di Raffaello De Masi

Appunti di programmazione
Le basi della programmazione O.O.

Dopo le fanfare iniziali e i rulli di tamburo, come in tutte le cose della vita, viene il lavoro quotidiano, fatto di mattoni messi l'uno sull'altro per raggiungere certi risultati. E così avviene anche per la programmazione strutturata. Da quando abbiamo aperto questa rubrica, due puntate fa, non abbiamo fatto altro che parlare della facilità e della utilità della programmazione strutturata; poco però abbiamo fatto per mostrare come a tale tecnica si possa giungere

Per la verità (e questo lo diciamo rimangiandoci un poco le belle affermazioni dell'inizio), se programmare in O.O. è facile, è meno semplice prepararsi i tool per poter lavorare con questa tecnica. In altri termini, per giungere a un lavoro facile e spedito in termini di programmazione finale, occorre spendere un poco del proprio tempo per mettere a punto una serie di attrezzi, utility, subroutine (o come meglio preferite chiamarle) che poi possano servire alla bisogna per quello che meglio ci fa comodo. Ma non era poi questa la filosofia di cui abbiamo fin dall'inizio parlato?

Per evitare che, però, il nostro bel programma O.O. divenga una raccolta più o meno intricata di subroutine e di chiamate a procedure o a funzioni predefinite, occorre introdurre una serie di concetti di base, relativi a costrutti formali e a elementi e blocchi di programmazione che rappresentano, come dicevamo dianzi, i mattoni fondamentali della costruzione del programma stesso. Più che di vere e proprie istruzioni (ricordiamo che fin dall'inizio abbiamo precisato, checché ne dicano gli «esperti», che la programmazione O.O. non è proprietà o prerogativa di un particolare linguaggio) parleremo quindi di filosofia dell'ambiente, di concetti di base che poi, volta per volta, ognuno potrà applicare al suo idioma preferito. Ciononostante, proprio per poter semplificare, adotteremo quando sarà necessario esempi che cercheremo di non affidare sempre allo stesso linguaggio; Basic, C, Pascal, Fortran, gli avanzati Logo dell'ultima ora, un buon Prolog, sono tutti buoni candidati alla bisogna.

I principi e i concetti di base della programmazione O.O.

Ripetiamo, fino ad essere noiosi; parliamo di concetti di base, che probabilmente, proprio per la loro natura «concettuale» (mi si perdoni la ricursione) non hanno sovente alcun riferimento in un linguaggio formale, né possono essere riferiti a particolari approcci alla programmazione. Ma, proprio perché si tratta di concetti, sarà poi, come vedremo, possibile implementarli nei particolari linguaggi in maniera più o meno immediata e agevole.

I principi fondamentali su cui si articola la programmazione orientata all'oggetto (su Macintosh come su qualunque altra macchina) sono articolati sulla interazione e sulla funzionalità di quattro elementi fondamentali; l'oggetto, la classe, il metodo e la successione (detta anche ereditarietà). Si tratta di concetti collegati tra loro e del tutto interdipendenti, per cui di seguito di essi tenteremo una descrizione analitica; c'è da precisare, comunque, che trattandosi di tool concorrenti a formare un nuovo concetto di ambiente di programmazione, è opportuno cercare di comprendere il senso dell'uno senza prescindere da quello degli altri. In altri termini, oltre al significato e alla funzione intrinseca oggettiva, occorre tener conto delle relazioni intercorrenti tra le varie parti; la potenza e l'efficienza di programmazione raddoppia, per il solo uso combinato delle diverse tecniche, la potenza totale, e, cosa che non guasta, la facilità stessa di redazione di un programma.

L'oggetto

Si definisce oggetto la semplice combinazione di notazioni di programmazione convenzionale, attraverso l'uso di dati e di procedure. Sarebbe la stessa cosa di quanto, ad esempio, si vede nella definizione di procedure da Pascal, o di Long FN da Basic, ma qui c'è già una piccola differenza. Mentre nella programmazione convenzionale le due componenti sono rappresentate indipendentemente e richiedono, da parte del programmatore il maneggio e l'individuazione della tecnica della loro interazione, i sistemi di programmazione O.O. integrano le due parti fino a farne un blocco unico, l'oggetto, appunto.

