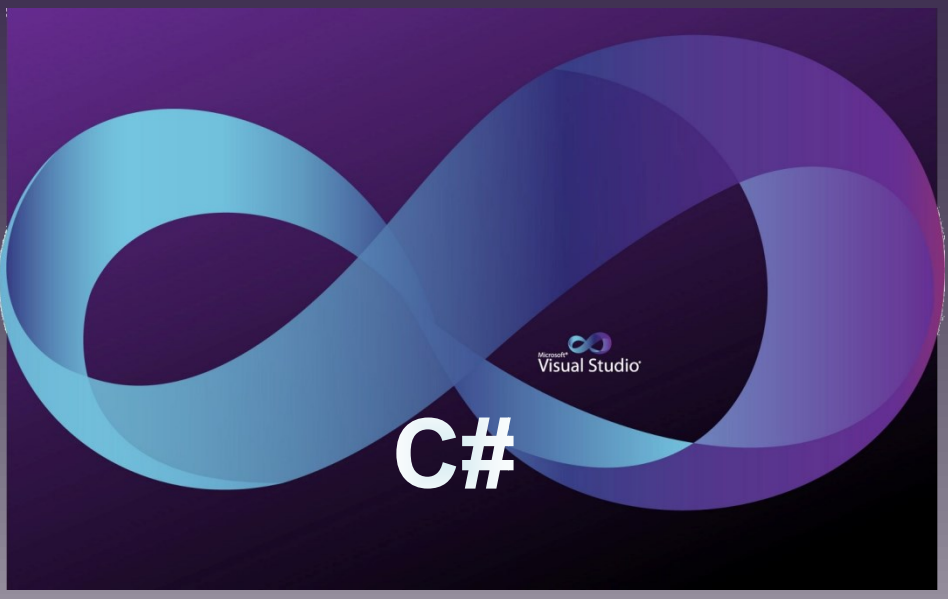
ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI



CORSO DI PROGRAMMAZIONE

OBJECT ORIENTED PROGRAMMING: I METODI

DISPENSA 15.02

15-02_OOP_Metodi_[19]

Versione del: 16/12/2020
Revisione numero: 19

Prof. Andrea Zoccheddu
Dipartimento di Informatica

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

I METODI DEGLI OGGETTI

I METODI DEGLI OGGETTI

METODI DI ISTANZA

COSA SI INTENDE PER METODO?

In una classe è possibile definire delle operazioni che svolgono compiti e/o elaborazioni di dati per far evolvere l'oggetto e per colloquiare con l'esterno.

Le operazioni che un oggetto può svolgere hanno il nome di metodi. In sintesi un metodo è un'operazione che, a partire da oggetto, esegue elaborazione e può restituire un risultato e/o un effetto sui dati dell'oggetto.

La definizione dei metodi è quella già analizzata in questo corso per le funzioni (appunto i metodi del programma), la cui sintassi assume la forma seguente:

VC#

```
public class NomeClasse
{
   . . .
   descrittore TipoMetodo NomeMetodo ( lista parametri)
   {
      //corpo del metodo
   }
}
```

Per esempio una classe può essere così definita:

VC#

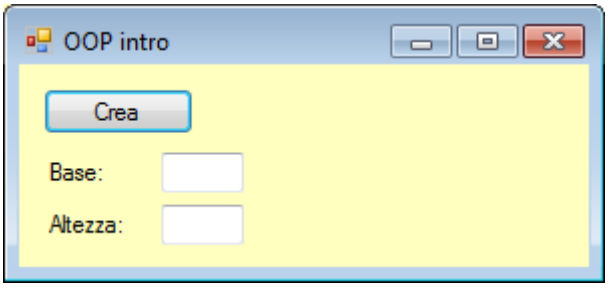
```
public class Rectangle
{
   private double side1, side2;
   public double Area ( )
   {
      //corpo del metodo
      return side1 * side2 ;
   }
}
```

Nell'esempio precedente, il metodo è di tipo double (se non restituisce alcun valore, il metodo può essere definito di tipo void); il metodo non ha parametri poiché i dati su cui lavora sono interni all'oggetto stesso.

