1 L'astrazione

Il mondo in cui viviamo è caratterizzato da una notevole complessità. Osservando ciò che ci circonda ci accorgiamo, infatti, che ogni cosa è più o meno complessa. Si pensi, per esempio, al numero di nozioni contenute in questo libro e alla difficoltà richiesta per impararle, oppure ai componenti di un computer o alle diverse parti del motore del nostro mezzo di trasporto abituale. È improbabile che un individuo, riferendosi a questi oggetti, sia in grado di comprenderli e descriverli in ogni particolare. Ben pochi automobilisti, infatti, conoscono le funzioni di tutte le componenti del motore della loro automobile.

Ancora più ardua risulta la costruzione di sistemi complessi. Immaginiamo di dover costruire un modellino d'aereo partendo dalle materie prime: plastica e ferro. Come per la realizzazione di un aeromodello si utilizzano componenti realizzate da altri, così nella descrizione della realtà si opera una semplificazione, limitandosi alla considerazione delle sole parti considerate rilevanti per la trattazione.

Se, per esempio, vogliamo descrivere le caratteristiche del motore della nostra automobile, potremmo soffermarci sul numero di cilindri (per esempio, 6) e sulla capacità di ciascun cilindro (per esempio, 500 cc), sul tipo di alimentazione (per esempio, benzina) e sul tipo di carburazione (per esempio, iniezione elettronica).

Tutte queste informazioni ci permettono di individuare un certo tipo di propulsore, ma non ci mettono sicuramente in grado di conoscerne tutte le caratteristiche. In altre parole, non potremmo mai realizzare un motore con tali caratteristiche senza informazioni aggiuntive (per esempio, come si costruisce un pistone o come si montano le fasce elastiche).

Questo processo di semplificazione della realtà complessa è detto **astrazione**.

Grazie all'astrazione prepariamo delle carte geografiche che ci permettono di raggiungere un determinato punto della città senza dover descrivere nei minimi dettagli l'aspetto e il colore delle abitazioni che si trovano in tale punto, oppure le informazioni anagrafiche sugli abitanti della zona. Una tale piantina non ci fornisce alcuna informazione storica sugli edifici o sui personaggi che vi hanno vissuto, ma questo non è lo scopo di una cartina stradale. Viceversa, consente di raggiungere i diversi punti della città in essa descritti, anche se ignoti.

Esistono diversi livelli di astrazione in base alle proprie esigenze: la descrizione del proprio scooter avrà dei livelli di astrazione diversi da quelli utilizzati dal meccanico per la riparazione dello scooter. Dovendo infatti impiegare il nostro ciclomotore, parleremo delle modalità di messa in moto, di rifornimento e di guida, mentre non tratteremo il diametro del foro della biella che compete invece al meccanico.

L'astrazione riveste un ruolo importante nella produzione del software. Un'applicazione software solitamente rappresenta un'astrazione della realtà. Si pensi a un programma di simulazione del volo. Nelle fasi che portano alla realizzazione del programma si crea un modello della realtà che si vuole riprodurre sullo schermo. Tale modello non è altro che la semplificazione delle componenti di un volo che rivestono un qualche interesse per i nostri scopi. Potremmo infatti essere interessati a riprodurre il rumore dell'aereo, oppure gli effetti provocati dal vento di coda, o, ancora, a riprodurre entrambe le caratteristiche.

Completata questa prima fase molto importante, detta di *astrazione* o **analisi del problema**, si passa alla sua implementazione attraverso la **programmazione** nel linguaggio prescelto.

