7. La programmazione ad oggetti

7.1. Elementi introduttivi

Nel Paragrafo 2 abbiamo introdotto i tipi di dati personalizzati definiti tramite l'espressione Type … End Type e abbiamo visto come tale struttura permetta di raggruppare in un unico contenitore differenti tipi di dati. L'unica limitazione, la diciamo adesso, è proprio quella che un tipo di dati definito dall'utente consente di rappresentare solo dati.

Supponiamo di voler gestire un certo numero d'impiegati. Per farlo potremmo pensare di definire il tipo impiegato nel modo seguente:

```
Type Impiegato
   ID As String
   Nome As String
   Costo_Orario As Double
End Type
```

Supponiamo ora di voler calcolare la paga da conferire ad un certo impiegato, a fronte del numero di ore da lui svolte. Per farlo potremmo pensare di scrivere una procedura come quello sottostante, che stampa a video il valore della retribuzione dovuta

```
Private Sub Paga(Im As Impiegato, Ore As Integer)
Dim S As String
   S = CStr(Im.Costo_Orario * Ore)
   S = Im.Nome & " ha guadagnato " & S & "$"
   Debug.Print S
End Sub
```

Un esempio d'utilizzo è mostrato di seguito:

```
Dim E As Impiegato
   E.Costo_Orario = 1.5
   E.Nome = "Pippo"
   E.ID = "Emp1"
   Call Paga(E, Ore)
End Sub
```

Funziona tutto, ma sicuramente, invece di scrivere delle funzioni e/o delle procedure che operano su un tipo personalizzato, sarebbe più utile associare procedure e/o funzioni direttamente al tipo di dato. Gli oggetti ci permettono esattamente di fare ciò.

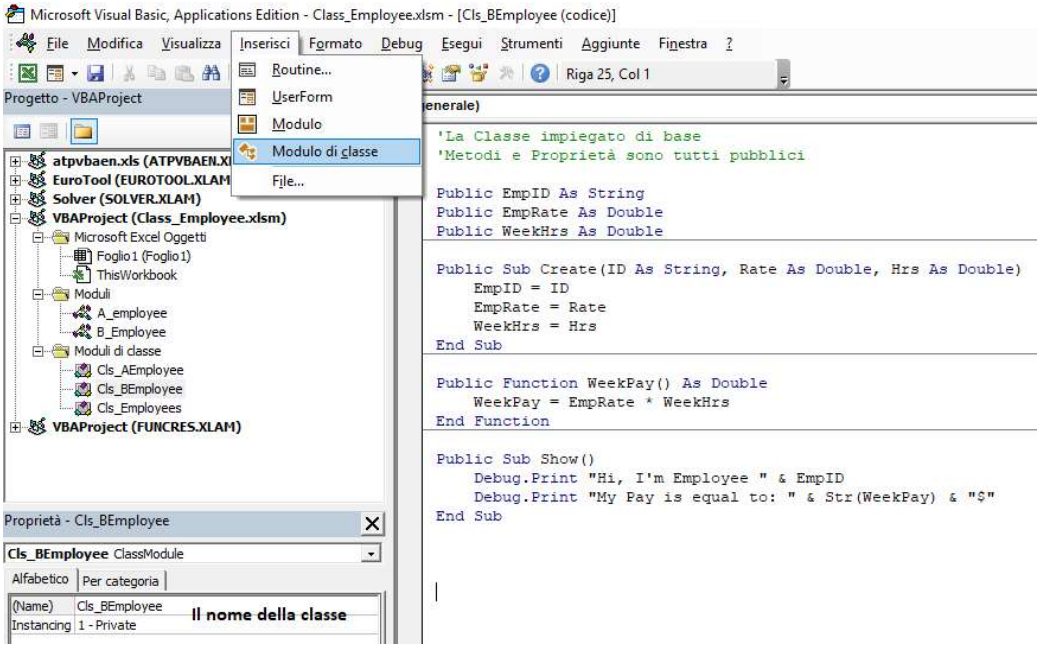
Un oggetto, infatti è contraddistinto dalle seguenti caratteristiche:

- Ha un set di Proprietà - Le proprietà sono attributi della cosa che un oggetto rappresenta; possiamo pensarle come le variabili che costituiscono un tipo definito dall'utente. Nell'esempio dell'impiegato saranno l'ID, il nome e il Costo Orario. Notiamo subito che un oggetto può anche contenere riferimenti ad altri oggetti, per cui una proprietà può far riferimento ad un altro oggetto. Ad esempio, un impiegato può far riferimento ad un superiore e il superiore può essere definito come un oggetto.