Il metodo può utilizzare gli attributi anche se questi sono privati.

PROGETTO GUIDATO

- Avviare Visual Studio e creare un nuovo progetto;



- Creare una nuova classe **Rectangle**:

VC#

```
class Rectangle
{
    //attributi
    public double side1, side2;
    //costruttore
    public Rectangle(double p1, double p2)
    {
        side1 = p1;
        side2 = p2;
    }
    //metodo Area
    public double Area()
    {
        return side1 * side2;
    }
    //metodo Perimetro
    public double Perimetro()
    {
        return 2 * (side1 + side2);
    }
}
```

- Dopo aver progettato il Form1 come nella figura illustrata, associare al pulsante il codice:

VC#

```
private void button1_Click(object sender, EventArgs e)
{
    double tmp1 = Convert.ToDouble(textBox1.Text);
    double tmp2 = Convert.ToDouble(textBox2.Text);
    Rectangle r1 = new Rectangle(tmp1, tmp2);
    double area1 = r1.Area();
    MessageBox.Show("Rettangolo r1 con area:\n" + area1);
    double peri1 = r1.Perimetro();
    MessageBox.Show("Rettangolo r1 con perimetro:\n" + peri1);
}
```

- Provare il progetto con valori 3.5 e 4.5

INVOCAZIONE DI METODI DI ISTANZA

In Visual C#, anche l'accesso ai metodi è molto simile a quello relativo agli attributi: in entrambi i casi si usa la notazione puntata.

La principale differenza è l'obbligo di usare le parentesi tonde, anche se non ci sono parametri da passare. Nel caso dell'esempio del Rettangolo appena proposto, è possibile invocarlo nel seguente modo:

VC#
```
Rectangle figura = new Rectangle(3 , 7);
double risultato = figura.Area();
```

Nell'esempio l'invocazione del metodo si nota nell'ultima riga di codice. Il metodo è invocato a partire dall'oggetto figura, prima dichiarato e istanziato, usando la notazione col punto.

Il metodo Area deve essere invocato con le parentesi tonde, sebbene non vi siano argomenti da passare.

TIPOLOGIA DI METODI

In Visual C#, anche la notazione puntata è molto simile a quello relativo agli attributi: in entrambi i casi è possibile definire la visibilità coi descrittori public e private (e in seguito vedremo altre tipologie di metodi). I metodi pubblici possono essere invocati dall'esterno, mentre quelli privati sono invisibili dall'esterno.

Per esempio una classe può essere definita come nel codice qui di seguito:

VC#
```
public class Voto
{
   private int voto;
   private void Incrementa ( )  //---metodo Incrementa è privato
   {
     if (voto < 10)
     voto++;
   }
   public void Migliora ( int punti ) //---metodo Migliora è pubblico
   {
     for (int k = 0; k < punti; k++)
           Incrementa ();
   }
}
```

Consideriamo l'esempio sopra proposto; il metodo Incrementa è privato e quindi NON può essere utilizzato dall'esterno; lo scopo del metodo Incrementa è di aumentare di +1 il voto incapsulato all'interno dell'oggetto, badando che non possa eccedere il valore 10.

Sebbene il metodo Incrementa sia privato, esso può essere invocato da un altro metodo e, nel nostro esempio, può essere invocato da Migliora; il metodo Migliora aumenta il voto di un certo numero di punti, ma per farlo utilizza il metodo Incrementa.

OVERLOAD DI METODI

Abbiamo già esaminato l'overload dei costruttori; in Visual C# è possibile utilizzare anche l'overload dei metodi. Il principio generale è lo stesso, con l'obbligo di definire i metodi omonimi in maniera non ambigua, ovvero sempre distinguibile dal compilatore. Per esempio è possibile costruire la seguente classe:

VC#
```
public class Voto
{
   public int voto;
   public void Incrementa ( )
   {
     if (voto < 10)
       voto++;
   }
   public void Incrementa ( int punti )
   {
     for (int k = 0; k < punti; k++)
       Incrementa ();
   }
}
```

Nel precedente esempio c'è un overload del metodo Incrementa.