2 Programmare con gli oggetti

Il computer esegue tutto ciò che viene richiesto ad esso, attraverso un linguaggio predefinito e compreso dalla macchina stessa. Il computer, infatti, svolge le funzioni richieste attraverso una sequenza di ordini (**istruzioni**), detta **programma**.
Nelle precedenti unità di apprendimento abbiamo infatti visto come sia possibile richiedere al calcolatore l'elaborazione di un calcolo anche molto complesso, semplicemente elencando i singoli passi da percorrere per giungere alla soluzione. La descrizione di ciascun passo è realizzata con un linguaggio simbolico, cioè non naturale, ma costruito opportunamente per comunicare con il computer. Nella fattispecie, ci siamo occupati del linguaggio C++.
Nei primi esempi di questo libro abbiamo usato un tipo di **programmazione procedurale**. In sostanza il programma, composto di dati e istruzioni, elenca le istruzioni che il computer deve eseguire secondo l'ordine che deve consentire di arrivare ai risultati (*output*) partendo dai dati forniti in ingresso (*input*). I diversi percorsi possibili, a seconda dei casi che si possono verificare durante l'esecuzione, sono gestiti attraverso le **strutture di controllo** (sequenza, selezione e ripetizione).
Questo stile di scrittura era di semplice realizzazione e adatto all'implementazione di piccoli programmi. Attorno al 1960, quando i produttori di software furono chiamati a scrivere programmi via via sempre più lunghi e potenti, sia per ragioni commerciali sia per le maggiori capacità dei computer, questo modo di implementare il codice si è dimostrato inefficiente in quanto difficile da modificare e da riutilizzare in altri progetti.
Queste esigenze dei programmi commerciali hanno portato alla definizione di un nuovo stile di programmazione, basato sull'organizzazione e la suddivisione dei programmi in **moduli** funzionalmente indipendenti, come abbiamo fatto negli esempi delle unità di apprendimento precedenti, utililizzando le **funzioni**.
La costruzione di programmi ordinati, basati sull'uso delle strutture di controllo e sull'organizzazione modulare del codice, si chiama **programmazione strutturata**.
In un programma ben strutturato, la scrittura della funzione principale, **main()**, non richiede la conoscenza di tutte le istruzioni contenute nelle sottofunzioni. Una volta che la sottofunzione è stata implementata correttamente, non ci si preoccupa più della sua gestione. La scrittura del programma si limita al richiamo della funzione di cui dobbiamo conoscere il numero e tipo di parametri di input previsti e il tipo restituito in output. Inoltre nulla vieta di implementare nel tempo un modulo più efficiente che svolga le stesse funzionalità, senza dover modificare il resto del programma.
Vent'anni di programmazione strutturata hanno evidenziato che i benefici apportati da questa tecnica di programmazione non sono sufficienti a garantire un abbondante riutilizzo del software implementato. A partire dal 1980 si è quindi sviluppato un nuovo stile di scrittura del codice che ne consente un reimpiego sempre maggiore. L'idea, di per sé molto semplice, rivoluziona il modo di implementare il codice inserendo un nuovo approccio, rappresentato dai dati del problema. Questo modo di programmare si chiama **programmazione orientata agli oggetti**.
La programmazione orientata agli oggetti non presuppone l'eliminazione delle tecniche precedenti, ma piuttosto le completa, aggiungendo loro una nuova dimensione.

La **programmazione orientata agli oggetti**, in breve **OOP** (*Object-Oriented Programming*) prende il mome dall'elemento su cui si basa: l'*oggetto*.

Il programma, realizzato con orientamento ad oggetti, si sviluppa attraverso le interazioni tra gli oggetti: durante l'esecuzione del programma, gli oggetti possono cambiare il loro stato e possono richiedere l'esecuzione di operazioni associate ad altri oggetti.

La programmazione orientata agli oggetti procura diversi vantaggi:

- facilità di **lettura** e di **comprensione** del codice, anche per persone diverse dall'autore;
- **manutenzione** del programma nel tempo per correzioni o miglioramenti;
- **robustezza** del programma in situazioni critiche o in operazioni che coinvolgono grandi quantità di dati;
- **riusabilità** di parti di codice o di moduli funzionali all'interno di altri programmi.

Gli **oggetti** rappresentano le entità del problema o della realtà che si vuole automatizzare con l'informatica. Un oggetto è in grado di memorizzare le informazioni che riguardano il suo stato; è anche possibile associare ad un oggetto un insieme di operazioni che esso può compiere.

In particolare, un oggetto può essere definito elencando sia le sue caratteristiche, sia il modo con cui interagisce con l'ambiente esterno, cioè i suoi comportamenti.