- Ha un set di Metodi – I metodi sono in un certo senso le "azioni" svolte o comunque associate all'oggetto. Si tratta di funzioni e/o procedure di proprietà dell'oggetto. Come vedremo, calcola paga potrebbe essere un metodo dell'oggetto impiegato.
- Può avere degli eventi - Pur non essendo esattamente una parte del modello di programmazione degli oggetti, gli eventi sono un utile strumento che permette a un oggetto di scambiare informazioni con il programma che ha creato l'oggetto. In altri termini, un evento è una subroutine residente nel programma che ha creato l'oggetto, ed è chiamata dall'oggetto. Un evento è una tecnica utile che elimina la necessità del programma di monitorare costantemente le modifiche di un oggetto. In questo modo, l'oggetto chiama l'evento per informare il programma di una modifica allo stato dell'oggetto.

7.2. Creare oggetti con VBA

A differenza di altri linguaggi in cui esistono apposite istruzioni per la creazione di un oggetto (o meglio di una classe), per creare una classe in VBA è necessario creare (nell'editor) uno speciale modulo, detto, appunto modulo di classe. Il nome attribuito a tale modulo sarà il nome della classe, come mostrato nella figura seguente.



Un modulo di classe

Notiamo che parliamo di classe quando indichiamo il tipo di dato che stiamo creando (ad esempio l'impiegato) e di oggetto quando creiamo una variabile di quel tipo ed assegniamo ad essa degli specifici valori (ad esempio lo specifico impiegato Mario Rossi). Una volta creato il modulo, resta tutto come prima. Le proprietà della classe (siano esse private o pubbliche) vengono dichiarate esattamente come variabili standard (con l'espressione Dim … As) così come i metodi della classe vengono creati esattamente come funzioni e procedure standard (con l'espressione Function/Sub … End Function/Sub).

Di seguito è riportata la classe impiegato (direttamente derivata dal tipo impiegato, illustrato precedentemente) che avrà il nome del modulo di classe in cui è scritto il codice (ad esempio Cls_Employee)

```
'LA CLASSE IMPIEGATO DI BASE

'Le Proprietà

Public EmpName As String
Public EmpID As String
Public EmpRate As Double
Public WeekHrs As Double 'Proprietà aggiunta rispetto al tipo personalizzato visto in precedenza

' I metodi


Public Sub Create(ID As String, Name As String, Rate As Double, Hrs As Double)
' Un costruttore
   EmpName = Name
   EmpID = ID
   EmpRate = Rate
   WeekHrs = Hrs
End Sub

Public Function WeekPay() As Double
   WeekPay = EmpRate * WeekHrs
End Function
Public Sub Show()
   Debug.Print "Hi, I'm Employee " & EmpID
   Debug.Print "My Pay is equal to: " & Str(WeekPay) & "$"
End Sub
```

Rispetto al tipo personalizzato, abbiamo introdotto:

- la proprietà WeekHrs che contiene le ore lavorate nell'ultima settimana;
- Il metodo Create, una sorta di costruttore[5] che permette di assegnare un valore a tutte le proprietà di un oggetto;
- Il metodo WeekPay che restituisce la paga settimanale dell'impiegato;
- Il metodo Show che mostra a video la descrizione dell'impiegato e la sua paga

Per usare la classe, ossia per creare uno o più oggetti appartenenti a questa classe è necessario definire una o più variabili di tipo oggetto in un qualsiasi modulo (non un modulo di classe). In particolare:

- la dichiarazione di una variabile di tipo oggetto è esattamente uguale alla dichiarazione di una variabile standard (e/o di una variabile di tipo definito dall'utente). In pratica si usa la ben nota sintassi **Dim** Nome_oggetto **As** Nome_Classe, dove Nome_Classe è il nome del modulo di classe in cui è scritto il codice della classe;
- l'assegnazione della variabile è invece leggermente differente, dato che richiede l'uso delle keyword **Set** e **New**, come mostrato di seguito:
  Set Nome_Oggetto = **New** Nome_classe

