di Raffaello De Masi

La programmazione Object Oriented

quarta parte

Dopo aver parlato così bene della programmazione object oriented viene voglia di pensare che finora abbiamo perso tempo! E forse, come concetto generale, non si è sbagliato. Invece è stato sbagliato finora l'alone che è stato voluto assegnare a questa scienza, che così si può chiamare, agganciandola al carrozzone di questo o quel linguaggio; come abbiamo detto più volte, OO non è sinonimo di Pascal, C, Forth o altro; è sinonimo di filosofia di pensiero e di tecnica di redazione dei programmi

Giusto per chiarire quanto abbiamo finora detto, preciseremo che, nel nostro dire, parleremo di Pascal, di C, di Forth e, perché no, di Basic. Ciononostante, sebbene questa trattazione sia stata da me organizzata senza far riferimento ad alcun linguaggio in particolare, ma solo mostrando la struttura di una singola applicazione o routine-procedura, per meglio comparare la maggiore o minore efficienza o fatica nell'implementazione tra questa e quella realizzazione disponibile nell'universo dei linguaggi disponibili per Macintosh, in modo da lasciare libero il programmatore di adottare le sue personali preferenze in fatto di linguaggi, occorrerà ogni tanto far cenno a questa o quella implementazione Macintosh.

Ma nella filosofia della completa trasparenza della programmazione OO nei confronti dei linguaggi, ci piace ricordare un esempio di tool didattico realizzato da K. Schmucker nel suo volume dedicato all'argomento; proprio per sfatare il luogo comune della indissolubilità del binomio programmazione OO-linguaggio dedicato, egli realizzò un programma, QuadWorld, che realizzava vari tipi di quadrilateri. QuadWorld, come programma dimostrativo, funzionava una meraviglia; era realizzato in tre linguaggi diversi, e ne esistevano parziali implementazioni in altri linguaggi, che permettevano, a chi volesse, di adottare il suo linguaggio preferito.

QuadWorld permetteva di disegnare e manipolare diversi tipi di quadrilatero; rettangoli, quadrati, rombi, parallelogrammi, e altri oggetti di vario genere. Esso era costruito in modo molto semplice, permettendo di creare un poligono della forma desiderata che poteva poi essere ridimensionato, spostato, deformato o ruotato. Il quadrilatero costruito poteva essere realizzato utilizzando due rappresentazioni diverse; una grafica (disegnante direttamente l'oggetto sullo schermo) e una che presentava il quadrilatero stesso sotto forma di descrizione di testo.

Il pregio di questo programma, che peraltro aveva solo valore didattico, era la sua completa dipendenza dall'interfaccia Macintosh. Esso utilizzava tutte le qualità della grafica ad alta risoluzione dello schermo, e i suoi complessi e raffinati menu. Le caratteristiche originali che lo contraddistinguevano erano così distinte:

- un'ampia area di scroll, sensibile alle dimensioni del disegno;
- la capacità di inserire il quadrilatero desiderato nella maniera più facile e agevole possibile, senza soverchie preoccupazioni circa i particolari tipi di quadrilatero utilizzati;
- la possibilità, selezionando un oggetto, di avere immediatamente a disposizione il riscontro relativo, sotto forma di «nome» e di descrizione testuale;
- la possibilità di cancellare uno o tutti i quadrilateri;
- la possibilità di ruotare, modificare e ridimensionare gli oggetti, con a disposizione, in ogni momento un «Undo».

Sembra, detto così, poca cosa, in confronto alla potenza dei pacchetti di grafica disponibili sul mercato; e così è; ma QuadWorld non è certo stato costruito per fare grafica, ma solo per insegnare a realizzare i «pezzi» generalmente utilizzati nella redazione di un programma OO.

