

Academy .NET

Metodi

Sommario

- **Utilizzo dei metodi**

Cos'è un metodo?

Perché utilizzare i metodi?

Dichiarazione e invocazione dei metodi

- **Metodi con parametri**

Passaggio di parametri e restituzione di valori

Overloading

Debug e flusso del programma

Metodi: denominazione e best practices

Dichiarazione e invocazione metodi

- I metodi sono parti di codice denominate che possono essere richiamate in seguito

Definizione del metodo:

```
static void PrintHeader()  
{  
    Console.WriteLine("-----");  
}
```

Nome del metodo

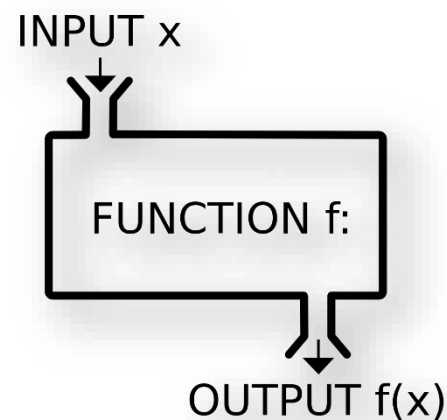
Corpo del metodo

- Invocazione (chiamata) del metodo più volte:

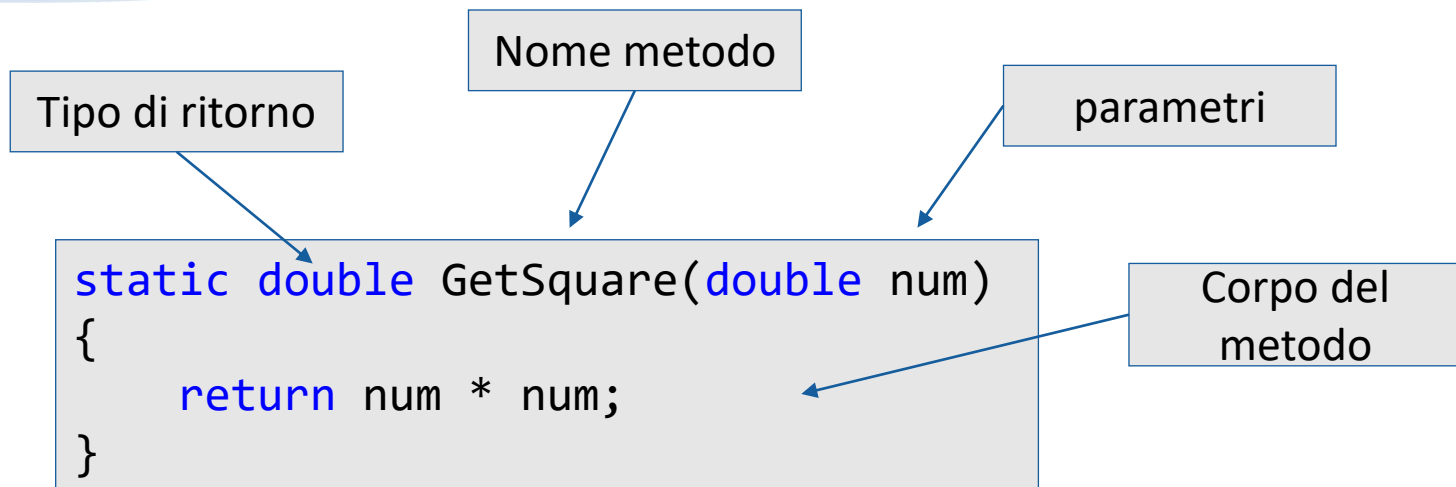
```
PrintHeader();  
PrintHeader();
```

Perchè usare i metodi?

