

Classi, Oggetti e Metodi...

– Objective-C –

Biagio Raucci

web site : www.raucci.net

Sommario

In questo tutorial verranno trattati alcuni concetti fondamentali della *programmazione orientata agli oggetti* (OOP) i.e. quel paradigma di programmazione, che prevede di raggruppare in un'unica entità (*la classe*) sia le strutture dati che le procedure che operano su di esse, creando per l'appunto un "oggetto" software dotato di proprietà (*dati*) e metodi (*procedure*) che operano sui dati dell'oggetto stesso. Partiremo, innanzitutto, dal concetto di classi per introdurre, via via – sempre in modo informale – quel po' di terminologia necessaria a muovere i primi passi nell'ambito di questo nuovo paradigma di programmazione.

Objective-C è un linguaggio di programmazione riflessivo orientato agli oggetti, sviluppato da Brad Cox alla metà degli anni '80 presso la Stepstone Corporation. Come lo stesso nome suggerisce, l'Objective-C è un'estensione a oggetti del linguaggio C. Esso mantiene la completa compatibilità col C (a differenza di quanto avviene, per esempio, nel caso del C++). Tra l'altro, anche a causa di questa compatibilità, Objective-C non è dotato di forte tipizzazione (caratteristica che invece esibiscono, tra gli altri, sia C++ che Java).

Le estensioni a oggetti con cui Objective-C arricchisce il modello semantico del C sono ispirate al linguaggio Smalltalk, in particolar modo alla gestione dei messaggi. Le caratteristiche del runtime system collocano l'Objective C tra i linguaggi ad oggetti dinamici. Sono supportati tutti gli elementi classici della programmazione a oggetti; ma non mancano concetti innovativi anche su questo fronte, come il meccanismo delle categorie e strumenti legati alla riflessione. La sua diffusione è principalmente legata al framework OpenStep di NeXT e al suo successore Cocoa, presente nel sistema operativo Mac OS X di Apple.

Valgono, per finire, le solite precisazioni: queste note sono il frutto della mia più completa incompetenza nella programmazione in Objective-C / Cocoa; sono note pensate, soprattutto, per me – a mio uso e consumo. Le metto qui, a disposizione di tutti, soprattutto per essere corretto. (Se, poi, risulteranno utili anche a qualche neofita, tanto di guadagnato.) I riferimenti per contattarmi li trovate in questi stessi fogli: se beccate qualche errore più o meno grave, segnalatemelo. Grazie.

Key words: Programming, C, Objective-C, Cocoa, Classi, Oggetti, Metodi.

1 Classi

Le classi sono uno strumento per costruire strutture dati che contengano non solo dati ma anche il codice per gestirli. Ogni classe contiene degli oggetti che hanno le stesse caratteristiche. Si avvale poi di relazioni di *ereditarietà* secondo le quali nuove classi di oggetti sono derivate da classi esistenti, ereditando le loro caratteristiche e estendendole con caratteristiche proprie.

Come tutti i costrutti che permettono di definire le strutture dati, una classe definisce un nuovo tipo di dato.

I membri di una classe sono:

attributi (dati, esattamente come i membri di un record)

metodi (procedure che operano su un oggetto)

Dal punto di vista matematico, una classe definisce un insieme *in modo intensivo*, ovvero definendone le caratteristiche invece che elencandone gli elementi. Se l'accesso agli attributi è ristretto ai soli membri della classe, le caratteristiche dell'insieme possono includere vincoli sui possibili valori che il record degli attributi può o non può assumere, e anche sulle possibili transizioni tra questi stati. Un oggetto può quindi essere visto come una macchina a stati finiti.

Una classe può dichiarare riservate una parte delle sue

Email address: raucci@gmail.com (Biagio Raucci).

proprietà e/o dei suoi metodi, e riservarne l'uso a sé stesso e/o a particolari tipi di oggetti a lui correlati.