Per la distinzione dei metodi è ininfluente il descrittore pubblico o privato, cioè non è sufficiente che uno sia pubblico e l'altro privato per distinguerli.

Per distinguere i metodi è necessario che abbiano una firma riconoscibile; in sostanza non basta che rendano un tipo diverso, né che i parametri siano di tipo diverso (c'è il rischio che siano compatibili): deve cambiare il numero di parametri oppure il tipo di parametro deve essere assolutamente riconoscibile (per esempio in un caso si usa string e nell'altro si usa double).

Se esistono metodi sovraccaricati (overloaded) è necessario che il compilatore sappia distinguerne l'invocazione:

VC#
```
Voto storia = new Voto();
storia.Incrementa();
storia.Incrementa(2);
MessageBox.Show("Esito: "+storia.voto);
```

La prima invocazione di Incrementa è riferita al metodo senza parametri, mentre quella con l'argomento 2 è riferita al metodo con un parametro.

ACCESSO DEGLI ATTRIBUTI DAI METODI

I metodi di un oggetto possono sempre accedere agli attributi dello stesso oggetto anche se questi ultimi sono privati. In effetti il concetto di privatezza è riferito alla impossibilità di un accesso dall'esterno alle informazioni riservate; ma i metodi appartengono all'oggetto stesso, a pari merito con gli attributi e quindi ad essi non si applicano restrizioni.

Un esempio a tal proposito è stato già proposto in questa dispensa.

PROTEGGERE IL DATO E ACCEDERVI DA METODI

Sebbene la questione della modifica di attributi incapsulati sarà trattata nella discussione sulle Proprietà di un oggetto, è giusto il caso di considerare un modo consueto di costruire oggetti «sicuri».

In generale è opportuno incapsulare gli attributi dell'oggetto e affidare la loro modifica a opportuni metodi; questi metodi sono generalmente chiamati Get e Set (Rendi e Prendi).

VC#

```
public class Voto
{
   private int _voto;

   public int GetVoto ( ) //---metodo di accesso in scrittura
   {
      return _voto;
   }

   public void SetVoto ( int voto ) //---metodo di accesso in lettura
   {
      this._voto = voto;
   }
}
```

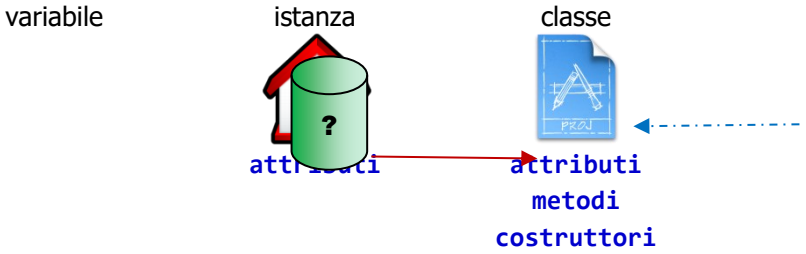
Una classe quindi potrebbe essere costruita in un modo analogo al seguente:

Una sintassi tradizionale prevede di scrivere i nomi di attributi privati con un tratto basso iniziale.

ACCESSO AI DATI E CASI PARTICOLARI

PAROLA CHIAVE THIS

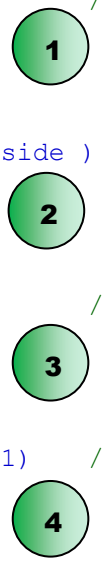
In precedenza è stata usata la parola chiave **this**. Questa parola riservata del linguaggio serve per fare riferimento all'oggetto stesso. La parola this può essere usata solo all'interno di una classe, nel momento della sua definizione. Quando una classe utilizza this, significa che si sta discutendo dell'istanza che ci sarà di quella classe. Consideriamo la seguente figura, che illustra come una variabile faccia riferimento ad un'istanza e quest'ultima è costruita basandosi sulla classe che ne costituisce il progetto.



