

SquareIt(n); // Passing the variable by value.

System.Console.WriteLine("The value after calling the method: {0}", n);

}

}

Output

The value before calling the method: 5

The value inside the method: 25

The value after calling the method: 5

Illustrazione del codice

La variabile n, essendo un tipo di valore, contiene i propri dati (il valore 5). Quando si richiama SquareIt, il contenuto di n viene copiato nel parametro x, che viene elevato al quadrato all'interno del metodo. In Main, tuttavia, il valore di n è sempre lo stesso, sia prima che dopo la chiamata al metodo SquareIt. La modifica apportata nel metodo ha effetto solo sulla variabile locale x.

Esempio: passaggio di tipi di valore per riferimento

L'esempio riportato di seguito è analogo a quello precedente, con la differenza che il parametro viene passato utilizzando la parola chiave **ref**. Il valore del parametro risulta modificato dopo la chiamata al metodo.

class PassingValByRef

{

static void SquareIt(ref int x)

// The parameter x is passed by reference.

// Changes to x will affect the original value of x.

{

x \*= x;

System.Console.WriteLine("The value inside the method: {0}", x);

}

static void Main()

{

int n = 5;

System.Console.WriteLine("The value before calling the method: {0}", n);

SquareIt(ref n); // Passing the variable by reference.

System.Console.WriteLine("The value after calling the method: {0}", n);

}

}

Output

The value before calling the method: 5

The value inside the method: 25

The value after calling the method: 25

Illustrazione del codice

In questo esempio non viene passato il valore di n, ma solo un riferimento a n. Il parametro x non è di tipo int ma è un riferimento a un valore **int**, in questo caso un riferimento a n. Pertanto, quando la variabile x viene elevata al quadrato all'interno del metodo, l'operazione ha effetto sull'elemento a cui x fa riferimento, ovvero n.

# Parola chiave REF

class RefExample

{

static void Method(ref int i)

{

i = 44;

}

static void Main()

{

int val = 0;

Method(ref val);

// val is now 44

}

}

È necessario che un argomento passato a un parametro **ref** venga in primo luogo inizializzato. Gli argomenti del parametro out, al contrario, non devono essere inizializzati prima di essere passati.

Le parole chiave **ref** e **out** vengono gestite in modo differente in fase di esecuzione, ma allo stesso modo in fase di compilazione. Non è pertanto possibile sottoporre a overload due metodi di cui uno accetta un argomento **ref** e l'altro un argomento out. Questi due metodi, ad esempio, sono identici in termini di compilazione, quindi questo codice non verrà compilato:

class CS0663\_Example

{

// compiler error CS0663: "cannot define overloaded

// methods that differ only on ref and out"

public void SampleMethod(ref int i) { }

public void SampleMethod(out int i) { }

}

L'overload può essere tuttavia eseguito se un metodo accetta un argomento **ref** o out e l'altro non ne utilizza, come riportato di seguito:

class RefOutOverloadExample

{

public void SampleMethod(int i) { }

public void SampleMethod(ref int i) { }

}

**N.B.**

Le proprietà non sono variabili e pertanto non possono essere passate come parametri **ref**.

# Parola chiave OUT

Con la parola chiave **out** gli argomenti vengono passati per riferimento. È simile alla parola chiave ref, con la differenza che con ref è necessario inizializzare la variabile prima che venga passata. Per utilizzare un parametro **out**, è necessario che la definizione del metodo e il metodo chiamante utilizzino in modo esplicito la parola chiave **out**. Esempio:

class OutExample

{

static void Method(out int i)

{

i = 44;

}

static void Main()

{

int value;

Method(out value);

// value is now 44

}

}

Anche se le variabili passate come argomenti **out** non devono essere precedentemente inizializzate, il metodo chiamante deve assegnare un valore prima che venga restituito.

Le parole chiave **ref** e **out** vengono gestite in modo differente in fase di esecuzione, ma allo stesso modo in fase di compilazione. Pertanto, se un metodo accetta un argomento **ref** e l'altro un argomento **out**, non è possibile eseguirne l'overload. Questi due metodi, ad esempio, sono identici in termini di compilazione, quindi questo codice non verrà compilato:

class CS0663\_Example

{

// compiler error CS0663: "cannot define overloaded

// methods that differ only on ref and out"

public void SampleMethod(out int i) { }

public void SampleMethod(ref int i) { }

}

L'overload può essere tuttavia eseguito se un metodo accetta un argomento **ref** o **out** e l'altro non ne utilizza, come riportato di seguito:

class RefOutOverloadExample

{

public void SampleMethod(int i) { }

public void SampleMethod(out int i) { }

}

**N.B.**

Le proprietà non sono variabili e pertanto non possono essere passate come parametri **out**.

Per informazioni sul passaggio delle matrici, vedere Passaggio di matrici mediante i parametri ref e out.

