L'incapsulamento

Informatica

Prof. Viglietti Francesco

a.s. 2012-13

Le proprietà

Nelle classi di esempio abbiamo utilizzato dei **campi istanza** che avevano il modificatore di accesso **public** con il quale il codice esterno alla classe poteva accedere direttamente ai loro valori.

In OOP è consigliabile garantire una maggiore "*ristrettezza*" riguardo l'accesso ai campi istanza della classe, cioè "*mascherare*" la natura e la struttura dei campi inclusi nella classe. L'utilizzatore della classe non deve interessarsi di come sono stati creati e/o manipolati i campi istanza dell'oggetto che dovrà gestire all'interno di una applicazione, ma li dovrà solamente utilizzare.

I progettisti di una classe, nascondono, rendendoli privati, tutti (o quasi) i campi istanza di una classe.

La domanda nasce spontanea: "Se i campi vengono dichiarati privati (**private**), come si possono utilizzare all'interno di una applicazione se non possiamo accedervi?"

Il C# mette a disposizione un costrutto particolare creato ad hoc per poter accedere indirettamente ai campi istanza di una classe: le **proprietà**

Get e set

Con le proprietà, siamo in grado di accedere ai campi istanza sia in lettura che in scrittura; questa è la sintassi:

public *tipo* *nome_proprietà*

```
{
   get      {          return nome_campo;        }
   set      {          nome_campo=value;         }
}
```

Il modificatore indica che la proprietà è **public**, quindi accessibile dal codice esterno alla classe, è dichiarata di un **tipo** tra quelli consentiti dal linguaggio e che è in stretta relazione con quello del campo istanza a cui farà riferimento ed infine ha un nome.

Le parole chiave **get** e **set** consentono l'effettivo accesso al campo istanza ed in particolare:

get → saremo in grado di leggere il contenuto del campo. Nel suo blocco di codice noterete la parola chiave **return**. Essa ci restituisce il valore del campo *nome_campo* (tra breve vedremo come).

set → saremo in grado di assegnare un preciso valore al campo *nome_campo* tramite la parola chiave **value**.

Classe studente nuova

Riprendiamo la classe **Studente** della lezione precedente, trasformiamola aggiungendovi due proprietà, eliminiamo il modificatore **public** rendendo entrambi i campi privati (**private)**. Inseriamo le proprietà relative ai campi:

```
class Studente
  {        // campi istanza privati della classe
    string nome;
    string cognome;     // definisco la proprietà per il campo nome
    public string Nome // è dello stesso tipo del campo istanza
      {
        get   // per la lettura
              {  return nome; // ritorna il valore stringa del campo   }
        set   // per l'assegnazione
              {  nome=value; // assegno un valore stringa al campo    }
      }  // definisco la proprietà per il campo cognome
      public string Cognome // è dello stesso tipo del campo istanza
      {
          get   // per la lettura
                {  return cognome; // ritorna il  valore stringa del campo   }
          set    // per l'assegnazione
                {  cognome=value; // assegno un valore stringa al campo  }
      }
} // fine blocco di codice della classe
```

…spiegazione

Ora che i campi istanza sono "*spariti*", in che modo le proprietà verranno usate nel codice? Le proprietà, hanno un nome, e tramite questo accederemo ai campi, esse possono essere considerate come semplici variabili e come tali verranno usate. Quando assegniamo un valore ad una variabile siamo soliti scrivere, per es: **n=124;** con le proprietà faremo la stessa cosa ma, avendo a che fare con un oggetto, dobbiamo usare la notazione puntata (operatore punto); quindi se abbiamo istanziato un oggetto **Alunno** tramite la classe **Studente** e vogliamo assegnargli un nome ed un cognome faremo così:

Alunno.**Nome**="Mario"; // per il nome Alunno.**Cognome**="Rossi"; // per il cognome

queste istruzioni, hanno effettuato un'assegnazione di un valore per i campi privati della classe richiamando automaticamente il costrutto **set** attraverso le rispettive **proprietà** (**Nome** e **Cognome**). La parola chiave **value** rappresenta il valore assegnato da set, dall'esterno al dato interno alla classe.

Se vogliamo leggere i valori contenuti nei campi privati della classe faremo:

Console.WriteLine(Alunno.N**ome**); // per il nome

Console.WriteLine(Alunno.C**ognome**); // per il cognome

le istruzioni richiamano automaticamente il costrutto **get** delle rispettive proprietà che ci restituirà il valore del campo privato associato tramite la parola chiave **return**.

esempio

```
1   using System;
2   class class1
3   {
4      private int x;
5      private int y;
6      public class1(int a)
7      {
8         this.y = a;
9      }
10     public int X
11     {
12        get {   return x;   }
13        set {    x = value;  if (y % 2 == 0) x = 0;
14           else x = 1;     }
15     }
16  }
17  class Test
18  {
19     public static void Main()
20     {
21        Console.Write("Inserisci un numero: ");
22        int a = int.Parse(Console.ReadLine());
23        class1 c = new class1(a);
24        c.X = -1;
25        Console.WriteLine(c.X);
26     }
27  }
```