Quando l'istanza è costruita, si predispone la memoria per accogliere tutti i dati (attributi) previsti dalla sua classe. Per risparmiare memoria, però, i metodi si conservano tutti nello stesso luogo, ma quando si invoca un metodo, questo deve fare riferimento alla sua istanza.

VC#

```
public partial class Square
{
   private int _side;
   public int GetSide ( )          //---metodo senza parametri
   {
     return this._side;
   }
   public void SetVoto ( int side ) //---metodo con 1 parametro
   {
     this._side = side;
   }
   public Square ( side )          //---costruttore con 1 parametro
   {
     this._side = side ;
   }
   public Square ( ) : this (1)    //---costruttore senza parametri
   {
   }
}
```



Analizziamo adesso l'esempio appena proposto.

❶ il primo caso di this non ha un'utilità immediata; in questo caso **this** significa «questo oggetto» e quindi la scrittura va intesa nel senso di «usa l'attributo di questo oggetto» e quindi poteva anche non essere usata; è utile comunque per ribadire che l'identificatore appartiene all'oggetto

❷ il secondo caso di this serve per distinguere il parametro dall'attributo, e questo uso è utile specie quando essi abbiano lo stesso nome; anche in questo caso **this** significa «questo oggetto» e quindi la scrittura va intesa nel senso di «usa l'attributo di questo oggetto»

❸ il terzo caso di this è identico al secondo (distinguere il parametro dall'attributo) ma è usato dentro un costruttore; in questo caso **this** significa «l'oggetto in fase di costruzione» e fa riferimento alla memoria che è stata appena allocata

❹ il quarto caso di this è usato per invocare un costruttore della stessa classe; in questo caso **this** significa «la classe dell'oggetto in fase di costruzione» e fa riferimento alla classe di modello per la memoria appena allocata

INVOCARE METODI DA ALTRI METODI

Anche per le invocazioni di metodi da altri metodi è possibile usare la parola **this**.

VC#

```
public partial class Square
{
   public int Area ( )              //---metodo senza parametri
   {                                                          5
     return this._side * this._side;
   }
   public double Diagonal ()        //---metodo senza parametri
   {                                                          6
     return Math.Sqrt( this.Area() );
   }
}
```

❺ nel quinto uso di this (due volte) semplicemente è l'analogo del primo esempio già discusso; **this** serve per indicare «questa istanza di oggetto»

❻ nel sesto caso di this invece si vede come il metodo **Diagonal** sia capace di invocare un altro metodo della stessa classe, e (sebbene non obbligatorio) fa uso di **this** per ribadire che i due metodi appartengono allo stesso oggetto.

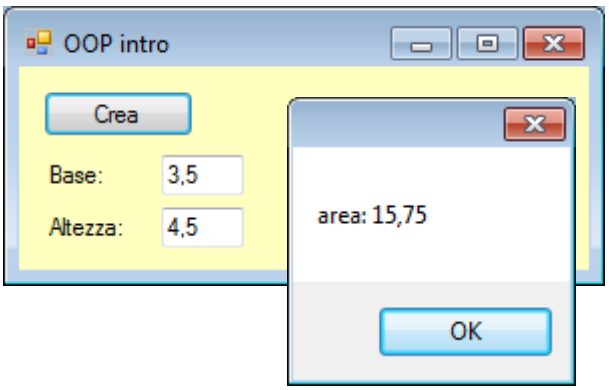
In una buona programmazione per oggetti gli attributi di una classe saranno privati

Si definiranno quindi metodi pubblici per l'accesso in lettura e scrittura

La classe fornirà poi costruttori pubblici e altri metodi pubblici per poterla rendere fruibile dall'esterno

PROGETTO GUIDATO CON COSTRUTTORI SOVRACCARICATI

- Si crei un nuovo progetto in Visual Studio
- Si approntì un Form1 simile alla figura qui a lato



- Si dichiari una classe come nel codice seguente:

VC#

```
public class Rectangle
{
    private double side1, side2;
    //costruttori
    public Rectangle(double x, double y)
    {
        side1 = x; side2 = y;
    }
    public Rectangle(double x)
    {
        side1 = x; side2 = x;
    }
    //metodi
    public double Area()
    {
        return side1 * side2;
    }
    public double Diagonal()
    {
        return Math.Sqrt(Area());
    }
}
```