2 Gli oggetti

Un *oggetto*¹, così come è pensato nella programmazione orientata agli oggetti, è una cosa – qualcosa cioè di molto simile ad un oggetto reale con il quale è possibile, appunto, interagire. Questo modo di concepire un costruito di base della programmazione agli oggetti è in netto contrasto con quello che avviene, ad esempio, con linguaggi quali il C o, più in generale, con i linguaggi procedurali. In C, infatti, tipicamente si pensa a cosa si vuole fare dapprima e poi si passa a trattare gli oggetti... tutto l'opposto di quello che si fa con la programmazione orientata agli oggetti, insomma. Cerchiamo di chiarire il concetto esposto con un semplice esempio. Prendiamo il vostro cellulare. Un cellulare è ovviamente un oggetto, il vostro oggetto. Chiaramente, il cellulare che avete non è un cellulare qualsiasi: è il vostro cellulare. Fabbriato, ad esempio, in quella fabbrica del Giappone, o di Detroit. Il vostro cellulare ha il proprio numero d'identificazione che lo rende "unico" rispetto a tutti gli altri cellulari. Insomma – per dirla con il linguaggio proprio della programmazione agli oggetti – il vostro cellulare è un'istanza di un cellulare. E continuando con la terminologia, "cellulare" è il nome della classe da cui l'istanza è stata creata. Sicché ogni volta che un nuovo cellulare viene prodotto, una nuova istanza della classe **cellulare** viene generata i.e. viene generato un nuovo *oggetto*. Il vostro cellulare – un oggetto, appunto –, è, ad esempio, nero, ha lo sportellino, i pulsanti colorati e il display a colori. E con questo cellulare voi fate certe azioni: ci telefonate, ci giocate, ci ascoltate la musica, e così via. Con quel cellulare – il vostro cellulare, appunto – eseguite delle azioni che possono essere eseguite, in genere, anche su altri cellulari.

3 Istanze e Metodi

L'unica occorrenza di una classe è detta *istanza* e le azioni che possono essere eseguite su un'istanza sono detti *metodi*. Applicando un metodo ad un oggetto possiamo modificarne lo *stato* dell'oggetto stesso. Se – tanto per esser chiari – il nostro metodo è "caricare la scheda del cellulare", dopo che il metodo è stato eseguito il credito della scheda montata sul mio cellulare sarà aumentato. Il metodo, insomma, ha avuto effetto sullo stato della scheda del mio cellulare.

A questo punto è d'obbligo una brevissima sintesi dei concetti esposti. Gli oggetti sono delle rappresentazioni uniche di una classe, e ciascun oggetto possiede alcune informazioni (dati) che sono tipiche di quel oggetto. I

¹ Un oggetto è una istanza di una classe. Un oggetto occupa memoria, la sua classe definisce come sono organizzati i dati in questa memoria.

metodi, invece, provvedono a modificare i dati degli oggetti.

Bene – direte voi – ma Objective-C come tratta questi concetti? Procediamo con ordine. Per applicare i metodi agli oggetti o alle classi, la sintassi è:

```
1 [ClassOrInstance method ];
```

Osservate attentamente la sintassi²: tra le parentesi sono presenti due "chiamate": il nome dell'*oggetto* (particolare istanza della classe) o quello della classe e il particolare metodo su cui si vuole agire. Quando si chiama un'istanza di una classe o la classe stessa per eseguire delle azioni stiamo mandando un *messaggio*; il destinatario del messaggio è detto *receiver*. Un modo, quindi, più generale per descrivere la sintassi precedente è:

```
1 [receiver message ];
```

Volendo in qualche modo sintetizzare quanto detto nell'esempio precedentemente fatto, potremmo scrivere innanzitutto:

```
1 yourMobile = [Mobile new ];
```

Ovvero: mandiamo un messaggio alla classe **Mobile** (il *receiver* del messaggio) chiedendole di fornirci un nuovo cellulare. Il risultato di questa chiamata è un oggetto (che rappresenta, appunto, il vostro cellulare) che viene memorizzato nella variabile **yourMobile**. Su tale oggetto (istanza della classe **Mobile**) applicheremo alcuni metodi. Qualche esempio?

```
1 [yourMobile ricaricaBatteria ];  
2 [yourMobile ricaricaCredito ];  
3 [yourMobile componiNumero ];  
4 [yourMobile attivaVivavoce ];
```

A questo punto vale la pena soffermarsi con un esempio concreto: giusto per chiarire ulteriormente quanto detto in precedenza.

