FAQ: proprietà

Linguaggio C#

Anno 2017/2018

Indice generale

1 Cos'è una proprietà?

Una proprietà è un "espediente" sintattico del linguaggio che fornisce l'accesso a un attributo di un oggetto. L'attributo appare come una variabile, ma in realtà l'accesso viene realizzato mediante una coppia di metodi (o uno soltanto).

Ad esempio, il seguente codice mostra un record, `Persona`, che definisce la proprietà `Nome`:

```
public class Persona
{
    private string nome;
    public string Nome
    {
        get { return nome; }
        set { nome = value; }
    }
}
```

`Persona` viene tradotto in:

```
public class Persona
{
    private string nome;
    public string Get_Nome() //-> get accessor
    {
        return nome;
    }

    public void Set_Nome(string value) //-> get accessor
    {
        nome = value;
    }
}
```

Mentre il codice che usa la proprietà:

```
Persona p = new Persona { Nome = "Gianni", Età = 20 };

var nome = p.Nome;  //-> invoca il get accessor
p.Nome = "Filippo";  //-> invoca il set accessor
```

viene tradotto in:

```
Persona p = new Persona { Nome = "Gianni", Età = 20 };

var nome = p.Nome;
var nome = p.Get_Nome();

p.Nome = "Filippo";
p.Set_Nome("Filippo");
```

2 Una proprietà è dunque simile a una variabile?

NO. Una proprietà non è una variabile, esattamente come non lo è un metodo.

Nella maggior parte dei casi si può usare la proprietà *come se fosse* una variabile; ma esistono situazioni nelle quali la sua reale natura emerge chiaramente.

Nel seguente codice, `Persona` definisce due campi pubblici. Il metodo `ModificaEtà()` usa un parametro `ref` per modificare l'età di una persona, aumentandola di 10 anni.

```
public class Persona
{
    public string Nome;
    public int Età;
}

...
static void Main(string[] args)
{
    Persona p = new Persona { Nome = "Gianni", Età = 20 };
    ModificaEtà(ref p.Età); //  Età -> 30

}

static void ModificaEtà(ref int età)  // modifica la variabile originale
{
    età = età + 10;
}
```

L'uso di `ref` implica che, nell'invocazione del metodo, venga passato l'indirizzo del campo `Età`, in modo che le modifiche siano eseguite sulla variabile originale e non su una copia.

Ma se `Persona` definisce delle proprietà invece che dei campi, la situazione cambia: il codice in `Main()` diventa sintatticamente scorretto:

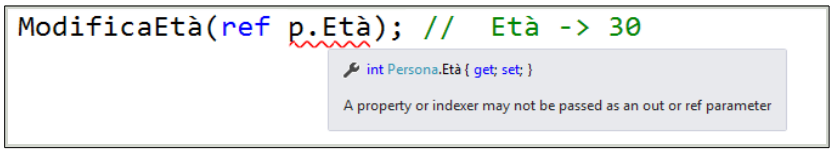
```
public class Persona
{
    public string Nome { get; set; }
    public int Età { get; set; }
}

...

static void Main(string[] args)
{
    Persona p = new Persona { Nome = "Gianni", Età = 20 };
    ModificaEtà(ref p.Età); // Una proprietà non è indirizzabile!
}

static void ModificaEtà(ref int età)  // modifica la variabile originale
{
    età = età + 10;
}
```

Viene prodotto l'errore:



Poiché una proprietà *non è* una variabile, non ha un indirizzo di memoria; dunque non può essere passata come argomento `ref` o `out`.

3 Una proprietà è sempre collegata a un campo privato?

NO. Dipende dalla funzione svolta dalla proprietà, anche se nei record è un caso molto comune. Ma in molte situazioni, compresi i record, non è così.

Il seguente codice mostra un record `Persona` con due campi pubblici e una proprietà che non è associata a un campo particolare:

```
public class Persona
{
    public string Nome;
    public string Cognome;

    public string Nominativo
    {
        get { return Cognome + ", " + Nome; }
    }
}
```

4 Cos'è il *backing field*?

Il termine *backing field* (letteralmente: *il campo che sta dietro*), definisce convenzionalmente il campo privato a cui la proprietà fornisce l'accesso (per quelle proprietà che lo fanno).