- Si prepari un codice per il pulante button1:

VC#

```
private void button1_Click(object sender, EventArgs e)
{
    double alt = Convert.ToDouble(textBox1.Text);
    double bas = Convert.ToDouble(textBox2.Text);
    Rectangle r = new Rectangle(alt , bas);
    double risultato = r.Diagonal();
    MessageBox.Show("diagonale: " + risultato);
}
```

- Si esegua il progetto e si esamini con attenzione il modo in cui è costruita la classe

ESERCIZI

ESERCIZI BASE

ESERCIZIO 1 PUNTO CARTESIANO

- Dichiarare un oggetto Punto, con le due coordinate come attributi privati
- Definire tre costruttori:senza parametri (origine), con un parametro (su retta y = x) e con due parametri
- Definire i metodi Get e Set
- Definire il metodo Norma che calcola la distanza dall'origine
- Preparare un'interfaccia che consente di invocare i metodi

ESERCIZIO 2 STUDENTE

- Dichiarare un oggetto Studente, con attributi pubblici Nome, Città, e un attributo privato di tipo array di interi per contenere 11 voti
- Definire un costruttore per creare uno studente con nome e città presi da parametro e un vettore con 11 voti casuali compresi tra 1 e 10
- Definire un metodo pubblico Media() che rende la media dei voti
- Definire un metodo pubblico Insufficienza() che rende il numero di insufficienze
- Definire un metodo pubblico Promosso() che rende true se tutti i voti sono >= a 6;
- Definire un metodo pubblico Promuovi() che trasforma le insufficienze in 6
- Preparare un'interfaccia per usare i metodi

ESERCIZIO 3 CORSO DI STUDENTI

- Dichiarare l'oggetto studente del precedente esercizio
- Dichiarare un oggetto Corso, con un attributo pubblico di tipo array di Studenti per contenere 31 studenti
- Definire un costruttore per creare nel vettore 31 studenti tutti con nome «Enea» e città «Ilio» e ciascuno con 11 voti casuali compresi tra 1 e 10
- Definire un metodo pubblico Media() che rende la media dei voti del corso
- Definire un metodo pubblico Insufficienti() che rende il numero studenti con almeno tre insufficienze
- Definire un metodo pubblico Promossi() che rende il numero di studenti senza insufficienze
- Definire un metodo pubblico Decreto() che trasforma le insufficienze di tutti gli studenti in 6
- Preparare un'interfaccia per usare i metodi

ESERCIZIO 4 SQUADRA SPORTIVA

- Dichiarare un oggetto pubblico Squadra, con attributi privati Nome, Città, Giocate, Vinte, Pareggiate
- Definire un costruttore con i parametri Nome e Città e uno senza parametri che definisce stringhe vuote invocando il precedente
- Definire i metodi **Get()** e **Set()** per modificare e sapere Nome e Città
- Definire il metodo privato **Gioca()** che aumenta di +1 Giocate,
- Definire il metodo **Vince()** che invoca Gioca() e aumenta di +1 Vinte,
- Definire il metodo **Pareggia()** che invoca Gioca() e che aumenta di +1 Pareggiate,
- Definire il metodo **Perde()** che invoca Gioca() e basta
- Definire un metodo **Punti()** che rende il valore di 3 x Vinte + 1 x Pareggiate

- Preparare un'interfaccia che consente di stilare la classifica
- L'interfaccia crea tre squadre con nomi e città impostate dal programmatore
- Poi i pulsanti consentono di vincere o pareggiare e di visualizzare la classifica in tempo reale