² La sintassi aggiunta rispetto al C è intesa al supporto della programmazione ad oggetti. Il modello di programmazione dell'Objective-C è basato sullo scambio di messaggi tra oggetti così come avviene in Smalltalk. Tale modello è differente da quello di Simula, che viene usato in numerosi linguaggi quali, tra gli altri, il C++. Questa distinzione è semanticamente importante e consiste principalmente nel fatto che in Objective-C non si chiama un metodo, ma si invia un messaggio. Si dice che un oggetto chiamato *ogg* la cui classe implementa il metodo *faiQualcosa*, risponde al messaggio *faiQualcosa*. L'invio del messaggio *faiQualcosa* all'oggetto *ogg* è espresso da: `[ogg faiQualcosa];`

4 Una classe per lavorare con le somme

Iniziamo, dapprima, a definire una classe in Objective-C³; successivamente impareremo a lavorare con una sua istanza.

Supponiamo di aver bisogno di scrivere un programma (*sic*) per eseguire la somma di due numeri interi. Chi è completamente a digiuno della programmazione ad oggetti si butta subito sulla scrittura procedurale e abbozzerà qualche cosa di molto simile al codice che segue:

```
1 #include <stdio.h>
2
3 int main (int argc, const char * argv[]) {
4     // insert code here...
5     int a = 1;
6     int b = 2;
7     printf ("The sum of %i and %i is %i \n", a,b,a+b);
8     return 0;
9 }
```

In questo programma la somma viene ad essere rappresentata in termini dei suoi addendi *a* e *b*.

Sul versante della programmazione ad oggetti, invece, struttureremo il programma in tre file: la sezione denominata `@interface`, quella `@implementation` e, per finire, `program`. Chi sono e cosa fanno questi segmenti? Procediamo con ordine. La sezione `@interface` descrive la classe, i suoi dati e i suoi metodi; semplicemente? è una dichiarazione d'intenti. La sezione denominata, invece, `@implementation` contiene il codice che implementa i metodi dichiarati nella sezione `@interface` mentre, per finire, la sezione `program` contiene il codice del programma vero e proprio.

4.1 La sezione `@interface`

Una possibile `@interface` al nostro programma può essere scritta nel seguente modo:

```
1 /*
2     Name           : Somma.h
3     Interface of class : Somma.m
4     Created by      : Biagio Raucci
5     Date            : 10/06/06.
6 */
7
```

³ L'Objective-C è un sottile strato posto sopra il linguaggio C; C quindi è un sottoinsieme stretto dell'Objective-C. Ne consegue che è possibile compilare un qualsiasi programma scritto in C con un compilatore Objective C. La gran parte della sintassi (clausole del preprocessore, espressioni, dichiarazioni e chiamate di funzioni) è derivata da quella del C, mentre la sintassi relativa alle caratteristiche object-oriented è stata creata per ottenere la comunicazione a scambio di messaggi simile a quella di Smalltalk.

```
8 #import <Cocoa/Cocoa.h>
9
10 @interface Somma : NSObject {
11     int    a;
12     int    b;
13 }
14 - (void)  setAddent1: (int) x;
15 - (void)  setAddent2: (int) y;
16 - (int)   getA;
17 - (int)   getB;
18 @end
```

