Classi, metodi e proprietà in C#

Il concetto di **classe** è alla base di ogni linguaggio di programmazione orientato agli oggetti ed ha la potenza di definire le caratteristiche di un insieme di oggetti che hanno proprietà e compiono azioni uguali. Rappresenta tutti gli oggetti che appartengono ad una certa classe, appunto.
Per fare un'esempio si pensi alle automobili: ogni automobile è diversa da un'altra ma tutte hanno quattro ruote un propulsore e compiono l'azione di avanzare o frenare. Questi elementi comuni le fanno rientrare in un unico concetto. Un'unica classe.

Più praticamente possiamo vedere *una classe è una collezione di variabili, metodi e proprietà*. Anche in questo caso, **C#** riprende la sintassi di Java dove per definire una classe si usa il **costrutto class.**

```
public class Persona
{
  // CAMPI o VARIABILI
  string mNome;
  string mCognome;
  string mAltezza;
  //Altre variabili...

  // COSTRUTTORE
  public Persona(string Nome, string Cognome)
  {
    //Imposta le proprietà iniziali della classe.
    mNome = Nome;
    mCognome = Cognome;
  }

  // METODI
  public void StampaMessaggio()
  { }
  private void SetDataNascita()
  { }

  // PROPRIETA'
  public string Altezza
  {
    get { return mAltezza; }
    set { mAltezza = value; }
  }
}
```

A volte una classe è identificata anche come "**tipo di dato**", infatti porta con sé sia la rappresentazione dei dati, sia le operazioni

sugli stessi, in realtà il concetto di classe è più vasto.

Modificatori di Variabili, Classi, Metodi, Proprietà

Nella dichiarazione di *variabili*, *classi*, *metodi* e *proprietà* si possono utilizzare i **Modificatori**, oltre al **Tipo**.
I Modificatori indicano ad esempio come l'oggetto è ***Visibile*** dal codice o l'***Accesso*** che si può avere ad esso.

[PUBLIC|PROTECTED|PRIVATE][STATIC][FINAL] TIPO Nome [= Valore]

PUBLIC Con **public**, una routine diventa accessibile da tutte le istanze della classe.
PRIVATE private, invece, impedisce che la routine sia visibile al di fuori della classe in cui è definita. Se non si specifica il modificatore di accesso, **private è il valore predefinito.**
PROTECTED Sarà visibile nell'ambito in cui definiamo le classi che ereditano da Persona ma non potrà essere utilizzata nel codice in cui istanziamo oggetti di questa classe. In pratica, **protected è simile a private**, con la differenza che il metodo è visibile anche alle classi che ereditano da quella principale.
INTERNAL ciò che è dichiarato come **internal** è visibile solo all'interno dell'assembly dell'applicazione.

STATIC Una ***variabile*** dichiarata **static** è visibile da tutte le istanze di quella classe. Un ***metodo*** **static** è utilizzabile senza instanziare la classe che lo contiene. Non agisce su una istanza specifica come invece fanno i metodi di istanza (non statici).

FINAL Una ***variabile*** **final** può essere inizializzata nel programma una sola volta. E un oggetto dichiarato **static + final** è una ***costante***.

Oggetti e costruttori

Possiamo però considerare le istanze di una classe, ovvero gli **oggetti** che realizzano il concetto di classe, come **variabili** definite da un certo tipo di dato.

```
//Crea un oggetto p che è istanza della classe Persona
Persona p = new Persona("Marco", "Minerva");
```

Per ogni **classe istanziata**, viene **creato un'oggetto** ed è invocato un metodo detto **costruttore** della classe.
Il costruttore, così come avviene in Java e in C++, ha lo stesso nome della classe e non restituisce alcun valore (nel nostro esempio il metodo public Persona).
Il costruttore è utilizzato per **impostare le proprietà** che la classe deve avere al momento della sua creazione: nel nostro esempio, quando si crea una istanza della classe Persona, le sue variabili `mNome` e `mCognome` assumono i valori specificati nel relativo argomento del costruttore.

Metodi, cosa sono e come si dichiarano in C#

I metodi rappresentano le modalità con cui si può accedere agli oggetti per stimolarli a mettere in atto azioni o comportamenti. Più pragmaticamente si tratta di routines (funzioni o procedure), vediamo come dichiararle.

I metodi di una classe possono essere definiti come *public*, *private*, *protected* oppure *internal* attraverso i **modificatori di accesso**.

Ad esempio, utilizzando ancora la nostra classe `Persona`:

```
// Dichiariamo ed istanziamo un oggetto della classe Parsona
Persona p = new Persona("Marco", "Minerva");

p.StampaMessaggio();  // Corretto, StampaMessaggio è public
p.SetDataNascita();  // Errato, SetDataNascita è private, quindi non è accessibile
```

Dopo il modificatore di accesso del metodo, nella dichiarazione è necessario specificare il **tipo di dato restituito**.

