

Introduzione al linguaggio Java 1.0

Ernesto "ventuno" Mudu

27 Aprile 2007

Copyright (c) 2007 Ernesto 'ventuno' Mudu. È garantito il permesso di copiare, distribuire e/o modificare questo documento seguendo i termini della Licenza per Documentazione Libera GNU, Versione 1.1 o ogni versione successiva pubblicata dalla Free Software Foundation; senza Sezioni Non Modificabili, nessun Testo Copertina, e con nessun Testo di Retro Copertina. Una copia della licenza è acclusa nella sezione intitolata Licenza per Documentazione Libera GNU:

*a tutte le donne,
impercettibile spinta
ad andare avanti
impercettibile segno
della presenza divina(?)*

Parte I

Il paradigma ad oggetti

1 Introduzione

La programmazione è l'arte di scrivere programmi per il computer in modo da espanderne le potenzialità unici limiti sono quello tecnologico, ovvero la capacità del calcolatore sul quale il programma deve funzionare e quello della fantasia del programmatore stesso. Esistono due paradigmi di programmazione, ovvero due modi diversi di programmare la programmazione procedurale e la programmazione ad oggetti, nel prossimo capitolo cercheremo di spiegare la differenza tra le due tipologie.

2 Procedurale vs Oggetti

Nella programmazione procedurale, il programmatore scrive blocchi ordinati di codice (funzioni), il compilatore quindi, prende ogni singola riga di codice e la trasforma in binario, con lo scopo di renderla comprensibile alla macchina. Nella programmazione ad oggetti, invece, la realtà non viene vista come tante righe di codice una di seguito all'altra, ma come un insieme di oggetti con *proprietà* (o variabili) e *metodi* (o funzioni) ben definiti che interagiscono tra loro.

3 La programmazione ad oggetti: concetti generali

Per comprendere cosa si intenda con oggetti e di come essi vengano modellati dal programmatore, possiamo prendere un qualsiasi oggetto di uso comune, ad esempio una banale sedia, e cercare di elencarne proprietà e metodi, così come potrebbe intenderli chi deve scrivere del codice.

Una sedia ha determinate proprietà:

- il colore;
- il peso;
- l'altezza;
- il peso massimo che può sostenere;
- ...

e ha anche diverse funzioni:

- possiamo usarla per sederci;
- per poggiarci sopra un libro;
- ...

Questo è un esempio semplice che permette di capire come ogni oggetto reale sia in realtà sintetizzabile attraverso i concetti espressi. Ma come applico gli oggetti nella programmazione? Un'applicazione della programmazione ad oggetti, potrebbe ad esempio, essere quella di definire i singoli elementi di un'interfaccia grafica (i pulsanti, le finestre, ...) come elementi dotati di proprietà specifiche (il colore, lo stato, le dimensioni, ...) che interagiscono tra di loro attraverso i singoli metodi (il pulsante viene premuto e la finestra si chiude, la finestra cambia colore quando il mouse passa su di essa, ...).

4 La programmazione ad oggetti: nello specifico

Quando programmiamo ad oggetti, la parola d'ordine è *astrazione*:

astrazione: *s. f. 1 (filos.) operazione attraverso la quale l'intelletto ricava concetti universali dalla conoscenza di oggetti individuali, prescindendo dalle determinazioni particolari degli oggetti stessi [...]*¹

questo significa che nel cercare di modellare un oggetto, non dovrò cercare di rappresentarne uno specifico, ma cercare di avvicinarmi sempre di più alla sua idea generale, ad esempio nel modellare il concetto di sedia, devo avere proprietà e metodi sufficienti da poter rappresentare tanto una sedia comune quanto una più complicata sedia da ufficio. Si parà infatti di *classi* che verranno *istanziate* per la creazione di un oggetto vero e proprio. Nella programmazione ad oggetti risulta fondamentale la fase di progettazione del software, ovvero quella fase in cui il programmatore deve modellare le singole classi e i rapporti tra di esse, una classe può essere rappresentata attraverso una tabella come quella che vediamo di seguito:

Nome della classe
proprietà: tipo
metodo: valore ritornato

4.1 Incapsulamento

L'incapsulamento è la proprietà per cui un oggetto contiene (*incapsula*) al suo interno le proprietà e i metodi che accedono ai dati stessi. Lo scopo principale dell'incapsulamento è appunto dare accesso ai dati incapsulati

¹<http://www.garzantilinguistica.it>

solo attraverso metodi definiti (quelli che chiameremo *getters* e *setters* nei prossimi capitoli, quando ci addentreremo nel discorso relativo al linguaggio Java). L'incapsulamento permette di vedere l'oggetto come una scatola nera di cui, attraverso l'Interfaccia sappiamo cosa fa e come interagisce con l'esterno ma non come lo fa . I vantaggi principali portati dall'incapsulamento sono: robustezza, indipendenza e l'estrema riusabilità degli oggetti creati².

