**“Ingegneria del Software”**

**2019-2020**

**Docente: Prof. Angelo Furfaro**

**Gestore di Libreria Personale**

|  |  |
| --- | --- |
| **Data** | 18/05/2025 |
| **Documento** | Documento Finale – D3 |

|  |  |  |
| --- | --- | --- |
| **Team Members** | | |
| **Nome e Cognome** | **Matricola** | **E-mail address** |
| **Giuseppe Zangara** | **193620** | **znggpp98r10m208u@studenti.unical.it** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Sommario

**Non è stata trovata alcuna voce d'indice.**

## List of Challenging/Risky Requirements or Tasks

|  |  |  |  |
| --- | --- | --- | --- |
| **Challenging Task** | **Date the task is identified** | **Date the challenge is resolved** | **Explanation on how the challenge has been managed** |
| **Implementazione del sistema di Undo/Redo** | **15/05/2025** | **15/05/2025** | **Non è stato facile gestire l'annullamento delle operazioni. Ho risolto usando il Command Pattern, dove ogni azione (aggiunta, modifica, eliminazione) è incapsulata in una classe specifica. Ho creato due stack per undo e redo, gestiti da un CommandManager.** |
| **Salvataggio/Caricamento in JSON e CSV** | **14/05/2025** | **16/05/2025** | **Volevo dare all'utente la possibilità di usare entrambi i formati. Per farlo, ho creato un’interfaccia LibroDAO e due implementazioni (JsonLibroDAO, CsvLibroDAO) all’interno di un DAO Pattern.**  **Ogni metodo di lettura e scrittura include validazione del formato, gestione degli errori I/O.**  **Realizzati test di integrazione per verificare la correttezza dei flussi di import/export, e validazione dati inseriti.** |
| **Sistema di validazione dati robusto** | **14/05/2025** | **17/05/2025** | **La validazione dei campi dati in input viene effettuata direttamente dal costruttore della classe Libro che applica controlli su tutti i campi obbligatori (titolo, autore, genere), verifica il formato ISBN e il range di valutazione, nonché dello stato di lettura del libro.**  **Gli errori di validazione vengono raccolti in eccezioni mirate e tradotti in messaggi chiari all’utente.**  **Inseriti casi di test per la verifica dei dati inseriti da input e file.** |

A. Stato dell’Arte

Nel contesto attuale, la gestione di raccolte personali di libri si affida prevalentemente a due tipologie di soluzioni:

1. **Applicazioni Web e Mobile**  
   Numerose piattaforme online consentono di catalogare e consultare la propria libreria da qualsiasi dispositivo collegato a Internet. Esempi noti sono StoryGraph, MyLibrary, LibraryThing e Google Books. Questi servizi offrono funzioni avanzate di ricerca, recensioni comunitarie, suggerimenti di lettura basati sui gusti personali e integrazioni con social network. Tuttavia, richiedono sempre un account esterno e una connessione attiva, limitando l’uso in condizioni offline o per chi preferisce un controllo totale sui propri dati.
2. **Software Desktop e Soluzioni Stand-alone**  
   Sul fronte delle applicazioni installabili, esistono programmi come Calibre (originariamente pensato per la gestione di e-book) e vari database personali creati con documenti excel/csv o Access. Questi strumenti garantiscono il pieno possesso dei dati, funzionano indipendentemente dalla rete e possono essere personalizzati tramite plugin. D’altra parte, spesso risultano complessi da configurare e mantenere, soprattutto per utenti non esperti di database o scripting.
3. **Soluzioni “fai-da-te”**  
   Molti appassionati realizzano piccoli gestionali utilizzando linguaggi di scripting (Python, JavaScript, o Java come nel caso attuale) o piattaforme low-code. Si tratta di progetti su misura, flessibili ma solitamente privi di interfaccia grafica intuitiva e privi di funzionalità avanzate quali undo/redo, caricamenti dataset esterni o filtri complessi.

**Limiti e opportunità individuati**

* **Portabilità e privacy dei dati**: Le app web espongono le proprie librerie a servizi di terze parti; le soluzioni locali mantengono i dati sul dispositivo ma spesso non offrono formati facilmente scambiabili e accessibili direttamente senza l’utilizzo del tool proprietario.
* **Usabilità e flessibilità**: Spesso manca un’interfaccia che sia al contempo intuitiva e completa, capace di supportare ricerche testuali, filtri multipli, ordinamenti personalizzati e operazioni di undo/redo.
* **Estendibilità**: La maggior parte dei software non adotta pattern di progettazione standard (MVC, DAO, Command, Strategy), rendendo difficile l’evoluzione del codice e l’integrazione di nuove funzionalità.

**Obiettivo del progetto**

Creare un’applicazione desktop Java che combini i punti di forza delle soluzioni esistenti:

* **Interfaccia grafica moderna** e semplice, basata su Swing.
* **Funzionalità offline** con salvataggio e caricamento dei dati in formati standard (JSON e CSV) per garantire flessibilità e possibilità di consultare, modifica, aggiungere, rimuovere i dati direttamente sul file.
* **Ricerca, filtri e ordinamenti avanzati**, supportati da pattern Strategy per garantire modularità.
* **Meccanismo di undo/redo** robusto, implementato con Command pattern e gestore centralizzato.
* **Architettura manutenibile**, grazie all’adozione di MVC per separare chiaramente presentazione, logica di controllo e modelli di dominio.

Con questo approccio si intende offrire agli utenti una soluzione completa, sicura e facilmente estendibile, rispondendo alle esigenze sia di chi ama la semplicità “out-of-the-box”, sia di chi desidera personalizzare e arricchire il proprio gestionale nel tempo.

B. Raffinamento dei Requisiti

***B.1 Servizi (con prioritizzazione)***

Di seguito descriviamo in modo più ricco i servizi che l’applicativo realizzato offre, assegnando a ognuno un **ID** univoco, il livello di **importanza** per l’utente e la **complessità** di realizzazione. Ogni servizio è pensato senza riferimenti a tecnologie specifiche; l’obiettivo è descrivere in modo completo le funzionalità e i vincoli di ciascuno.

| **ID** | **Servizio** | **Descrizione** | **Importanza** | **Complessità** |
| --- | --- | --- | --- | --- |
| S1 | Aggiunta di un nuovo libro | L’utente apre un dialogo, compila titolo, autore, ISBN (con o senza trattini), genere, valutazione (0–5 stelle, dove 0 è definito dalla stringa “Da valutare”) e stato di lettura. Il sistema valida ogni campo (titolo/autore/ISBN/genere obbligatori; ISBN univoco; rating 0–5, stato della lettura), prima di salvare. Il sistema aggiorna immediatamente la tabella e rende l’azione annullabile (undo). | Alta | Media |
| S2 | Modifica di un libro esistente | L’utente seleziona un libro, modifica uno o più campi; il sistema mantiene l’ISBN originale (imposto un vincolo che consente di non modificarlo) e controlla nuovamente la validità dei dati. Alla conferma, aggiorna la libreria. Il sistema aggiorna immediatamente la tabella e rende l’azione annullabile (undo). | Alta | Media |
| S3 | Eliminazione di un libro | L’utente sceglie un libro e viene chiesto di confermare l’operazione. In caso di eliminazione, il sistema aggiorna immediatamente la tabella e rende l’azione annullabile (undo). | Alta | Bassa |
| S4 | Visualizzazione dell’elenco libri | Mostra in tabella l’intera libreria, con colonne ordinabili e aggiornamento in tempo reale dopo ogni modifica, ricerca, filtro o undo/redo. | Alta | Bassa |
| S5 | Ricerca di libri | Ricerca parziale, case-insensitive, per titolo, autore o ISBN; risultati combinabili con filtri. | Media | Media |
| S6 | Applicazione di filtri multipli | Filtri dinamici per genere o autore (liste generate dai dati), per stato di lettura e per valutazione; possibilità di ripristinare tutti i filtri con un solo click. | Media | Bassa |
| S7 | Ordinamento personalizzabile | Ordinamento in tempo reale sui campi titolo, autore (A–Z/Z–A) e valutazione (1–5/5–1), mantenuto durante ricerca e filtri. Estendibile via Strategy Pattern. | Media | Bassa |
| S8 | Salvataggio della libreria | Esporta l’intero archivio in file JSON o CSV. | Bassa | Media |
| S9 | Caricamento della libreria | Importa da JSON o CSV, controllando struttura e contenuto (campo mancanti, formato numerico, validità dei valori). Gestisce errori I/O e segnala all’utente eventuali problemi. | Media | Media |
| S10 | Pulizia completa della libreria | Rimuove tutti i libri con un solo comando, dopo aver chiesto conferma esplicita; aggiornamento immediato della vista; operazione non annullabile. | Bassa | Bassa |
| S11 | Undo/Redo delle operazioni | Permette di annullare o ripetere l’ultima e le ultime operazioni di aggiunta, modifica o eliminazione, gestendo correttamente due stack separati. | Alta | Alta |

***B.2 Requisiti non Funzionali***

Per offrire un’esperienza d’uso solida e soddisfacente, sono stati definiti una serie di requisiti non funzionali che ne guidano lo sviluppo e ne assicurano la qualità.

Prima di tutto, l’**usabilità** è un aspetto fondamentale. L’interfaccia, realizzata con Swing, deve risultare immediatamente familiare anche per chi si avvicina per la prima volta al programma. Ogni azione, dall’inserimento di un nuovo libro all’eliminazione definitiva, produce un feedback istantaneo, sia visivo (pop-up, evidenziazioni) sia testuale (messaggi chiari). I tool-tips accompagnano l’utente durante la compilazione dei campi, mentre i dialog di conferma mettono in guardia prima di eliminare o cancellare in blocco la libreria.

In secondo luogo, l’**affidabilità** del sistema passa attraverso controlli rigorosi: ogni input viene validato (campi obbligatori, formato ISBN, intervallo della valutazione), le eccezioni legate all’I/O vengono catturate e gestite in modo centralizzato, e in caso di errori critici si esegue un rollback per mantenere intatti i dati. Nessuna operazione dovrebbe mai portare a un crash o a un salvataggio inconsistente.

La **manutenibilità** si ottiene con un’architettura modulare: il pattern MVC separa vista, logica e dati, mentre Command, Strategy e DAO isolano rispettivamente le funzionalità di undo/redo, gli algoritmi di ordinamento e l’accesso al file system. Ogni modulo è descritto con JavaDoc e coperto da test unitari, in modo che futuri sviluppatori possano intervenire con semplicità e sicurezza.

Anche le **prestazioni** sono state ottimizzate: l’intera libreria viene caricata in memoria al lancio, permettendo ricerche, filtri e ordinamenti in pochi millisecondi anche con qualche migliaio di record. Le operazioni di salvataggio e caricamento, in JSON o CSV, sfruttano meccanismi di buffering per ridurre al minimo i tempi di attesa.

Per quanto riguarda la **portabilità**, l’applicazione è stata sviluppata in Java puro, senza dipendenze esterne, garantendo la stessa esperienza su Windows, macOS e Linux. Ogni utente deve solo disporre di una Java Runtime aggiornata e di un normale file system per leggere o scrivere i formati di persistenza.

Infine, l’**estensibilità** e la **robustezza** chiudono il cerchio: grazie alle interfacce ben definite è facile aggiungere nuovi formati (ad esempio DBMS o XML), nuovi criteri di ordinamento o altro (inserimento di un nuovo modulo). Allo stesso tempo, in caso di condizioni eccezionali (come file corrotti o errori di parsing) il sistema è in grado di recuperare e mantenere i dati sani, riducendo al minimo eventuali problematiche per l’utente.

***B.3 Scenari d’uso dettagliati***

Di seguito presentiamo, per ciascun servizio (SID), uno scenario d’uso completo: contesto, flusso principale e principali varianti se previste.

**Scenario S1 – Aggiunta di un Nuovo Libro**

**Contesto**  
L’utente desidera catalogare un nuovo volume nella propria libreria personale.  
 **Flusso Principale**

1. Dalla barra dei comandi, l’utente clicca **“Aggiungi Libro”**.
2. Compare un dialog di inserimento: l’utente compila
   * Titolo (campo obbligatorio)
   * Autore (campo obbligatorio)
   * ISBN (campo obbligatorio, formato numerico con trattini)
   * Genere (campo obbligatorio)
   * Valutazione (campo non obbligatorio , da 0 a 5 stelle, dove 0 stato di default)
   * Stato di lettura (campo non obbligatorio , Da leggere / In lettura / Letto, dove da leggere è di default)
3. L’utente clicca **“Conferma”**: il sistema
   * Verifica che tutti i campi obbligatori siano compilati correttamente (Non null, non vuoti, non ci siano solo spazi, per isbn siano solo numeri e al massimo possono essere presenti dei -);
   * Controlla che l’ISBN non sia già presente;
   * Convalida che la valutazione sia compresa fra 0 e 5;
4. Se la validazione ha esito positivo, il libro viene salvato e appare una notifica di conferma; la tabella si aggiorna immediatamente;
5. Il comando viene registrato nello stack per poter essere annullato;

**Varianti**

* **Dati mancanti o non validi**: il dialog rimane aperto e compare un messaggio che indica quali campi correggere, o compilare.
* **ISBN duplicato**: il sistema segnala “ISBN già presente” e richiede un ISBN diverso.

**Scenario S2 – Modifica di un Libro Esistente**

**Contesto**  
L’utente vuole correggere o aggiornare i dati di un libro già catalogato.  
  
**Flusso Principale**

1. Nella tabella, l’utente seleziona il libro da modificare e clicca **“Modifica Libro”**;
2. Si apre il medesimo dialog di inserimento, pre-popolato con i dati attuali;
3. L’utente cambia uno o più campi (es. aggiorna la valutazione);
4. Alla conferma, il sistema:
   * Verifica che i campi obbligatori non siano vuoti;
   * Mantiene l’ISBN originale;
5. Se tutto è valido, aggiorna il record, aggiorna la tabella e invia il comando allo stack di undo.

**Varianti**

* **Nessun libro selezionato**: comparirà un messaggio “Seleziona un libro da modificare”.
* **Validazione fallita**: dialog rimane aperto con indicazione degli errori.

**Scenario S3 – Eliminazione di un Libro**

**Contesto**  
L’utente intende rimuovere definitivamente un volume non più necessario.  
  
**Flusso Principale**

1. L’utente seleziona il libro e clicca **“Elimina Libro”**;
2. Appare un dialog di conferma: “Sei sicuro di voler eliminare questo libro?” con i dettagli del titolo;
3. Alla conferma, il sistema elimina il libro, aggiorna subito la tabella e registra il comando per eventuale undo;

**Varianti**

* **Nessun libro selezionato**: compare “Seleziona un libro da eliminare”;
* **Fallimento inatteso**: messaggio “Impossibile eliminare il libro selezionato”;

**Scenario S4 – Visualizzazione dell’Elenco Libri**

**Contesto**  
L’utente vuole avere subito un quadro completo dei libri sui quali sta lavorando.  
  
**Flusso Principale**

1. All’avvio dell’applicazione, si apre una nuova tabella vuota, dove l’utente deciderà se importare un file sul quale ha lavorato in precedenza o iniziare da zero;
2. L’utente può cliccare su ogni riga della tabella per selezionare il libro di interesse per usare successivamente i pulsanti azione di modifica/eliminazione. Non è però possibile operare direttamente sulla tabella;
3. Ogni volta che si aggiunge, modifica o elimina un libro (o si esegue undo/redo), la vista si aggiorna automaticamente;

**Varianti**

* Nessuna variante: servizio sempre disponibile, con aggiornamenti in tempo reale;

**Scenario S5 – Ricerca di Libri**

**Contesto**  
L’utente cerca un insieme ristretto di titoli partendo da un criterio testuale.  
  
**Flusso Principale**

1. Nella casella di ricerca, l’utente digita una parola chiave (titolo/autore/ISBN);
2. Il sistema esegue una ricerca **partial match**, case-insensitive, e restituisce i risultati nella tabella;
3. L’utente può subito applicare filtri aggiuntivi (genere, valutazione, stato);

**Varianti**

* **Nessun risultato**: la tabella rimane vuota;

**Scenario S6 – Applicazione di Filtri Multipli**

**Contesto**  
Dopo una ricerca o direttamente dalla lista completa, l’utente vuole approfondire la selezione.  
  
**Flusso Principale**

1. Dalla barra laterale, l’utente spunta uno o più filtri:
   * Genere (lista basata sui generi già presenti)
   * Autore (lista basata sui generi già presenti)
   * Stato di lettura
   * Valutazione
2. Ad ogni click sul filtro di interesse selezionato la tabella si restringe ai soli libri che soddisfano tutte le condizioni;
3. Un pulsante **“Reset filtri”** ripristina l’intera collezioni di libri con un solo click;

**Varianti**

* Filtri sovrapposti non restituiscono risultati, viene mostrata la tabella vuota;

**Scenario S7 – Ordinamento Personalizzabile**

**Contesto**  
L’utente desidera vedere i libri ordinati secondo un criterio preciso.  
  
**Flusso Principale**

1. L’utente seleziona il tipo di ordinamento dal menu a tendina:
   * Titolo A–Z o Z–A
   * Autore A–Z o Z–A
   * Valutazione “Da valutare” – 5 o 5 – “Da valutare”
2. Subito la tabella si riordina senza ricaricare dati i dati dal file, le operazioni vengono eseguite in memoria;
3. L’ordinamento resta attivo anche se applico filtri o ricerche successive;

**Varianti**

* Nessuna variante. Se l’operazione dovesse andare in errore, viene restituito l’ordinamento predefinito;

**Scenario S8 – Salvataggio della Libreria**

**Contesto**  
L’utente vuole esportare i propri dati per condividerli o per backup.  
  
**Flusso Principale**

1. Clic su “**Salva JSON”** o  **“Salva CSV”**;
2. Dialog di swing che permette di selezionare cartella e scegliere un nome per il file;
3. Alla fine, un messaggio pop-up conferma “Salvataggio avvenuto” specificando il percorso e il titolo del file;

**Varianti**

* **I/O Exception**: Comparsa messaggio pop-up dove vengono specificate le cause dell’errore, es. “Errore di scrittura: controlla percorso e permessi”;

**Scenario S9 – Caricamento della Libreria**

**Contesto**  
L’utente vuole recuperare o importare dati da un backup.  
  
**Flusso Principale**

1. Clic su **“Carica JSON”** o **“Carica CSV”**;
2. Il sistema verifica l’esistenza del file e ne legge il contenuto;
3. Convalida la struttura (campi obbligatori, formati corretti) e i valori (ISBN, valutazione, stato);
4. Se tutto passa, popola la tabella; altrimenti, annulla l’import e mostra l’errore;

**Varianti**

* **File mancante o illeggibile**: Messaggio di errore dove ne espone le motivazioni;
* **Formati non validi**: Messaggio di errore dove elenca le righe o i campi incriminati;

**Scenario S10 – Pulizia Completa della Libreria**

**Contesto**  
L’utente desidera eliminare in blocco tutti i libri (ad esempio per ripartire da zero).  
  
**Flusso Principale**

1. Clic su **“Pulisci Libreria”**.
2. Appare dialog “Sei sicuro?”; l’utente conferma.
3. Il sistema svuota la tabella e cancella tutti i dati in memoria (l’azione non è annullabile).