Commentiamolo. Quando si definisce una nuova classe occorre, innanzitutto, precisare al compilatore qual'è la *classe madre*, i.e. da quale classe occorre generare la nuova classe. Fatto questo occorrerà, poi, specificare che tipo di dati verranno memorizzati negli oggetti di questa nuova classe, ovvero occorrerà descrivere i membri (detti *instance variables*) che la classe dovrà contenere. Poi, per finire, occorre definire il tipo di operazioni (i *metodi*, appunto) che possono essere usati quando si verrà a lavorare con gli oggetti derivanti dalla classe in esame. Nel caso specifico:

- il nome della nuova classe è `Somma`;
- tale classe eredita le caratteristiche della classe `NSObject`: questa classe è definita nel file `Cocoa/Cocoa.h` – ecco perché occorre importarlo nel nostro file proprio all'inizio del programma;
- le variabili sono racchiuse all'interno delle parentesi graffe; in particolare abbiamo due variabili intere per la nostra classe `Somma`: `a` e `b`;
- gli *instance methods* sono:
 - - (void) `setAddent1:` (int) `x`;
 - - (void) `setAddent2:` (int) `y`;
 - - (int) `getA`;
 - - (int) `getB`;

Quando si dichiara un nuovo metodo occorre precisare al compilatore se il metodo restituisce o meno un valore e, nel caso in cui venga restituito un valore, qual è il suo tipo. Sicché, la dichiarazione:

```
1 - (void) setAddent1: (int) x;
```

specifica che l'istanza del metodo chiamato `setAddent1:` (nota che i due punti al termine del nome del metodo indicano che il metodo stesso prende degli argomenti) non restituisce alcun valore ma ha bisogno di un valore intero (`int`) in ingresso. Il nome di questo intero è `x`. Per contro la dichiarazione:

```
1 - (int) getB;
```

ci informa, semplicemente, che l'istanza del metodo `getB` restituisce un intero ma non necessita di nessun argomento.

4.2 La sezione @implementation

Questa sezione contiene il codice per i metodi che sono stati dichiarati nella sezione @interface.

```
1  /*
2   Name           : Somma.m
3   Interface of class : Somma.m
4   Created by      : Biagio Raucci
5   Date           : 10/06/06.
6  */
7
8  #import "Somma.h"
9
10
11 @implementation Somma
12
13 -(void) setAddent1: (int) x
14 {
15     a = x;
16 }
17
18 -(void) setAddent2: (int) y
19 {
20     b = y;
21 }
22
23 -(int) getA ;
24 {
25     return a ;
26 }
27
28 - (int) getB
29 {
30     return ( b );
31 }
32
33 @end
```

Il metodo `setAddent1:` [rispettivamente `setAddent2:`] prende come argomento un metodo chiamato `x` [rispettivamente `y`] e semplicemente lo memorizza nella variabile `a` [`b`]. Il metodo `getA` [rispettivamente `getB`] restituisce, semplicemente, la variabile⁴ `a` [`b`].

4.3 la sezione program

La sezione `program` contiene il codice per risolvere il particolare problema.

Di seguito ritroviamo una proposta della sezione in oggetto:

```
1  /*
2   Name           : objsomma.m
3   Created by      : Biagio Raucci
4   Date           : 10/06/06.
5  */
```