Messaggio	Effetto del messaggio
Descrive	Crea a fornisce una descrizione dell'oggetto creato
Raggio	Calcola il raggio di un cerchio descritto attraverso il suo centro e un punto della circonferenza
Rett_inscritto	Calcola le caratteristiche del quadrato inscritto nel cerchio
Rett_circoscritto	Come sopra , ma per il quadrato circoscritto
Ellisse	Trasforma il cerchio in una ellisse dalle determinate caratteristiche e dimensioni
	.....

Figura a - Assetto di un ipotetico programma che definisce quattro oggetti destinati a maneggiare cerchi.

Questo comporta una tecnica un poco differente del maneggio della chiamata all'oggetto; mentre nella procedura classica si inviano a questa i dati da manipolare, all'oggetto si invia un «messaggio»; l'oggetto legge il messaggio, lo interpreta, e di conseguenza esegue una delle procedure in esso contenute, procedura che opera sui dati privati in suo possesso.

Facciamo un esempio? Semplice; immaginiamo di redigere, in un linguaggio immaginario l'oggetto:

```
Object: rettangolo_1
  Dati interni: X1Y1:Punto,
                X2Y2:Punto
  Messages: centro.
```

dove il messaggio [centro] rappresenta l'istruzione attraverso cui viene calcolato il baricentro dell'oggetto e restituisce la relativa informazione. Come si vede, per eseguire questo calcolo viene solo inviato alla procedura il messaggio [centro]; esso, da solo, si incarica di tutto.

Questo tipo di approccio nasconde, per la verità, i particolari su cui lavora la procedura stessa e i dati su cui il programmino lavora. Un esempio più chiaro della stessa procedura potrebbe essere:

```
Object: rettangolo_2
  Dati interni: X1,Y1:INTEGER,
                X2,Y2:INTEGER
                Punto_centrale_X:INTEGER
                Punto_centrale_Y:INTEGER
    Punto_centrale_X=(X1+X2)/2
    Punto_centrale_Y=(Y1+Y2)/2
  Messages: centro.
```

che esegue esattamente la stessa cosa ma che risulta più chiara nell'uso dei dati che lo stesso programma maneggia. Il primo utilizza due soli dati, appartenenti a una non meglio classificata categoria definita «Punti»; il secondo utilizza quattro variabili intere. Una vera differenza tra le due procedure, anzi per essere più chiari, tra i due «oggetti» non esiste. Il vero punto cruciale del problema è che, sebbene i due programmi sviluppino i loro calcoli e le loro operazioni interne in maniera del tutto diversa, i risultati (e questo non solo dal punto di vista numerico) non hanno alcuna differenza tra di loro.

In altre parole, e in questo utilizzando una metafora, non mia, ben diffusa nel campo della programmazione O.O., gli oggetti sono come unità modulari (di cui non interessano le caratteristiche di funzionamento interno), o, se si preferisce, dei piccoli computer, che accettano istruzioni e restituiscono messaggi come parte di un sistema più ampio e articolato.

Le classi

Tutto il principio di modularità, interdipendenza e possibilità di riuso della programmazione orientata all'oggetto andrebbe a farsi benedire se si dovesse descrivere la struttura, il tipo e le modalità di funzionamento di un messaggio per ogni oggetto separatamente. Gruppi di oggetti possono comportarsi allo stesso modo per lo stesso tipo di messaggi, e, per tale motivo, possono essere descritti una sola volta descrivendo la loro classe.

Una cosa del genere avviene, ad esempio, in Pascal, dove è possibile definire tipi di dati unici, rispondenti alle funzionalità e alle necessità imposte dal programmatore (proprio per restare nell'esempio precedente, avremmo potuto definire il tipo [Punto] come personale), e che possono essere utilizzati con diversi valori di variabile.