[5] *A differenza di altri linguaggi VBA non permette di utilizzare dei costruttori veri e propri. Un costruttore oltre a settare le proprietà dovrebbe anche creare l'oggetto. Nel nostro caso, invece, dobbiamo prima creare l'oggetto manualmente e solo dopo possiamo chiamare il costruttore.*

- una volta assegnata la variabile oggetto è possibile accedere (in lettura e scrittura) a tutti le proprietà (ossia alle variabili dichiarate pubbliche) ed è possibile utilizzare tutti i suoi metodi di tipo pubblico. Per farlo è sufficiente far seguire al nome della variabile oggetto il punto "." seguito dal nome della proprietà e/o del metodo desiderato.

Un esempio d'utilizzo della classe Cls_Employee è mostrata di seguito

```
Dim Em As Cls_Employee 'Dichiariamo la variabile oggetto

Set Em = New Cls_Employee 'Creiamo l'oggetto
Em.Name = "Pippo"  'Assegnamo manualmente una proprietà
Set Em = Nothing 'Cancelliamo l'oggetto

Set Em = New Cls_Employee 'Lo ricreiamo
Em.Create "B1", "Topolino", 10.5, 40 'Settiamo in una volta tutte le sue proprietà col costruttore
Em.Show 'Chiamiamo il metodo Show
Set Em = Nothing
```

Si noti l'uso della keyword **Nothing** che permette di cancellare un oggetto, ossia di liberare lo spazio di memoria ad esso associato.

7.3. Collection

Così come è possibile raggruppare più variabili dello stesso tipo all'interno di un array, è possibile raggruppare più oggetti dello stesso tipo (ma anche di tipo diverso) utilizzando le Collections[6]. Le collections sono molto simili ad array monodimensionali, ma sono esse stesse oggetti. In pratica sono oggetti dotati delle seguenti proprietà e dei seguenti metodi:

Nome	Tipo	Sintassi	Effetto
Count	Proprietà	[-]	Restituisce il numero di oggetti contenuti nella collection
Add	Metodo	.Add(Item,[key],[before],[after])	Aggiunge Item alla collection. Se fornita, key è un'etichetta che permette di effettuare una ricerca non solo per indice, ma anche per etichetta. Item, di default, viene messo in fondo alla collection (indice massimo), a meno che non venga passato uno dei parametri before o after che consentono di definire in che posizione mettere il nuovo item
Remove	Metodo	.Remove(Index)	Rimuove l'elemento nella posizione o con l'etichetta specificata da Index
Item	Metodo	.Item(Index)	Assegna, ad un'altra variabile oggetto, l'oggetto della collection definito da Index

[6] *In effetti anche per raggruppare gli oggetti potremmo usare degli array, ma questa non sarebbe un buona pratica di programmazione*

Di seguito è riportata una procedura che genera casualmente un numero di impiegati (5 di default), e assegna loro un ID e una paga oraria casuale, li assegna uno alla volta ad una collection chiamata Employees (ndr. si noti che, convenzionalmente, le collezioni vengono indicate con nomi che finiscono con la esse per indicare il "plurale"). Dopodiché, si cicla sulla collection per mostrare a video le informazioni di tutti gli impiegati aggiunti alla collection stessa. Infine, la collection viene cancellata; così facendo vengono cancellati automaticamente anche tutti gli oggetti ad essa appartenenti.

