Appendice D

Diagrammi di classe e sistemi orientati agli oggetti

Andrea Gini

Un effetto della strategia di incapsulamento è quello di spingere il programmatore a esprimere il comportamento di un sistema a oggetti unicamente attraverso l'interfaccia di programmazione delle classi. In questo senso, quando un programmatore si trova a dover utilizzare una libreria di classi realizzate da qualcun altro, non è interessato a come essa sia stata effettivamente implementata: di norma, è sufficiente conoscere le firme dei metodi, le relazioni di parentela tra le classi, le associazioni e le dipendenze, informazioni che non dipendono dall'implementazione dei singoli metodi.

Il diagramma di classe è un formalismo che permette di rappresentare per via grafica tutte queste informazioni, nascondendo nel contempo i dettagli di livello inferiore. L'uso dei diagrammi di classe permette di vedere un insieme di classi Java da una prospettiva più alta rispetto a quella fornita dal codice sorgente, simile a quella che si ha quando si guarda una piantina per vedere com'è fatta una città. La piantina non contiene tutti i dettagli della zona rappresentata, come la posizione delle singole abitazioni o dei negozi, ma riporta informazioni sufficienti per orientarsi con precisione.

I diagrammi di classe fanno parte di UML (Unified Modeling Language), un insieme di notazioni grafiche che permette di fornire una rappresentazione dei diversi aspetti di un sistema software orientato agli oggetti, indipendentemente dal linguaggio di programmazione effettivamente utilizzato. L'UML comprende sette tipi diversi di diagrammi, che permettono di modellare i vari aspetti dell'architettura e del comportamento di un sistema software prima di iniziarne lo sviluppo. I diagrammi UML costituiscono una parte fondamentale della documentazione di un sistema informativo, e forniscono una guida essenziale in fase di studio o di manutenzione del sistema stesso.

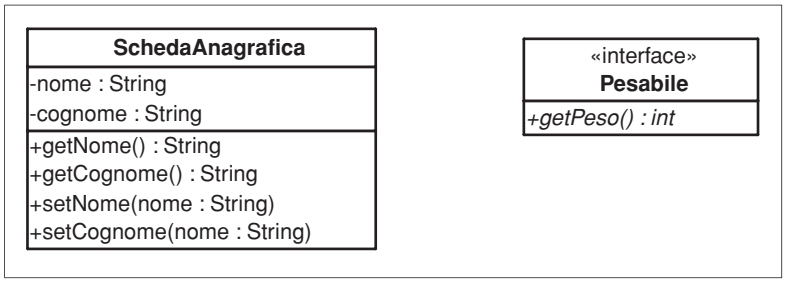
L'UML non è un linguaggio di programmazione, anche se negli ultimi anni gli ambienti di sviluppo hanno iniziato a includere strumenti che permettono di produrre codice a partire dai

diagrammi e viceversa. I seguenti paragrafi vogliono fornire una guida essenziale ai diagrammi di classe, l'unico formalismo UML presente in questo libro.

Classi e interfacce UML

In UML le classi e le interfacce sono rappresentate come rettangoli, suddivisi in tre aree: l'area superiore contiene il nome della classe o dell'interfaccia, quella intermedia l'elenco degli attributi e quella inferiore l'elenco dei metodi:

Figura D.1 – *Un esempio di classe e di interfaccia in UML.*



Entrambi i diagrammi non contengono alcun dettaglio sul contenuto dei metodi: il comportamento di una classe o di un'interfaccia UML è espresso unicamente tramite il nome dei suoi metodi. Le firme di metodi e attributi seguono una convenzione differente rispetto a quella adottata in Java: in questo caso, il nome precede il tipo, e tra i due compare un simbolo di due punti (:) come separatore. I parametri dei metodi, quando presenti, seguono la stessa convenzione. Il simbolo più (+), presente all'inizio, denota un modificatore public, mentre il trattino (-) indica private e il cancelletto (#) significa protected.

Il diagramma di interfaccia presenta alcune differenze rispetto a quello di classe:

- Al di sopra del nome compare un'etichetta "interface".
- Gli attributi (normalmente assenti) sono sottolineati, a indicare che si tratta di attributi statici immutabili.
- I metodi sono scritti in corsivo, per indicare che sono privi di implementazione.