Infatti, e lo vedremo nelle nostre discussioni, qualunque sia il linguaggio

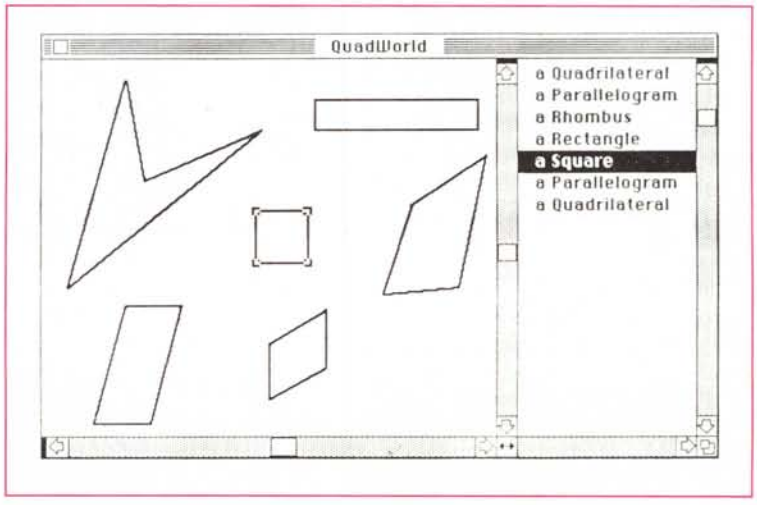

Figura A - La finestra principale di QuadWorld.

utilizzato per la versione preferita di QuadWorld, esso si basa su un costrutto piramidale di classi e metodi. Un esempio è la figura A che alleghiamo, dove il quadrato discende dal rettangolo (essendo una forma più specializzata di esso), il rombo discende dal parallelogrammo e così via. La cosa più interessante è che l'intera gerarchia è poi tutta confluente in una classe generale, quella dei «Quadrilateri».

Ovviamente questo tipo di struttura può essere implementata più o meno facilmente a seconda del linguaggio utilizzato. Così, a seconda delle varie implementazioni, possono essere presenti piccole variazioni nella struttura del linguaggio.

QuadWorld è costruito partendo da una applicazione MacApp, è compatibile con tutti i linguaggi supportati da questo ambiente, e può essere facilmente adattato a Lisa Toolkit e Smalltalk MVC, per i linguaggi che non lo sono.

Il linguaggio Object Pascal

Cominciamo a parlare del primo linguaggio comparso sul mercato e dichiaratamente dedicato alla programmazione OO, il ben noto e ormai famoso Object Pascal. Per la verità si tratta del secondo linguaggio, ma la prima realizzazione di esso, il Clascal, era piuttosto specializzato, essendo dedicato al Lisa Office System. Grazie anche alle esperienze eseguite sullo sfortunato predecessore del Mac Object Pascal si presentò subito come un linguaggio elegante, facile da realizzare, efficiente e rapido. Il risultato più evidente di tutto ciò fu che la maggior parte delle complesse routine presenti nel Clascal scomparvero in quanto assorbite da metacomandi complessi e facili da usare; l'utente finale fu così incoraggiato grandemente ad intraprendere la strada di questo ambiente dal fatto che non era più necessario imparare elenchi interminabili di nuovi comandi e procedure, né nuovi e più approfonditi concetti rispetto a quelli già noti. Non a caso Object Pascal fu scritto dal team di Clascal, che fece frutto delle esperienze già pregresse e usufruì della collaborazione del gran guru del Pascal Niklaus Wirth, che per l'occasione fu invitato a collaborare a Cupertino. A questo si aggiunse il supporto di alcuni utenti di Clascal, che collaborarono a disegnare in maniera efficiente il nuovo linguaggio.

Trasferire le tecniche di programmazione OO in un linguaggio all'uopo dedicato può essere realizzato in due modi; disegnando completamente un nuovo linguaggio, basato su questi concetti (detto comunemente, approccio «puro»), o utilizzando come base un linguaggio già esistente, inserendo in essi i nuovi concetti propri di tale ambiente (linguaggi ibridi). Appartiene alla prima categoria un solo linguaggio, SmallTalk, realizzato originalmente attraverso un impegno notevole e articolato; tutti gli altri linguaggi, compreso lo stesso Object Pascal, il C++ e il Neon, sono ibridi. In questo caso però, e in particolare nei linguaggi appena nominati, le operazioni di modfica furono eseguite in maniera così lieve e raffinata che il risultato fu l'avere a disposizione degli eccellenti linguaggi C, Forth e Pascal, con in più, a disposizione, tutte le nuove possibilità messe a disposizione dalle classi, dai metodi, dai messaggi e da tutte le belle altre cose che abbiamo visto nella puntata passata.