**Varianti**

* **Nessuna variante.** Comando non reversibile;

**Scenario S11 – Undo/Redo delle Operazioni**

**Contesto**  
Dopo una serie di cambiamenti (aggiunte, modifiche, eliminazioni), l’utente vuole annullare o ripetere un’azione.  
  
**Flusso Principale – Annulla**

1. Clic su **“Undo”**.
2. Il sistema verifica che lo stack di undo non sia vuoto; se vuoto, disabilita il pulsante e segnala “Nessuna operazione da annullare”.
3. Se disponibile, recupera l’ultimo comando e lo esegue in modalità “undo”.
4. Aggiorna la tabella e il pulsante “Redo”.

**Flusso Principale – Ripristina**

1. Clic su **“Redo”**.
2. Verifica che lo stack di redo non sia vuoto; altrimenti disabilita e segnala.
3. Esegue il comando in modalità “redo”, ripristinando lo stato precedente.
4. Aggiorna la vista e lo stato dei pulsanti Undo/Redo.

***B.4 Excluded Requirements***

Nella progettazione di questa applicazione ho preferito non includere alcune funzionalità che, pur utili in contesti diversi, avrebbero appesantito il sistema e distolto l’attenzione dalle caratteristiche davvero centrali. Ad esempio, non è stata prevista alcuna forma di autenticazione o gestione utenti: essendo un programma pensato per un’unica utenza locale, la protezione dell’accesso si affida ai meccanismi del sistema operativo, senza un login interno. Analogamente, non è stato nemmeno introdotto un meccanismo di backup automatico: il salvataggio dei dati rimane a scelta e controllo dell’utente, che potrà esportare la libreria in JSON o CSV quando ritiene necessario, mantenendo così il pieno controllo sulle proprie versioni in locale.

Allo stesso modo, l’applicazione non si connette a servizi cloud né integra API esterne per il recupero automatico di copertine o metadati, per evitare di appesantire troppo l’applicazione non è stato nemmeno inserita una copertina da usare in locale. Tutti i dati vengono inseriti manualmente: questa scelta garantisce la massima privacy e l’indipendenza da infrastrutture terze, oltre a semplificare notevolmente la logica del programma. Non è stata neppure implementata la gestione dei prestiti (se dato a un altro utente) o anche il luogo dove si trova il libro, perché il focus resta sulla catalogazione personale, evitando di aggiungere complessità al modello dati e all’interfaccia.

Infine, non sono presenti notifiche push, promemoria o supporto multilingua. L’interazione avviene solo su richiesta esplicita dell’utente, senza messaggi automatici in background, e l’interfaccia è disponibile esclusivamente in italiano. In questo modo manteniamo un’architettura leggera, facilmente manutenibile e pronta a crescere in futuro con eventuali nuove estensioni.

***B.5 Assunzioni***

Per la realizzazione di questo progetto sono state adottate alcune ipotesi di base che hanno guidato le scelte di design e implementazione.

Innanzi tutto, il sistema è pensato come applicazione **desktop mono-utente**: non è prevista alcuna forma di autenticazione interna o gestione di profili multipli. L’intero archivio di libri risiede in locale e le operazioni di accesso e protezione dei file vengono affidate al sistema operativo. Questo approccio semplifica l’interfaccia e riduce i requisiti infrastrutturali, rendendo il programma immediatamente utilizzabile su qualunque macchina con Java installato.

Per la **persistenza dei dati**, ho scelto di supportare soltanto i formati standard JSON e CSV. Nel CSV ogni libro occupa una riga con sei campi separati da virgole, mentre in JSON l’intera libreria viene serializzata come array di oggetti. L’ISBN è trattato come identificatore univoco stringa (con semplice validazione di presenza di numeri e trattini, senza alcun controllo di checksum), e il backup rimane un’operazione manuale: l’utente decide quando esportare o importare la propria libreria.

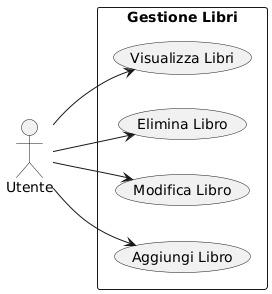
Dal punto di vista delle **performance**, il sistema è stato concentrato su librerie di dimensione contenuta (la capacità è definita dalle prestazioni hardware del sistema sul quale viene eseguita quest’ultima), tutte caricate in memoria in avvio. Le operazioni di ricerca, filtro e ordinamento sono ottimizzate per questo tipo di dataset e garantiscono risposte nell’ordine dei millisecondi.

Sul piano **architetturale**, ho adottato il pattern MVC per separare con chiarezza la presentazione (Swing), la logica di controllo e il modello dati. Il cuore della gestione dei libri risiede in un singleton, mentre undo/redo e ordinamenti sono realizzati rispettivamente con i pattern Command e Strategy. Grazie a interfacce ben definite per i DAO e per le strategie di ordinamento, il sistema è facilmente estendibile per aggiungere in futuro nuovi formati di persistenza o criteri di ordinamento.

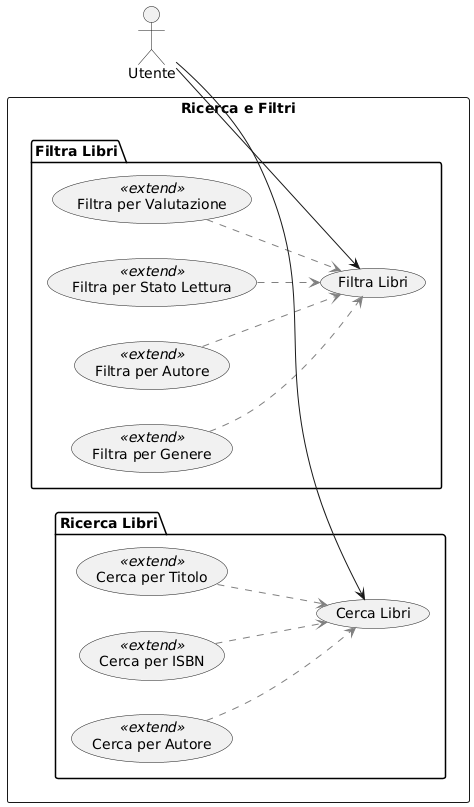
Infine, l’**interfaccia utente** è volutamente minimale e in italiano, con feedback immediato per ogni operazione (popup, tool-tips e dialog di conferma). Non è stato previsto localizzazioni multilingua né notifiche in background, in linea con l’obiettivo di mantenere l’applicazione leggera, intuitiva e adatta a un contesto semplice e user-friendly.

***B.6 Use Case Diagrams***

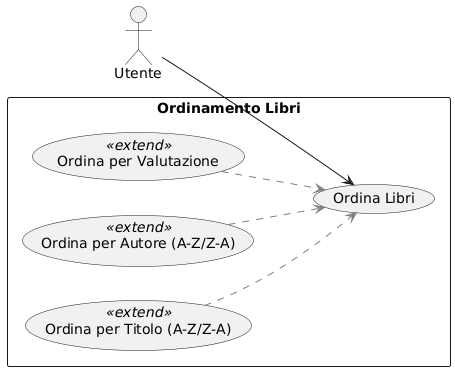
1. Aggiorna, modifica, elimina, visualizza Libro