- Programmazione più gestibile:
 - Divide grandi problemi in piccole sezioni
 - Migliore organizzazione del programma
 - Migliora la leggibilità del codice
 - Migliora la comprensibilità del codice
- Evita la ripetizione di codice
 - Migliora la manutenibilità del codice
- Riutilizzabilità del codice
 - Utilizzo di metodi esistenti più volte



Dichiarare i metodi



- I metodi si dichiarano all'interno di una classe
- **Main()** è un metodo
- Le variabili all'interno di metodi sono dette **variabili locali**

Chiamata al metodo

- I metodi vengono prima dichiarati, quindi invocati

```
static void PrintHeader()  
{  
    Console.WriteLine("-----");  
}
```

- I metodi possono essere invocati (chiamati) con il loro nome + ():

```
static void Main()  
{  
    PrintHeader();  
}
```


Parametri dei metodi

- I parametri dei metodi possono contenere ogni tipo di dato

```
static void PrintNumbers(int start, int end)
{
    for (int i = start; i <= end; i++)
    {
        Console.Write("{0} ", i);
    }
}
```

- Chiama il metodo con determinati valori (argomenti)

```
static void Main()
{
    PrintNumbers(5, 10);
}
```

Parametri dei metodi

- È possibile passare zero o più parametri
- È possibile passare parametri di diversi tipi
- Ogni parametro ha nome e tipo

Più parametri

```
static void PrintStudent(string name, int age, double grade)
{
    Console.WriteLine("Student: {0}; Age: {1}, Grade: {2}",
        name, age, grade);
}
```

Parametri con valori di default

- I parametri possono accettare valori predefiniti:

```
static void PrintNumbers(int start = 0, int end = 100)
{
    for (int i = start; i <= end; i++)
    {
        Console.Write("{0} ", i);
    }
}
```

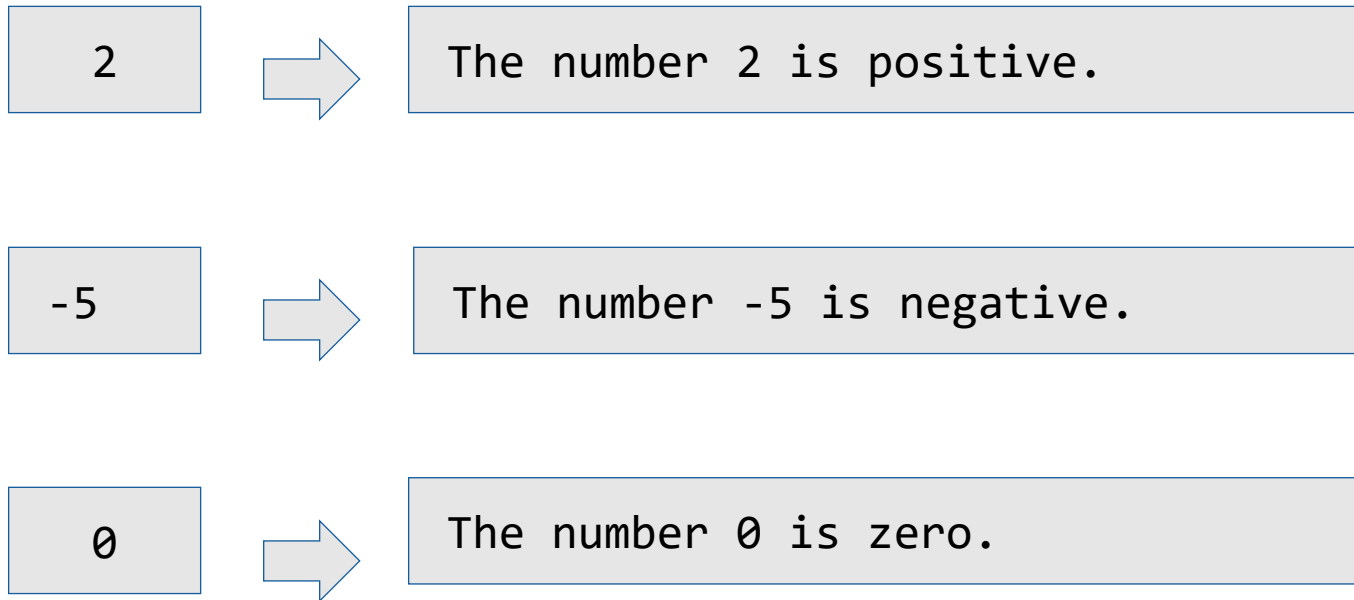
Valori predefiniti

- Il metodo sopra può essere chiamato in diversi modi:

```
PrintNumbers(5, 10);
PrintNumbers(15);
PrintNumbers();
PrintNumbers(end: 40, start: 35);
```