Letto così sembrerebbe in ogni caso, che l'uso di un discorso ibrido sia limitativo rispetto a quello che metterebbe a disposizione un linguaggio espressamente costruito; invece l'unico linguaggio puro esistente non ha avuto lo stesso successo di altri. Secondo gli utenti l'approccio ibrido ha migliori qualità per le seguenti ragioni:

- l'abitudine, l'uso e l'adozione di tecniche procedurali è troppo forte perché anche l'utente più raffinato ne possa fare agevolmente a meno. Con un linguaggio ibrido è possibile, caso per caso, adottare sistemi dell'una o dell'altra filosofia senza troppe rinunzie a costumi ormai consolidati (pare proprio la stessa filosofia dei pascalisti che scacciano il GOTO, ma lo usano senza far sapere niente a nessuno). In termini più tecnici (ma il succo del discorso è sempre lo stesso), con un linguaggio ibrido si può scegliere il miglior tool per ogni passo del programma che si sta realizzando senza forzare l'una o l'altra filosofia a strutture che magari non le sono connaturali.

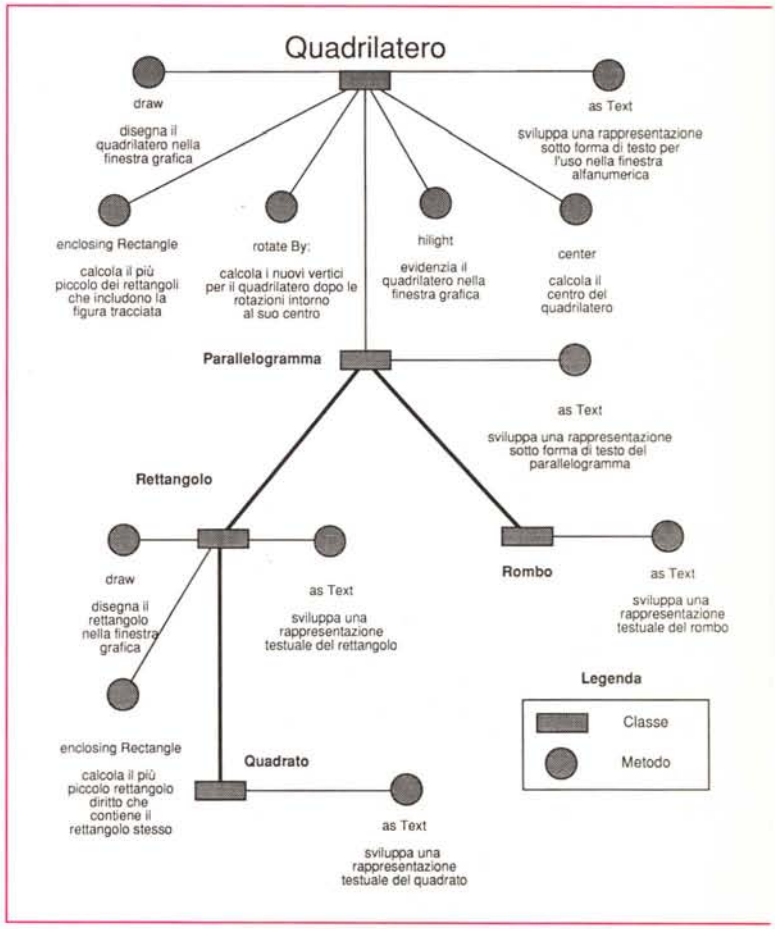

Figura B - La struttura principale di QuadWorld (da Schmucker, opera citata).

● I linguaggi ibridi permettono di usare librerie già precostituite o di riutilizzare pezzi di software già ben testati; la cosa è più vera in particolare per certi tipi di linguaggio, che hanno a disposizione librerie potenti, già create da altri (è il classico esempio del C, alle cui numerose librerie presenti sul mercato è possibile abbinare quello di MacApp, appunto nell'ambiente, visto che ne stiamo parlando, dell'Object Pascal.

● In genere un linguaggio ibrido, visto che sono in commercio implementazioni di diversi idiomi, invoglia decisamente al passaggio alla programmazione strutturata, in quanto il programmatore eviterà lo shock culturale del passaggio ad un nuovo linguaggio ex-novo, in quanto considererà il passaggio alla programmazione OO come un upgrading del linguaggio già usato, senza stravolgimenti radicali.

● Per lo stesso motivo, essendo già in parte ben conosciuti dal programmatore, i linguaggi strutturati saranno appresi in minor tempo, essendoci la sola necessità di imparare l'estensione, orientata all'oggetto, del linguaggio già conosciuto.

● I linguaggi ibridi permettono di testare alcuni costrutti attraverso le due differenti tecniche di chiamata al metodo e alla procedura, per verificare quale possa essere la più efficace. Ad esempio, il chiudersi in un linguaggio OO senza contatti con tecniche esterne ad esso può essere limitativo, come succede spesso in complessi calcoli numerici.