**Esempio**

La dichiarazione di un metodo **out** risulta utile quando si desidera che un metodo restituisca più valori. Un metodo che utilizza un parametro **out** può comunque restituire variabili come tipo restituito (vedere return), ma può anche restituire uno o più oggetti a un metodo chiamante come parametri **out**. Nell'esempio riportato di seguito viene utilizzato il parametro **out** per restituire tre variabili con un'unica chiamata al metodo. Si noti che il terzo argomento è assegnato a null. In questo modo i metodi possono restituire valori in modo facoltativo.

class OutReturnExample

{

static void Method(out int i, out string s1, out string s2)

{

i = 44;

s1 = "I've been returned";

s2 = null;

}

static void Main()

{

int value;

string str1, str2;

Method(out value, out str1, out str2);

// value is now 44

// str1 is now "I've been returned"

// str2 is (still) null;

}

}

# Esempio: scambio di tipi di valore

# Un esempio comune di modifica dei valori dei parametri passati è costituito dal metodo Swap, in cui si passano due variabili, x e y, il cui contenuto viene scambiato dal metodo. È necessario passare i parametri al metodo Swap per riferimento. In caso contrario, si opererà semplicemente su una copia locale dei parametri all'interno del metodo. Di seguito è illustrato un esempio del metodo Swap che utilizza i parametri di riferimento:

# static void SwapByRef(ref int x, ref int y)

# {

# int temp = x;

# x = y;

# y = temp;

# }

# static void Main()

# {

# int i = 2, j = 3;

# System.Console.WriteLine("i = {0} j = {1}" , i, j);

# SwapByRef (ref i, ref j);

# System.Console.WriteLine("i = {0} j = {1}" , i, j);

# }

# Output:

# i = 2 j = 3

# i = 3 j = 2

# Passaggio di matrici come parametri

# È possibile passare matrici ai metodi come parametri. Le matrici sono infatti tipi di parametri, pertanto il metodo può modificare il valore degli elementi.

# Passaggio di matrici unidimensionali come parametri

# È possibile passare una matrice unidimensionale inizializzata a un metodo. Esempio:

# PrintArray(theArray);

# Il metodo chiamato nella riga sopra riportata può essere definito nel modo seguente:

# void PrintArray(int[] arr)

# {

# // method code

# }

# È inoltre possibile inizializzare e passare una nuova matrice in un unico passaggio. Esempio:

# PrintArray(new int[] { 1, 3, 5, 7, 9 });

# Esempio 1

# Nell'esempio che segue una matrice di stringhe viene inizializzata e passata come parametro al metodo PrintArray, in cui ne vengono visualizzati gli elementi:

# class ArrayClass

# {

# static void PrintArray(string[] arr)

# {

# for (int i = 0; i < arr.Length; i++)

# {

# System.Console.Write(arr[i] + "{0}", i < arr.Length - 1 ? " " : "");

# }

# System.Console.WriteLine();

# }

# static void Main()

# {

# // Declare and initialize an array:

# string[] weekDays = new string[] { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

# // Pass the array as a parameter:

# PrintArray(weekDays);

# }

# }

# Output 1

# Sun Mon Tue Wed Thu Fri Sat

# Passaggio di matrici multidimensionali come parametri

# È possibile passare una matrice multidimensionale inizializzata a un metodo. Se ad esempio theArray è una matrice bidimensionale:

# PrintArray(theArray);

# Il metodo chiamato nella riga sopra riportata può essere definito nel modo seguente:

# void PrintArray(int[,] arr)

# {

# // method code

# }

# È inoltre possibile inizializzare e passare una nuova matrice in un unico passaggio. Esempio:

# PrintArray(new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } });

# Esempio 2

# In questo esempio una matrice bidimensionale viene inizializzata e passata al metodo PrintArray, in cui ne vengono visualizzati gli elementi.

# class ArrayClass2D

# {

# static void PrintArray(int[,] arr)

# {

# // Display the array elements:

# for (int i = 0; i < 4; i++)

# {

# for (int j = 0; j < 2; j++)

# {

# System.Console.WriteLine("Element({0},{1})={2}", i, j, arr[i, j]);

# }

# }

# }

# static void Main()

# {

# // Pass the array as a parameter:

# PrintArray(new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } });

# }

# }

# Output 2

# Element(0,0)=1

# Element(0,1)=2

# Element(1,0)=3

# Element(1,1)=4

# Element(2,0)=5

# Element(2,1)=6

# Element(3,0)=7

# Element(3,1)=8

# Passaggio di matrici mediante ref e out

# Analogamente a tutti i parametri out, anche il parametro out di un tipo matrice deve essere assegnato prima dell'uso, ovvero assegnato dal chiamato. Esempio:

# static void TestMethod1(out int[] arr)

# {

# arr = new int[10]; // definite assignment of arr

# }

# Come tutti i parametri ref, anche il parametro ref di un tipo matrice deve essere assolutamente assegnato dal chiamante. Non è pertanto necessario che venga assegnato dalla procedura che viene chiamata. Il parametro ref di un tipo matrice può risultare modificato in conseguenza della chiamata. È ad esempio possibile che alla matrice venga assegnato il valore null o che essa venga inizializzata con una matrice diversa. Esempio:

# static void TestMethod2(ref int[] arr)

# {

# arr = new int[10]; // arr initialized to a different array

# }

# Nei due esempi che seguono viene illustrata la differenza tra out e ref quando vengono utilizzati per passare matrici a metodi.