Non a caso, in Object Pascal, una classe è riferita a un «tipo» d'oggetto ed è dichiarata in maniera esattamente eguale al tipo RECORD, specifico di questo linguaggio. Gli oggetti infatti sono riferiti come «chiamate» alle rispettive classi, così come, proprio in Pascal, le variabili sono chiamate ai rispettivi tipi di dati.

A dimostrare l'estrema analogia delle strutture, una classe viene organizzata e specificata descrivendo i nomi e i tipi di variabile contenuti nella chiamata a uno o più oggetti in essa contenuti, come pure riferendosi alla lista dei messaggi e delle risposte (anche in termini di variabili) a cui le chiamate (o gli oggetti) danno origine. Questi dati vengono definiti «variabili di chiamata» e il set di messaggi, con una definizione non proprio calzante, «protocollo di classe».

Precisato ciò, se nel programma ci sono diverse occasioni di calcolo del centro di un rettangolo, potremo definire una volta per tutte la classe:

```
Class : centro_rettangolo
    Variabili di chiamata: X1Y1:Punto,
                           X2Y2:Punto
     Messages: centro(restituisci Punto).

e, nel corso del programma avere chiamate come

Object name : rettangolo_1  Class name: centro_rettangolo
    Dati interni: X1Y1:(100,100),
                  X2Y2:(200,200)
...

Object name : rettangolo_2  Class name: centro_rettangolo
    Dati interni: X1Y1:(250,100),
                  X2Y2:(450,330)
...

Object name : rettangolo_3  Class name: centro_rettangolo
    Dati interni: X1Y1:(125,200),
                  X2Y2:(900,1500)
```

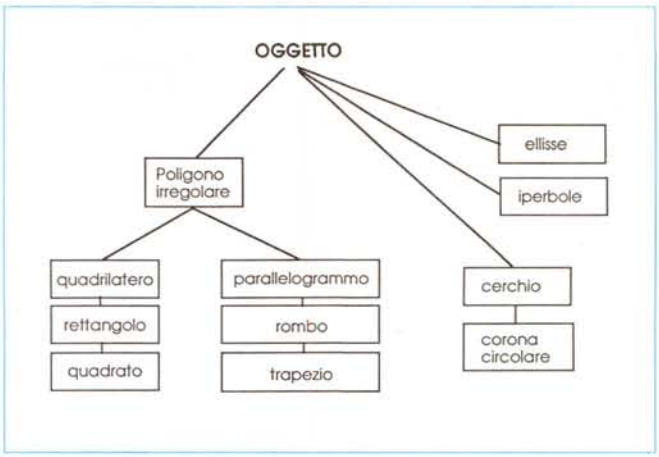

Figura b - Un esempio di struttura gerarchica ed ereditarietà diretta; tutti i linguaggi O.O. dispongono di questo tipo di struttura ad albero.

L'idea di base, certo non lontana da quella che anima le funzioni o le procedure di altri linguaggi (ma si ricordi che l'O.O. non è un linguaggio) è che la classe descrive, una volta per tutte, la struttura delle chiamate, e l'oggetto contiene i dati delle variabili. È un poco come considerare una classe come una specie di valigia a scomparti, in cui sono contenuti oggetti e nella quale se ne possono aggiungere altri.

Ovviamente, gli esempi presentati precedentemente sono assolutamente banali (e, per assurdo, chi legge troverebbe del tutto inutile e forse sovrabbondante il loro stesso uso). Tanto per fare un esempio più complesso immaginiamo di dover costruire un programma di grafica che disegna cerchi: un assetto strutturale attraverso classi dovrebbe tener conto e definire almeno cinque oggetti che per comodità indichiamo in figura a.

L'aver costruito tante routine per eseguire centrature, rotazioni, calcolo dei parametri del cerchio, trasformazioni e così via non deve coinvolgere l'utente finale su come, effettivamente, queste routine devono funzionare. Addirittura, come abbiamo visto nella precedente occasione, due routine possono eseguire la stessa funzione giungendo a risultati analoghi attraverso procedimenti diversi (o addirittura manipolando dati diversi).