Gli **attributi** rappresentano gli elementi che caratterizzano l'oggetto, utili per descrivere le sue caratteristiche e definirne il suo stato.
I **metodi** rappresentano i comportamenti e le funzionalità che l'oggetto mette a disposizione.

La struttura di un oggetto è completamente descritta quando vengono elencate le caratteristiche e i comportamenti dell'oggetto.

Per esempio, un libro è un *oggetto*: le caratteristiche del libro (titolo, peso, argomento trattato, numero di pagine, ecc.) corrispondono agli *attributi* dell'oggetto, mentre le azioni che si possono esercitare sull'oggetto (aprirlo a una certa pagina, leggerlo, sfogliarlo, sottolinearlo) corrispondono ai *metodi* di cui dispone l'oggetto considerato.

3 La classe

Gli oggetti sono definiti dalle classi.

La **classe** è la descrizione astratta degli oggetti attraverso gli attributi e i metodi. Per utilizzare un oggetto occorre crearlo come esemplare della classe, cioé come **istanza** di una classe.

Nella programmazione orientata agli oggetti, dopo aver identificato l'oggetto, solitamente rappresentato da un sostantivo, occorre individuarne gli attributi, ossia le caratteristiche rilevanti, solitamente rappresentati dagli aggettivi, ed infine i metodi, cioè le azioni da applicare agli oggetti, generalmente indicate dai verbi.

Dovendo, per esempio, realizzare un programma che gestisca la clientela di un'azienda, ossia memorizzi i dati anagrafici, consentendo l'inserimento di nuovi clienti, le eventuali modifiche e cancellazioni di quelli già inseriti, notiamo che i sostantivi sono l'azienda e i clienti.

Ciascun cliente prevede delle informazioni (o *attributi*), come il nome, l'indirizzo e il numero di telefono, e delle funzionalità, quali l'acquisizione delle informazioni (*inserimento*), la *modifica* di alcune di queste e la loro *cancellazione*.

Una classe viene rappresentata con uno schema grafico detto **diagramma della classe**, che ne evidenzia il nome, gli attributi e i metodi.

Per esempio, il diagramma della classe *Cliente* è così strutturato:

Cliente
nome cognome indirizzo telefono
Inserisci Modifica Cancella

Lo schema precedente rappresenta la classe: la prima zona contiene il nome della classe, la seconda l'elenco degli *attributi* e la terza l'elenco dei *metodi*.
Gli elementi che formano la classe, attributi e metodi, si chiamano **membri** della classe. Il raggruppamento dei membri conferisce alla classe il significato di un'unità di programmazione, riutilizzabile in altri programmi: il controllo dell'esecuzione diventa più facile e la modifica degli oggetti nel tempo diventa più semplice. Questi aspetti descrivono il concetto di **incapsulamento**, che è uno dei concetti alla base della programmazione ad oggetti.

> Il termine **incapsulamento** indica la qualità degli oggetti di poter incorporare al loro interno sia gli attributi sia i metodi, cioè le caratteristiche e i comportamenti dell'oggetto.

Tutto ciò che si riferisce ad un certo oggetto è racchiuso e contenuto all'interno dell'oggetto stesso. Si crea come una capsula, una barriera concettuale, che isola l'oggetto dalle cose esterne. Si dice che gli attributi e i metodi sono incapsulati nell'oggetto. In questo modo tutte le informazioni utili che riguardano un oggetto sono ben localizzate. Questa imposizione, che obbliga a raccogliere tutto quello che riguarda una singola entità all'interno di un oggetto, è uno dei vantaggi che viene offerto dalla programmazione orientata agli oggetti.

Creando tre istanze, *cliente1*, *cliente2* e *cliente3* della classe *Cliente*, si ottengono tre oggetti:

cliente1	**cliente2**	**cliente3**
nome = Simone cognome = Bianchi indirizzo = via Europa 3 telefono= (02)345467	nome = Antonio cognome = Bruni indirizzo = piazza Trieste 5 telefono = (06)89674536	nome = Giacomo cognome = Rossi indirizzo = via Torino 10 telefono = (030)234561

Ogni classe possiede un particolare metodo predefinito, detto **costruttore**, che viene attivato quando si crea un oggetto.
Dopo aver creato l'oggetto, è possibile modificare le sue proprietà o invocare l'attivazione di un suo metodo.