4.2 Ereditarietà

Ogni classe madre può generare delle classi figlie, che ereditano le stesse proprietà e gli stessi metodi, questo permette di aggiungere nuovi metodi senza dover modificare la classe originale o di modificare quelli già esistenti, secondo le proprie esigenze.

4.3 Poliformismo

Corrisponde alla possibilità che i metodi, una volta ereditati, potranno essere ridefiniti, ed assumere quindi una nuova forma.

Parte II

Il linguaggio Java

5 Cenni storici, e tecnici

Java è un linguaggio creato dalla *Sun Microsystems* e rilasciato nel 1995. Dal 2006 la Sun ha rilasciato sotto licenza GPL (*General Public License*) la propria implementazione del linguaggio.

Proprietà tanto ambiziosa quanto importante del linguaggio Java, è che risulta sempre indipendente dalla piattaforma, una volta scritto e compilato il programma, infatti, dovrebbe poter girare su qualsiasi hardware, ove sia presente una *Java Virtual Machine* (JVM), ovvero una macchina virtuale, in grado di leggere un codice intermedio: il *bytecode*, a metà tra il linguaggio Java ed il linguaggio macchina, Java risulta essere quindi, un ibrido tra linguaggio compilato ed interpretato.

Ci concentreremo, ora, sul linguaggio, andremo infine a vedere come Java implementi le varie caratteristiche della programmazione ad oggetti (incapsulamento, ereditarietà, etc.).

²http://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti#Incapsulamento

6 Il linguaggio

Esempio classico, dal momento in cui insegna un linguaggio di programmazione è l'HelloWorld, vediamo di seguito, e occupiamoci poi dell'analisi del codice:

```
1 public class HelloWorld {
2 private String frase;
3 HelloWorld(String fras){
4 frase = fras;
5 }
6 public void saluta () {
7 System.out.println(frase);
8 }
9 public static void main(String[] args){
10 HelloWorld hw = new HelloWorld('Ciao Mondo');
11 hw.saluta(); //commento inutile
12 }
```

In realtà l'esempio proposto è un po' più complicato del classico HelloWorld, ma è una cosa voluta, serve infatti a proporre i diversi aspetti della programmazione in Java, spieghiamo prima come dovrebbe funzionare il programma: la classe HelloWorld, è composta da una proprietà (*frase*, ovvero una stringa che costituisce il nostro saluto, ed un metodo (*saluto* che non fa altro che richiamare la funzione di Java che permette di stampare a video una stringa di caratteri (nel nostro caso l'output del programma sarà *Ciao Mondo*). Vediamo ora il codice, cercando di spiegare come esso funzioni, riga per riga:

- alla riga 1, si apre la classe, il corpo della stessa deve essere compreso tra parentesi graffe, l'ultima si chiude alla riga 12;
- riga 2, abbiamo la dichiarazione di una variabile, come possiamo vedere il nome della variabile deve essere preceduto da un modificatore di visibilità e dal tipo, in questo caso la variabile è di tipo String, perché deve contenere una stringa di testo, quella che verrà poi stampata a schermo con il metodo *saluta*;
- righe 3-5, il *costruttore* è un metodo speciale, che deve sempre avere lo stesso nome della classe a cui si riferisce e, viene invocato ogni volta che la classe viene istanziata, il costruttore può essere usato per assegnare i valori iniziali alle variabili, al momento della creazione dell'oggetto, in questo caso assegnamo la stringa passata come argomento al costruttore, che costituirà poi il nostro saluto;
- righe 6-8, dichiarazione di un metodo, come per la classe il corpo deve essere compreso tra le parentesi graffe e come le variabili deve

essere preceduto dal suo modificatore di visibilità e dal tipo di variabile ritornato nel nostro caso *saluta* non fa altro che richiamare la funzione di Java (*println*) che stampa a video una stringa di caratteri, dato che il metodo non deve ritornare nessun tipo di variabile usiamo la parola chiave *void*;

- righe 9-11, ogni programma Java, deve avere in una delle sue classi, un metodo *main*, che viene richiamato ogni volta che viene eseguito il programma, come possiamo notare si tratta di un metodo statico questo vuol dire che non fa riferimento a nessun oggetto specifico, essendo il suo contenuto molto importante lo analizziamo riga per riga:
 - riga 10, viene creato un nuovo oggetto della classe *HelloWorld*, chiamato *hw*, siamo obbligati a scrivere una stringa come argomento del costruttore;
 - riga 11, utilizziamo il metodo *saluta*, attraverso l'oggetto *hw*, appena creato. Dopo le doppie slash *//*, troviamo un commento, questa parte viene ignorata al momento della compilazione, è utile al programmatore per commentare il codice, in alternativa può essere compreso tra */** e **/*, se il commento si trova su più righe.