Nel seguente codice, il campo `nome` è il *backing field* della proprietà `Nome`:

```
public class Persona
{
    private string nome;  // backing field della proprietà Nome.
    public string Nome
    {
        get { return nome; }
        set { nome = value; }
    }
}
```

5 Esistono delle regole precise sui nomi dei *backing field* e delle proprietà corrispondenti?

NO. Esistono delle convenzioni. Di norma, le proprietà sono pubbliche, mentre i *backing field* sono privati, ergo: le prime hanno lo stesso nome dei secondi, ma con la prima lettera maiuscola.

Altri stabiliscono di prefissare i campi privati con il carattere *underscore*.

6 Cos'è una *proprietà derivata*

Convenzionalmente, si dice *derivata* una proprietà che non ha un *backing field*, ma che restituisce il risultato di un'espressione.

Nel codice seguente, `Nominativo` è una proprietà derivata.

```
public class Persona
{
    public string Nome;
    public string Cognome;

    public string Nominativo
    {
        get { return Cognome + ", " + Nome; }
    }
}
```

Le proprietà derivate rappresentano un classico esempio di come l'interfaccia pubblica di una classe possa incapsulare la sua implementazione. Nel caso precedente, un oggetto di tipo `Persona` definisce tre attributi `Nome`, `Cognome` e `Nominativo`, ma memorizza soltanto due campi.

7 Una proprietà deve avere entrambi gli accessor?

NO. Una proprietà deve avere almeno un *accessor*, ma non obbligatoriamente entrambi.

Una proprietà con il solo *get accessor* si definisce ***get-only***. Una proprietà con il solo *set accessor* si definisce ***set-only***.

Le proprietà *set-only* sono molto rare. Molto spesso, le proprietà *get-only* sono proprietà derivate.

8 I due accessor possono essere vuoti?

SI e **NO**. Il *get accessor* non può essere vuoto, mentre il *set accessor* sì. Il perché è evidente, se si considerano gli *accessor* per quello che sono: dei metodi.

Il seguente codice mostra l'implementazione della proprietà `Nome` e la relativa traduzione eseguita dal linguaggio:

```
public class Persona
{
    private string nome;
    public string Nome
    {
        get { }  //-> non consentito!
        set { }  //-> consentito
    }

    private string nome;

    public string Get_Nome()  // il metodo dichiara un tipo di ritorno
    {                         // e dunque DEVE restituire un valore!
    }

    public void Set_Nome(string value) // un metodo void può essere vuoto.
    {
    }
}
```

Il *get accessor* corrisponde a un metodo che restituisce un valore; coerentemente con le regole del linguaggio, deve farlo!

9 Cos'è una proprietà automatica?

Una proprietà automatica semplifica il lavoro del programmatore, facendo sì che sia il linguaggio a definire automaticamente il *backing field* e gli *accessor*.

Il seguente esempio mostra come una proprietà automatica viene tradotta dal linguaggio:

```
public class Persona
{
    // codice scritto dal programmatore
    public string Nome { get; set; }

    // codice prodotto dal linguaggio
    private string @nome;
    public string Nome
    {
        get { return @nome; }
        set { @nome = value; }
    }
}
```

È evidente che una proprietà automatica è utile soltanto come sostituto di una proprietà con *backing field*, oppure di un campo pubblico.

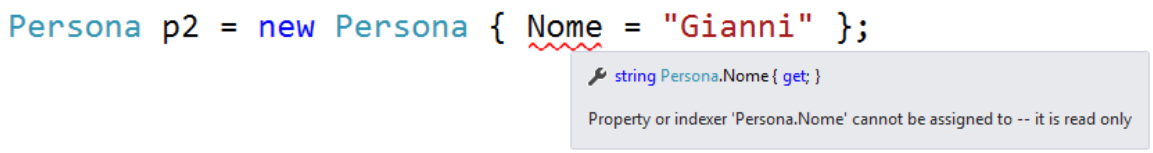
10 Una proprietà automatica può essere *get-only*?