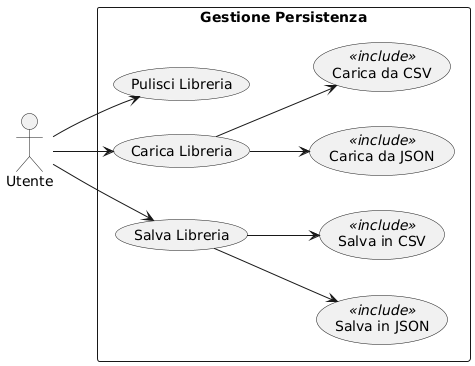
2. Ricerca e filtri



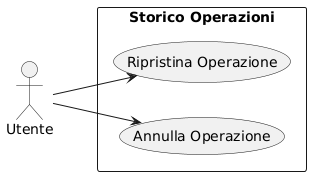
3. Ordinamento



4. Gestione persistenza

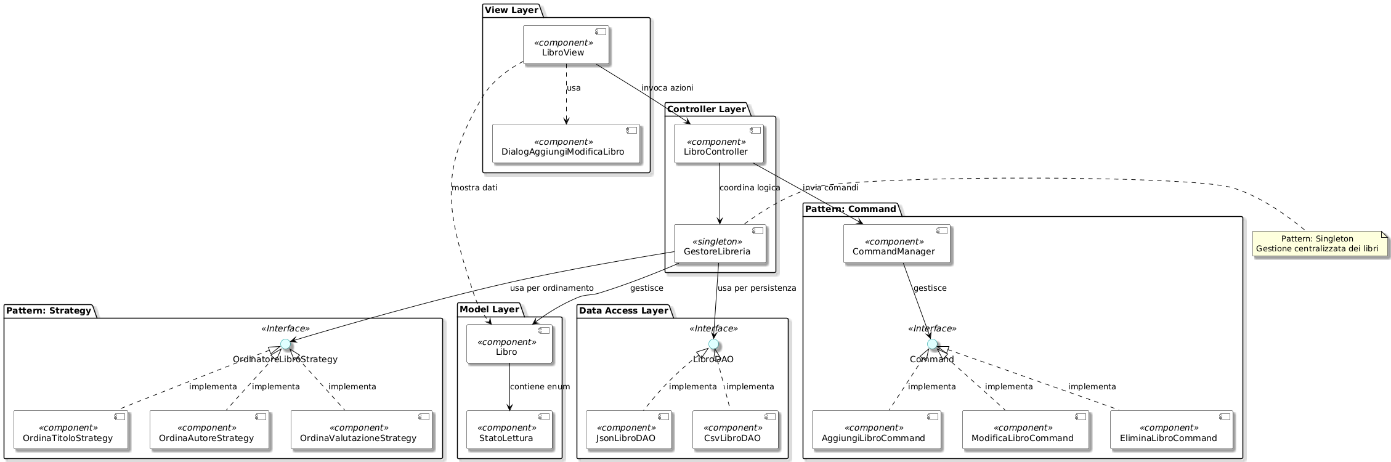


5. Storico operazioni



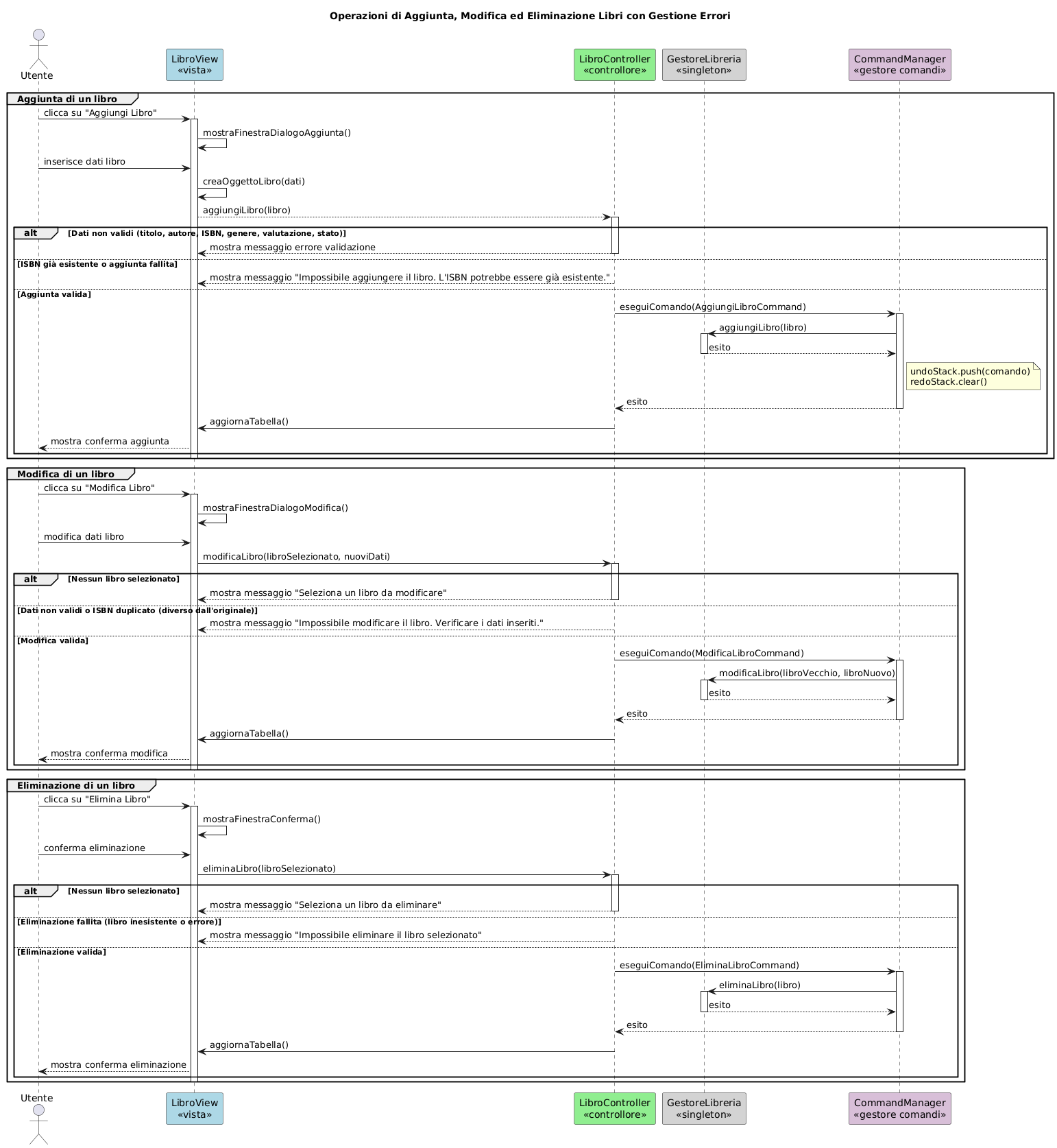
C. Architettura Software

***C.1 The static view of the system: Component Diagram***

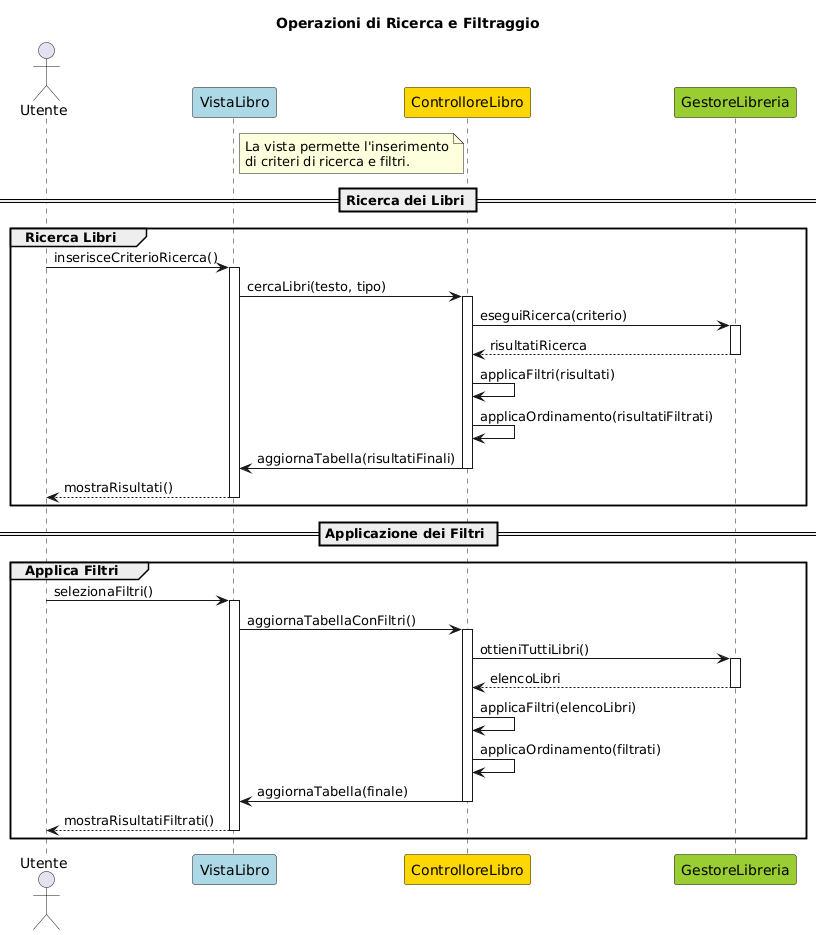


***C.2 The dynamic view of the software architecture: Sequence Diagram***

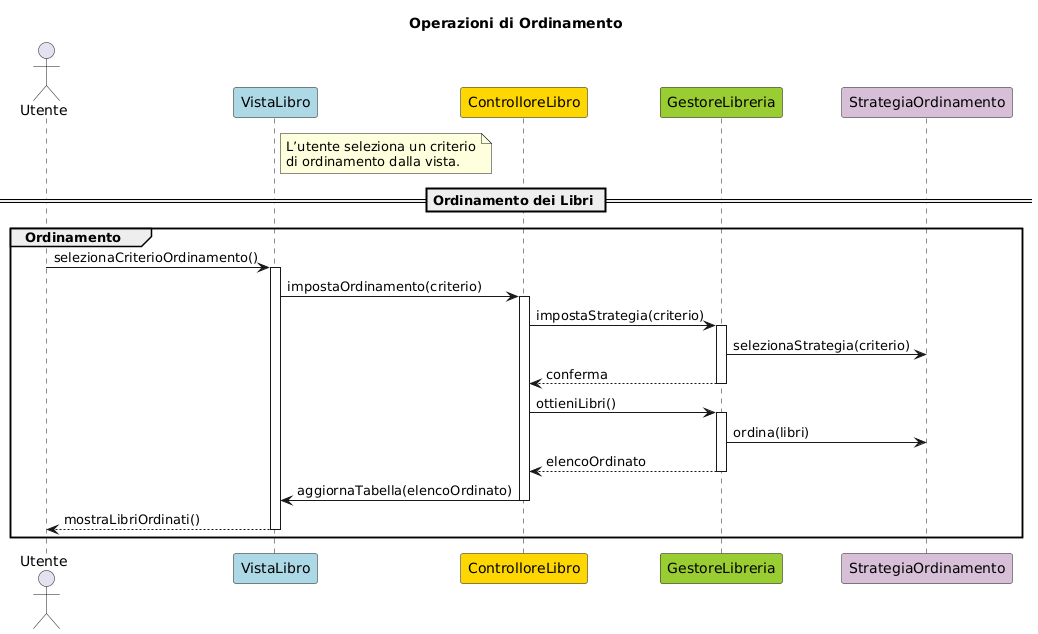
1. Operazioni di aggiunta, modifica ed eliminazione libri con gestione errori



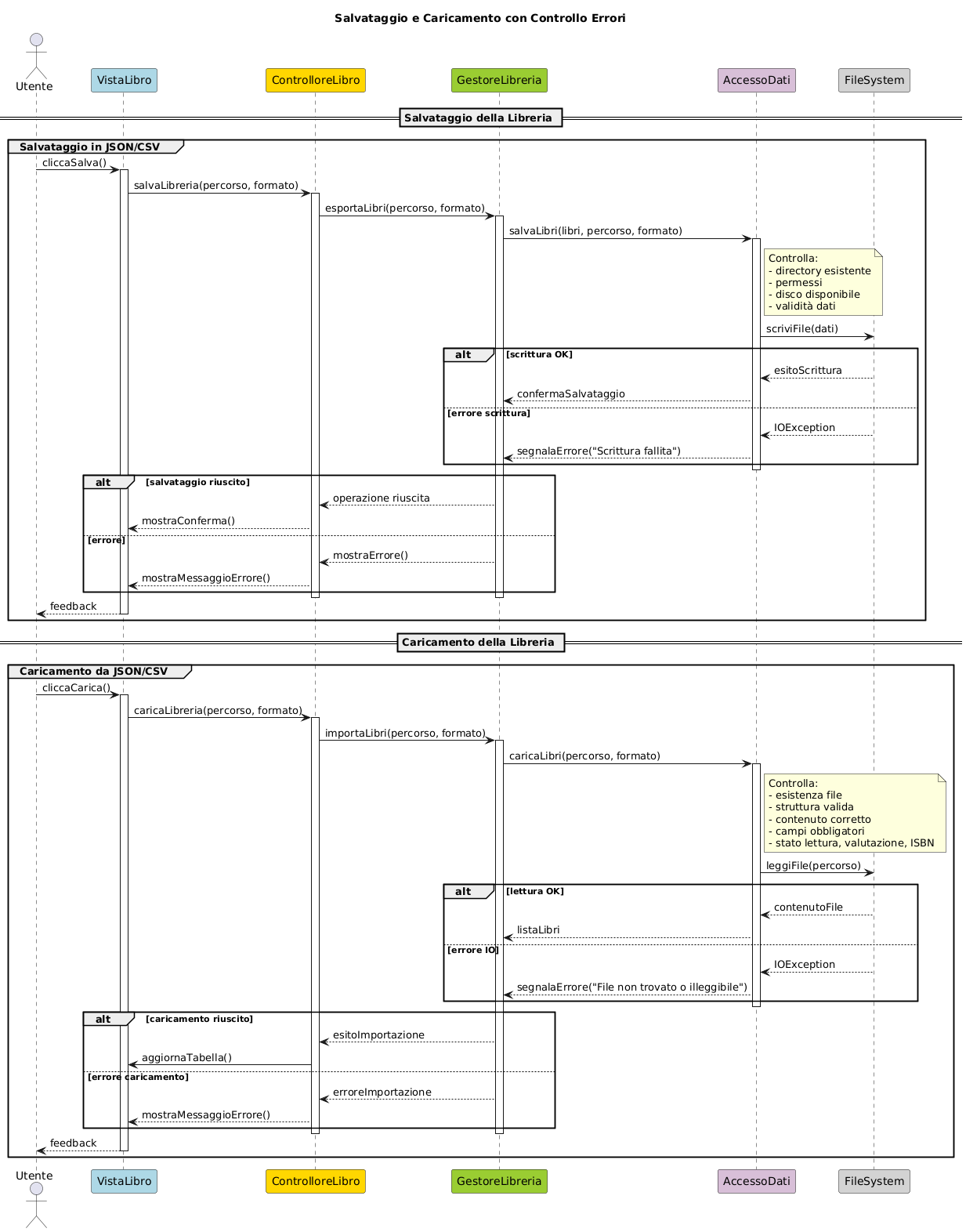
2. Operazioni di ricerca e filtraggio



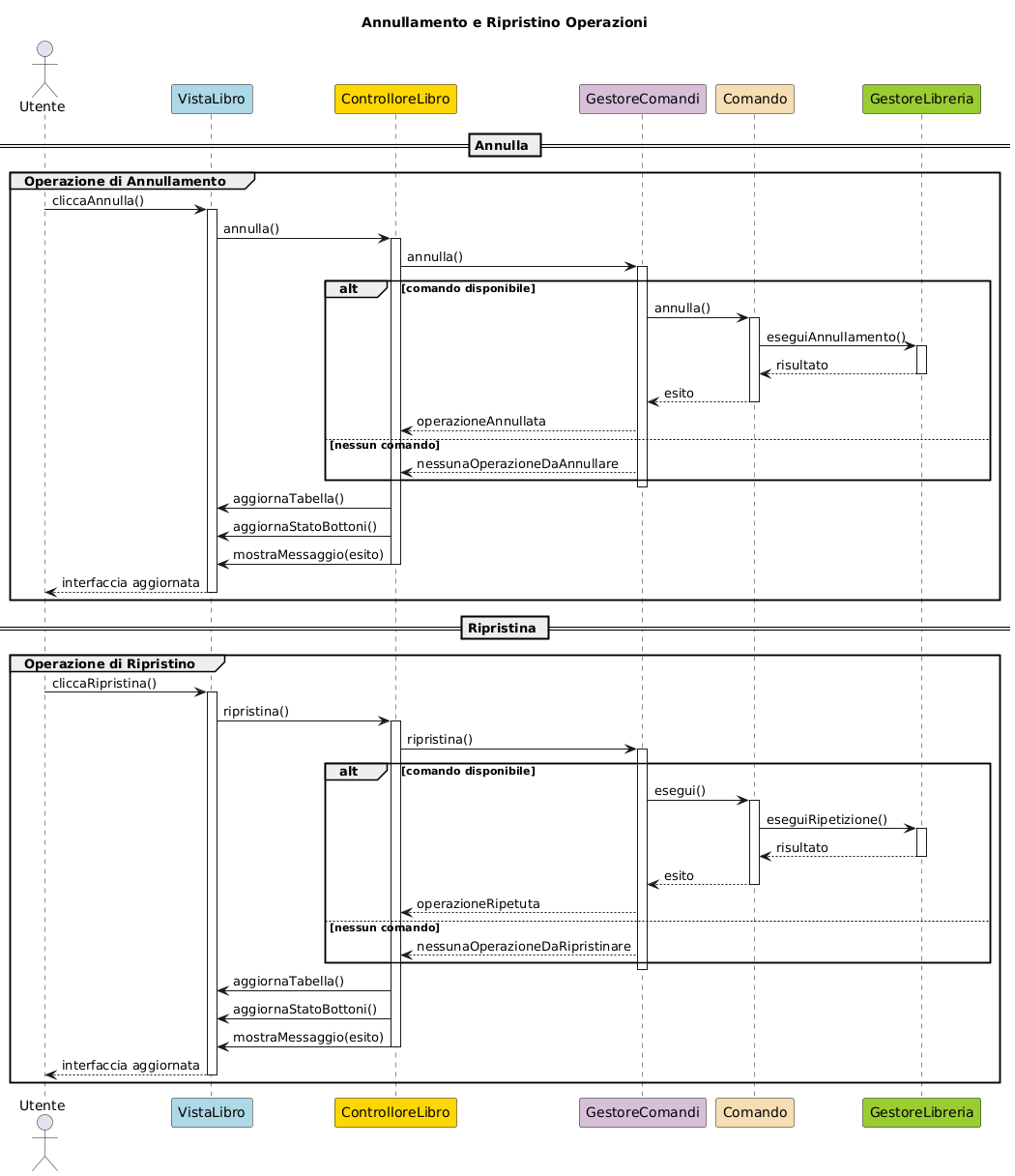
3. Ordinamento



4. Salvataggio e caricamento della libreria



5. Storico Operazioni – Undo/Redo



D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS)

All’interno dell’applicazione non è stato utilizzato un vero e proprio database relazionale, ma viene fatto affidamento a due semplici formati di persistenza, JSON e CSV, per conservare la collezione di libri. Anche in assenza di un DBMS, formalizzare il modello dei dati resta essenziale: definire con precisione entità, attributi e relazioni ci aiuta a mantenere chiarezza, coerenza e facilità di estensione nel tempo.

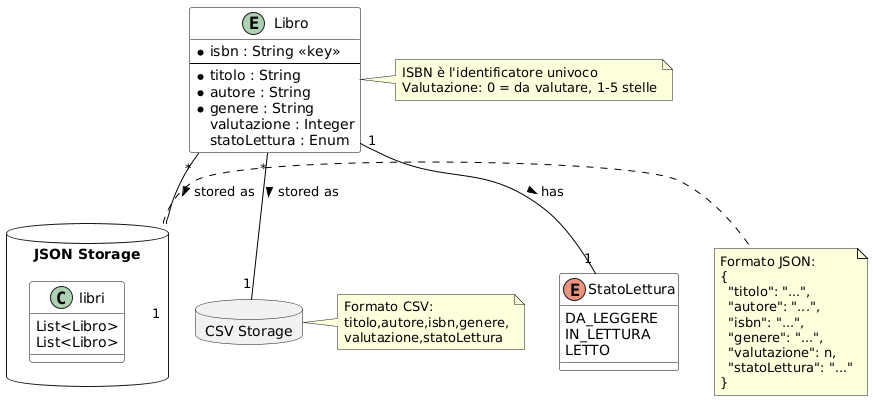
**Modello dei dati**

**Entità principale: Libro**  
Ogni libro è identificato in modo univoco dal suo ISBN (trattato come stringa, con soli numeri e trattini). A questa “chiave” si associano poi tutti gli altri campi:

* **Titolo** (stringa, obbligatorio)
* **Autore** (stringa, obbligatorio)
* **Genere** (stringa, obbligatorio)
* **Valutazione** (intero compreso tra 0 e 5; il valore 0 indica “Da valutare”. Default “Da valutare”)
* **Stato di lettura** (enumerazione con tre valori: Da leggere, In lettura, Letto. Default “Da leggere”)

**Diagramma ER**

*Il diagramma sottostante mostra l’entità principale “Libro”, l’enumerazione “StatoLettura” e le “tabelle” virtuali di persistenza in JSON e CSV.*



**Spiegazione del modello**

L’unica entità del sistema è **Libro**, identificata in modo univoco dall’attributo isbn (stringa che contiene numeri e trattini). Ad ogni libro sono associati i campi obbligatori titolo, autore e genere, più gli attributi valutazione (intero da 0 a 5) e statoLettura (un valore dell’enumerazione con i possibili stati “DA\_LEGGERE”, “IN\_LETTURA” e “LETTO”).

Lo **stato di lettura** è modellato come un piccolo insieme predefinito di costanti e, pur non essendo una tabella relazionale vera e propria, agisce come lookup per garantire coerenza dei valori.

La persistenza avviene in due formati alternativi:

* **JSON**: la libreria intera è serializzata in un array di oggetti, ognuno dei quali riproduce i campi dell’entità Libro.
* **CSV**: ogni libro è rappresentato da una singola riga con sei colonne, nell’ordine isbn, titolo, autore, genere, valutazione, statoLettura.

**Componenti di accesso ai dati**

Dal punto di vista dell’architettura applicativa, abbiamo un’unica interfaccia **LibroDAO** che definisce le operazioni di base (salva, carica). Due classi concrete la implementano:

* **JsonLibroDAO**, che si occupa di serializzare e deserializzare la lista di libri in formato JSON.
* **CsvLibroDAO**, che legge e scrive lo stesso elenco in formato CSV.

In questo modo, il resto dell’applicazione (controller, view, undo/redo, ricerca, filtri, ordinamento) può lavorare sempre sugli oggetti Libro senza preoccuparsi del formato sottostante. Se in futuro volessimo aggiungere un terzo formato (ad esempio XML o un vero database), basterà scrivere un’altra implementazione di LibroDAO e registrarla nel meccanismo di configurazione.

**Confronto con sistemi esistenti**

L’applicazione non si interfaccia con database già esistenti né importa da altri applicativi: la libreria viene costruita interamente dall’utente, riga dopo riga, nei file di persistenza. Questo approccio rende l’applicazione completamente indipendente da infrastrutture esterne e potente nella sua semplicità.

E. Scelte Progettuali (Design Decisions)

Durante lo sviluppo dell'applicazione, ho effettuato delle scelte progettuali fondamentali, tra queste le cinque decisioni architetturali principali, ognuna delle quali è stata motivata dall'obiettivo di realizzare un sistema chiaro, flessibile e facilmente manutenibile. Di seguito, fornirò una descrizione dettagliata, evidenziando sia le motivazioni alla base di tali decisioni sia il loro impatto complessivo sul progetto.

**1. Architettura MVC**

Per mantenere una netta separazione delle responsabilità, ho deciso di adottare il modello **Model-View-Controller**: il *Model* contiene i dati dei libri e le regole di validazione, la *View* gestisce l’interfaccia grafica (Swing) e il *Controller* coordina l’interazione tra utente e dati. In questo modo, modifiche all’aspetto della finestra o alle logiche di salvataggio non si pestano i piedi tra loro, e diventa facile testare ogni componente in modo isolato.

**Implementazione**

* **Model:** la classe Libro e l’enum StatoLettura descrivono i dati.
* **View:** LibroView (estende JFrame) contiene la tabella, i pulsanti e i dialog, e si limita a notificare eventi.
* **Controller:** LibroController intercetta le azioni dell’utente, invoca il GestoreLibreria per modificare i dati e chiede alla vista di aggiornarsi.

**Impatto**  
Grazie a MVC, modifiche all’interfaccia (ad esempio un restyling dei dialog) non hanno effetto sui dati o sulla logica di business, e viceversa.