Tutto questo però non significa che conoscere un linguaggio Pascal permette immediatamente di redigere programmi in Object Pascal; tanto per capirci Object Pascal permette di scrivere programmi tradizionali così come qualunque linguaggio più generico, ma per usarlo in maniera OO occorre un minimo (si fa per dire) di allenamento e fatica, non tanto per imparare in sé i nuovi comandi e le nuove procedure a disposizione, ma per entrare nello spirito dell'ambiente stesso e per costruire naturalmente e senza «traduzioni» nuove routine e blocchi di programma. Il vantaggio di non buttare via tutta l'esperienza trascorsa ed evitare la fatica di dover imparare una nuova strada consente di trasferire più facilmente concetti già esistenti al nuovo ambiente; in un linguaggio OO l'unità di modulo è la classe, in un linguaggio convenzionale procedurale l'unità è, appunto, la procedura (o la funzione, nel caso del C); da qui a passare alla costruzione di blocchi, con la loro brava ereditarietà, il passo è breve.

Classi e oggetti in Object Pascal

Object Pascal è un linguaggio potente e ben costruito, che conserva la stessa chiarezza del Pascal da cui deriva; la sua modularità e struttura OO gli deriva da alcune differenze di base che potremo riassumere in questa puntata, rimandando alla prossima volta la trattazione di MacApp.

In effetti la grande risorsa di Object Pascal è data dalla potenza applicativa dei suoi oggetti e classi.

Il Pascal standard ha quattro tipi strutturati: array, serie, file e record. Queste, che sono classi a tutti gli effetti in un linguaggio OO, in Object sono solo la base di una serie ben più importante; nuove classi sono definite come serie addizionali di tipi strutturati, in estensione alla sintassi e alla semantica dei comandi di tipo Pascal. Un nuovo tipo strutturato, chiamato oggetto, somiglia da vicino al tipo record standard (ma non lo sostituisce), e la somiglianza si spinge a tal punto da rappresentare gli stessi tipi di record, composti di campi di differenti tipo di dati. Ma la vera differenza sta nel fatto che Object Pascal possiede due tipi nuovi di oggetti; campi dati, che contengono i dati di variabile associati agli oggetti della classe stessa o direttamente i puntatori ai dati; e i campi metodo che puntano alle procedure e alle funzioni che implementano i metodi della classe.

L'implementazione Apple di Object Pascal per Macintosh si chiama Macintosh WorkShop Pascal. In questa implementazione esistono alcune caratteristiche degne di nota e piuttosto pregevoli; Macintosh WorkShop Pascal permette di definire librerie compilate separate. Queste librerie, che qui sono chiamate unità, sono definite in due parti; la parte di interfaccia, che definisce le costanti conosciute esternamente, le variabili, i tipi e le chiamate per procedure esterne già definite e per le funzioni; e la parte di implementazione, che crea il corpo delle procedure e delle funzioni, e che determina la disponibilità tra l'altro, delle librerie esterne. Questo permette di fornire le procedure stesse in forma compilata all'eventuale utente che di esse necessita, per poter usare in un nuovo programma la procedura creata nuova di zecca, non è necessario disporre del codice sorgente (che l'autore, per ovvi motivi, potrebbe non voler mettere a disposizione); è sufficiente disporre della procedura compilata e della interfaccia d'unità, generalmente piuttosto semplificata. Un esempio classico di sistema di chiamata a procedure precompilate sono le [call] alle routine QuickDraw, che permettono di creare interfacce utente di splendida fattura anche con il più gracile Basic.

Nel definire le classi in un Pascal Object Oriented sono adottate in particolare tecniche piuttosto standard. Questa dualità appena descritta è il vero cardine del discorso: la parte di interfaccia descrive l'organizzazione dei dati destinati ad essere manipolati dalle nuove classi, e la parte implementativa costruisce i «metodi», le nuove procedure che vengono realizzate quando un messaggio di «attivazione» viene loro inviato. Lo stretto collegamento che esiste tra interfaccia e implementazione contribuisce a rendere più efficace l'idea della costruzione in mattoni che finora ci eravamo fatti della programmazione Object Oriented. In questo caso i laterizi sono rappresentati dai metodi, le interfacce fanno invece la parte della malta. Ovvio, continuando l'analogia che nessuno si sognerebbe di impastare i mattoni sotto casa.