Si osservi una tipica implementazione Java del diagramma di classe presente in figura 1:

```
public class SchedaAnagrafica {
```

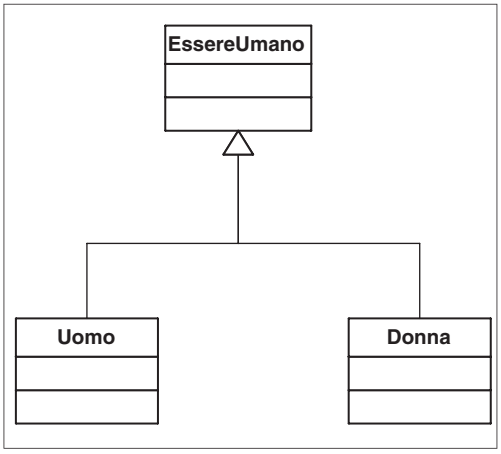
```
    private String nome;
    private String cognome;

    public String getNome() {
      return nome;
    }
    public void setNome(String nome) {
            this.nome = nome;
    }
    public String getCognome() {
      return cognome;
    }
    public void setCognome(String cognome) {
      this.cognome = cognome;
    }
}
```

Spesso il diagramma di classe presenta un livello di dettaglio inferiore rispetto al codice sottostante: tipicamente, si usa un diagramma per descrivere un particolare aspetto di una classe, e si omettono i metodi e gli attributi che non concorrono a definire tale comportamento. In questo libro, i diagrammi di classe sono stati disegnati secondo questa convenzione.

Ereditarietà e realizzazione

L'ereditarietà è rappresentata in UML con una freccia dalla punta triangolare, che parte dalla classe figlia e punta alla classe padre:

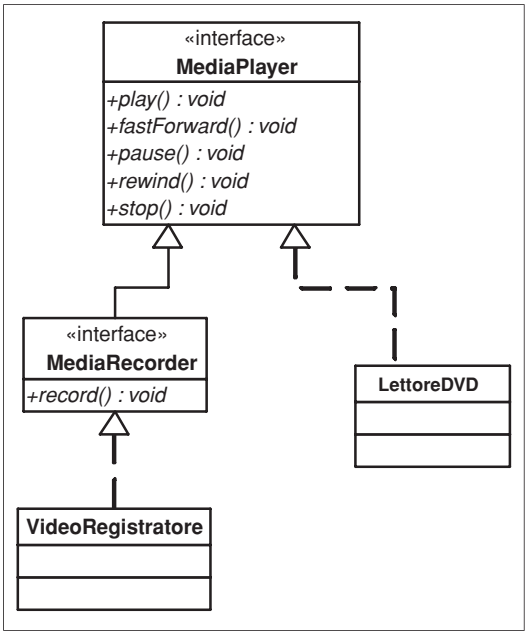
Figura D.2 – *Ereditarietà tra le classi.*



La realizzazione, equivalente all'implementazione di un'interfaccia in Java, viene rappresentata con una freccia simile a quella usata per l'ereditarietà, ma tratteggiata. Si noti che si ricorre alla realizzazione solo quando una classe implementa un'interfaccia, mentre se un'interfaccia ne estende un'altra si utilizza la normale ereditarietà.

In figura D.3 è possibile vedere un diagramma di classe contenente una relazione di ereditarietà tra interfacce (l'interfaccia MediaRecorder è figlia dell'interfaccia MediaPlayer) e due casi di realizzazione (la classe LettoreDVD realizza l'interfaccia MediaPlayer, mentre la classe VideoRegistratore realizza MediaRecorder).