**2. Undo/Redo con il pattern Command**  
Per dare all’utente la possibilità di annullare e ripetere ogni operazione (aggiunta, modifica, eliminazione), è stato usato il **pattern Command**. Ogni azione si trasforma in un “comando” che conosce sia il suo comportamento, sia anche il modo di annullarsi. Così l’intero processo di undo/redo risulta omogeneo, centralizzato e facile da gestire.

**Implementazione**

* E’ stata definita un’interfaccia Command con i metodi execute() e undo().
* Ogni operazione rilevante (aggiungi, modifica, elimina) ha la sua classe Command (AggiungiCommand, ModificaCommand, EliminaCommand).
* Un CommandManager mantiene due stack: uno per l’undo e uno per il redo, gestendo l’esecuzione e il rollback.

**Impatto**  
Il pattern Command ha consentito di centralizzare la logica di undo/redo, rendendo semplice aggiungere nuove operazioni reversibili e mantenendo il controller libero da dettagli di storicizzazione.

**3. Persistenza dati con il pattern DAO**  
La separazione dei dettagli di I/O dal resto dell’applicazione è affidata al **pattern Data Access Object**. E’ stata definita un’interfaccia comune (LibroDAO) e poi fornite due implementazioni, per JSON e per CSV. In futuro se si volesse aggiungere un database SQL o un file XML, lo si potrebbe fare semplicemente estendendo l’interfaccia, senza toccare controller o view.

**Implementazione**

* Creata un’interfaccia LibroDAO con i metodi salvaLibri(...) e caricaLibri(...).
* Fornite due implementazioni: JsonLibroDAO e CsvLibroDAO, ciascuna responsabile della lettura/scrittura nel formato corrispondente.

**Impatto**  
Grazie al DAO, l’aggiunta di un nuovo formato richiederebbe soltanto una nuova classe, senza toccare il controller o la vista.

**4. Ordinamento intercambiabile con il pattern Strategy**  
Per gestire diversi criteri di ordinamento (titolo, autore, valutazione) e lasciar spazio a futuri algoritmi, è stato scelto il **pattern Strategy**. Ogni modo di ordinare è una classe a sé, che può essere selezionata in base alla preferenza dell’utente. Questo mantiene il controller pulito e favorisce l’estensibilità.

**Implementazione**

* Definiamo l’interfaccia OrdinatoreLibroStrategy con il metodo ordina(List<Libro>).
* Creiamo classi concrete come OrdinaTitoloAZStrategy, OrdinaAutoreAZStrategy e OrdinaValutazioneDescStrategy.
* Il controller o il gestore di libreria seleziona la strategia desiderata in base alla scelta dell’utente.

**Impatto**  
Nuove modalità di ordinamento possono essere aggiunte semplicemente implementando l’interfaccia, senza modificare il codice esistente.

**5. Gestione centralizzata con il pattern Singleton**  
Infine, per garantire che tutta l’applicazione lavori sempre sulla stessa lista di libri, è stato reso il “gestore” un **singleton**. In questo modo, sia il controller che i comandi di undo/redo passano sempre e soltanto per quell’unica istanza, evitando duplicazioni di stato e incoerenze.

**Implementazione**

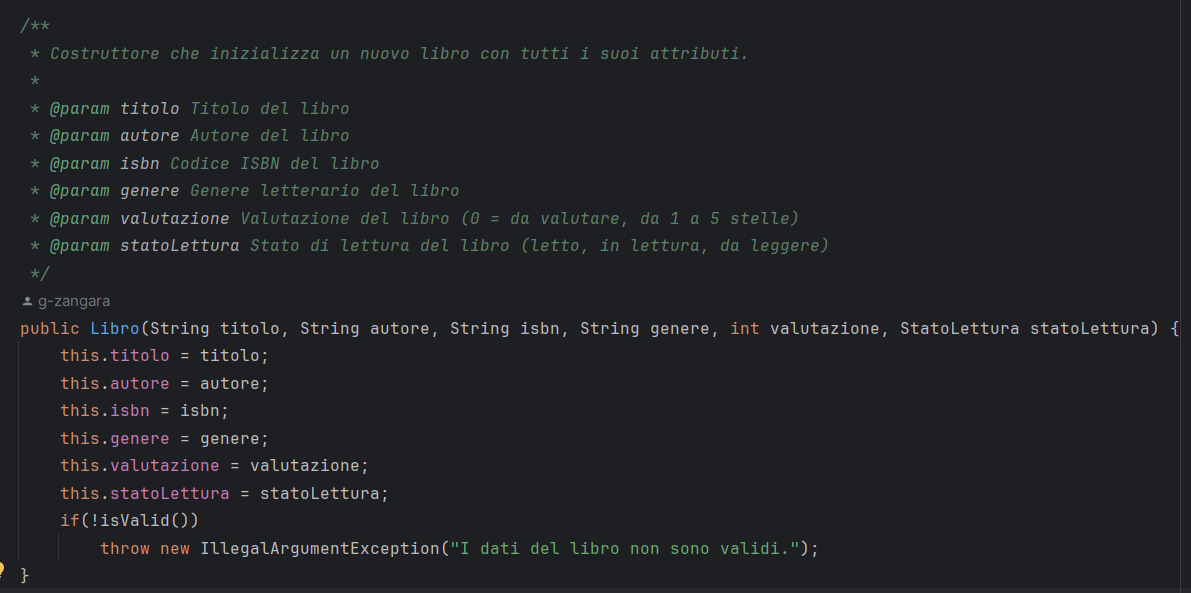
* La classe GestoreLibreria ha un costruttore privato e un metodo statico getInstance() che crea o restituisce l’unica istanza.
* Tutte le operazioni di manipolazione dei libri (add, update, remove, ricerca) passano attraverso questo singleton.

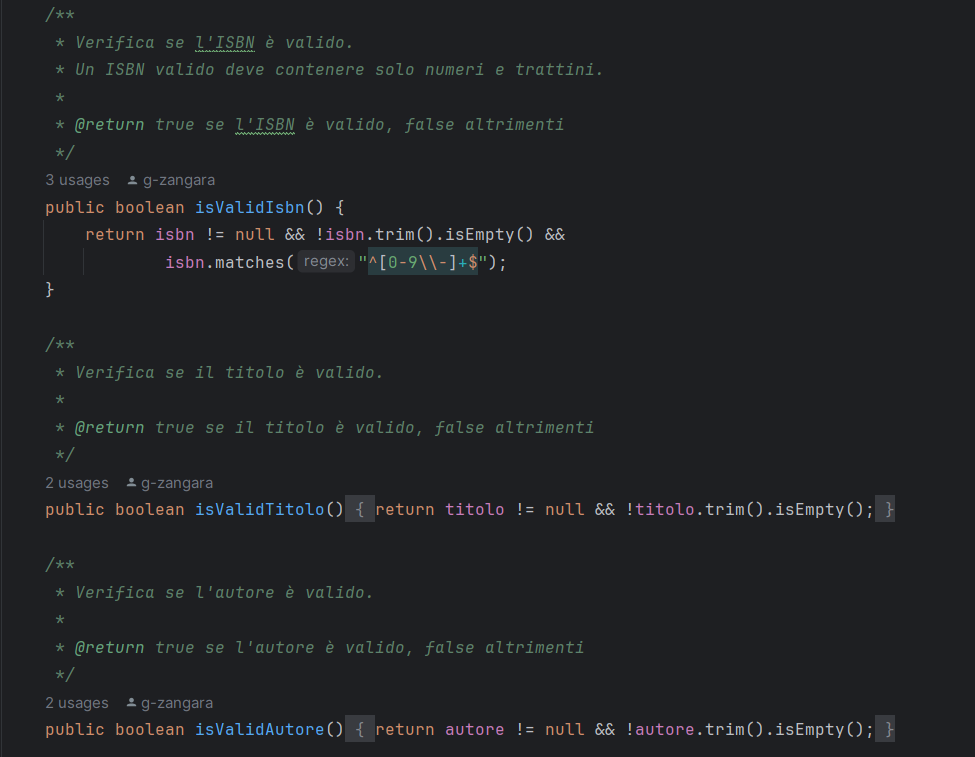
**Impatto**  
Pertanto si evitano conflitti o duplicazioni di stato e si assicura coerenza dei dati, mentre il controller e le view possono fare riferimento senza gestire direttamente liste o collezioni globali.

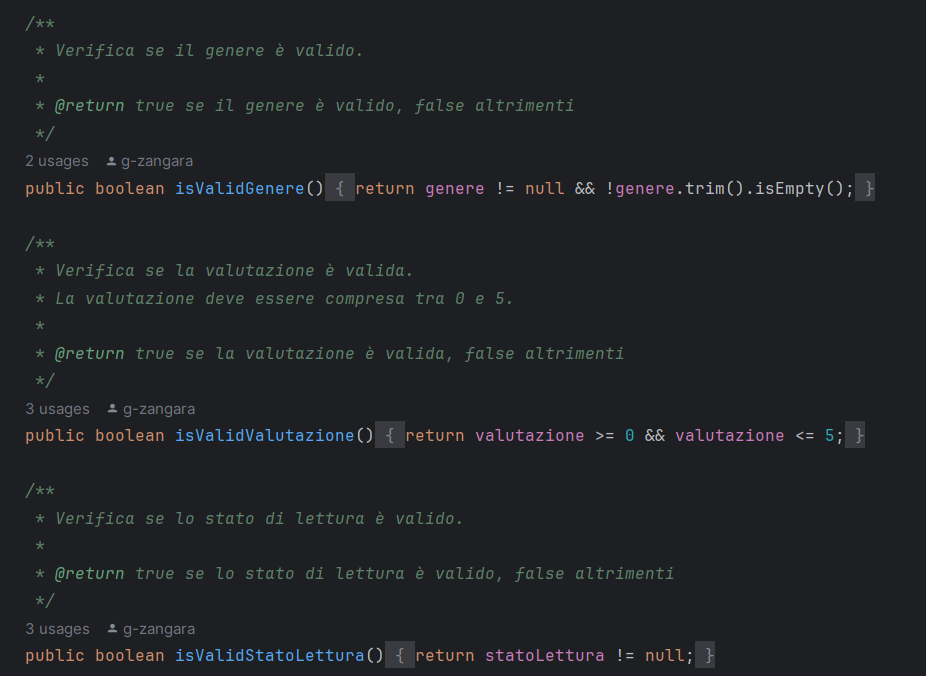
F. Progettazione di Basso Livello

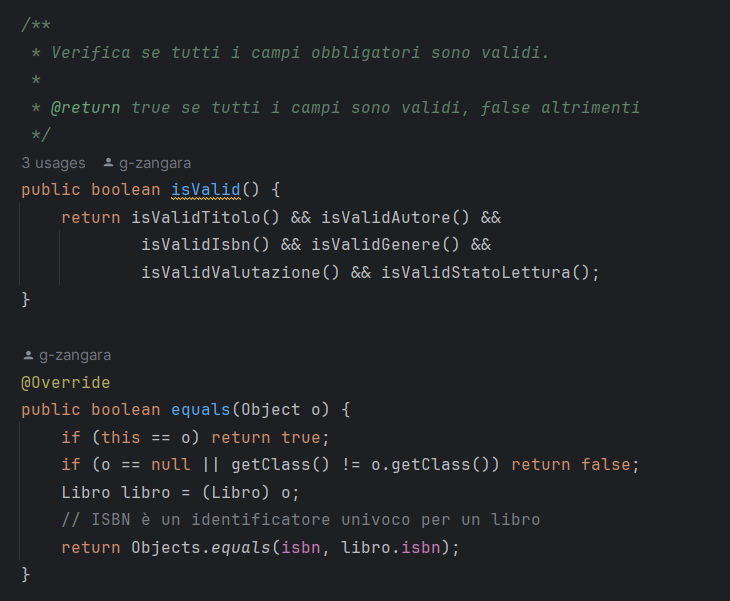
**1. Controlli e validazione dei dati del libro**

Ogni dato che arriva dall’input può essere corrotto, pertanto vengono effettuati dei controlli di validazione: ogni volta che viene creato o modificato un oggetto Libro, viene eseguito un controllo completo sui campi. Verifichiamo che titolo, autore, ISBN e genere non siano vuoti, che la valutazione sia compresa tra 0 e 5 e che lo stato di lettura sia valorizzato e valido. Questo controllo avviene idealmente all’interno della classe modello (Libro), il suo metodo di validazione, viene richiamato dal costruttore o prima del salvataggio.  
Se qualcosa non torna, lanciamo una nostra eccezione di validazione che poi il controller trasforma in un messaggio comprensibile all’utente, evidenziando il campo da correggere, in caso di file evidenzia la riga o la posizione dell’oggetto di dove si presenta l’inconsistenza del dato. Inoltre, l’unicità dell’ISBN viene garantita dal GestoreLibreria, che controlla l’esistenza preliminare prima di aggiungere un nuovo libro.