SI. Una proprietà automatica con il solo *get accessor* rappresenta un *campo immutabile*. Infatti è possibile assegnare un valore nella proprietà soltanto nel costruttore, dopodiché non può più essere modificata, nemmeno nei metodi della classe.

```
public class Persona
{
    public Persona(string nome)
    {
        Nome = nome;  // soltanto nel costruttore è possibile assegnare alla proprietà
    }

    public string Nome { get; }  //->proprietà immutabile
}
```

Nota bene: l'immutabilità della proprietà impedisce di impostarla mediante inizializzazione dell'oggetto, come mostra il seguente *screen shot* (supposto di aver aggiunto un costruttore vuoto).



11 Posso avere una proprietà *get-only*, ma inizializzabile?

SI. Basta definire un *set accessor* privato.

```
public class Persona
{
    public string Nome { get; private set; }  //-> inizializzabile
}
```

```
Persona p = new Persona { Nome = "Gianni" };  // OK
```

Dopo aver creato l'oggetto, `p.Nome` non è più modificabile dall'esterno.

12 Nei record, perché dovrei usare delle proprietà invece dei campi pubblici?

Non esiste una regola generale che stabilisce se usare proprietà o campi; in molti scenari le due opzioni sono equivalenti. Ma non sempre.

Nelle applicazioni con interfaccia grafica, ad esempio, (WinForms, WPF, Xamarin, etc) utilizzare

proprietà invece di campi può fare la differenza. Infatti, i controlli grafici (*listbox*, *textbox*, etc) sono in grado di visualizzare automaticamente i dati memorizzati negli oggetti, purché tali dati siano definiti mediante delle proprietà.

Ad esempio, supponi di avere un'applicazione WinForms che debba visualizzare i nomi di un elenco di persone mediante un *listbox*. Il record `Persona` è definito nel seguente modo:

```
public class Persona
{
    public string Nome;
    public int Età;
}
```

Segue il codice dell'applicazione che crea i dati e li visualizza (`lstPersone` è il *listbox*):

```
List<Persona> persone;
private void Form1_Load(object sender, EventArgs e)
{
    persone = new List<Persona>()
    {
        new Persona { Nome = "Filippo", Età = 18 },
        new Persona { Nome = "Sonia", Età = 25 },
        new Persona { Nome = "Andrea", Età = 42 }
    };

    VisualizzaPersone();
}

void VisualizzaPersone()
{
    lstPersone.Items.Clear();
    foreach (var p in persone)
    {
        lstPersone.Items.Add(p.Nome);
    }
}
```

Ebbene, `Visualizza()` non sfrutta la funzionalità di *databinding* offerta dal *listbox*. Questa fa sì che sia il *listbox* a caricare direttamente i record, visualizzando per ognuno di essi il campo stabilito dal programmatore. Unico vincolo: è necessario usare proprietà e non campi pubblici.

```
public class Persona
{
    public string Nome { get; set; }
    public int Età { get; set; }
}
```

Segue la versione di `Visualizza()` che sfrutta la funzionalità di *databinding*:

```
void VisualizzaPersone()
{
    lstPersone.DisplayMember = "Nome";  // stabilisce il campo da visualizzare
    lstPersone.DataSource = persone;  // visualizza automaticamente i dati
}
```

Nei progetti "Class Library" usare le proprietà (e non i campi) nei record pubblici

Quando un record viene usato soltanto all'interno del progetto nel quale è definito, la questione "proprietà ↔ campi pubblici" non è particolarmente rilevante. Il programmatore, se lo ritiene opportuno, può sempre cambiare idea, senza che questo provochi un impatto sul resto dell'applicazione.

La situazione cambia quando il progetto è una libreria che, presumibilmente, sarà referenziata da altri progetti, anche di altri programmatori. Se il record è privato (e dunque inaccessibile agli altri progetti) non ci sono problemi, ma se è pubblico, un'eventuale modifica alla sua interfaccia pubblica può "rompere" la compatibilità con i progetti che usano la libreria.

In questo caso, la scelta migliore (e generalmente adottata) è usare le proprietà.