Si noti che per ciclare su tutti gli elementi della collection si potrebbe usare un classico ciclo For … next del tipo:

```
For i = 1 To (Collections.Count )
        Collections.Item(i).Show 'Si mostra l'iesimo oggetto della collection
Next i
```

È però possibile, e spesso più conveniente, usare un ciclo del tipo **For Each In … Next**, la cui sintassi è la seguente:

```
For Each Obj In Collections
        [… instructions … ]
Next Obj
```

Dove Obj è una variabile di tipo oggetto che viene assegnata, di volta in volta, all'oggetto situato nella successiva posizione della collection. Alla fine del ciclo la variabile Obj viene automaticametne cancellata (come se venisse scritto Set Obj = Nothing).
Notiamo infine come tale ciclo possa essere usato non solo su collection, ma anche nel caso di vettori di tipo variant.

```
Public Sub Emp_Coll(Optional N_Emp As Integer = 5)
Dim Emp As Cls_Employee
Dim Employees As Collection
Dim i As Integer
  Set Employees = New Collection
  For i = 1 To N_Emp
    Set Emp = New Cls_Employee
    Emp.Create "A" & i, Application.WorksheetFunction.RandBetween(5, 10), _
                                                Application.WorksheetFunction.RandBetween(20, 40)
    Employees.Add Emp, Emp.EmpID 'Oggetto aggiunto e chiave di ricerca
    Set Emp = Nothing
  Next i
  For Each Emp In Employees
    Emp.Show
  Next Emp
  If Emp Is Nothing Then Debug.Print "Emp è stato cancellato" ' Is Nothing  in questo caso è sempre vera
```

```
    Debug.Print "Accesso posizionale"
    Employees(N_Emp).Show 'Accesso posizionale all'ultimo impiegato
    Debug.Print "Accesso per chiave"
    Employees("A" & CStr(i - 1)).Show 'Accesso per chiave all'ultimo impiegato
    Set Employees = Nothing
End Sub
```

Si noti, infine, come l'assegnazione casuale dell'ID e della paga oraria avvenga mediante l'istruzione Application.WorksheetFunction.RandBetween(). In particolare, Application.WorksheetFunction permette di accede a tutte le funzioni predefinite di Excel; RandBetween corrisponde (in inglese) alla funzione Casuale.Tra.

7.4. Routine di Proprietà

Nel paragrafo 7.2 abbiamo dato per implicito che, per definire una proprietà sia sufficiente creare una variabile pubblica all'inizio del modulo di classe. In effetti non è sempre così, infatti esistono due diversi tipi di proprietà: variabili di livello classe pubbliche e routine di proprietà.

Come visto in precedenza, le variabili di livello classe devono essere definite, come variabili pubbliche, prima di qualsiasi subroutine, funzione o proprietà. In termini pratici, ciò significa che tutte le variabili di livello classe dovrebbero trovarsi all'inizio del modulo di classe. Essendo pubbliche tali variabili saranno direttamente accessibili anche all'esterno della definizione della classe con la solita sintassi basata sull'operatore punto (es. Oggetto.Proprietà). Anche se per i cultori della programmazione ad oggetto, l'utilizzo di variabili pubbliche a livello di classe non è consigliabile[7], in termini pratici, quando possibile conviene sempre creare le proprietà mediante l'utilizzo di variabili pubbliche a livello di classe.

In alcuni casi però è conveniente (o come vedremo tra poco addirittura necessario), definire delle proprietà mediante "routine di proprietà". Quest'approccio consente infatti di limitare l'accesso, ad esempio consentendo un accesso in sola scrittura, alle variabili di classe e/o di effettuare verifiche sui valori assegnati alle variabili di classe. Operativamente, per creare proprietà usando "routine di proprietà" è necessario:

- Definire una variabile di classe di tipo privato (quindi non accessibile all'esterno della classe)
- Usare una particolare procedura di lettura, definita **Propety Get()**, per accedere in lettura al valore contenuto nella variabile di classe privata;
- Usare una particolare procedura di scrittura, definita **Propety Let()**, per accedere in scrittura al valore contenuto nella variabile di classe privata;

Tali proprietà si definiscono esattamente come qualsiasi procedura, semplicemente sostituendo alla keyword Sub la Keyword Property Get (o Set).

È importante notare che, una volta creato l'oggetto, il nome che è stato assegnato a tali procedure corrisponderà al nome della proprietà utilizzabile per interagire con l'oggetto creato. Per questo motivo è buona norma dare lo stesso nome alle proprietà che leggono e che scrivono il valore di una stessa variabile.

