Documento a cura di Simone Remoli.

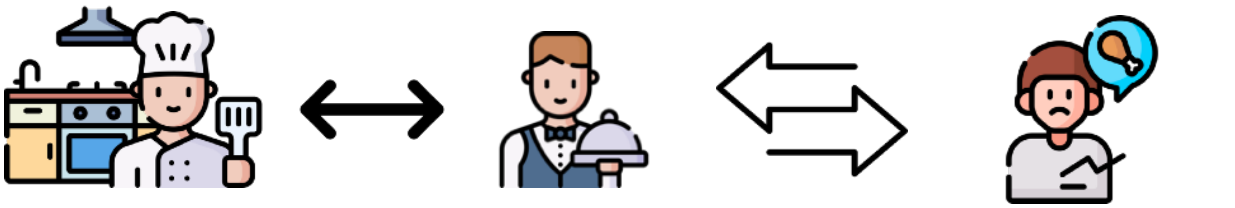
Design Pattern.

PATTERN DAO (Data Access Object)

Immagina di recarti in un ristorante con una fame incontrollabile, desideroso di gustare un delizioso piatto di **spaghetti aglio, olio e peperoncino**.
Non appena ti accomodi al tavolo, *attendi con impazienza l'arrivo del cameriere* per ordinare il tuo piatto.
Il ruolo del cameriere è fondamentale: **prende la tua ordinazione**, **la comunica allo chef in cucina** e, una volta pronto, **ti serve il piatto** **senza che tu debba preoccuparti di come sia stato preparato**.
Se il cameriere non esistesse, dovresti entrare direttamente in cucina, cercare gli ingredienti, cucinare da solo il piatto e poi tornare al tavolo per mangiarlo.
Questo processo sarebbe inefficiente e poco pratico.

Immagina che lo chef rappresenti il **database**, ossia il livello di persistenza, e che il cameriere sia il **DAO (Data Access Object)**.
Il DAO funge da intermediario tra l'applicazione e il database, gestendo l'accesso ai dati senza che il resto del sistema debba occuparsi di dettagli come connessioni, query SQL o cambiamenti nella struttura del database.

Se un giorno il ristorante cambia chef (ovvero il database viene sostituito o modificato), tu, in quanto cliente (l'applicazione), non devi preoccuparti di nulla: sarà il cameriere (DAO) a sapere come interagire con il nuovo chef, garantendo che il tuo ordine venga comunque servito correttamente.