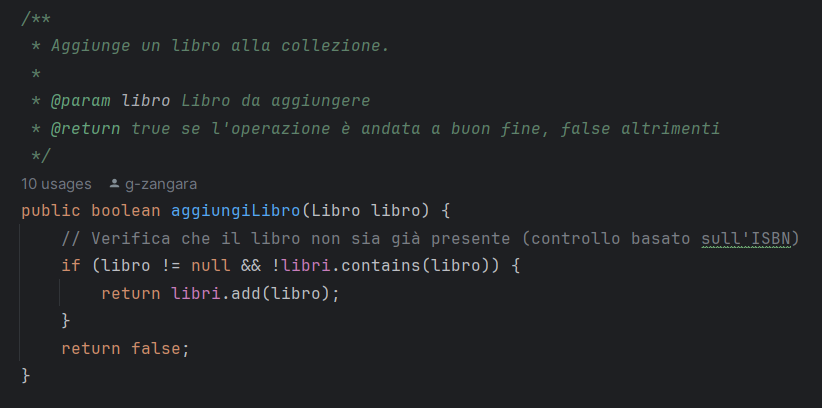






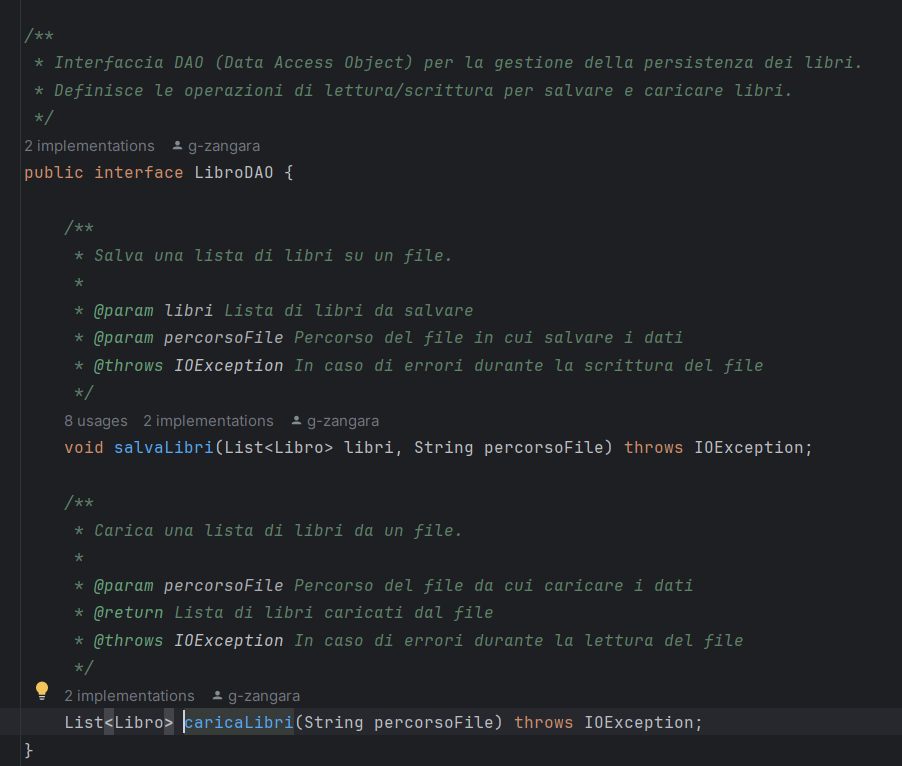


Metodo di aggiunta in GestoreLibrera

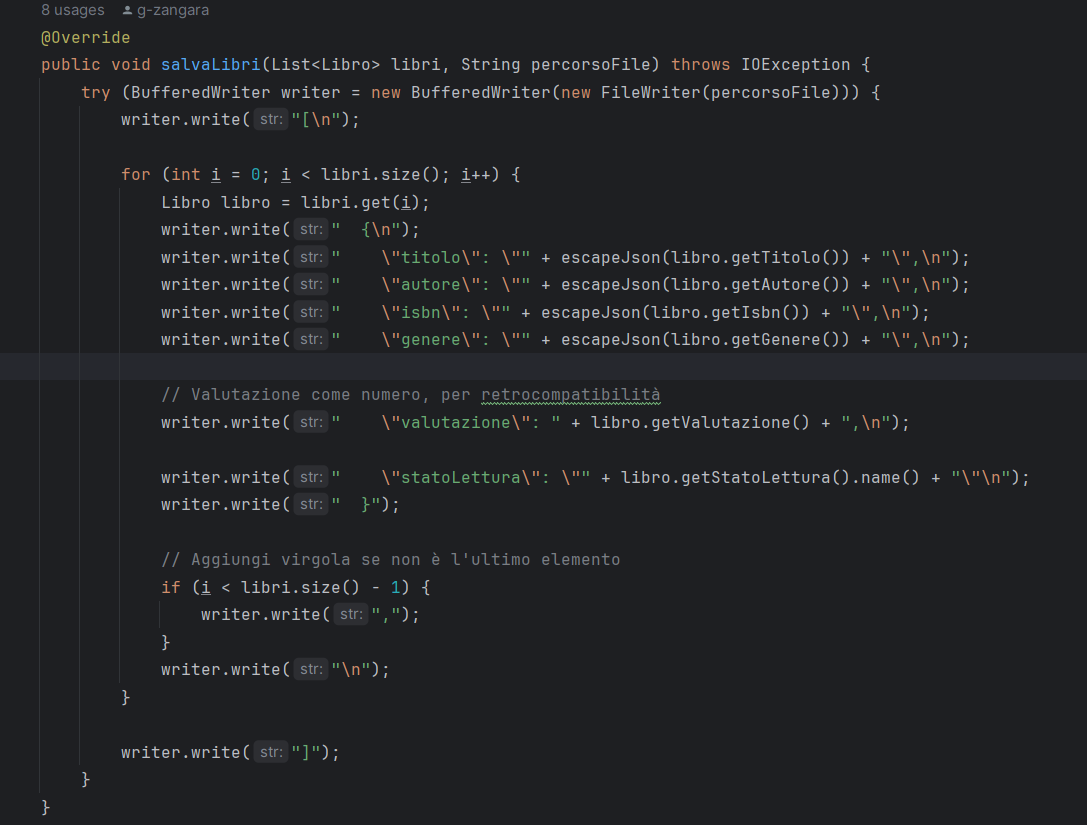


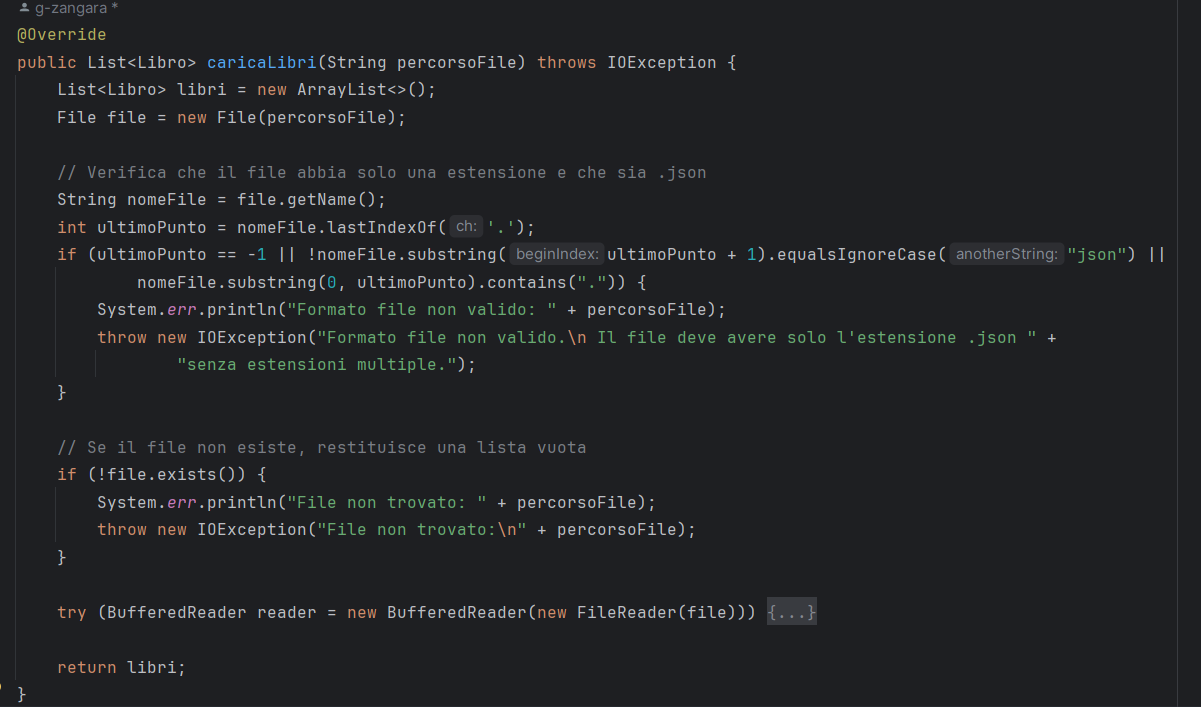
**2. Accesso ai dati: pattern DAO**

Per tenere pulita la logica di business da ogni dettaglio di I/O file, è stata definita l’interfaccia LibroDAO e due implementazioni concrete: una per il formato JSON e una per il CSV. Ogni operazione di salvataggio o caricamento è gestita all’interno di blocchi try–catch che catturano eventuali IOException e le rilanciano come eccezioni specifiche di persistenza.  
In questo modo, se si volesse in futuro aggiungere XML, un database relazionale o qualsiasi altra sorgente dati, basterebbe fornire un nuovo DAO rispettando la stessa interfaccia, senza toccare controller o view.

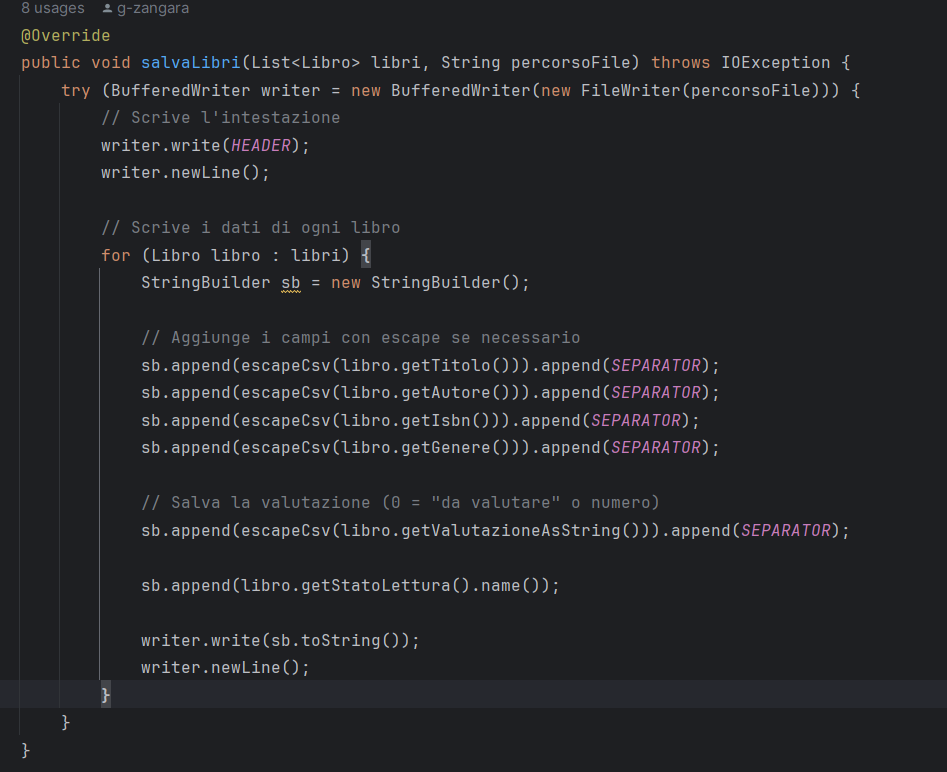


Implementazione JsonLibroDAO

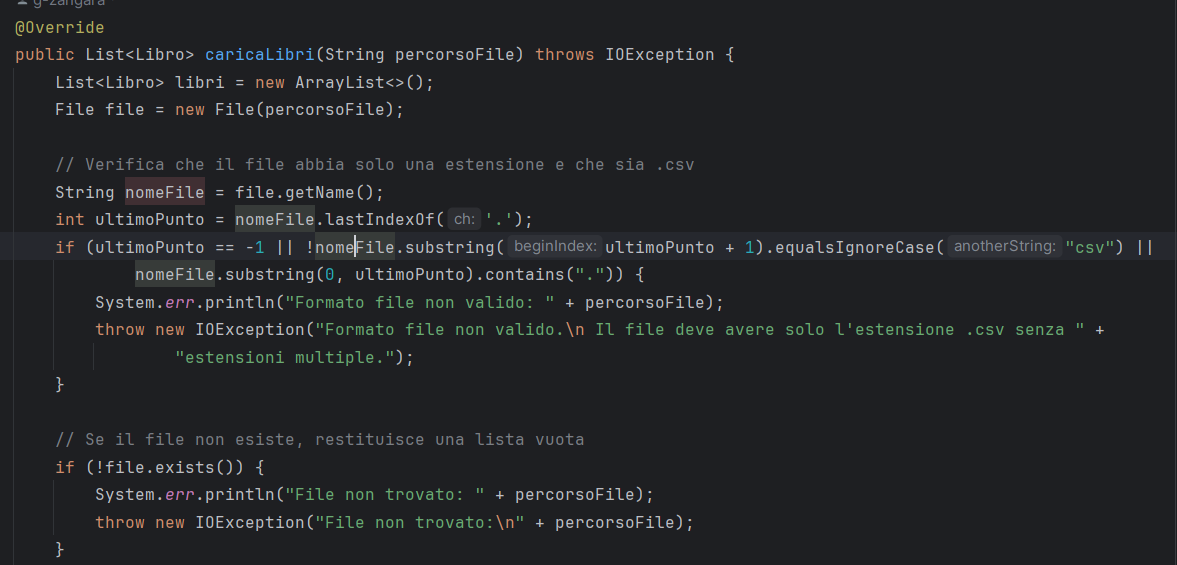




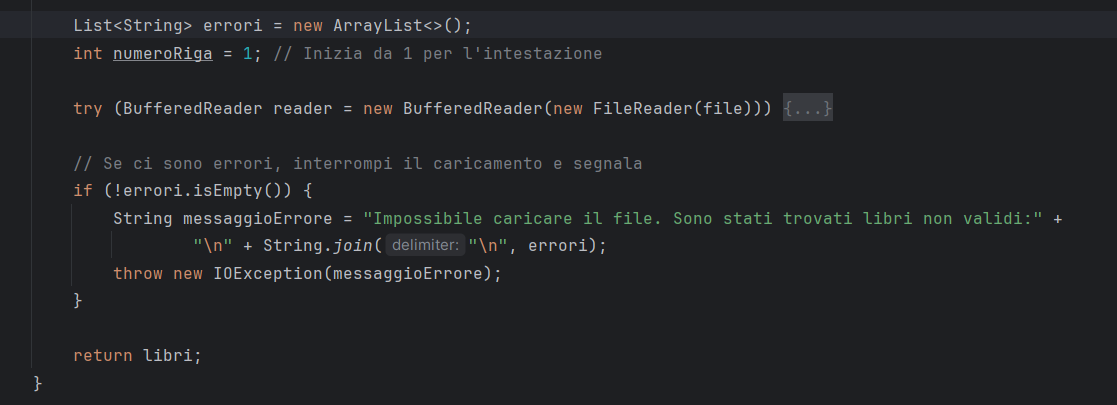
Implementazione di CsvLibroDAO



CaricaLibri di CsvLibroDAO parte1

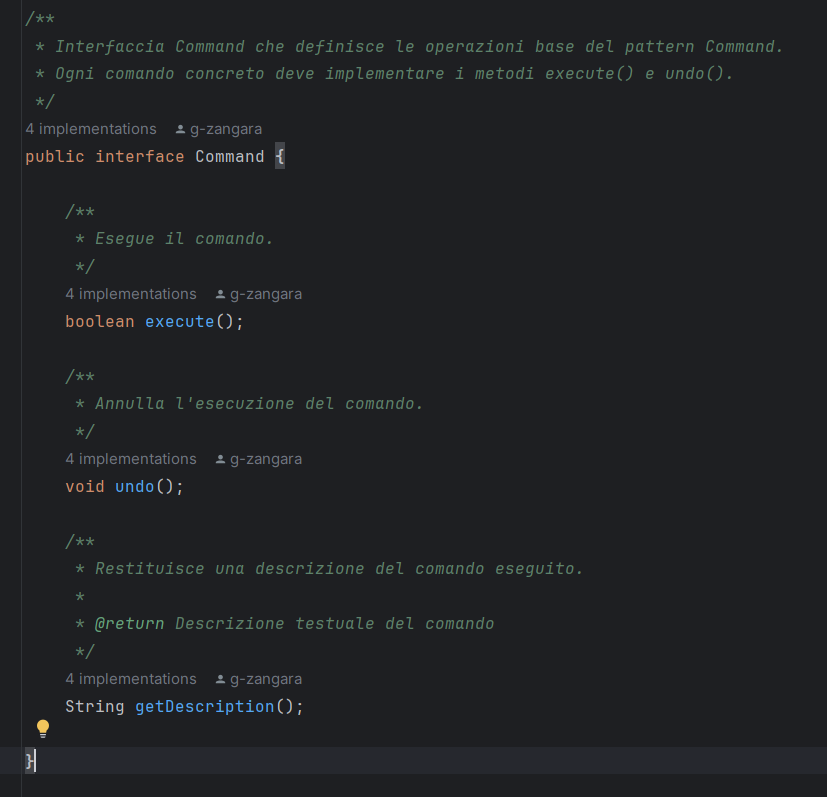


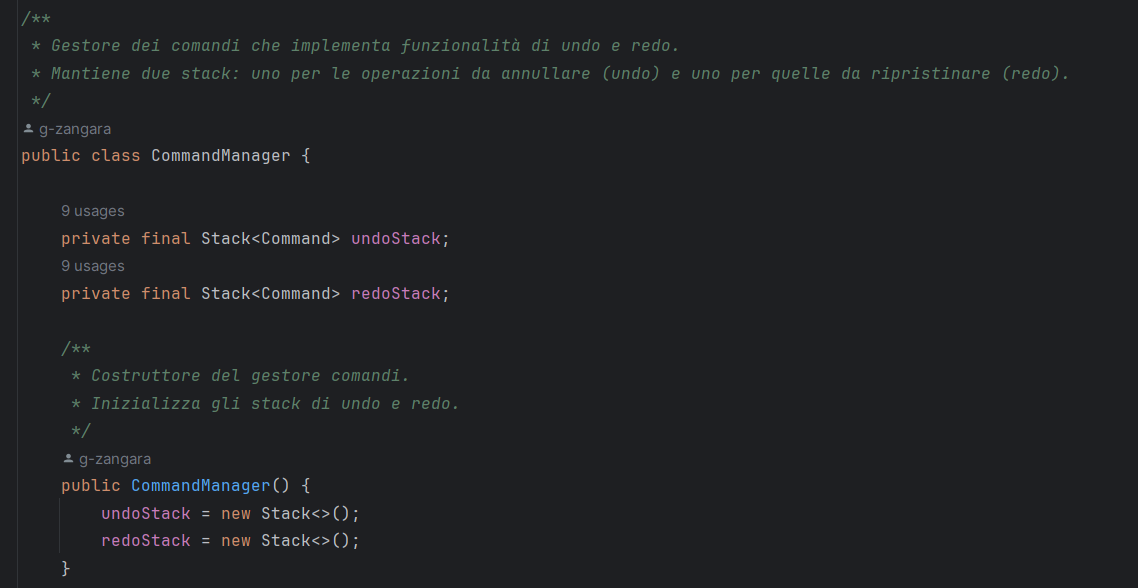
CaricaLibri di CsvLibroDAO parte2

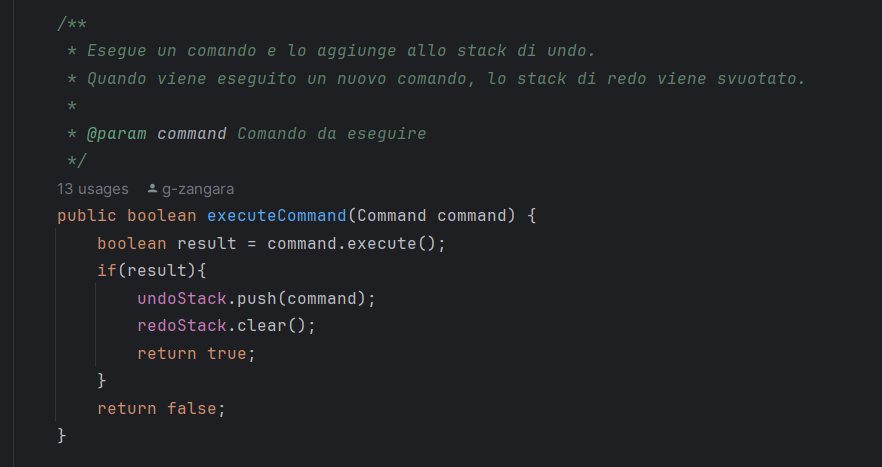


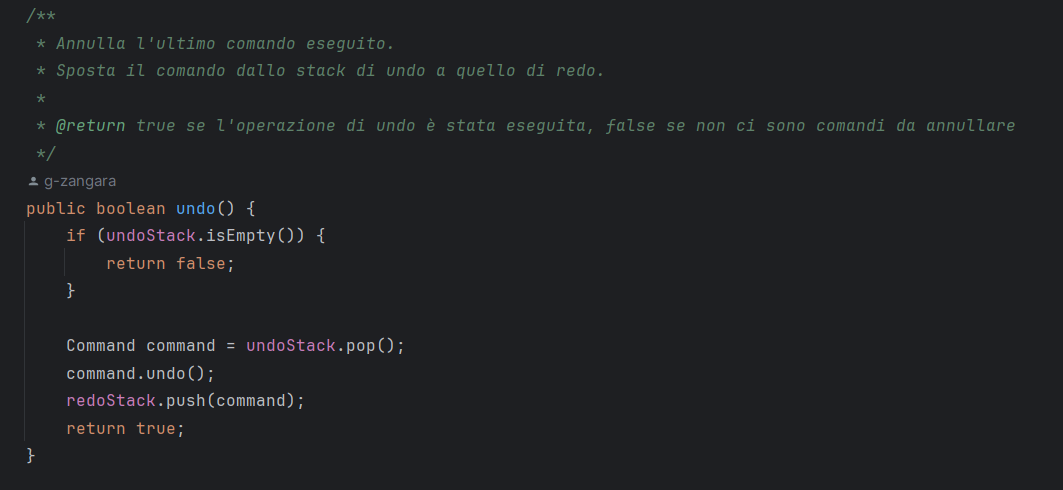
**3. Undo/Redo centralizzato con Command Pattern**