[7] *In termini tecnici non garantisce l'incapsulamento, una degli elementi tipici di una classe insieme ad ereditarietà e polimorfismo (argomenti non trattati in questa dispensa e, comunque non totalmente implementati da VBA).*

Esiste anche una terza procedura di assegnamento chiamata **Property Set()**; l'unica differenza tra un'istruzione Property Set e un'istruzione Property Let è che Property Set si utilizza quando si lavora con gli oggetti, mentre normalmente si assegnerebbero i valori usando l'istruzione Let.
Riprendiamo l'esempio precedente (classe impiegato) e vediamo subito come dotarlo di routine proprietà. Supponiamo di voler usare una proprietà (ad esempio chiamata WeeklyHrs) per dare all'utilizzatore della classe la possibilità di inserire le ore totali settimanali lavorate da un impiegato. Tali ore saranno in parte ore di lavoro standard e ore di lavoro straordinario; per evitare che l'utilizzatore della classe debba anche effettuare tale suddivisione, deleghiamo alla classe tale compito, oltre a quello di controllare la validità del valore cumulato di ore ricevuto in input.
A tal fine introduciamo:

- due costanti che definiscono il numero massimo di ore totali in una settimana e il numero massimo di ore di straordinario in una settimana;
- due variabili di classe private che conterranno, rispettivamente, il numero di ore settimanali standard e straordinario;
- una proprietà Let per assegnare il numero totale di ore settimanali. Tale procedura verificherà l'input e lo suddividerà in ore straordinarie e ore lavorative standard e assegnerà tali valori alle due variabili private prima definite (si noti che non è necessario includere una variabile, dato che tale valore è semplicemente dato dalla somma delle altre due).
- tre proprietà Get due per leggere il valore delle ore standard e di quelle di lavoro ordinario (si noti che, così facendo l'accesso a tali variabili è solo in lettura) e una per leggere il valore delle ore totali (si noti che "leggere" è improprio dato che in effetti, anche se l'utilizzatore della classe non potrà saperlo, non esiste una variabile riassuntiva "ore settimanali totali"; tale valore si ricava, infatti, dalla somma delle ore standard e di quelle straordinarie, assegnate alle relative variabili private di classe).

Il codice complessivo è il seguente

```
Public EmpName As String
Public EmpID As String
Public EmpRate As Double
Private NormalH As Double
Private OverH As Double
Const Standard_Week_Hours = 40
Const Max_Week_Hours = 60
Property Let WeeklyHrs(Hrs As Double)
  If Hrs < 0 Then Hrs = 0
  Hrs = WorksheetFunction.Min(Max_Week_Hours, Hrs)
  NormalH = WorksheetFunction.Min(Standard_Week_Hours, Hrs)
  OverH = WorksheetFunction.Max(0, Hrs - NormalH)
End Property
Property Get WeeklyHrs() As Double
  WeeklyHrs = NormalH + OverH
End Property
Property Get NormalHrs() As Double
  NormalHrs = NormalH
End Property
```

```
Property Get OverHrs() As Double
   OverHrs = OverH
End Property
Public Function EmpWeeklyPay() As Double
   EmpWeeklyPay = (NormalH * EmpRate) + (OverH * EmpRate * 1.5)
End Function
```

Supponendo di aver creato un oggetto Emp di tipo impiegato, scrivendo ad esempio Emp.OverHrs si potrà accedere al numero di ore straordinario lavorate dall'impiegato.

Concludiamo questo paragrafo enunciando un caso in cui, in VBA, l'uso di routine di proprietà è assolutamente necessario. Infatti, le proprietà semplici non possono essere usate per restituire una matrice, una stringa a lunghezza fissa, una costante o una struttura complessa creata con un'istruzione Type. Se occorre una proprietà che restituisca una di queste cose, è necessario creare un'adeguata variabile di classe privata e una routine Property che opera su tale variabile privata.