# Esempio 1

# In questo esempio la matrice theArray viene dichiarata nel chiamante (il metodo Main) e inizializzata nel metodo FillArray. Gli elementi vengono quindi restituiti al chiamante e visualizzati.

# class TestOut

# {

# static void FillArray(out int[] arr)

# {

# // Initialize the array:

# arr = new int[5] { 1, 2, 3, 4, 5 };

# }

# static void Main()

# {

# int[] theArray; // Initialization is not required

# // Pass the array to the callee using out:

# FillArray(out theArray);

# // Display the array elements:

# System.Console.WriteLine("Array elements are:");

# for (int i = 0; i < theArray.Length; i++)

# {

# System.Console.Write(theArray[i] + " ");

# }

# }

# }

# Output 1

# Array elements are:

# 1 2 3 4 5

# Esempio 2

# In questo esempio la matrice theArray viene inizializzata nel chiamante (il metodo Main) e passata al metodo FillArray utilizzando il parametro ref. Alcuni degli elementi della matrice vengono aggiornati nel metodo FillArray. Gli elementi vengono quindi restituiti al chiamante e visualizzati.

# class TestRef

# {

# static void FillArray(ref int[] arr)

# {

# // Create the array on demand:

# if (arr == null)

# {

# arr = new int[10];

# }

# // Fill the array:

# arr[0] = 1111;

# arr[4] = 5555;

# }

# static void Main()

# {

# // Initialize the array:

# int[] theArray = { 1, 2, 3, 4, 5 };

# // Pass the array using ref:

# FillArray(ref theArray);

# // Display the updated array:

# System.Console.WriteLine("Array elements are:");

# for (int i = 0; i < theArray.Length; i++)

# {

# System.Console.Write(theArray[i] + " ");

# }

# }

# }

# Output 2

# Array elements are:

# 1111 2 3 4 5555

# Main() e argomenti della riga di comando

# Il metodo Main costituisce il punto di ingresso di un'applicazione console C# o di un'applicazione Windows. Librerie e servizi non richiedono un metodo Main come punto di ingresso. All'avvio dell'applicazione, Main è il primo metodo richiamato. In un programma C# può esistere un solo punto di ingresso.

# Panoramica

# Il metodo Main è il punto di ingresso di un programma EXE, ovvero il punto in cui il controllo del programma inizia e termina.

# Il Main viene dichiarato all'interno di una classe o di uno struct. Main deve essere statico e non deve essere pubblico. (Nell'esempio precedente, ha l'accesso predefinito di privato.) Non è necessario che la classe o la struttura che lo contiene sia statica.

# Main può avere un tipo restituito void o int.

# Il metodo Main può essere dichiarato con o senza un parametro string[] contenente gli argomenti della riga di comando. Quando si utilizza Visual Studio per creare applicazioni Windows Form, è possibile aggiungere il parametro manualmente o utilizzare la classe Environment per ottenere gli argomenti della riga di comando. I parametri vengono letti come argomenti della riga di comando a indice zero.

# Visualizzare gli argomenti della riga di comando

# Gli argomenti forniti per un file eseguibile sulla riga di comando sono accessibili tramite un parametro facoltativo a Main. Gli argomenti vengono forniti sotto forma di una matrice di stringhe. Ogni elemento della matrice contiene un solo argomento. Gli spazi tra gli argomenti vengono rimossi. Si considerino ad esempio le seguenti chiamate della riga di comando di un file eseguibile fittizio:

|  |  |
| --- | --- |
| **Input sulla riga di comando** | **Matrice di stringhe passate a Main** |
| **executable.exe a b c** | "a"  "b"  "c" |
| **executable.exe one two** | "one"  "two" |
| **executable.exe "one two" three** | "one two"  "three" |

# Esempio

# In questo esempio vengono visualizzati gli argomenti della riga di comando passati a un'applicazione della riga di comando. L'output illustrato è relativo alla prima voce della tabella precedente.

# static void Main(string[] args)

# {

# // The Length property provides the number of array elements

# Console.WriteLine("parameter count = {0}", args.Length);

# for (int i = 0; i < args.Length; i++)

# {

# Console.WriteLine("Arg[{0}] = [{1}]", i, args[i]);

# }

# Console.ReadLine();

# }

# /\* Output (assumes 3 cmd line args):

# parameter count = 3

# Arg[0] = [a]

# Arg[1] = [b]

# Arg[2] = [c]

# \*/

# N.B.: In sede di sviluppo utilizzare la casella di testo Command line arguments che si trova in DEBUG, Programma Properties.