Problema - segno di numero intero

- Crea un metodo che stampa il segno di un numero intero **n**:



Soluzione - segno di numero intero

```
static void PrintSign(int number)
{
    if (number > 0)
        Console.WriteLine("The number {0} is positive", number);
    else if (number < 0)
        Console.WriteLine("The number {0} is negative.", number);
    else
        Console.WriteLine("The number {0} is zero.", number);
}

static void Main()
{ PrintSign(int.Parse(Console.ReadLine())); }
```

Tipi di ritorno dei metodi

- Tipo **void** - non restituisce valore (esegue solo il codice)

```
static void AddOne(int n)
{
    n += 1;
    Console.WriteLine(n);
}
```

Nessun tipo di ritorno

- Altri tipi: valori restituiti, in base al tipo restituito del metodo

```
static int PlusOne(int n)
{
    return n + 1;
}
```

Return **int**

return

- La parola chiave **return** interrompe immediatamente l'esecuzione del metodo.
- Restituisce il valore specificato:

```
static string ReadFullName()  
{  
    string firstName = Console.ReadLine();  
    string lastName = Console.ReadLine();  
    return firstName + " " + lastName;  
}
```

return string



- I metodi Void possono essere terminati semplicemente usando **return**

return;

Valori di ritorno

- **Il valore restituito può essere:**

Assegnato a una variabile

```
int max = GetMax(5, 10);
```

Usato nell'espressione

```
decimal total = GetPrice() * quantity * 1.20m;
```

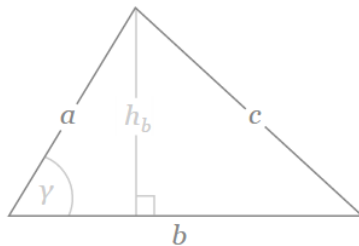
Passato a un altro metodo

```
int age = int.Parse(Console.ReadLine());
```

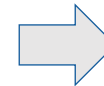

Problema: calcolare l'area del triangolo

- Crea un metodo che calcoli e restituisca l'area di un triangolo per **base** e **altezza** date:

$$A = \frac{h_b b}{2}$$



$$\begin{aligned} b &= 3 \\ h_b &= 4 \end{aligned}$$



$$A = 6$$

Soluzione: calcolare l'area del triangolo

```
static double CalcTriangleArea(double width, double height)
{
    return width * height / 2;
}
```

```
static void Main()
{
    double width = double.Parse(Console.ReadLine());
    double height = double.Parse(Console.ReadLine());
    Console.WriteLine(CalcTriangleArea(width, height));
}
```

Firma del metodo

- La combinazione del nome del metodo e dei parametri è chiamata **firma**

```
static void Print(string text)
{
    Console.WriteLine(text);
}
```

Firma del metodo



- La firma distingue metodi con lo stesso nome
- Quando metodi con lo stesso nome hanno firme diverse, questo viene chiamato "**overloading**" del metodo

Overloading dei metodi

- Utilizzo dello stesso nome per più metodi con firme diverse (nome e parametri del metodo)

```
static void Print(string text)
{
    Console.WriteLine(text);
}
static void Print(int number)
{
    Console.WriteLine(number);
}
static void Print(string text, int number)
{
    Console.WriteLine(text + ' ' + number);
}
```


Firma e tipi di ritorno

Il tipo restituito dal metodo non fa parte della sua firma-

- Considera l'esempio:

```
static void Print(string text)
{
    Console.WriteLine(text);
}
static string Print(string text)
{
    return text;
}
```

Errore di
compilazione



Come farebbe il compilatore a sapere quale metodo chiamare?