Sempre in riferimento all'uso di proprietà con tipi definiti dall'utente, potrebbero verificarsi dei problemi qualora si cercasse di assegnare, in un'unica istruzione, un valore a uno degli elementi della struttura.
Supponiamo che la classe contenga le seguenti istruzioni:

```
Public Type T_Coordinate
        Latitudine As Single
        Longitudine As Single
End Type
Private Map_Coordinate As T_Coordinate
Public Property Get Coordinate As T_Coordinate
        Coordinate = Map_Coordinate
End Property
Public Property Let Coordinate (Tcrd As T_Coordinate)
        Map_Coordinates = Tcrd
End Property
```

Assumiamo ora, che sia stato instanziato un oggetto col nome Mappa, è allora possibile leggere la latitudine delle coordinate inserite in questo modo:

```
Lat = Mappa.Coordinate.Latitudine
```

In questo modo si usa la proprietà Coordinate per accedere alla variabile Map_Coordinate e l'operatore punto (relativo al tipo definito dall'utente), per leggere il valore della latitudine.
Poiché questa funziona, potremmo essere tentati di usare anche le seguenti istruzioni:

```
Mappa.Coordinate.Latitudine = 47.63
Mappa.Coordinate.Longitudine = 122.13
```

Tuttavia, nonostante non venga segnalato nessun errore, dopo aver eseguito entrambi i comandi, longitudine sarà pari a 112.13, ma latitudine sarà pari a 0. Sebbene questo possa sembrare un bug di Visual Basic, in realtà non lo è. Quando si fa riferimento all'elemento Latitudine nella prima istruzione, Visual Basic crea una variabile temporanea T_Coordinate e imposta il valore Latitudine a 47.63. Poiché la variabile temporanea è riempita con gli zeri quando viene allocata e non c'è un valore assegnato

esplicitamente a Longitudine, questo contiene un valore di zero. Così, quando viene chiamata la routine Let Coordinate, con la variabile temporanea creata da Visual Basic, l'elemento Latitudine sarà impostato a 47.63 e l'elemento Longitudine sarà impostato a zero. La stessa situazione si verifica quando esegui la seconda istruzione. Poiché non è stato assegnato un valore a Latitudine nella variabile temporanea, il valore precedente di 47.63 sarà sovrascritto da zero, e questo annulla la modifica fatta nella prima istruzione.
Ci sono un paio di modi per evitare questo problema. Il primo modo, e probabilmente il migliore, è creare una classe anziché usare un'istruzione Type. Tuttavia, se si volesse proprio utilizzare un'istruzione Type, sarebbe necessario creare "manualmente" una variabile temporanea a cui assegnare esplicitamente i valori di latitudine e di longitudine, come mostrato di seguito:

```
Dim TempVar As T_Coordinate
TempVar.Latitudine = 47.63
TempVar.Longitudine = 122.13
Mappa.Coordinate = TempVar
```

7.5. L'operatore With

Quando si lavora con gli oggetti può essere utile sfruttare l'operatore **With** che permette di evitare di riscrivere tutte le volte il nome dell'oggetto quando vogliamo modificarne le proprietà. Supponiamo, ad esempio di voler settare le proprietà di un impiegato, come visto, per farlo possiamo procedere come di seguito indicato:

```
Set Emp = New Cls_Employee
Emp.Empname = "Pippo"
Emp.EmpID = "ID1"
Emp.EmpRate = 20.5
Emp.WeekHrs = 38
```

Usando With, possiamo risparmiare un po' di tempo (non molto per la verità), come mostrato di seguito:

```
Set Emp = New Cls_Employee
  With Emp
     .Empname = "Pippo"
     .EmpID = "ID1"
     .EmpRate = 20.5
     .WeekHrs = 38
  End With
```

7.6. Risolvere i riferimenti, l'operatore Me

A volte capita che una variabile locale e una variabile a livello di classe abbiano lo stesso nome. Questo può accadere quando si vuole dare a un parametro in un metodo lo stesso
nome di una proprietà. Per differenziare una variabile di livello classe da una variabile o un parametro locale, si può aggiungere il prefisso "Me." Alla variabile di livello classe, come nel seguente esempio:

```
  If Me.Nome <> Nome Then …
```

In questa istruzione, la variabile Me.Nome si riferisce a una variabile a livello di classe, mentre la

variabile Nome non qualificata si riferisce a una variabile o a un parametro locale.
La parola chiave Me può essere usata anche per qualificare un qualsiasi elemento pubblico o privato di una classe contenuto nel codice all'interno della classe, tra cui variabili di livello classe, subroutine, funzioni e routine di proprietà.