Vi sembra ancora troppo banale. Bene; cercate di applicare tutto quello che abbiamo detto finora alla gestione delle icone, delle finestre o dei menu.

Tutti sappiamo che questo avviene attraverso chiamate al toolbox; col nostro bravo programmino chiamiamo l'istruzione [window], sbattiamo giù una serie di opzioni e di parametri aggiuntivi, e, se proprio non siamo dei muli, ecco la finestra aprirsi, spostarsi, ridimensionarsi o zoomare secondo i nostri bisogni.

Chi tiene conto, conserva, aggiorna i dati relativi alle dimensioni, alle coordinate relative e assolute, allo stato, al refreshing dello schermo, chi, nel nostro meraviglioso Word, gestisce la tabulazione, il colore dello schermo, il tipo e la varietà, in grandezza dei caratteri, aggiorna il righello, esegue e gestisce le tabelle? Pensate solo un momento a un programma, scritto in linguaggio normale, che crei un file che «ricorda» la formattazione, rigo per rigo, di un documento redatto con un word processor. Creerebbe un mostro di file in cui almeno due terzi del contenuto sarebbero rappresentati da caratteri di controllo, dati relativi al setup e così via. Invece un File creato da Word è poco più lungo del numero di caratteri ASCII in esso contenuti. E allora?

Provate a fare un esperimento. Se possedete Word 5, create un documento con questa versione e leggetelo con la versione precedente. Vedrete che, all'inizio e alla fine, saranno presenti alcuni righi (non più di tre o quattro) contenenti una serie di caratteri strani; sono chiamate a oggetti, presenti nel programma di base. Bastano quelle poche chiamate, lette al lancio del file, a formattare l'intero documento.

Un altro esempio? Subito! Ricordate JustText, un word processor che ebbe una certa fortuna qualche anno fa? Dedicato a chi lavorava solo con LaserWriter, non eseguiva alcuna formattazione a video; le chiamate ai caratteri erano eseguite attraverso l'inserimento di un codice «embebbed» direttamente nel testo. Secondo voi che faceva quel codice? Chiamava un oggetto (o addirittura, per formattazioni complesse, una classe) in fase di stesura di un file intermedio (detto Print File) che poi veniva inviato alla stampante. Semplice, no?

Un altro, tanto per finire? Vi siete mai chiesti come è costituito un file grafico, ad esempio redatto con MacDraw o con PixelPaint? Come si fa a conservare un rettangolo in un file? Vi posso aiutare? Ricordate che programmi come Draw, Claris Cad, Cricket Draw, sono programmi di grafica che maneggiano oggetti; non è una coincidenza di denominazioni.

Utilizzando una precisa metafora di Kurt J. Schmuker, con questo tipo di programmazione il punto di vista per lo sviluppo delle applicazioni passa dall'interno all'esterno. La fatica principale del programmatore sarà quella di interconnettere vari oggetti (uno che sa come far funzionare le window, un altro i menu, un altro i messaggi di errore, un altro come gestire le celle di spreadsheet, uno che conosce i codici di formattazione di un w.p. o di un database, uno che è specializzato, magari attraverso una serie di librerie, nella gestione della grafica, e così via), senza per questo sapere cosa accade nella gestione interna dei comandi. Così se bisogna costruire un programma che utilizza menu, anche gerarchici, frecce di scroll e si deve altresì accedere ad un network AppleTalk, è possibile assemblare e testare parti diverse delle librerie e incollarle insieme secondo la necessità. E queste librerie dove trovarle? Una sola parola! A questo serve MacApp e questo MacApp offre: una libreria di «classi» (appunto) che fornisce tutti i mezzi di cui si ha bisogno per sviluppare una applicazione Mac.