In questo caso è stata inserita una piccola elaborazione nell'ambito del **set**, come si vede dalla riga 13-14.
La parte di codice che interessa il **get** viene invocata in fase di lettura mentre **set** viene utilizzata in fase di scrittura. E' possibile utilizzarne solo una delle due e pertanto, sulla base di quanto detto in presenza del solo **get** saremo davanti ad una classe read-only mentre in caso se è stato definito solo **set** la classe sarà write-only.
Le proprietà possono anche avere compiti di validazione dei dati prima che questi siano scritti e in questo senso risultano molto utili; ad es:
if (value > 10) x = value;
else x = 0;

Indicizzatori

Si basano sull'uso di indici, proprio come gli array e infatti sono conosciuti come "smartarrays". Tramite gli indicizzatori è permesso accedere agli oggetti con le medesime modalità degli array quindi tramite l'operatore []. Come viene spiegato un po'ovunque così come una proprietà sembra simile ad un campo così un indicizzatore appare simile ad un array. Concettualmente, per quanto richiamino le proprietà sono un po' meno immediati da comprendere rispetto a queste e, come vedremo ci sono anche alcune importanti differenze.

Formalmente un indicizzatore è fatto come segue, usando sempre la keyword **this**:

```
[modificatore][tipo di ritorno] this [argomenti]
{
  get  {  ..... }
  set  {  .....  }
}
```

il modificatore è opzionale.

esempi… Indicizzatori

```
1   using System;
2   class class1
3   {
4       private int[] x = new int[]{3,4,7,10};
5       public int this[int i]
6       {
7           get   {  return x[i];  }
8           set   {  x[i] = value; }
9       }
10  }
11  class Test
12  {
13      public static void Main()
14      {
15          class1 c1 = new class1();
16          c1[3] = 3;
17          Console.WriteLine(c1[3]);
18      }
19  }
```

```
1   using System;
2   class class1
3   {
4       private int[] x = new int[]{3,4,7,10};
5       private int[] y = new int[]{11,12,13,14};
6       public int this[int i, int xy]
7       {
8           get   {  if (xy == 0) return x[i];
9               else return y[i];  }
10          set   {  if (xy == 0) x[i] = value;
11              else y[i] = value; }
12      }
13  }
14  class Test
15  {
16      public static void Main()
17      {
18          class1 c1 = new class1();
19          Console.WriteLine(c1[3,0]);
20          Console.WriteLine(c1[3,1]);
21      }
22  }
```

Indicizzatori

Se l'uso degli indicizzatori ricorda da vicino le proprietà esistono comunque, delle differenze. La prima è che le proprietà sono individuate attraverso i nomi mentre una differenziazione tra gli indicizzatori è possibile solo tramite le signature dal momento che tutte sono introdotte dalla parola this. Inoltre gli indicizzatori usano parametri, cosa che non avviene quando lavoriamo con le proprietà. Queste sono identificate da un nome mentre gli indicizzatori lavorano tramite indici.

esercizio su classe

Vogliamo provare a costruire una classe e definire degli attributi e dei metodi per poi utilizzarla in un programma (ricordiamo sempre che la classe è un oggetto, e come tale deve e può essere usato e messo in comunicazione con altri oggetti). La classe che vogliamo creare è la classe *Calciatore*. Individuiamo degli attributi per questa classe:

- nome
- ruolo
- squadra
- goalSegnati

…evidentemente i primi tre sono delle stringhe e l'ultimo è un intero.

es. su classe … soluzione

```
class Calciatore
{  // attributi
string nome;
string squadra;
string ruolo;
int golSegnati;
```

Dobbiamo creare un I metodo che serve per istanziare un oggetto di tipo calciatore, questo metodo l'abbiamo chiamato *costruttore*. Cosa deve fare? Semplicemente ricevere dei parametri in ingresso e assegnarli all'oggetto classe (in pratica a se stesso).

```
// costruttore (operazione)
public Calciatore(string nome, string squadra, string ruolo)
{  this.nome = nome;
this.squadra = squadra;
this.ruolo = ruolo;
golSegnati = 0;
}
```

l'espressione ***this*** serve per assegnare il valore (passato come parametro) all'attributo della classe, sta ad indicare: *assegna al campo nome di questa classe il valore nome, …, inizializza il valore di gol Segnati a 0*(da notare che non compare **this**, non c'è possibilità di confonderlo con i parametri, non essendoci un parametro per i goal segnati).

Senza il costruttore l'inizializzazione sarebbe stata quella di default!

es. su classe … soluzione

Costruiamo altri due metodi:

```
// metodo (operazione)
public void AggiornaGolSegnati(int gol)
{
golSegnati += gol;
}
// metodo (operazione)
public void VisualizzaGol()
{
Console.WriteLine("{0} – gol segnati: {1}", nome, golSegnati);
}
```

Il primo riceve come parametro il numero di goal segnati e aggiorna il suo score,
il secondo semplicemente manda a video il valore dell'attributo della classe.
Vediamo ora di mettere in un main questa classe e i rispettivi metodi e vediamo di usarli.

```
class MainClass
{
static void Main()
{
Calciatore c = new Calciatore("Filippo Inzaghi", "Milan", "Attaccante");
c.VisualizzaGol();
c.AggiornaGolSegnati(2);
c.VisualizzaGol();
}
}
```

Prova a creare lo stesso esercizio in visuale.