Un'ulteriore fase, prima dell'implementazione del codice, è dedicata all'individuazione di eventuali generalizzazioni e, muovendosi dal generale al particolare, all'individuazione di alcune specificità dell'oggetto esaminato.

Si creano così delle classi molto generali, che possiamo chiamare **classi di base** (o **superclassi**), nelle quali vengono implementate le componenti comuni a un considerevole numero di oggetti, sfruttando le proprietà di **information hiding** (o *data hiding*).

> Il termine **information hiding** indica il mascheramento delle modalità di implementazione di un'oggetto, rendendone disponibili all'esterno solo le funzionalità.

Chi utilizza un oggetto è interessato alle caratteristiche e alle funzioni che mette a disposizione, ma non sempre è interessato a conoscerne l'implementazione, cioè il modo con cui i metodi sono realizzati.

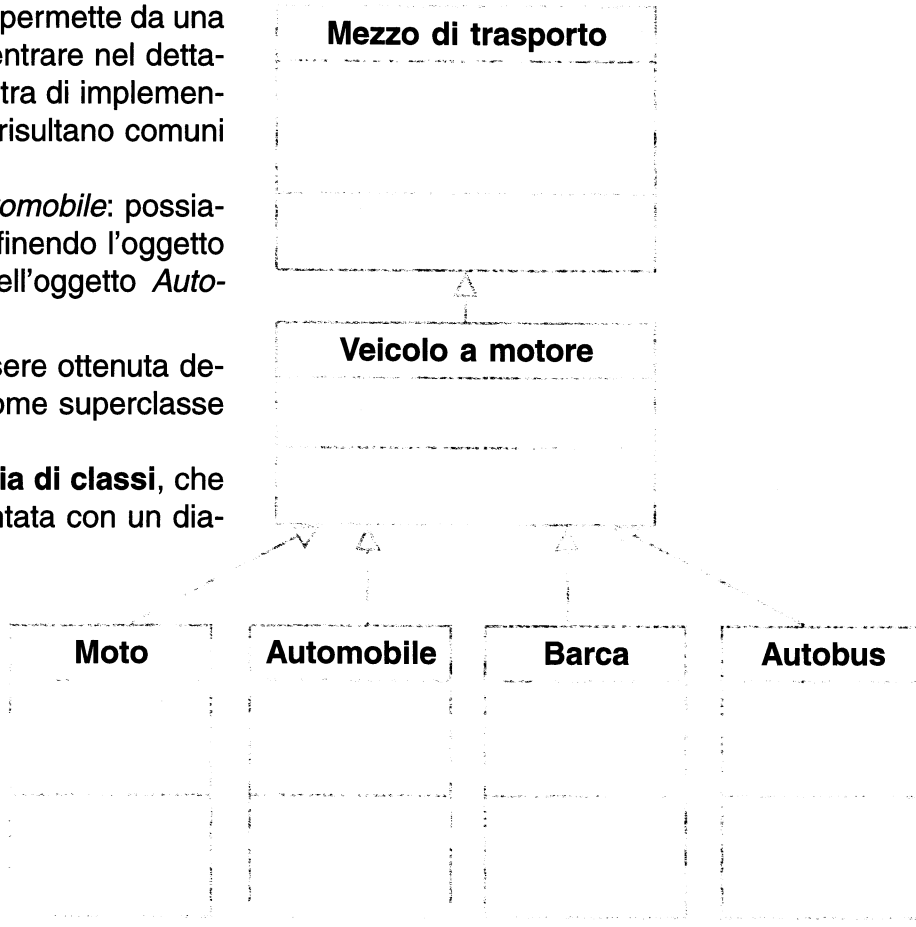
Molto spesso l'oggetto è stato costruito da un programmatore, diverso da colui che lo userà. A quest'ultimo interessa sapere invece come interagire con l'oggetto, quali sono i metodi che mette a disposizione e come poterli richiamare.