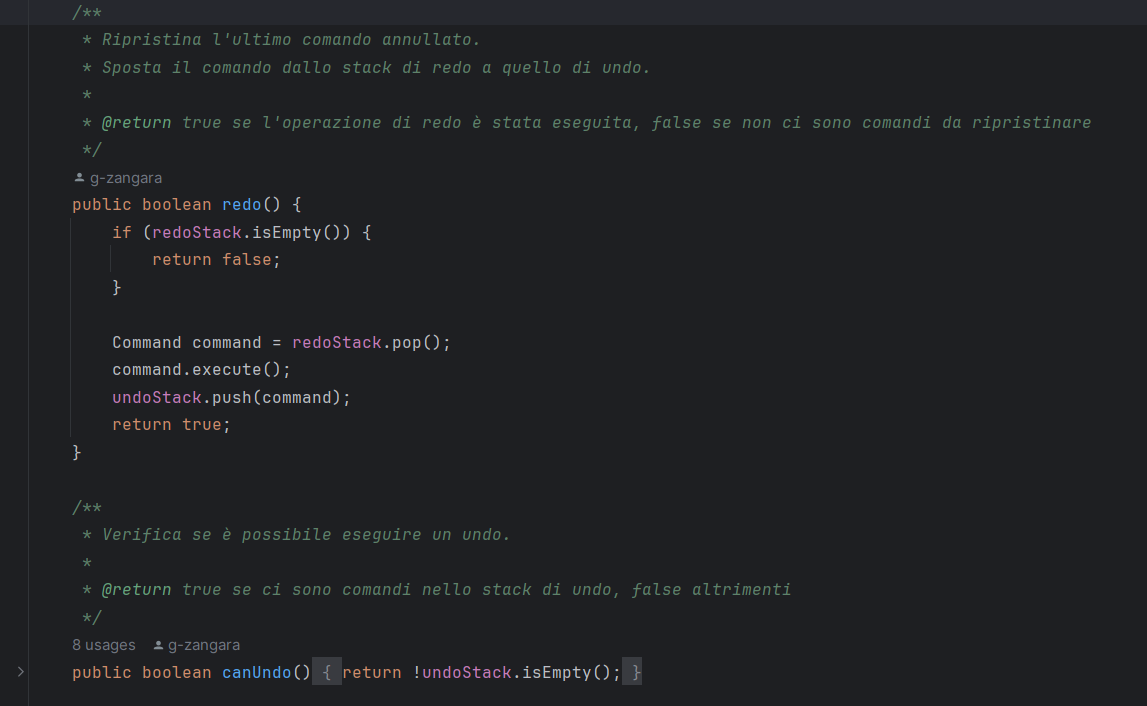
Tutte le operazioni che modificano la libreria (aggiunta, modifica, eliminazione) diventano “comandi” dotati di due metodi: execute() per applicare l’azione e undo() per annullarla. Un gestore comandi (commandManager) mantiene due stack: uno per le esecuzioni da poter annullare e uno per le ripetizioni da poter “rifare”.  
Così, il controller può semplicemente invocare commandManager.execute(comando), e il resto del meccanismo (storico, rollback, abilitazione/disabilitazione dei pulsanti) è gestito in maniera uniforme.

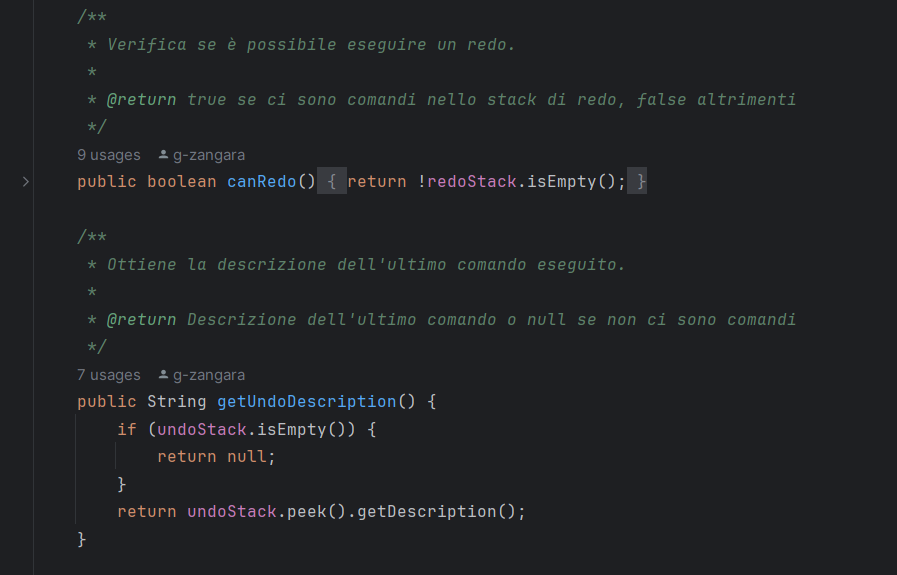


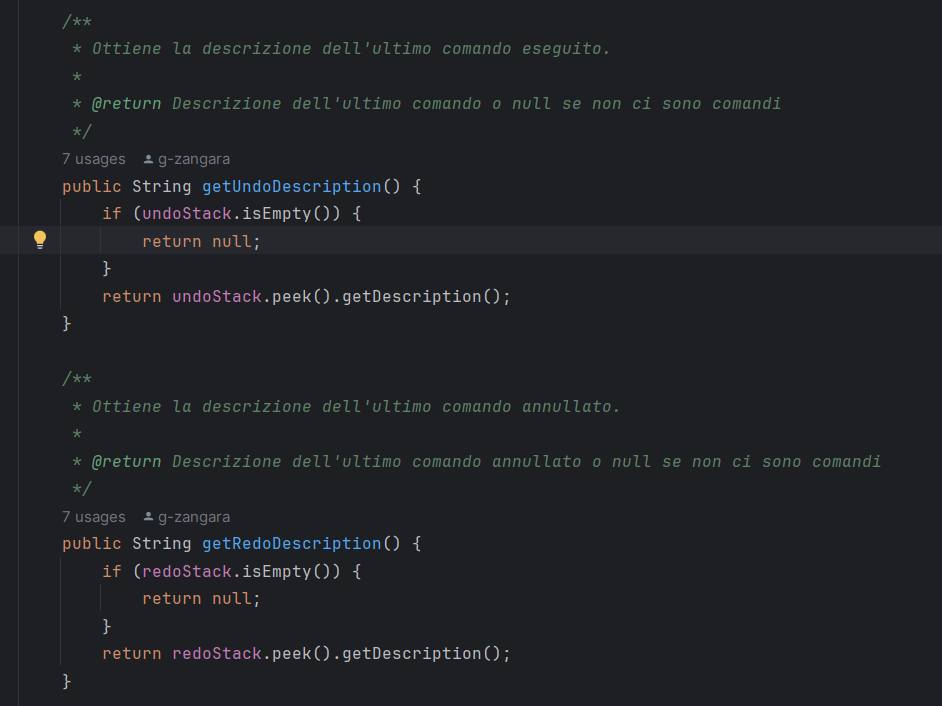


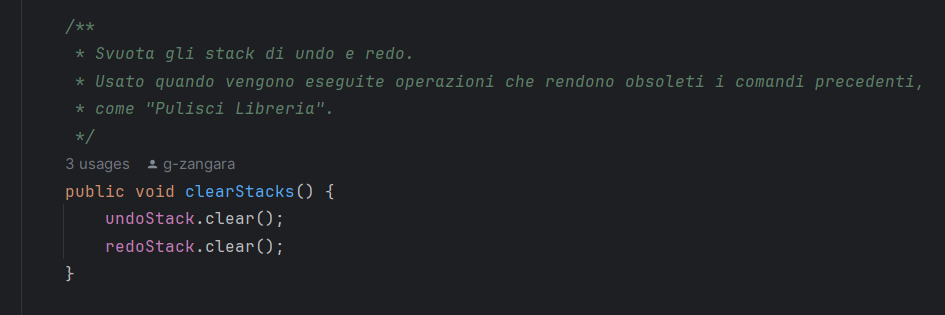






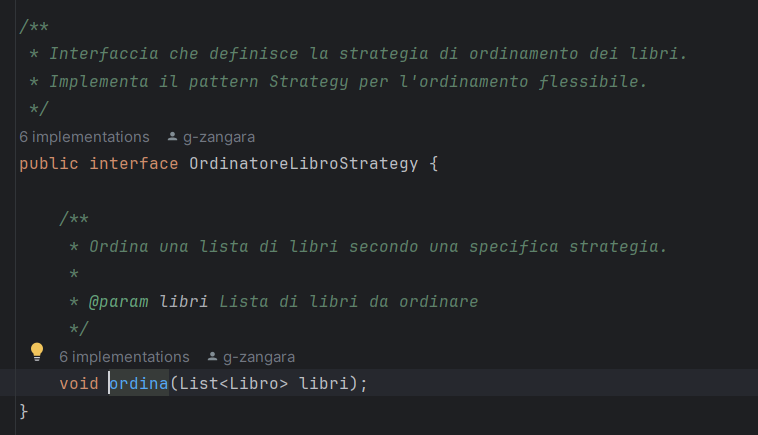


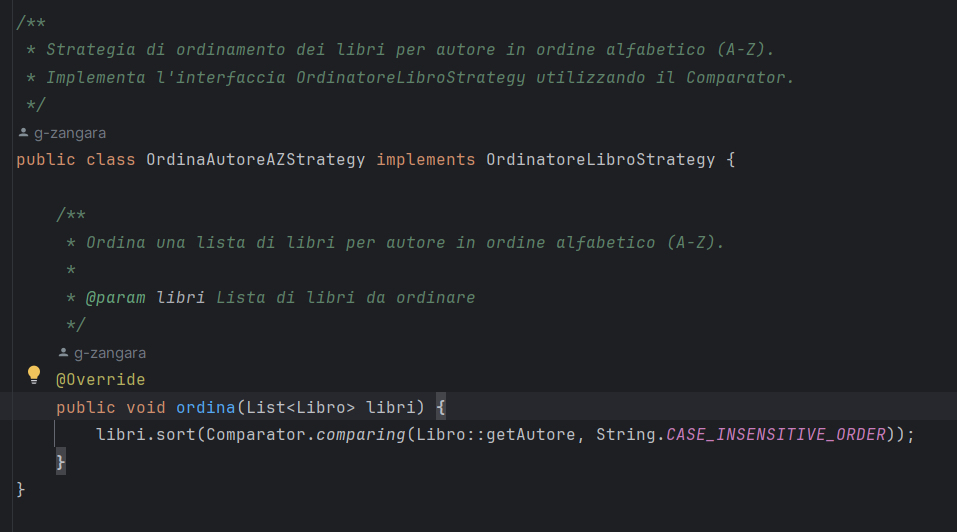


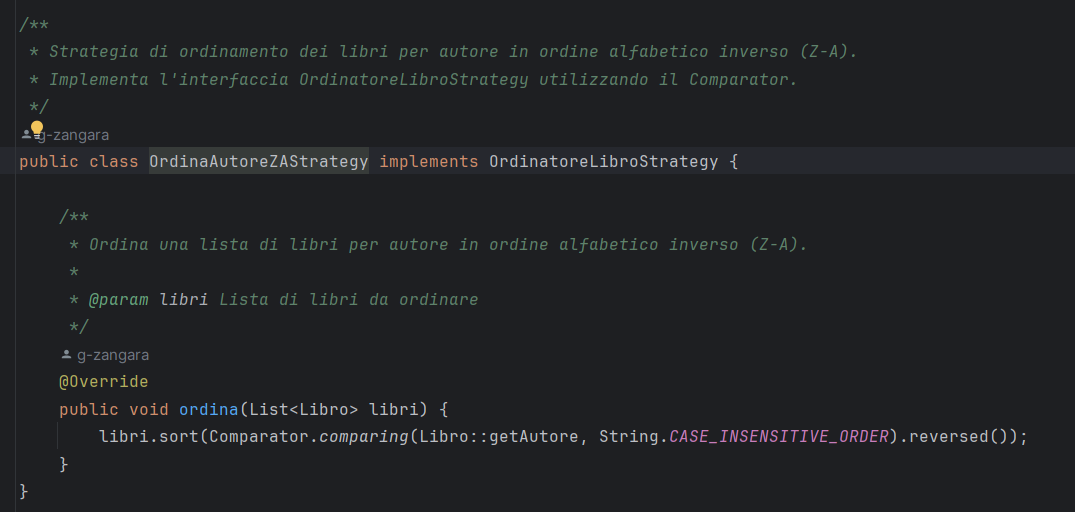


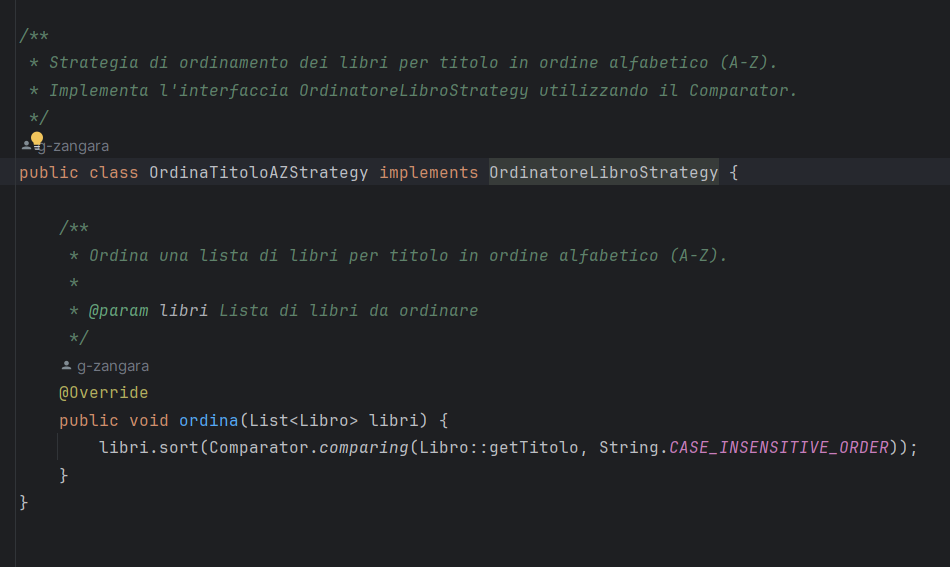
**4. Ordinamenti plug-in con Strategy Pattern**

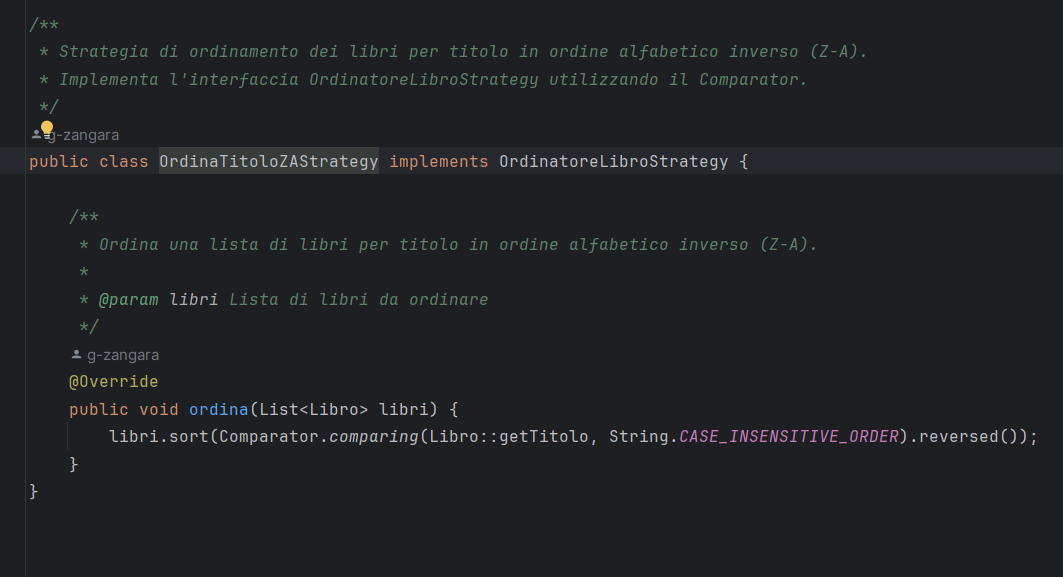
Per offrire ordinamenti differenti e lasciare la porta aperta a futuri criteri, ciascun algoritmo di ordinamento è incapsulato in una classe che implementa la stessa interfaccia. Quando l’utente sceglie “Ordina per titolo A–Z”, “Ordina per autore A–Z” o “per valutazione 1-5”, e viceversa il controller seleziona la strategia corrispondente ed esegue l’ordinamento sulla lista di libri.  
Questo permette l’aggiunta di nuovi criteri (ad esempio per data di inserimento o numero di pagine, qualora sia prevista un’estensione del model) semplicemente creando una nuova strategia, senza modificare il flusso principale.