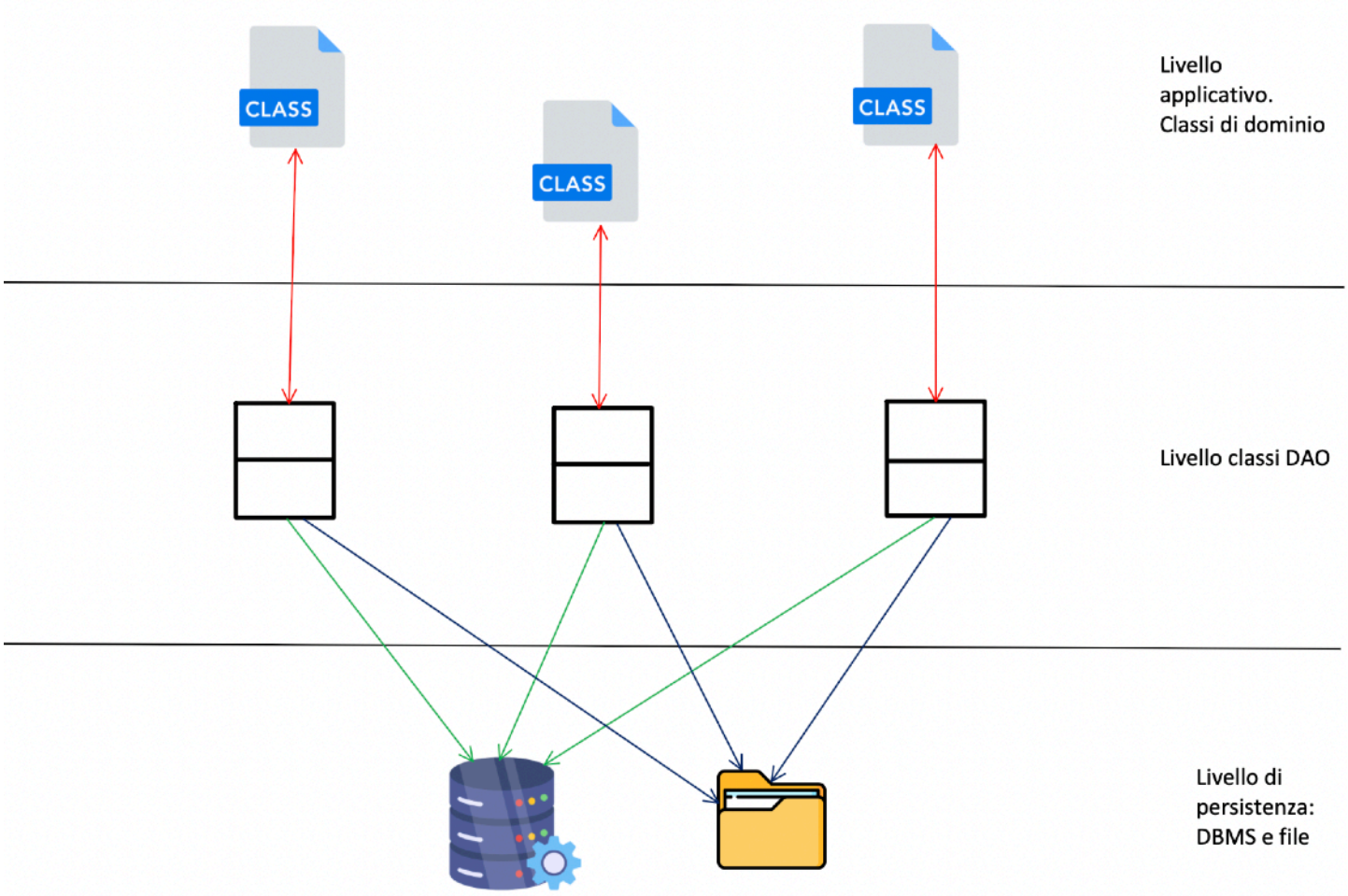


Database **DAO** **Classe di Dominio Applicativo.**

Nel mondo della progettazione dividiamo i nostri dati in 3 macrostrutture: **livello di persistenza**, **classi DAO** e **classi di dominio**.

In Java, il DAO è una classe.
Le classi DAO generano a tempo d'esecuzione oggetti di dominio dal database al dominio applicativo e viceversa.
Le DAO si comportano da ponte tra la logica applicativa e il livello di persistenza.

Lo schema finale è il seguente:



In un database relazionale ogni relazione ha il suo DAO.
In genere, ogni tabella nel database ha il suo DAO dedicato!
Un DAO non si può relazionare a più tabelle.

🟢 Vantaggi:

✅ **Organizzazione migliore** → Ogni classe gestisce solo la sua tabella.

✅ **Facilità di manutenzione** → Se una tabella cambia, modifichi solo il DAO corrispondente.

✅ **Codice più chiaro** → Sai subito dove trovare il codice che interagisce con una tabella.

A volte i DAO sono gestiti da una **DAO Factory**, che permette di ottenere il DAO giusto in base al database usato.

In questo modo, se in futuro vuoi cambiare database, puoi sostituire facilmente l'implementazione del DAO senza modificare il codice dell'applicazione!

Ogni volta che apriamo una connessione al database, usiamo risorse preziose. Se il nostro DAO non gestisce bene le connessioni, il server può rallentare o addirittura bloccarsi!

A volte, un DAO recupera **troppi** dati o **troppo pochi**.

Tipica query: *SELECT * FROM utenti;*

Se ci sono 1 milione di utenti, perché caricare tutto quando ne serve solo uno?

✅ Soluzione: Query specifiche.

Questa era una leggera panoramica sul pattern DAO.

ESEMPIO A LEZIONE DEL PROF.ING. A. PELLEGRINI.

Tralasciando i numerosi **code smells** presenti nel codice, volutamente introdotti in quanto l'obiettivo di questo esempio **non è focalizzarsi sulle convenzioni o sulla riduzione di codice ridondante**, bensì **sulla logica sottesa all'estrazione dei dati attraverso una o più classi DAO**, il seguente **progetto**, di dimensioni ridotte, si compone di sei classi:

- **Una classe principale (Main)**, responsabile dell'avvio e della gestione del flusso di esecuzione.
- **Tre classi DAO (Data Access Object)**, incaricate di interfacciarsi con il livello di persistenza, astrarre le operazioni di accesso ai dati e garantire l'indipendenza della logica applicativa rispetto ai dettagli implementativi del database.
- **Due classi di dominio**, le quali rappresentano le entità fondamentali del modello.

L'intento di questa struttura è quello di enfatizzare il ruolo del **DAO** come intermediario tra la logica applicativa e il database, evidenziando l'importanza della separazione delle responsabilità e della gestione efficiente dei dati.

Attenzione.

Si dà per assunto che il lettore abbia già **scaricato il codice** relativo alla gestione degli attori e dei film e che abbia predisposto autonomamente l'intero **stack software** necessario per l'esecuzione del progetto. Ciò include l'installazione e la configurazione del **database fornito dal Professore**, ossia **SakilaDB**, e la disponibilità di un software di gestione del database, come **DBeaver** o **MySQL Workbench**.
Il codice sorgente **non verrà mostrato** in questa sede, ma verrà illustrata la **logica di funzionamento** su cui si basa, con particolare attenzione alla gestione dell'accesso ai dati attraverso le classi **DAO (Data Access Object)** e al loro ruolo nell'astrazione del livello di persistenza.