Flow di esecuzione

Dopo l'esecuzione di un metodo, il programma continua.

```
static void Main()
{
    Console.WriteLine("before method executes");
    DoSomething();
    Console.WriteLine("after method executes");
}
static void DoSomething()
{
    Console.WriteLine("I'am inside DoSomething()");
}
```

Chiamata del
metodo

Nominare i metodi

- Linee guida per la denominazione dei metodi
Utilizzare nomi significativi e immediati
I nomi dei metodi dovrebbero rispondere alla domanda: Cosa fa questo metodo?

FindStudent, LoadReport

Nominare i parametri

Nomi dei parametri del metodo:

Forma preferita: [sostantivo] o [Aggettivo] + [sostantivo]

Utilizzare il **camelCase**

Nome significativo

Lo scopo dovrebbe essere compreso con facilità

```
firstName, report, speedKmH, fontSizeInPixels, font
```

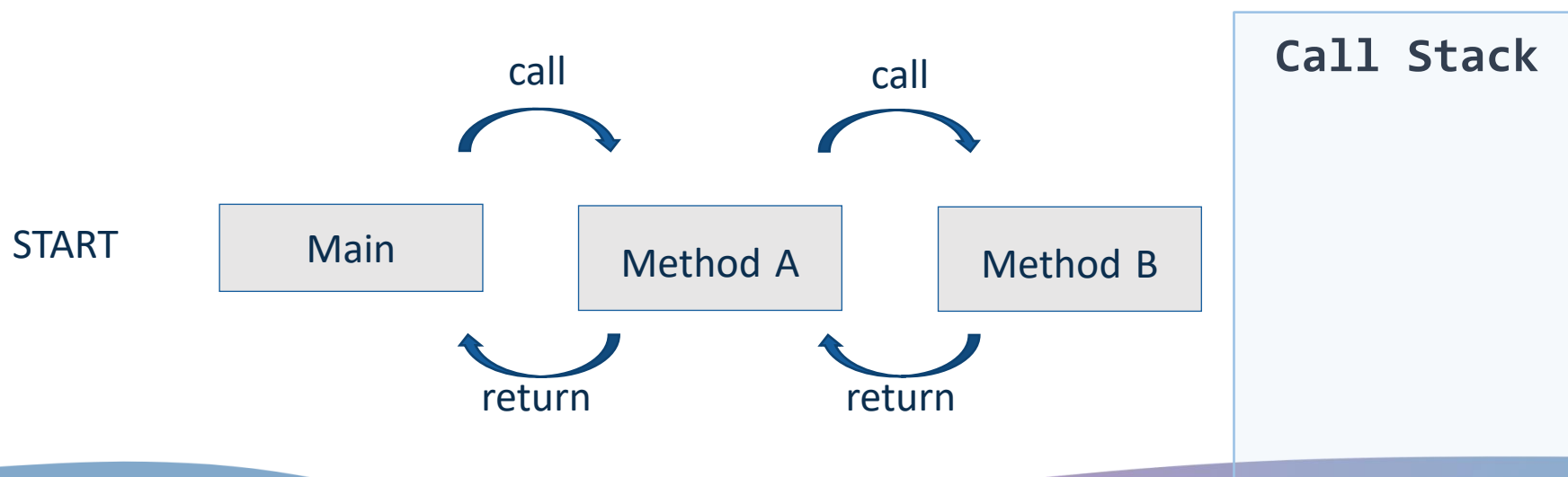

Metodi – Best Practices

- Ogni metodo dovrebbe eseguire una **singola attività** ben definita
 - Il nome del metodo dovrebbe descrivere l'attività in modo chiaro e non ambiguo
- É preferibile evitare metodi troppo lunghi

```
private static void PrintReceipt()  
{  
    PrintHeader();  
    PrintBody();  
    PrintFooter();  
}
```

Lo stack

- "Lo **stack**" memorizza le informazioni sulle subroutine (metodi) attive di un programma per computer
- Tiene traccia del punto in cui ogni subroutine attiva deve restituire il controllo al termine dell'esecuzione



Domande & approfondimenti

Academy .NET