Ovviamente la parte implementativa di una unità, il cuore del sistema, può essere usata per più di un metodo delle classi definite nella parte di interfaccia. A causa del fatto che la parte implementativa generale nasconde tutto quello che contiene nel corpo stesso del programma, è possibile definire qualunque numero di costanti private globali, oggetti globali o, anche classi globali, tutte private.

MacApp, l'ambiente applicativo espandibile proprio di Macintosh, non è altro che un gruppo di classi Object Pascal, o in termini più semplici, un gruppo di procedure precompilate che servono a costruire più facilmente applicazioni che rispettano lo standard Macintosh. Si tratta davvero di una serie di tool che permettono di costruire un programma Mac in maniera facile e veloce; per molti programmatori l'abbinata MacApp-Linguaggio OO rappresenta il solo tool giornaliero per creare programmi anche estremamente complessi. Addirittura essi tendono a identificare le due cose tanto che quando sentono parlare di programmazione OO non immaginano di aver già percorso metà della strada per giungere al risultato.

Da ciò a parlare finalmente di MacApp il passo è anche più breve di quelli fatti in precedenza; non ci resta che parlare quindi di questo ambiente di sviluppo, supportato direttamente da Apple e che rappresenta, probabilmente il più grande sforzo programmatorio nell'ambito dei linguaggi e ambienti di sviluppo realizzati in ambiente Macintosh. È questo l'argomento che affronteremo la prossima volta. MC

SOFTWARE TECNICO :

- Contabilità Imprese Edili e Studi Tecnici
- Gestione Gare d'Appalto e Lavori Pubblici
- Gestione Albi professionali
- Topografia in 2D e 3D
- Progettazione stradale, cartografia
- Software per Gestione dBase dal CAD e scannerizzazione immagini
- Software Tecnico manutenzione Ascensori
- Manutenzione ed amm.ne Immobili

CAD:

- Architettonici e applicativi AUTOCAD* (alcuni esempi):
- Termotecnica
- Gestione topografica di cave di marmo, cave di inerti, discariche, bacini.
- Terreni agricoli
- Arredamento interni - Cucine - Bagni

AUTOCAD E' UN MARCHIO AUTODESK

SOFTWARE MEDICO:

- Gestione Medici di base
- Ostetricia - Ginecologia
- Medicina generale (Realizzazione di procedure per altre specializzazioni) - Oculistica

SOFTWARE GESTIONALE APPLICATIVO (alcuni esempi)

- Contabilità - Abbigliamento - Ottica

RICERCA OPERATIVA E MODELLI DI

OTTIMIZZAZIONE CON APPLICAZIONI

SPECIFICHE GIA' SVILUPPATE.

SOFTWARE DISPONIBILE PER AMBIENTI MS-DOS, WINDOWS, UNIX.

I PREZZI SONO IVA ESCLUSA

SOFTWARE ORIZZONTALE (esempi):

TURBO C ++ IT	Lit. 300.000
TURBO C ++ P. IT	Lit. 450.000
WINDOWS 3.0 IT	Lit. 260.000
INFORMIX WINGZ	Lit. 620.000
MS-OS 2 1.1	Lit. 580.000
ALDUS PAGE MAKER	Lit. 1.290.000
EXCEL IT	Lit. 670.000
AUTOCAD 10 386	Lit. 6.700.000
CLIPPER 5.0	Lit. 940.000

HARDWARE (esempi):

PC AT 16 Mhz. - 1 MB RAM - 1 FDD 1.2 MB - 1 HD 40 MB - SK VIDEO VGA - MONITOR VGA MONOCROMATICO -

Configurazione completa Lit. 1.490.000

PC 386 33 Mhz. - 2 MB RAM - 1 FDD 1.2 MB - 1 HD 40 MB - SK VIDEO VGA - MONITOR VGA MONOCROMATICO -

Configurazione completa Lit. 3.890.000

Questi sono alcuni esempi delle nostre offerte software-hardware. Per l'invio del catalogo SOFTWARE-HARDWARE telefonare ai numeri sottoindicati.

STAMPANTI (esempi):

PANASONIC LASER	Lit. 2.800.000
OKI 380 24 AGHI	Lit. 810.000
CITIZEN SWIFT 9	Lit. 445.000

MICROSYS
Soluzioni software ed hardware

Via Germanico,24
00192 - ROMA
Tel. 06/3251763-4-5

SI RICERCANO RIVENDITORI ED AGENTI