Le classi sono le seguenti:

- Main.java;
- ActorDAO.java;
- FilmDAO.java;
- FilmActorDAO.java;
- Actor.java;
- Film.java;

I metadati nel database nelle relative tabelle sono:

- **Actor**(actor_id, first_name, last_name);
- **Film**(film_id, title, description, release_year, language_id,…);
- **Film_Actor**(actor_id, film_id);

La tabella pivot Film_Actor ci indica quale attore ha recitato in quale film. Precedentemente relazione N:N (*si dà per assunto*).

Spiegazione.

Obiettivo del codice: recuperare dal database tutti i film in cui ha lavorato un determinato attore.

Il file **Main.java** acquisisce da input **il nome e il cognome dell'attore** inseriti dall'utente. Una volta ottenuti questi dati, **delega** alla classe **ActorDAO.java** il compito di **recuperare le informazioni dell'attore** corrispondente, incapsulandole all'interno di una **collezione di oggetti di dominio** di tipo **Actor**.

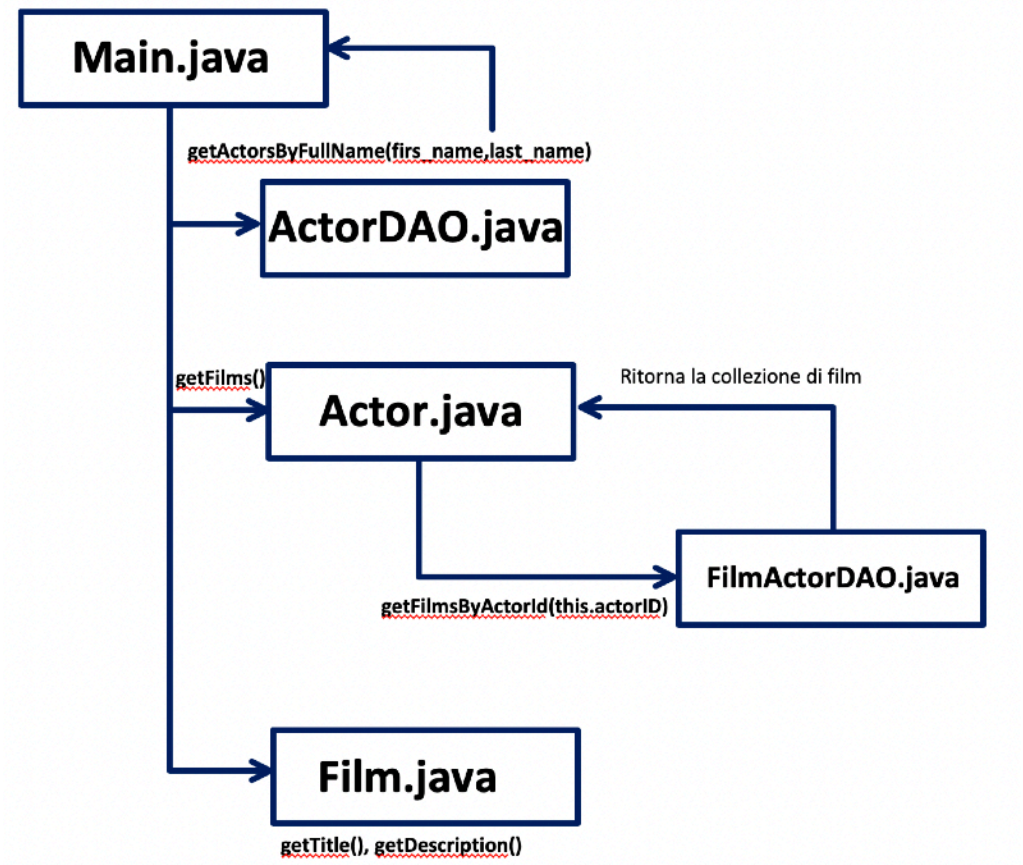
La classe **ActorDAO.java** si occupa dell'**interrogazione del database**, raccoglie tutte le informazioni relative all'attore ricercato e le organizza all'interno di una **collezione di oggetti di dominio** di tipo **Actor**, che viene poi restituita al **Main.java**.

A questo punto, **Main.java itera** sulla collezione di attori restituita, **stampando i dettagli** di ciascun attore trovato. Inoltre, per ogni attore presente nella collezione, richiama una **funzione specifica della classe Actor.java**.

All'interno di questa funzione, viene invocato un metodo della classe **FilmActorDAO.java**, il quale, sulla base dell'**id dell'attore**, effettua una query al database per ottenere **tutti i film in cui l'attore ha recitato**. La collezione dei film così ottenuta viene quindi restituita al **Main.java**.

Infine, **Main.java itera** su questa collezione di film e per ciascuno di essi **stampa il titolo e la descrizione**, offrendo all'utente una panoramica completa della filmografia dell'attore richiesto.

Questo approccio garantisce una netta **separazione delle responsabilità** tra i diversi componenti del sistema, con il DAO che si occupa esclusivamente dell'**accesso ai dati**, mentre la logica applicativa si concentra sull'elaborazione e presentazione delle informazioni all'utente.



Questo è tutto sul Pattern DAO.