

Tabella Coesione e Accoppiamento

CLASSE	LIVELLO DI COESIONE	LIVELLO DI ACCOPPIAMENTO	MOTIVAZIONE
Utente	Funzionale	Per Dati	Tutti i metodi operano sugli stessi dati dell'entità utente contribuendo a una singola funzionalità. Nessuna dipendenza esterna.
ControlloreRegUtente	Procedurale	Per Controllo	Tutti i metodi contribuiscono alla funzionalità di registrazione utenti. Comunica con altre classi tramite chiamate a funzioni statiche e non.
ControlloreVisUtenti	Procedurale	Per Controllo	Tutti i metodi contribuiscono alla funzionalità di visualizzazione e gestione utenti. Comunica tramite chiamate a funzioni statiche e non.
UtenteInvalidoException	Logica	Per Dati	Classe base per gestire eccezioni di validazione utente (operazioni logicamente simili). Nessuna dipendenza.

UtenteDuplicatoException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri.
UtenteMailException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri.
UtenteNomeCognomeException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri.
UtentePrestitoAttivoException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri..
Libro	Funzionale	Per Dati	Tutti i metodi operano sugli stessi dati dell'entità libro contribuendo a una singola funzionalità. Nessuna dipendenza esterna.

ControlloreRegLibro	Procedurale	Per Controllo	Tutti i metodi contribuiscono alla funzionalità di registrazione libri. Comunica con altre classi tramite chiamate a funzioni statiche e non.
ControlloreVisLibri	Procedurale	Per Controllo	Tutti i metodi contribuiscono alla funzionalità di visualizzazione e gestione libri. Comunica tramite chiamate a funzioni statiche e non.
LibroInvalidoException	Logica	Per Dati	Classe base per gestire eccezioni di validazione libro (operazioni logicamente simili). Nessuna dipendenza.
LibroDuplicatoException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri.
LibroDataPubblicazioneException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri.

LibroNumeroCopieException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri.
LibroPrestitoAttivoException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri.
Prestito	Funzionale	Per Dati	Tutti i metodi operano sui dati del prestito contribuendo a una singola funzionalità. Dipende da Utente, Libro e Stato tramite parametri.
ControlloreRegPrestito	Procedurale	Per Dati	Tutti i metodi contribuiscono alla funzionalità di registrazione prestiti. Passa strutture dati complesse (Prestito contenente Utente e Libro).
ControlloreVisPrestiti	Procedurale	Per Dati	Tutti i metodi contribuiscono alla funzionalità di visualizzazione e gestione prestiti. Comunica tramite parametri semplici.

PrestitoInvalidoException	Logica	Per Dati	Classe base per gestire eccezioni di validazione prestito (operazioni logicamente simili). Nessuna dipendenza.
PrestitoDataScadenzaException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri.
PrestitoDuplicatoException	Logica	Per Dati	Gestisce un tipo specifico di errore. Dipende dalla classe base tramite ereditarietà con passaggio parametri.
ComparatoreCognomeNomeUtente	Funzionale	Per Controllo	Classe che implementa la logica di filtraggio degli utenti
ComparatoreTitoloLibro	Funzionale	Per Controllo	Classe che implementa la logica di filtraggio dei libri

ControlloreHome	Procedurale	Per Timbro	Durante l'esecuzione del metodo di salvataggio richiede il contenuto di tutti gli archivi
-----------------	-------------	------------	---