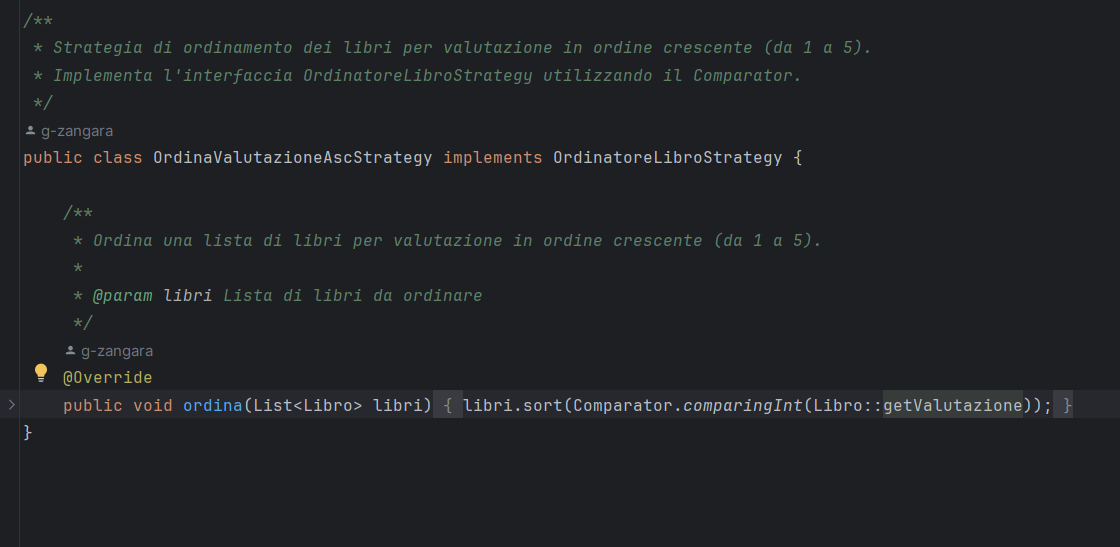


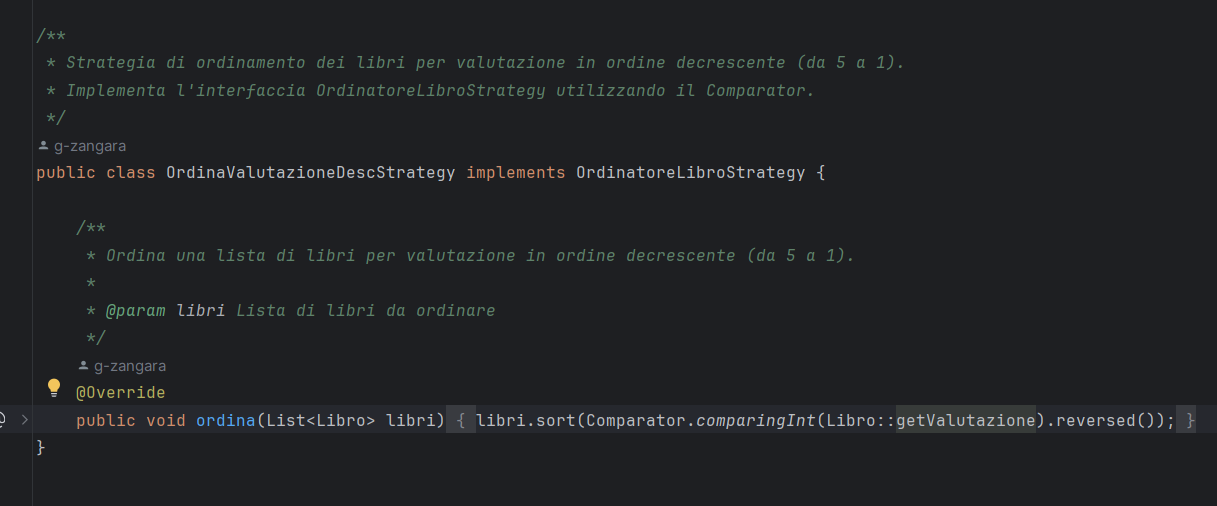






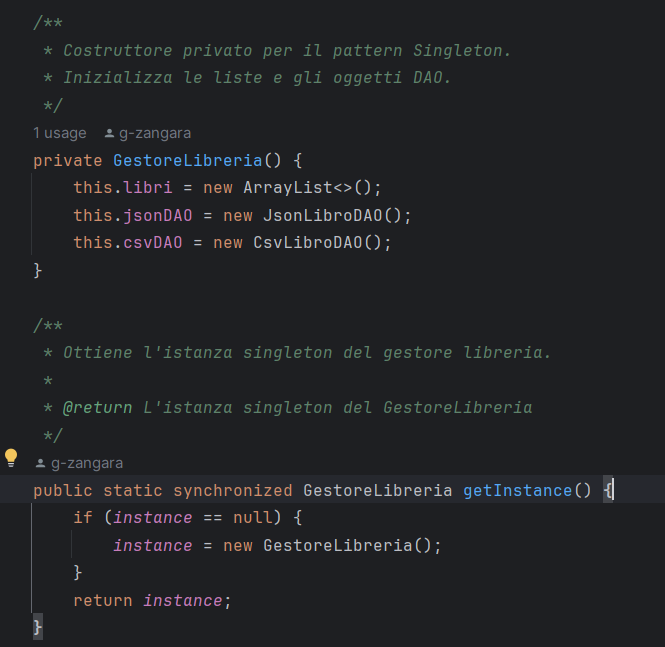






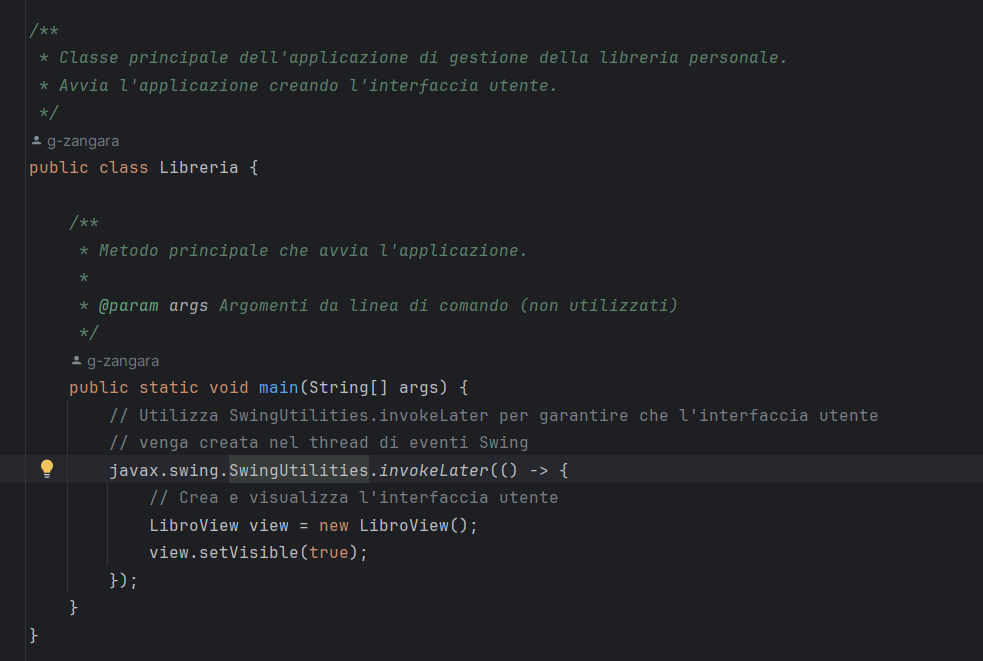
**5. Gestione centralizzata e sicura dello stato con Singleton e thread-safe**

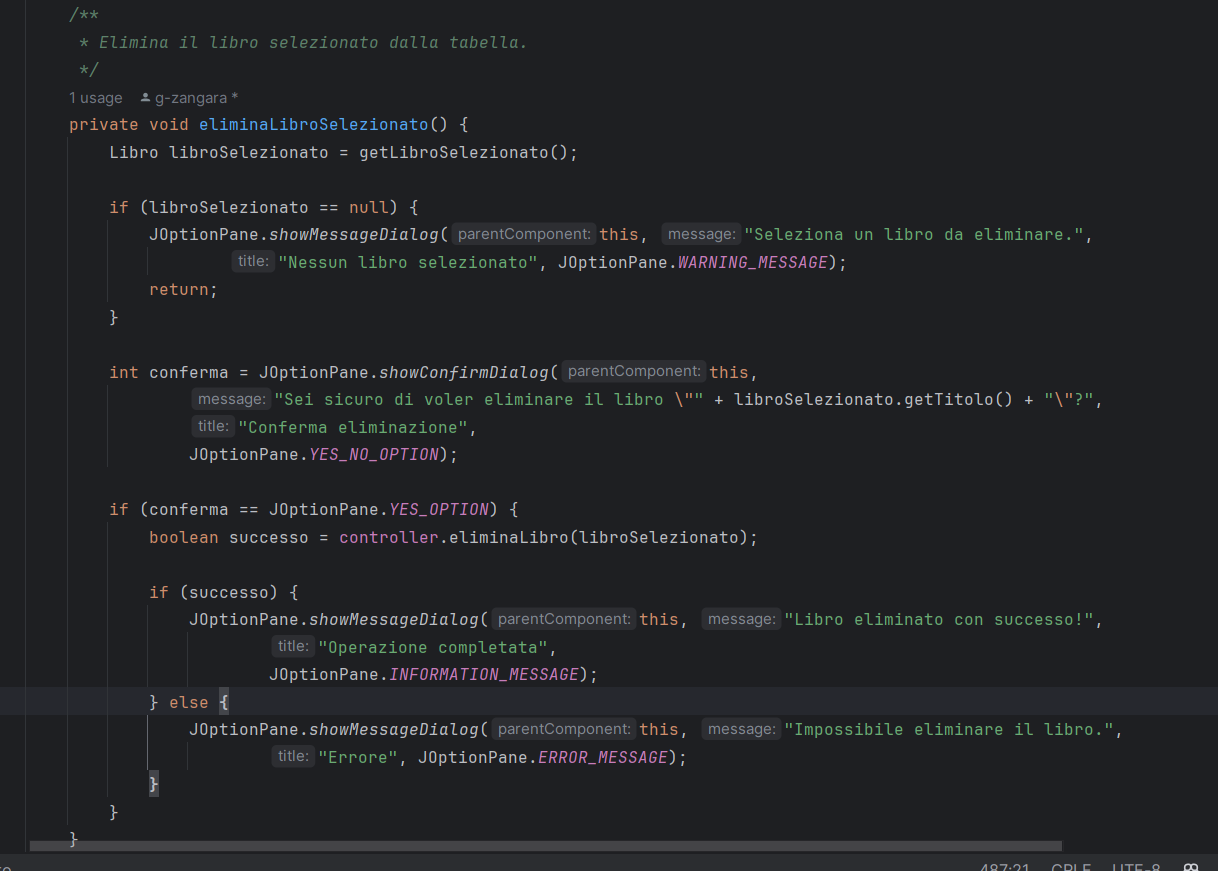
Per evitare liste multiple o incoerenze quando diverse parti del programma leggono o scrivono la collezione di libri, GestoreLibreria è un singleton: un’unica istanza condivisa da controller, comandi e view.  
Inoltre, per garantire sicurezza in caso di operazioni concorrenti (ad esempio undo/redo mentre arriva un secondo click), la lista è protetta con strutture thread-safe e i metodi critici sono sincronizzati.

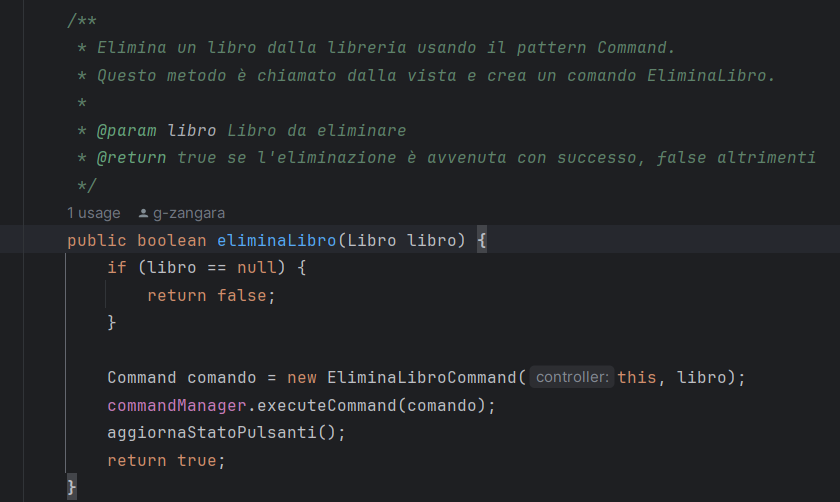


**6. Interazione pulita tra View e Controller**

La LibroView osserva il modello e si aggiorna solo quando viene notificata dal controller, che richiama metodi di refresher (aggiornaTabella) in modo thread-safe (usando ad esempio SwingUtilities.invokeLater). In questo modo si evitano blocchi o eccezioni di accesso errato alla GUI.

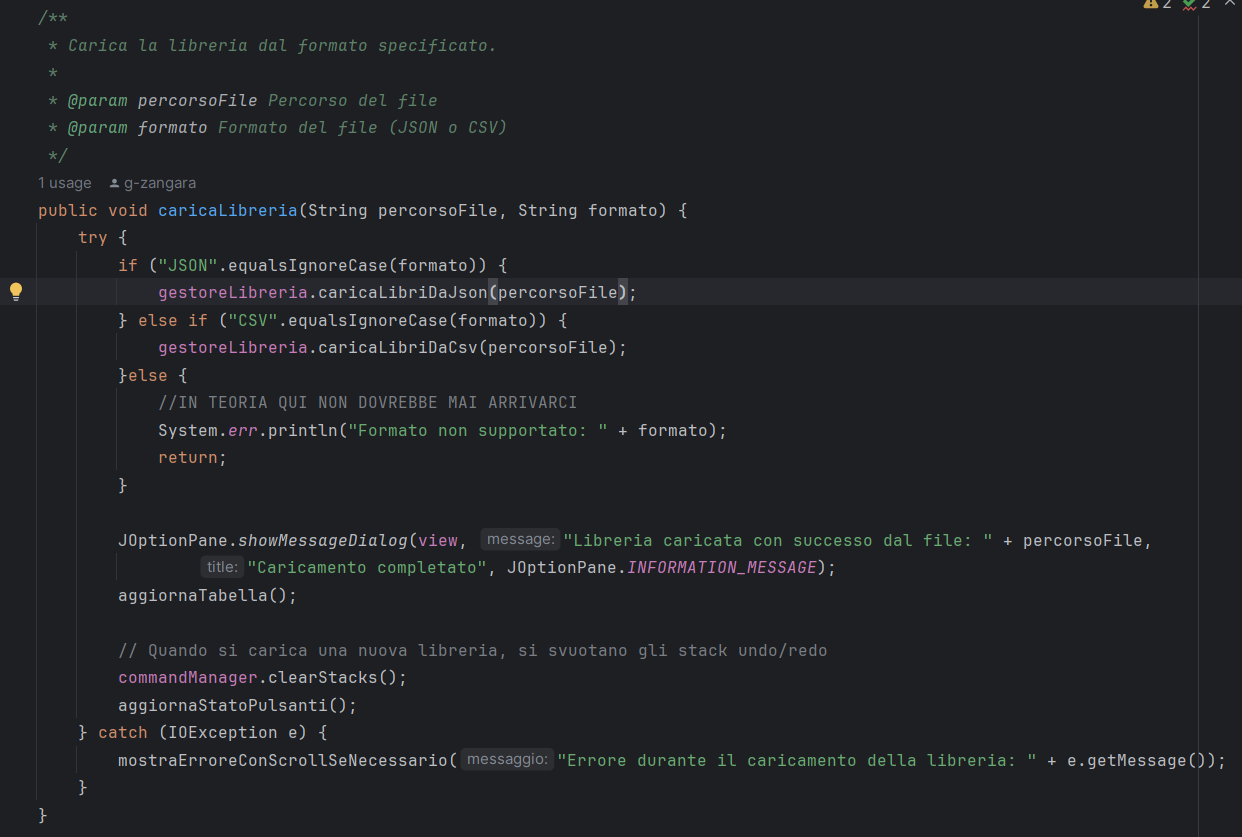






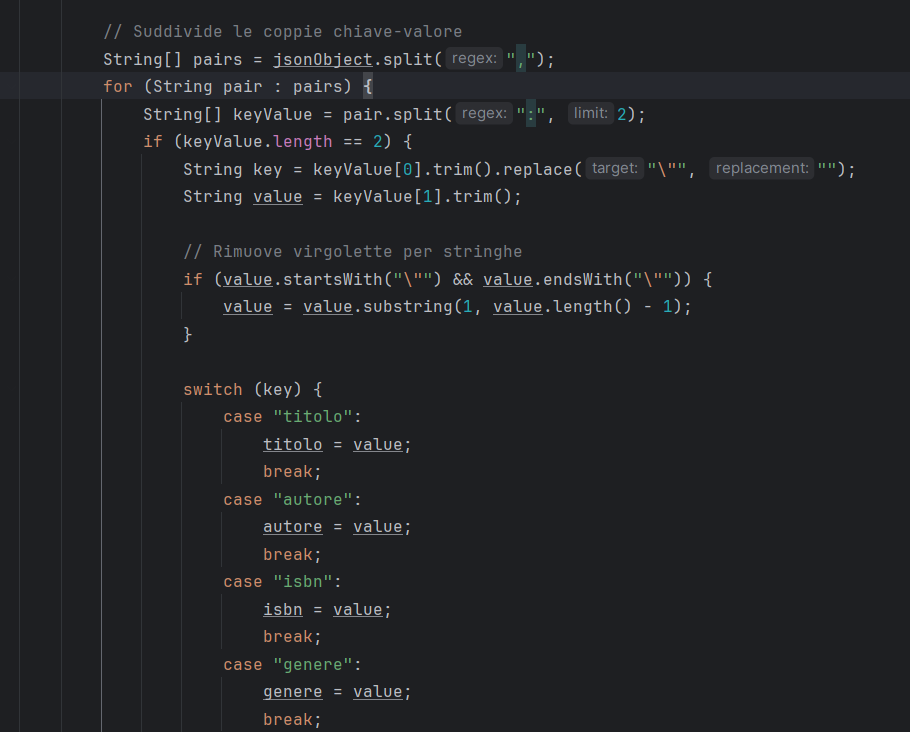
**7. Validazione di input/output e gestione errori nei file**