⁴ Si fa esplicitamente notare che non è possibile accedere direttamente alle variabili `a` e `b` perché queste sono *nascoste-data encapsulation*.

```
6
7 #import <Foundation/Foundation.h>
8 #import "Somma.h"
9 int main (int argc, const char * argv[]) {
10     NSAutoreleasePool * pool =
11         [[NSAutoreleasePool alloc] init];
12
13     Somma *miaSomma ;
14
15     /*
16     miaSomma is an object of type Somma,
17     i.e. miaSomma is used to store values
18     from your new Somma class. The (*) in
19     front of miaSomma is required.
20     Technically, it says that miaSomma is
21     a pointer to a Somma
22     */
23
24     // create an instance of a Somma
25
26     miaSomma = [[Somma alloc] init ] ;
27
28     /*
29     alternatively you may write:
30     miaSomma = [Somma alloc];
31     miaSomma = [miaSomma init];
32     */
33
34     NSLog(@"My first class.\n");
35
36     /*
37     You can use getA method for obtain the
38     value stored in instance variable named a
39     */
40
41     NSLog( @"Value a= %d\n", [miaSomma getA ]);
42
43     /*
44     You can use getB method for obtain the
45     value stored in instance variable named b
46     */
47
48     NSLog( @"Value b= %d\n", [miaSomma getB ]);
49
50     /*
51     You can use setAddent1: method to set the
52     value of instance variable a to 10
53     */
54
55     [ miaSomma setAddent1: 10 ];
56
57     /*
58     You can verify that the value is written to
59     instance variable correctly
60     */
61
62     NSLog( @"Value a = %d\n", [miaSomma getA ]);
63
64     /*
65     You can use setAddent2: method to set the
66     value of instance variable b to 20
67     */
68
69     [ miaSomma setAddent2: 20 ];
```

```

70
71  /*
72  You can verify that the value is written to
73  instance variable correctly
74  */
75
76  NSLog( @"Value b = %d\n", [miaSomma getB ]);
77
78  /*
79  You can evaluate the sum of a and b variables
80  */
81
82  NSLog( @"Value a+b = %d\n", [miaSomma getA]+
83  [miaSomma getB]);
84
85  return 0;
86
87  [pool release];
88  return 0;
89  }

```

Prima di tutto notiamo che NSLog è una funzione della FoundationKit per stampare le istruzioni di debug sulla console. All'interno della sezione main si definisce una variabile, miaSomma, con l'istruzione seguente:

```

1  Somma  *miaSomma ;

```

La linea precedente dice, semplicemente, che miaSomma sia un oggetto di tipo Somma, i.e. miaSomma viene usata per memorizzare i valori per la classe Somma. Passiamo, adesso, a creare un oggetto. Per fare ciò procederemo in due step:

- **Allochiamo**, prima di tutto, uno spazio di memoria per salvare il nuovo oggetto:

```

1  miaSomma = [Somma alloc];

```

Quando si invia il messaggio alloc ad una classe si crea un oggetto.

- Il secondo passo consiste nell'**inizializzare** un oggetto dopo averlo allocato:

```

1  miaSomma = [miaSomma init];

```

Scorrendo il programma si notano le istruzioni:

```

1  [ miaSomma setAddent1: 10 ];
2  [ miaSomma setAddent2: 20 ];

```

attraverso le quali si stabiliscono i valori agli addendi della somma, valori che vengono stampati a video attraverso le istruzioni:

```

1  NSLog( @"Value a = %d\n", [miaSomma getA ]);
2  NSLog( @"Value b = %d\n", [miaSomma getB ]);

```

L'istruzione release permette di liberare la memoria.

Riferimenti bibliografici

- [1] AA.VV. Numerical Recipes Cambridge University Press. <http://www.library.cornell.edu/nr/>
- [2] James Ducas Davidson and Apple Computer, Inc. Cocoa with Objective-C O'RELLY
- [3] Brian W. Kernighan, Dennis M. Ritchie, Linguaggio C ANSI ed. Jackson Libri.
- [4] Livio Sandel, MacCocoa – available at <http://www.macocoa.omitech.it/index.htm>
- [5] Marco Coisson, Introduzione a Cocoa – available on Marco Coisson's home page http://homepage.mac.com/marco_coisson/
- [6] A.A.V.V., Wikipedia <http://it.wikipedia.org/>
- [7] Byron S. Gottfried, Programmare in C – McGraw-Hill
- [8] Barninga Z!, Tricky C <http://tinyurl.com/9s3dg>
- [9] Il linguaggio C – appunti delle lezioni, Vittorio Ghini <http://www.cs.unibo.it/~ghini/>