Questo modo di intendere gli oggetti induce a considerarli come delle scatole nere (*black box*). I dettagli sulle caratteristiche e la struttura dell'oggetto sono nascosti all'interno, garantendo l'*information hiding*.

Partendo dai principi dell'*information hiding* si è sviluppato un nuovo modo di concepire il software come formato da scatole, che mettono a disposizione un insieme di funzionalità. È nata un'industria di componenti software. Il contenuto di questi componenti resta nascosto, mentre se ne conosce il modo con cui operano. Per creare nuovi programmi si uniscono diverse componenti, facendole interagire nel modo desiderato: si pensi per esempio alle componenti che costituiscono le interfacce grafiche nei moderni computer (finestre, menu, barre degli strumenti, bottoni).

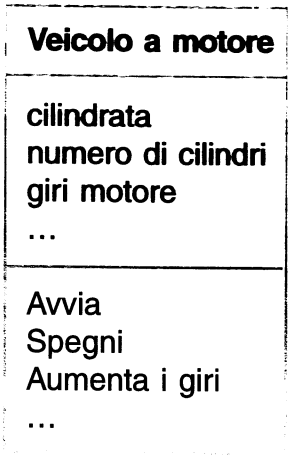
Questo modo di pensare generalizzato permette da una parte di riferirsi ad una classe, senza entrare nel dettaglio degli attributi e dei metodi, e dall'altra di implementare una sola volta parti di codice che risultano comuni a progetti diversi.
Si consideri per esempio l'oggetto *Automobile*: possiamo effettuare una generalizzazione definendo l'oggetto *Veicoli a motore* come superclasse dell'oggetto *Automobile.*

Una generalizzazione ulteriore può essere ottenuta definendo l'oggetto *Mezzo di trasporto* come superclasse dell'oggetto *Veicolo a motore*.
Si ottiene in questo modo una **gerarchia di classi**, che può essere opportunamente rappresentata con un diagramma come nella figura a destra.



Le frecce partono dalla classe di livello inferiore verso la classe di livello superiore. La freccia si legge con la frase "è un" (in inglese **is a**): per esempio, l'autobus *è un* veicolo a motore. Ogni rettangolo, diviso in tre zone, descrive la classe con nome, attributi e metodi.

Per esempio, la classe *Veicolo a motore* è rappresentata in dettaglio dallo schema della figura a destra.
Nel descrivere, per esempio, un motore possiamo fare riferimento alle sue proprietà, quali la cilindrata, il numero di cilindri e il numero di giri. Sempre in relazione all'oggetto *Veicolo a motore*, possiamo pensare che sia possibile avviare il motore, spegnerlo, aumentarne i giri, ecc.



I simboli utilizzati negli schemi grafici precedenti rispettano gli standard del linguaggio **UML** (*Unified Modeling Language*), un insieme di regole riconosciute a livello internazionale per specificare, rappresentare, progettare e realizzare software, in particolare, nel contesto della programmazione, per descrivere le classi e gli oggetti.
Dopo aver implementato una classe di base, si può passare alla creazione di una classe più specifica (o **sottoclasse**), strettamente legata alla classe che si deve trattare nel proprio problema specifico. Tale classe *deriva* dalla precedente attraverso il meccanismo dell'**ereditarietà**, che è un altro concetto fondamentale della programmazione ad oggetti.
Con riferimento al diagramma precedente, l'oggetto *Automobile* eredita le caratteristiche (attributi) e i comportamenti (metodi) dell'oggetto *Veicolo a motore*.
Di conseguenza l'oggetto *Automobile* ha in dotazione, per effetto dell'ereditarietà, tutte le proprietà che l'oggetto *Veicolo a motore* ha messo a disposizione, per esempio *cilindrata* e *Avvia*.
Oltre all'ereditarietà, una classe derivata, per poter essere implementata in modo efficace, sfrutta le possibilità offerte dal **polimorfismo**, cioè la possibilità di definire più versioni di uno stesso metodo.