7.7. Memoization tramite l'uso di una classe – Class Initialize e Class Terminate

Nel paragrafo 6.1 abbiamo visto come utilizzare una variabile globale per implementare una funzione che calcola la serie di Fibonacci mediante ricorsione e memoization. Usando una classe possiamo ottenere il tutto in maniera più robusta e compatta.
È infatti sufficiente usare:

- Una variabile privata di tipo array - Fib_series() - a livello di classe in cui memorizzare i valori della sequenza precedentemente calcolati;
- Un metodo pubblico – F_Compute(N) - che calcola l'ennesimo numero della sequenza e che, se necessario, ridimensiona le dimensioni di Fib_series();
- Un metodo privato – Cfib(M) – che implementa ricorsivamente il calcolo della sequenza e che si preoccupa di registrare i nuovi valori trovati all'interno di Fib_series.
- Un metodo privato che si preoccupa di dimensionare Fib_Series per la prima volta e di assegnargli i primi numeri della sequenza di Fibonacci.

Il codice completo (scritto in un modulo di Class Cls_Fib) è riportato di seguito:

```
Private Fib_Series() As Single 'La variabile a livello di classe

Private Sub Class_Initialize() 'Routine che viene eseguita automaticamente alla creazione dell'oggetto
   ReDim Fib_Series(1 To 6)
   Fib_Series(1) = 1
   Fib_Series(2) = 1
   Fib_Series(3) = 2
   Fib_Series(4) = 3
   Fib_Series(5) = 5
   Fib_Series(6) = 8
   ' ...
End Sub

Public Function F_Compute(N As Integer) As Single
   If N > UBound(Fib_Series) Then ReDim Preserve Fib_Series(1 To N)
   F_Compute = CFib(N)
End Function

Private Function CFib(N As Integer) As Single
   If Fib_Series(N) <> 0 Then
      CFib = Fib_Series(N)
      Exit Function
   End If
   CFib = CFib(N - 1) + CFib(N - 2)
   Fib_Series(N) = CFib
End Function
```

```
Public Sub Show_Sequence()
   For i = 1 To UBound(Fib_Series)
        Debug.Print Fib_Series(i)
   Next i
End Sub
```

Si noti che il vettore Fib_Series è inizializzato tramite il metodo privato **Class_Initialize()**; si tratta in effetti di un "evento" (ossia una procedura che si attiva automaticamente al verificarsi di una certa situazione), definito da VBA che si attiva automaticamente non appena un oggetto viene creato.
In pratica nel momento in cui scriveremo:

```
Set F = New Cls_fib
```

verrà eseguito tutto il codice contenuto nel metodo Class_Initialize

Questo evento è utile per inizializzare le variabili di livello classe ed eseguire tutte le istruzioni Set New necessarie a creare eventuali oggetti contenuti nella classe.

```
Set VariabileOggetto = New Class
```

Esiste un altro evento di questo genere, chiamato **Class_Terminate** che, analogamente al primo, contiene un codice che verrà avviato appena prima che un oggetto venga distrutto.
Questo evento è un luogo ideale per distruggere oggetti che sono locali alla classe, impostandoli a Nothing con un codice simile a questo:

```
Set VariabileOggetto = Nothing
```

Notiamo infine che gli eventi Class_Initialize e Class_Terminate sono attivati solo quando l'oggetto vero e proprio è creato o distrutto. La semplice impostazione di una variabile oggetto a un'altra non avvierà l'evento Class_Initialize. Analogamente, se due o più variabili puntano allo stesso oggetto, la semplice impostazione di una variabile oggetto a Nothing non avvierà l'evento Class_Terminate.
Concludiamo questo paragrafo, con un esempio di codice che utilizza la classe Cls_Fib:

```
Public Sub Prova_Fib()
Dim Fibonacci As Cls_fib
Dim F As Single
   Set Fibonacci = New Cls_fib 'L'evento Class_Activate viene eseguito
   Debug.Print "Starting Sequence"
   Fibonacci.Show_Sequence 'Viene scritta la sequenza di base 1 1 2 3 5 8 13
   F = Fibonacci.F_Compute(35)
   Debug.Print "Ending Sequence"
   Fibonacci.Show_Sequence 'Vengono scritti i primo 35 numeri della sequenza
End Sub
```