Vediamo adesso come Java implementa i concetti tipici della programmazione ad oggetti.

6.1 Incapsulamento

Come abbiamo visto per ogni variabile, nel momento della sua dichiarazione possiamo specificare il suo modificatore di visibilità, che può assumere i valori:

- *public*, se vogliamo che il valore della variabile sia leggibile e il suo valore sia modificabile all'esterno della classe;
- *private*, se vogliamo che la variabile sia visibile solo all'interno della classe stessa.

Il modificatore di visibilità viene normalmente impostato a *private*, per rispettare il principio dell'incapsulamento per accedere ai valori della variabile usiamo i metodi *get* e *set*. Ecco un semplice esempio:

```
1 public class HelloWorld {
2 private String frase;
3 public String setFrase(fras String){
4 frase = fras;
5 }
3 public String getFrase(fras String){
4 return frase;
5 }
6 }
```

6.2 Ereditarietà e polimorfismo

Se vogliamo creare una nuova classe, che erediti proprietà e metodi da un'altra usiamo, nella dichiarazione della classe figlia, la parola chiave *extends*, in questo modo:

```
1 public class DopoHelloWorld extends HelloWorld{
2 public void saluta () {
3 System.out.println('Addio Mondo');
4 }
```

In questo caso, la classe *DopoHelloWorld* utilizzando il polimorfismo, ridefinisce il corpo del metodo *saluta*, ovviamente non abbiamo bisogno di riscrivere il resto della classe in quando viene tutto automaticamente ereditato.

6.3 Classi astratte ed interfacce

Le classi astratte, sono classi che non possono essere istanziate, per utilizzarle dobbiamo quindi ereditare i metodi in un'altra classe, mentre le interfacce sono classi che contengono solo metodi statici ed astratti, le classi che le implementano (parola chiave *implements*) devono per forza fornire il corpo dei metodi presenti nell'interfaccia.

```
1 public interface UsareHelloWorld{
2 public int saluta () {
3 System.out.println('Addio Mondo');
4 }
```

```
1public class HelloWorld implements UsareHelloWorld{
2 public void saluta () {
3 System.out.println(frase); [...]
4 }
```

L'esempio appena riportato, impone alla classe *HelloWorld* di definire al suo interno un metodo che si chiami *saluta*.

6.4 Controlli e cicli

Il costrutto *if* permette di verificare se una data condizione si verifica o meno, se la condizione tra parentesi tonde è vera, viene eseguito il blocco di codice contenuto all'interno delle parentesi graffe, altrimenti (se esiste un costrutto *else*) viene eseguito il codice presente nel secondo blocco di parentesi, altri controlli possono essere effettuati con *elseif*

```
if (condizione) {  
  // blocco di codice eseguito se condizione è vera  
} elseif (condizione) {  
  // se condizione è falsa e condizione2 è vera viene eseguito questo blocco  
  //questa parte può essere omessa  
}  
} else {  
  // se condizione e condizione2 sono false questa parte viene eseguita  
  // anche questa parte può essere omessa  
}
```

I cicli sono di due tipi, *for* e *while*, ecco gli esempi, per facilitare la comprensione del ciclo *for*, vedremo anche un esempio pratico:

```
for (espressione1;espressione2;espressione3) {  
  // ...  
}  
for (i=1;i<10;i++) {  
  // assegnamo prima alla variabile i il valore 1  
  // se è minore di 10 ne aumentiamo il valore di 1 (i++)  
  //quindi questo blocco di codice viene eseguito 8 volte  
}
```

```
while (condizione) {  
  // blocco di codice eseguito se condizione è vera  
}
```

7 Conclusioni

Lo scopo di questo documento, è quello di essere utile dispensa per coloro che hanno seguito il corso di Java, durante la sessione di corsi primaverile dell'anno 2007 presso l'underscore_TO*hacklab³, e non è volutamente esaustiva sugli argomenti trattati, può essere considerata una modesta introduzione al mondo della programmazione ad oggetti, per una vera preparazione sono consigliati un buon manuale, e tanta buona volontà da parte dell'interessato.

³<http://www.autistici.org/underscore>

La sola pratica, infatti costituisce il miglior metodo di apprendimento, in questo, come in tanti altri campi dello scibile umano.

Indice

I	Il paradigma ad oggetti	2
1	Introduzione	2
2	Procedurale vs Oggetti	2
3	La programmazione ad oggetti: concetti generali	2
4	La programmazione ad oggetti: nello specifico	3
4.1	Incapsulamento	3
4.2	Ereditarietà	4
4.3	Poliformismo	4
II	Il linguaggio Java	4
5	Cenni storici, e tecnici	4
6	Il linguaggio	5
6.1	Incapsulamento	6
6.2	Ereditarietà e polimorfismo	7
6.3	Classi astratte ed interfacce	7
6.4	Controlli e cicli	8
7	Conclusioni	8