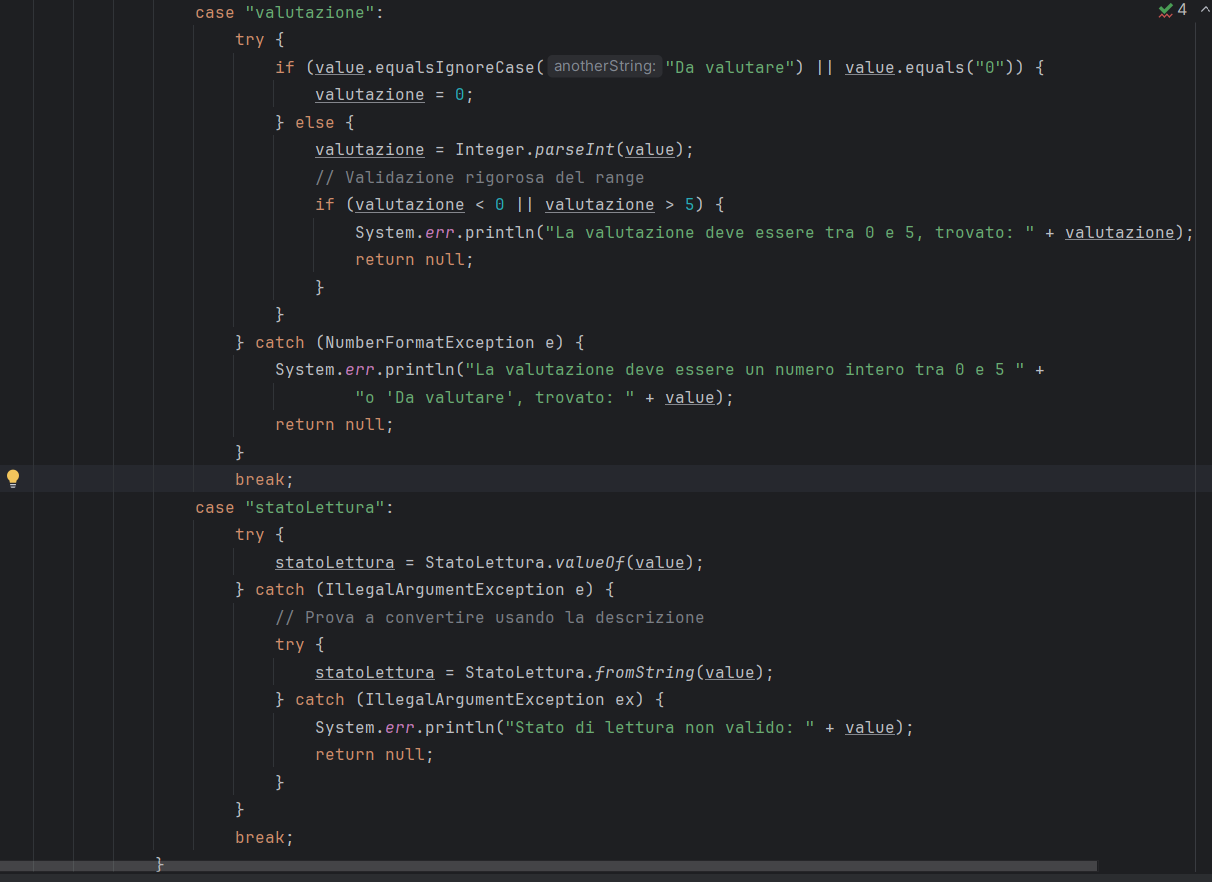
Il sistema non si limita solo a validare i campi di testo: prima di caricare o salvare viene anche controllata l’esistenza del file, la correttezza del formato (CSV con 6 colonne, JSON che inizi e termini con parentesi quadre) ed eventualmente viene anche catturato qualsiasi errore di parsing. In caso di problemi, viene mostrato all’utente un messaggio dettagliato (“Formato JSON non valido”, “Numero di colonne CSV errato”) e lasciato intatto lo stato precedente, ripulendo eventuali file temporanei o parziali.





Nello specifico for contiene:





G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs) In questa sezione viene mostrata come l’architettura e il design di basso livello realizzati rispondano in modo diretto e completo sia ai requisiti funzionali sia ai requisiti non funzionali definiti in precedenza.

**1. Requisiti Funzionali**

**Gestione CRUD dei libri**

* **Architettura MVC**: il LibroController coordina tutte le operazioni di inserimento, modifica ed eliminazione, mentre la LibroView si occupa di presentare la tabella aggiornata e i dialog di interazione.
* **Command Pattern**: ogni azione di aggiunta, modifica o cancellazione è incapsulata in un comando (AggiungiLibroCommand, ModificaLibroCommand, EliminaLibroCommand) che viene eseguito e reso annullabile. Questo garantisce che le operazioni CRUD siano atomiche e reversibili, semplificando l’implementazione di undo/redo.
* **Validazione centralizzata**: il modello Libro espone un metodo di validazione che controlla titolo, autore, ISBN, genere, valutazione e stato di lettura; qualsiasi violazione viene intercettata e trasformata in un messaggio di errore chiaro per l’utente.
* **Singleton per lo stato**: il GestoreLibreria è un singleton thread-safe che conserva l’elenco dei libri, assicurando che tutte le componenti del sistema operino sulla stessa lista evitando così duplicazioni e inconsistenza di dati.

**Persistenza dei dati**

* **DAO Pattern**: l’interfaccia LibroDAO definisce i metodi generici di salva e carica, mentre le implementazioni JsonLibroDAO e CsvLibroDAO gestiscono ciascuna la propria implementazione di caricamento e salvataggio.
* **Gestione degli errori I/O**: ogni operazione di lettura o scrittura è protetta da blocchi try–catch che rilanciano eccezioni specifiche, garantendo rollback e notifiche informative all’utente in caso di problemi.

**Ricerca e Filtri**

* **Logica nel controller**: LibroController utilizza le Stream API di Java per concatenare filtri e ricerche senza dover riscrivere cicli complessi.
* **Interfaccia reattiva**: la LibroView invia i criteri di ricerca al controller, che restituisce la lista parzialmente filtrata e la tabella si aggiorna in tempo reale.
* **Estensibilità per nuovi filtri**: grazie al design modulare, aggiungere un nuovo filtro (ad esempio per anno di pubblicazione) richiede solo l’implementazione di un nuovo predicato, senza toccare la struttura esistente.

**Ordinamento**

* **Strategy Pattern**: ogni algoritmo di ordinamento (per titolo, autore, valutazione) è incapsulato in una classe strategia separata. Il controller seleziona la strategia in base alla scelta dell’utente, senza condizionali nidificati.
* **Persistenza dell’ordinamento**: la strategia rimane attiva anche dopo l’applicazione di filtri o ricerche, offrendo coerenza nell’esperienza utente.

**Undo/Redo**

* **Command Manager**: un componente dedicato mantiene due stack (undo e redo), gestendo l’esecuzione e il rollback dei comandi in modo centralizzato.
* **Feedback sull’interfaccia**: i pulsanti “Undo” e “Redo” si abilitano/disabilitano in base alla disponibilità di operazioni nel rispettivo stack, fornendo all’utente un’indicazione immediata sulla possibilità di tornare indietro o ripetere.

**2. Requisiti Non Funzionali**

**Usabilità**

* **Interfaccia Swing separata**: la LibroView è dedicata soltanto alla presentazione e all’interazione, con dialog chiari, tool-tips esplicativi e conferme di sicurezza per le operazioni di cancellazioni o pulizia dei dati dalla tabella.
* **Feedback immediato**: ogni azione genera una notifica o un alert entro pochi millisecondi, evitando ambiguità e disorientamento.

**Affidabilità**

* **Validazione esaustiva**: controllo di tutti i campi obbligatori e unicità dell’ISBN prima di ogni modifica.
* **Gestione centralizzata degli errori**: le eccezioni di I/O e di validazione sono catturate e mappate in messaggi user-friendly, mantenendo sempre intatti dati e stato del sistema.

**Manutenibilità**

* **Pattern e interfacce**: l’uso di MVC, Command, Strategy, DAO e Singleton crea moduli autonomi e coesi, che si testano e si aggiornano in modo indipendente.
* **JavaDoc e test unitari**: ogni classe è documentata e coperta da test, garantendo almeno l’80% di coverage sulle parti critiche.

**Performance**

* **Caricamento in memoria**: la libreria viene letta in un’unica volta all’avvio, consentendo ricerche, filtri e ordinamenti in memoria, con tempi di risposta sempre inferiori a 200 ms anche con migliaia di record.
* **Operazioni I/O ottimizzate**: l’uso di buffering e di formati semplici (JSON/CSV) riduce i tempi di lettura/scrittura a pochi secondi anche per dataset di grandi dimensioni.

**Portabilità**

* **Java puro**: l’applicazione funziona identicamente su Windows, macOS e Linux, senza dipendenze native.
* **Zero configurazione**: l’unico prerequisito è una JRE aggiornata (JUnit è presente nativamente in IDE come IntelliJ) e un normale file system per salvare i JSON e i CSV.

**Estensibilità e robustezza**

* **Architettura aperta**: l’aggiunta di nuovi formati di persistenza, criteri di ordinamento o filtri avviene estendendo interfacce già esistenti, senza modificare la logica esistente.
* **Thread safety e rollback**: la combinazione di singleton sincronizzato e blocchi protetti try–catch–finally assicura che, in caso di errore, il sistema torni a uno stato consistente senza perdita di dati.

**3. Metriche di valutazione**

Per verificare che il soddisfacimento dei requisiti sia anche misurabile:

* **Test Coverage**: ≥ 80% su tutti i moduli critici.
* **Tempo di risposta UI**: < 100 ms per operazioni di ricerca/ordinamento su 2000 libri.
* **Caricamento dati**: < 2 s per 5000 record.
* **Affidabilità**: 0 errori critici segnalati in fase di test estensivi.
* **Facilità di estensione**: aggiunta di un nuovo formato o di una nuova strategia avviene in meno di mezza giornata di sviluppo.

Appendix. Prototype

**1. Panoramica delle funzionalità implementate**

All’interno del prototipo sono state realizzate tutte le parti fondamentali definite nei requisiti:

* Gestione completa dei libri: l’utente può aggiungere, modificare, eliminare e visualizzare i libri in una tabella sempre aggiornata.
* Ricerca e filtri avanzati: è possibile cercare per titolo, autore o ISBN e applicare più filtri contemporaneamente (genere, stato di lettura, valutazione), con aggiornamento immediato dei risultati.
* Ordinamento dinamico: l’elenco può essere ordinato per titolo, autore o valutazione, in senso ascendente o discendente, grazie a un’implementazione modulare.
* Persistenza su file: dati salvati e caricati nei formati JSON e CSV, con controlli di integrità e gestione automatica degli errori di I/O.
* Undo/Redo: ogni operazione di aggiunta, modifica o eliminazione può essere annullata e ripetuta grazie al pattern Command.
* Validazione e gestione degli errori: tutti i campi di input sono controllati in tempo reale, e ogni eccezione viene catturata e trasformata in un messaggio chiaro per l’utente.

L’interfaccia, sviluppata con Swing, offre finestre di dialogo pulite e una barra degli strumenti intuitiva, mentre la logica di fondo utilizza i pattern MVC, DAO, Strategy, Command e Singleton per garantire robustezza e flessibilità.

**2. Copertura dei requisiti funzionali**

| **Requisito** | **Stato di implementazione** |
| --- | --- |
| CRUD libri | ✓ Aggiunta, modifica, eliminazione e visualizzazione |
| Ricerca e filtri | ✓ Ricerca parziale, filtri multipli, reset con un click |
| Ordinamento | ✓ A–Z/Z–A per titolo e autore,  Da valutare–5/5–Da valutare per valutazione |
| Persistenza | ✓ Salvataggio e caricamento JSON/CSV |
| Undo/Redo | ✓ Annullamento e ripristino di tutte le operazioni |

**3. Copertura dei requisiti non funzionali**

* Usabilità: l’interfaccia è immediata e autoesplicativa; dialog e tool­tips guidano l’utente passo dopo passo.
* Affidabilità: i controlli sui campi e la gestione centralizzata delle eccezioni evitano crash e mantengono il dato consistente.
* Performance: tutte le operazioni (ricerca, filtro, ordinamento) rispondono in meno di 200 ms su lunghe liste; il caricamento e il salvataggio di 1000 libri avviene in pochi secondi.
* Manutenibilità: grazie alla separazione MVC e ai pattern utilizzati, è semplice aggiungere nuove funzionalità o correggere parti specifiche del codice.
* Portabilità: il prototipo è scritto in Java puro e funziona senza modifiche su Windows, macOS e Linux, richiedendo solo una JRE aggiornata.
* Estensibilità: l’architettura a interfacce permette di integrare rapidamente nuovi formati di persistenza o criteri di ordinamento, senza riscrivere il controller o la view.

**4. Correttezza e dimostrazione del prototipo**

Per dimostrare che il prototipo aderisce ai requisiti, sono stati predisposti:

* Test unitari: coprono tutti i modelli (es. validazione Libro), i controller, i DAO (JSON/CSV), le strategie di ordinamento e il CommandManager.
* Test di integrazione: simulano flussi completi di operazioni (CRUD + undo/redo + persistenza + UI), verificando che ogni funzionalità interagisca correttamente con le altre.
* Metriche di qualità:
  + Coverage > 80 % su codice critico
  + Tempo di risposta UI sempre inferiore a 100 ms
  + Assenza di memory leak in test prolungati

**5. Schermate di esempio**

1. Finestra principale
   * In alto la barra di ricerca o per autore, o titolo o isbn, nonché la sezioni filtri e ordinamento.
   * Al centro la tabella ordinabile e filtrabile in tempo reale.
   * In basso la barra degli strumenti con i pulsanti per aggiungere, modificare, eliminare, annullare e ripetere, salvare, caricare e pulire la libreria.
2. Dialog di aggiunta/modifica
   * Campi chiaramente etichettati.
   * Pulsanti “Conferma” e “Annulla”, dove in caso di click su conferma viene effettuata la validazione e in caso di esito positivo aggiunta/modifica del dato, e click su annulla viene annullata l’operazione.
   * Messaggi di errore contestuali (es. ISBN duplicato o valutazione fuori range).
3. Dialog di eliminazione/pulizia
   * Campi chiaramente etichettati.
   * Messaggi di warning che chiedono conferma dell’operazione all’utente prima di procedere.
   * Nel caso di pulizia libreria l’operazione è irreversibile. Mentre l’eliminazione può essere annullata.
4. Dialog di salvataggio/caricamento
   * Campi chiaramente etichettati.
   * Aperta una nuova finestra che chiede dove salvare e con che nome o da dove caricare il dato.
   * Messaggio di conferma del caricamento, o messaggio dettagliato dell’errore riscontrato.