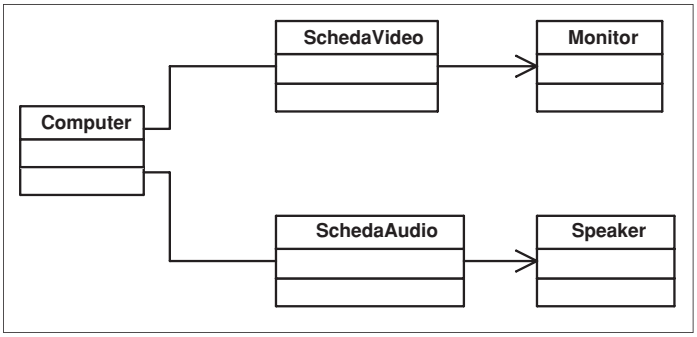
Figura D.3 – *Un diagramma che contiene sia l'ereditarietà sia la realizzazione.*



Associazione

L'associazione, rappresentata da una linea che congiunge due classi, denota una relazione di possesso. Un'associazione può essere bidirezionale o unidirezionale. Nel secondo caso, al posto di una linea semplice si utilizza una freccia. La freccia indica la direzione del flusso della comunicazione: in pratica, la classe da cui parte la freccia può chiamare i metodi di quella indicata dalla punta, ma non viceversa. L'equivalente Java dell'associazione è la presenza di un attributo in una classe, che di fatto denota il possesso di un particolare oggetto e la possibilità di invocare metodi su di esso.

Figura D.4 – *Classi unite da associazioni.*

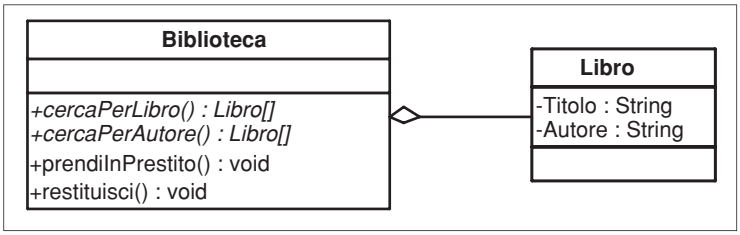


In figura D.4 è possibile osservare un insieme di classi caratterizzate da associazioni sia uni- sia bidirezionali: un computer è collegato alle schede audio e video da associazioni bi direzionali, a indicare che la comunicazione avviene in entrambe le direzioni; le due schede, invece, presentano un'associazione unidirezionale rispettivamente con gli speaker e il monitor, poiché non è permessa la comunicazione in senso inverso.

Aggregazione

Un tipo speciale di associazione è l'aggregazione, rappresentata da una linea tra due classi con un'estremità a diamante, che denota un'associazione uno a molti. In figura D.5 si può osservare una relazione uno a molti tra una biblioteca e i libri in essa contenuti.

Figura D.5 – *Un esempio di aggregazione.*

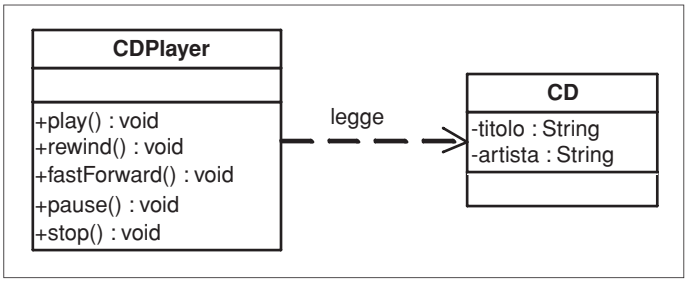


A parte la cardinalità, l'aggregazione equivale a un'associazione: nell'esempio di figura D.5 la classe Biblioteca possiede una collezione di libri e può invocare metodi su ognuno di essi. In Java, l'aggregazione corrisponde solitamente a un attributo di tipo Vector o HashTable, o più semplicemente a un array.

Dipendenza

La dipendenza è rappresentata da una freccia tratteggiata. Letteralmente, la dipendenza suggerisce che la classe puntata dalla freccia esista indipendentemente dalla classe da cui parte la freccia: in Java, questo significa che la prima può essere compilata e utilizzata anche in assenza della seconda.

Figura D.6 – *Relazione di dipendenza in UML.*



La figura D.6 presenta un esempio di relazione di dipendenza: il CD, inteso come supporto per musica e dati, esiste indipendentemente dal particolare lettore con cui lo si legge: le sue caratteristiche sono definite da un documento denominato Red Book, al quale si devono attenere i produttori di lettori CD. Si noti l'etichetta "play" che compare sopra la freccia: le etichette permettono di fornire maggiori informazioni sul tipo di relazione che sussiste tra due classi.