I metodi

Abbiamo definito poco fa la funzione delle classi di oggetti; ma, anche se MacAPP ci dà il prodotto bell'e fatto, pronto da usare, un minimo di soddisfazione agli oscuri programmatori che hanno redatto queste bellissime applicazioni è dovuto; e poi, non può essere (anzi quasi sempre è) che a noi occorra un oggetto fatto proprio in un modo particolare (che so, ad esempio, uno che costruisca trapezi tutti con un angolo alla base di 75°)? Occorre allora penetrare un poco più addentro al concetto di procedura, magari per rendersi conto di come essa manipola i dati.

Per definizione, le classi definiscono procedure che «sanno» come manipolare i dati specifici degli oggetti. Queste procedure sono chiamate metodi, e proprio i metodi maneggiano, gestiscono e ripartiscono i dati all'interno degli oggetti stessi. Per chiamare in causa uno di questi metodi, l'operatore invia un messaggio a una istanza di una classe. Un messaggio funziona come un selettore, una parola d'ordine che specifica quale metodo o oggetto va attivato; sarà lo stesso oggetto, per i motivi già diverse volte detti in precedenza, ad attivare le giuste procedure e i ricorsi appropriati per la soluzione del problema. Poiché gli oggetti, con il maturare dell'esperienza (e magari con il concatenamento degli

oggetti tra di loro) saranno sempre più sofisticati, l'azione svolta verso la macchina sarà sempre più simile all'invio di una serie di ordini più sofisticati e compatti.

Potremo così arrivare a programmazione estremamente sofisticata, con ordini del tipo «Traccia una serie di rettangoli concentrici con incrementi della base del 20% o la retta tangente al cerchio passante per questo punto» (che strana somiglianza con certi comandi di Claris Cad); questo tipo di programmazione viene in gergo, definita «chiamata per richiesta», e consiste in una tecnica di invocazione dei comandi che è fondamentale nella programmazione object-oriented. Con una libreria ben fornita e soprattutto ben articolata di tool, non sarà più necessario sapere cosa fare per giungere a un buon risultato, ma solo sapere quale chiamata invocare per giungere al risultato stesso.

L'ereditarietà

Il quarto e ultimo concetto di programmazione O.O. è l'ereditarietà; come un bambino eredita dal padre caratteristiche somatiche e comportamentali, così classi discendenti da altre possono, se necessario, ereditare variabili e metodi da classi progenitrici. Questo può semplificare in maniera cospicua la fatica di programmare, in quanto non è, in tal modo, necessario descrivere il messaggio specifico di ogni classe. Ogni classe diverrà quindi il compendio di quelle che l'hanno in certo qual modo generata, e sarà compito del programmatore solo definire i messaggi addizionali per la specifica classe, non presenti nei blocchi precedenti.

Questa ereditarietà può, a sua volta, essere divisa in due substrutture diverse; ereditarietà diretta, o come viene anche definita, a gerarchia semplice, ed ereditarietà multipla. Questa seconda sezione si presenta con caratteristiche piuttosto complesse e, inoltre, non è supportata da molti dei linguaggi descritti. La prima struttura è invece uno dei pilastri di base dell'object-orienting; essa entra in gioco quando una classe è descritta in termini di un'altra immediatamente precedente in termini di «parentela»; questa classe, «genitrice», è chiamata genericamente «superclasse» (o, in Object Pascal, antenato diretto). Questo porta ad una struttura ad albero più o meno complessa (ma sempre diretta) come quella descritta in figura b, con ogni classe identica, nella struttura di base, a quella precedente, tranne che per le caratteristiche addizionali che la individuano.

La nuova classe sarà, in altri termini, una più specializzata della precedente, e così (facciamo sempre riferimento alla figura b) una corona circolare apparterrà alla stessa categoria del cerchio, ma con caratteristiche più specializzate (non foss'altro per i due valori del cerchio di curvatura). Il nuovo oggetto (o la nuova classe) possiede quindi tutte le caratteristiche della precedente (come l'ellisse possiede quelle del cerchio), ma si differenzia da essa per certe sue caratteristiche particolari.

Come queste parti fondamentali interagiscono tra di loro? E come fare in modo da chiamarle in causa? Lo vedremo nella prossima puntata.

MC