È possibile indicare un qualsiasi tipo di dato: int, string, byte, etc. Si può specificare anche il nome di una classe, poiché, ricordiamo, una classe è un tipo.

Per **restituire il valore**, è necessario utilizzare la parola chiave **return**, seguita da una qualsiasi espressione che abbia lo stesso tipo del valore di ritorno del metodo. Tale istruzione causa anche l'uscita dal metodo. Ad esempio:

```
public int Quadrato(int N)
{
  return N * N;
  //Eventuale codice scritto qui non verrà eseguito.
}
```

In C# un metodo che restituisce un valore deve obbligatoriamente contenere la parola chiave ***return***, altrimenti, in fase di compilazione, si otterrà un errore.
Le funzioni di Visual Basic, invece, possono non presentare il return: in tal caso il valore restituito è quello di default per il particolare tipo di dati.

I metodi che non restituiscono un valore hanno tipo **void**. Si può comunque usare la parola return in un metodo void, che in questo caso ha il solo scopo di uscire dalla procedura: in questo caso, quindi return non deve essere seguito da alcuna espressione.

Parametri

I parametri consentono di passare ai metodi variabili. I parametri di un metodo ottengono i valori effettivi dagli *argomenti* specificati quando viene richiamato il metodo. Esistono quattro tipi di parametri: ***parametri di valore***, ***parametri di riferimento***, i ***parametri di output*** e ***matrici di parametri***.
Un *parametro di valore* viene usato per passare argomenti di input. Corrisponde a una variabile locale che ottiene il valore iniziale dall'argomento passato per il parametro. Le modifiche a un parametro di valore non modifica l'argomento esterno passato per il parametro.
Un *parametro di riferimento* viene usato per passare argomenti per riferimento (puntatore). L'argomento passato per un parametro di riferimento deve essere una variabile con un valore definito e,

durante l'esecuzione del metodo, il parametro di riferimento rappresenta lo stesso percorso di archiviazione della variabile di argomento. Un parametro di riferimento viene dichiarato con il modificatore `ref`.
Un *parametro di output* viene usato per passare argomenti per riferimento. È simile a un parametro di riferimento, ad eccezione del fatto che **non è necessario assegnare un valore** per l'argomento specificato dal chiamante. Un parametro di output viene dichiarato con il modificatore `out`.
Una *matrice di parametri* consente di passare un numero variabile di argomenti a un metodo. Una matrice di parametri viene dichiarata con il modificatore `params`.

Le proprietà della classe in C#

Una classe può contenere anche un altro tipo di metodi, le cosiddette **proprietà**. Si tratta di particolari routine che permettano di leggere e/o impostare i valori di determinate proprietà di una classe. Aggiungiamo, per esempio, la seguente proprietà alla classe Persona:

```
public string Altezza
{
  get { return mAltezza; }
  set { mAltezza = value; }
}
```

Essa permette di avere accesso in lettura/scrittura alla variabile `mNome`, che altrimenti non sarebbe visibile all'esterno della classe. Il blocco get deve restituire la variabile associata alla proprietà, quindi, come detto prima, deve contenere l'istruzione return.
Il codice del blocco set, invece, viene eseguito quando si modifica il valore della proprietà; la parola chiave value contiene il nuovo valore da assegnare. L'utilizzo delle proprietà è intuitivo:

```
Persona p = new Persona("Marco", "Minerva");

Console.WriteLine(p. Altezza);  //Stampa "190 cm" (richiama il blocco get)
p.Altezza = "190 cm";  //Imposta l'Altezza a "190 cm" (richiama il blocco set)
```

È lecito domandarsi **perché utilizzare le proprietà**, quando si potrebbe più semplicemente rendere pubblica la variabile `mAltezza`.
Conviene avere una classe con membri privati accessibili solo tramite proprietà quando si devono effettuare dei controlli sui valori assegnati ai membri stessi.
Ad esempio, supponiamo di voler impedire che l'utente inserisca una Altezza vuota: se impostiamo la variabile `mAltezza` come pubblica questo controllo non può essere effettuato.
Infatti si ha libero accesso alle variabili che possono essere impostate su qualunque stringa valida (compresa, dunque, la stringa vuota).

Per risolvere il problema, è sufficiente definire i membri come privati e **modificare la proprietà** `Altezza` nel modo seguente:

```
public string Altezza
{
  get { return mAltezza; }
  set
  {
    if (value == string.Empty)
      mAltezza = "(Valore non inserito)";
    else
      mAltezza = value;
  }
}
```

string.Empty è un campo a sola lettura che rappresenta la stringa vuota. Se Altezza viene impostata sulla stringa vuota, questo codice la definisce come `Valore non inserito`.
E' buona norma di coerenza effettuare sempre un controllo sugli argomenti, piuttosto che dare libero accesso alle variabili stesse.

Le proprietà possono anche essere di sola lettura, se dispongono del solo blocco **get**, oppure di sola scrittura, se hanno solo il codice associato al **set**.