ESERCIZIO 5 CAMPIONATO SPORTIVO

- Risolvere l'esercizio precedente che definisce una classe pubblica Squadra
- Dichiarare un oggetto Campionato, con un attributo privato vettore di Squadre
- Definire un costruttore senza parametri che prepara il vettore con 12 Squadre
- Definire un costruttore con un parametro vettore di stringhe che conterrà i nomi delle Squadre
- Definire un metodo Pubblico **Giocare()** che fa giocare tutti gli incontri possibili tra le Squadre scegliendo casualmente chi Vince e chi perde o se si pareggia
- Definire un metodo **Classifica()** che un vettore di stringhe ciascuna che riassume
- Preparare un'interfaccia coi seguenti pulsanti:
  button1 crea il Campionato
  button2 fa giocare il Campionato
  button3 visualizza in una listBox1 la classifica

ESERCIZIO 6 RETTANGOLO

- Si dichiari una classe Rettangolo che abbia attributi privati per entrambi i lati
  Si dichiari un attributo statico privato Random già istanziato
- Definire un costruttore senza parametri che crei un rettangolo con valori casuali per i lati
- Definire i metodi Get e Set per leggere e scrivere i valori dei lati del rettangolo
- Definire i metodi opportuni per ottenere i valori dell'area, del perimetro e della diagonale
- Definire dei metodi per raddoppiare e per dimezzare il lato del quadrato
- Preparare un'interfaccia per usare i metodi

ESERCIZIO 7 VETTORE DI RETTANGOLI

- Si risolva l'esercizio precedente che dichiara un oggetto rettangolo
- Si dichiari una classe Collezione che abbia un attributo privato vettore di rettangoli
- Definire un costruttore senza parametri che crei un vettore di rettangoli ciascuno costruito opportunamente
- Definire i metodi Get e Set per leggere e scrivere i valori del vettore di rettangoli con opportuni parametri e tipi
- Definire un metodo opportuno che restituisca la somma delle aree dei rettangoli
- Definire un metodo opportuno che restituisca la somma dei perimetri dei rettangoli
- Definire un metodo opportuno che restituisca la somma delle diagonali dei rettangoli
- Definire dei metodi per raddoppiare e per dimezzare entrambi i lati di tutti i rettangoli del vettore
- Preparare un'interfaccia per usare i metodi

ESERCIZIO 8 DATA

- Si deve implementare una classe Data che rappresenta una data del calendario Gregoriano; la data ha tre attributi privati: giorno, mese e anno.
- La data ha i seguenti costruttori pubblici:
- Senza parametri, crea la data 1/1/1900;
- Con tre parametri, crea la data secondo i parametri indicati;
- La data ha i seguenti metodi pubblici:
- ToString(), che rende la data in forma di unica stringa;
- Bisestile, che rende true se l'anno è bisestile, falso altrimenti;
- Leggi, imposta la data secondo tre parametri in modo valore;
- Scrivi, rende la data secondo tre parametri in modo risultato;
- Domani, che incrementa la data e la porta al giorno successivo;
- Ieri, che decrementa la data e la porta al giorno precedente;
- Giorno, che rende la stringa del giorno della settimana (es. Domenica);

ESERCIZIO 9 ORARIO

- Si deve implementare una classe Orario che rappresenta un orario sessagesimale; l'orario ha tre attributi privati: ore, minuti e secondi.
- L'orario ha i seguenti costruttori pubblici:
- Senza parametri, crea l'orario mezzanotte;
- Con tre parametri, crea l'orario secondo i parametri indicati;
- L'orario ha i seguenti metodi pubblici:
- Leggi, imposta l'orario secondo tre parametri in modo valore;
- Scrivi, rende l'orario secondo tre parametri in modo risultato;
- ToString(), che rende l'orario in forma di unica stringa;
- DopoSecondi, che incrementa l'orario di k secondi, passati per parametro;
- PrimaSecondi, che decrementa l'orario di k secondi, passati per parametro;

ESERCIZIO 10 CARTA (DA GIOCO)

- Si dichiari una classe Carta che abbia Valore e Seme come attributi privato
- Definire un costruttore senza parametri che crei l'asso di picche
- Definire i metodi Get e Set
- Definire il metodo ToString, che rende la stringa che rappresenta la carta
- Definire il metodo booleano Uguale che, presa una carta come parametro, rende true se il parametro è uguale alla carta
- Preparare un'interfaccia per usare i metodi

SOMMARIO