Valutazione Principi di Buona Progettazione e QA

- Per rispettare il principio **DRY** (*Don't Repeat Yourself*), la logica di validazione è stata centralizzata in metodi riutilizzabili e grazie l'utilizzo dell'interfaccia Archiviabile. La classe Utente espone `isNomeCognomeValido()`, `isMailValida()` e `isMatricolaValida()` che vengono chiamati sia durante la registrazione che durante la modifica, evitando duplicazioni. Analogamente, Libro fornisce `isDataValida()` e `isNumCopieValido()` per garantire coerenza nelle verifiche.
- Per rispettare il principio **YAGNI** (*You Aren't Going to Need It*), il sistema implementa solo le funzionalità richieste dall'analisi dei requisiti. Non sono state aggiunte feature speculative come storico modifiche, logging avanzato ,notifiche o grafiche accattivanti. Ogni metodo nei controller ha uno scopo preciso legato a un caso d'uso: `registraUtente()`, `rimuoviUtente()`, `estinguiPrestito()`. Questa scelta mantiene il codice snello e focalizzato sui requisiti effettivi del bibliotecario.
- La distribuzione delle responsabilità è equilibrata tra i moduli del sistema. I controller di registrazione gestiscono l'inserimento dati e la validazione iniziale, mentre i controller di visualizzazione si occupano della presentazione nelle TableView e delle modifiche inline. Le classi Principali (Utente, Libro, Prestito) incapsulano la logica di business e i metodi di confronto. Le eccezioni sono organizzate in gerarchie dedicate per ogni dominio. Questa suddivisione permette modifiche localizzate senza propagare cambiamenti in tutto il sistema.

- L'architettura adotta un **approccio modulare** basato su tre package principali (gestioneutente, gestione libro, gestioneprestito), ciascuno articolato in subpackage tematici. I subpackage registrazione contengono i controller per l'inserimento (ControlloreRegUtente, ControlloreRegLibro, ControlloreRegPrestito), visualizzazione contengono quelli per la consultazione e modifica (ControlloreVisUtenti, ControlloreVisLibri, ControlloreVisPrestiti), mentre eccezioni raggruppano le classi per la gestione degli errori specifici del dominio. Questa organizzazione facilita la navigazione del codice e rende evidenti i confini tra le diverse funzionalità.
- Il sistema rispetta il principio della **singola responsabilità** assegnando a ogni classe un compito specifico. Utente rappresenta l'entità studente con i suoi attributi e validazioni, ControlloreRegUtente gestisce esclusivamente l'interfaccia di registrazione, ControlloreVisUtenti si occupa solo della tabella di visualizzazione e delle operazioni di modifica/cancellazione. Le eccezioni hanno responsabilità ancora più mirate: UtenteDuplicatoException segnala solo matricole duplicate, UtenteMailException solo email non valide. Nessuna classe accumula funzionalità eterogenee che potrebbero renderla difficile da mantenere.
- Il principio **Open/Closed** emerge nelle gerarchie di eccezioni dove ogni classe base (UtenteInvalidoException, LibroInvalidoException, PrestitoInvalidoException) può essere estesa aggiungendo nuove sottoclassi per gestire ulteriori vincoli di validazione. Se domani fosse necessario validare un nuovo campo, basterebbe creare una nuova eccezione che estende la base appropriata, senza toccare controller o logica esistente. I metodi statici di validazione nelle classi principali possono cambiare implementazione interna mantenendo la stessa firma pubblica, garantendo l'adattabilità a scenari futuri

Per quanto riguarda gli attributi di qualità possiamo identificare principalmente:

QA INTERNI:

- **Portabilità:** Il sistema si preoccupa di salvare i dati su un file esterno e garantisce l'utilizzo dell'applicazione anche su due sistemi operativi, oltre a un occhio di riguardo verso l'**adattabilità** grazie al riutilizzo di metodi che può essere riadattato in vista di modelli identificativi diversi

QA ESTERNI:

- **Usabilità:** l'intuitività del sistema e della sua interfaccia grafica permette all'utilizzatore di semplificare le varie operazioni possibili
- **Sicurezza (Safety):** l'incapsulamento e la gestione delle eccezioni controllare permette di evitare accessi ai dati in maniera incontrollata ed eventuali arresti anomali del sistema che porterebbero alla perdita di tutte le modifiche fatte in archivio durante la sessione