

## Sommario

Cosa apprenderemo in questo modulo? .....	2
Modulo 4 - Incapsulamento .....	2
Ricapitolando !.....	3

### Cosa apprenderemo in questo modulo?

- Cos'è l'incapsulamento?
- A che serve?
- Come è genericamente implementato?
- Cosa sono i modificatori di accesso e visibilità ?

### Modulo 4 - Incapsulamento

L'incapsulamento è un concetto molto importante nella programmazione orientata agli oggetti. Il termine deriva da "incapsulare", "proteggere". L'idea è di rendere l'accesso ad una classe più ristretto, in modo da impedire la modifica non controllata allo stato di un oggetto, cosa che potrebbe generare incongruenze nel sistema. Inoltre, l'incapsulamento permette anche di proteggere il codice dall'uso di costrutti complessi, consentendo di utilizzare gli oggetti, senza preoccuparsi della loro effettiva implementazione.

Grazie a questo concetto, le caratteristiche delle classi sono separate dalla loro implementazione, in altre parole si definisce un "contratto" che ogni classe stabilisce con il mondo esterno. Il contratto indica le modalità di funzionamento di una classe (attraverso i suoi metodi, i parametri ed i risultati restituiti). Questo contratto prende il nome di interfaccia. Così sappiamo che quali sono i dati in input ad una classe e quali sono invece i dati in restituiti in output dalla stessa classe.

Per esempio, immaginate una persona che utilizza una macchina per il caffè. Esiste un "contratto", una "convenzione" su come usare la macchina, dove mettere l'acqua, dove mettere il chicco di caffè, ecc. Così, per utilizzarla basta semplicemente che l'utente conosca l'interfaccia della macchina per il caffè, senza preoccuparsi dei suoi meccanismi interni. Ora immaginate che la macchina per il caffè abbia un problema. Durante la manutenzione, il tecnico può sostituire tutte le parti interne e cambiare anche le modalità con cui verranno utilizzati gli ingredienti, ma per l'utente finale, la macchina rimarrà una macchina da caffè che è in grado di produrre caffè a partire da grani e acqua. Da notare che, sebbene l'utente non sappia come avviene l'elaborazione interna dei prodotti, egli è in grado tranquillamente di utilizzare il dispositivo. Diciamo che gli attributi della macchina da caffè e il suo comportamento sono incapsulati al suo interno, perché non possiamo cambiarli liberamente in quanto utenti non autorizzati.

Analogamente a come avviene per la macchina del caffè, qualsiasi classe incapsulata dovrebbe proteggere i propri attributi, non permettendo ad eventi esterni di modificarne i valori. Ciò può essere fatto attraverso la dichiarazione di attributi di tipo privato. In questo caso, vi è la garanzia che solo la classe stessa possa nel tempo cambiare il proprio stato (ovvero il valore dei suoi attributi) ed essere così protetta da operazioni attivate da eventi esterni (chiamate di metodi).

Considerate questo esempio: Supponiamo di avere una classe chiamata Lavatrice che simula il comportamento di una macchina lava panni. Una volta in funzione, non è più possibile modificare lo stato della macchina in qualsiasi modo, altrimenti si rischia di danneggiare il dispositivo ed impedire che il lavoro

## PROGRAMMAZIONE ORIENTATA AGLI OGGETTI - MODULO 4

venga eseguito nel modo migliore. Pertanto, per avanzare nel ciclo di lavaggio è sufficiente chiamare il metodo "prossimoPasso( )", che presumibilmente altera lo stato interno della macchina in maniera controllata proteggendo così il funzionamento della classe.

Ti starai chiedendo : Che cosa succede se ho bisogno di conoscere il valore corrente oppure ho bisogno di modificare il valore iniziale di un attributo di una classe? In questo caso, l'incapsulamento prevede che ciascun attributo abbia un paio di metodi ad esso associati, utilizzabile per **leggere** o **scrivere** il valore dell'attributo. In letteratura questi metodi sono anche noti come **GETTER** e **SETTER**, rispettivamente.

Il metodo di **lettura** (GETTER) è un metodo usato per scoprire quale sia il valore corrente di un attributo e di solito è definito come "leggi < nome attributo >" (nei linguaggi di programmazione è abitualmente denominato "*get.nome\_atributo*"). Il metodo di **scrittura** (SETTER), è a sua volta utilizzato per modificare dall'esterno il valore di un attributo attraverso un metodo di solito implementato con il nome di "scrivi < nome attributo >" "*set.nome\_atributo*").

Ci sono due concetti che sono intrinsecamente legati all'Incapsulamento Accoppiamento e Coesione :

### **Accoppiamento:**

Si riferisce al livello al quale una classe conosce ed utilizza i membri di un'altra classe, cioè, rappresenta una misura di interdipendenza tra diversi moduli (classi, oggetti). Più basso è il livello accoppiamento di una classe più ridotto sarà l'impatto di una eventuale modifica alla classe stessa.

### **Coesione:**

Si riferisce alla responsabilità della classe, ovvero ogni classe deve avere la sua responsabilità definita correttamente, in termini di attributi e metodi, che devono essere legati solo al rispetto delle proprie funzionalità. In genere, classi che hanno molte responsabilità non sono desiderabili.

In termini di solidità, un buon software deve avere una **alta coesione** ed un **basso accoppiamento**.

## **Ricapitolando !**

In questa sezione abbiamo appreso l'incapsulamento. Vedi il modulo successivo per un approfondimento del contenuto di questa sezione.