Documentazione diagrammi

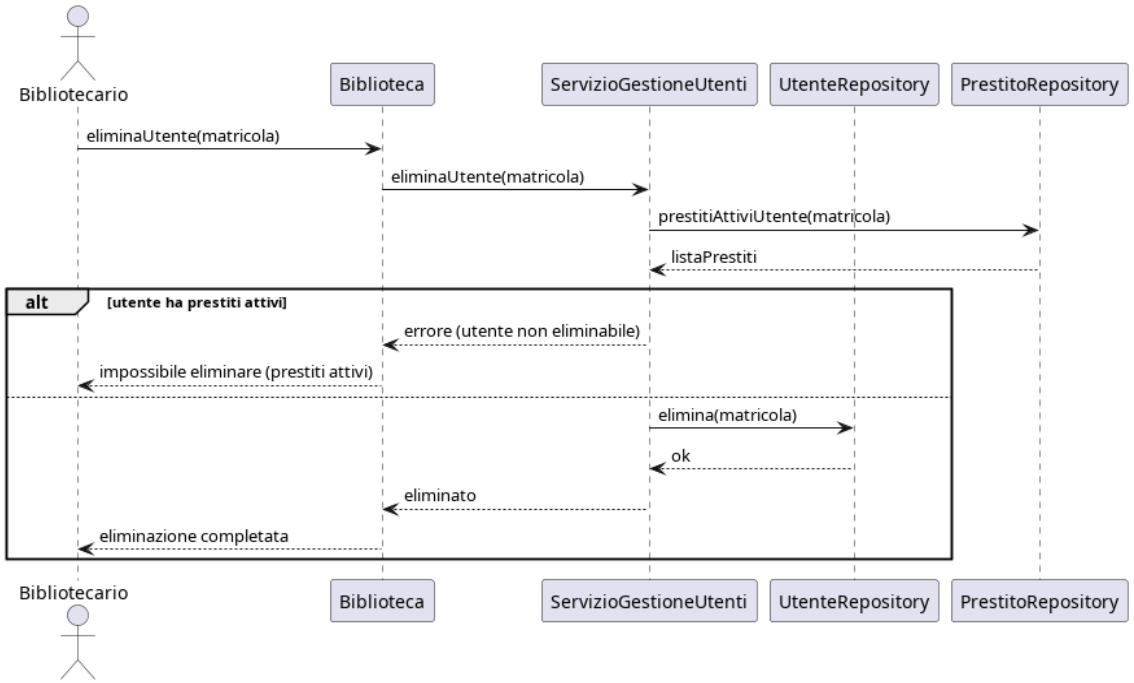
INDICE

Diagrammi di sequenza

Diagramma di classi

Diagrammi di sequenza

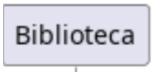
eliminaUtente



SCENARI E PARTECIPANTI:

Il diagramma di sequenza illustra il processo di eliminazione di un utente identificato da matricola da parte dell'*attore* **Bibliotecario,** con una gestione particolare del caso in cui l'utente abbia prestiti attivi(difatti il sistema deve verificare che l'utente non abbia prestiti attivi prima di procedere con l'eliminazione).
I partecipanti coinvolti sono:

attore che inizia l'interazione;



classe di controllo che coordina l'operazione;

classe di servizio che implementa la logica di business;

repository per la persistenza degli utenti;

repository per l'accesso ai dati dei prestiti.

ANALISI DELLA COESIONE:

- ***BIBLIOTECA***

  La classe **BIBLIOTECA** presenta COESIONE FUNZIONALE, il livello più alto nella scala di coesione, in quanto agisce come punto di coordinamento per il caso d'uso, svolgendo un singolo compito ben definito: gestire la richiesta di eliminazione dell'utente da parte del bibliotecario.
  Non include logica di business né accesso ai dati, limitandosi a delegare al livello di servizio appropriato.
  BIBLIOTECA rispetta questo principio, poiché funge da controller nel pattern architetturale.

- ***SERVIZIOGESTIONEUTENTI***

  La classe **SERVIZIOGESTIONEUTENTI** presenta COESIONE FUNZIONALE, in quanto il modulo realizza un unico compito: implementare la logica di business per l'eliminazione degli utenti, verificando che vi sia assenza di prestiti attivi e coordinando le interazioni con i repository.
  La classe rappresenta un esempio di buona progettazione perché si occupa della gestione di una specifica funzionalità di business.
  Tutte le operazioni del modulo sono strettamente correlate alla gestione degli utenti e lavorano sugli stessi dati(matricola dell'utente).

- ***UTENTEREPOSITORY***

  La classe **UTENTEREPOSITORY** presenta COESIONE FUNZIONALE, in quanto il repository incapsula completamente la responsabilità della persistenza degli utenti, fornendo un'interfaccia per le operazioni di accesso ai dati relativi all'Utente.
  Essa si occupa esclusivamente della persistenza e del recupero dei dati dell'Utente.

- ***PRESTITOREPOSITORY***

La classe **PRESTITOREPOSITORY** presenta COESIONE FUNZIONALE e mantiene la stessa qualità progettuale di **UTENTEREPOSITORY.** Questo modulo incapsula la responsabilità della gestione della persistenza e del recupero dei dati relativi all'elemento *Prestito* e si occupa esclusivamente delle operazioni di accesso ai dati per i prestiti.
Nel diagramma di sequenza il modulo espone l'operazione prestitiAttiviUtente che si occupa del recupero dell'elenco dei prestiti associati ad un determinato utente.

ANALISI DELL'ACCOPPIAMENTO:

BIBLIOTECA↔SERVIZIOGESTIONEUTENTI

Livello di accoppiamento: PER DATI

Biblioteca invoca il metodo eliminaUtente(matricola) passando solo l'informazione strettamente necessaria(la matricola dell'utente) e ricevendo in risposta l'esito dell'operazione.
Nel passaggio di informazioni da un modulo ad un altro non sono interessate strutture dati complesse, evitando così un possibile *accoppiamento per timbro.*

SERVIZIOGESTIONEUTENTI↔REPOSITORY

Livello di accoppiamento: PER DATI

Le interazioni tra **SERVIZIOGESTIONEUTENTI** e i repository **(UTENTEREPOSITORY E PRESTITOREPOSITORY)** mostrano un accoppiamento per dati, in quanto funzioni dei repository come prestitiAttiviUtente ed eliminaUtente scambiano con ServizioGestioneUtenti solo le informazioni necessarie(nel nostro caso la matricola).

PRINCIPI DI BUONA PROGETTAZIONE

- ***SINGLE RESPONSIBILITY PRINCIPLE(SRP)***

  Il design rispetta a pieno questo principio, dato che i componenti del codice svolgono un singolo compito ben definito. Infatti:

  - ➢ **BIBLIOTECA** coordina il caso d'uso;
  - ➢ **SERVIZIOGESTIONEUTENTI** implementa la logica dell'operazione;
  - ➢ **UTENTEREPOSITORY** gestisce la persistenza degli utenti;
  - ➢ **PRESTITOREPOSITORY** gestisce l'accesso ai dati dei prestiti;

.

- ***OPEN-CLOSED PRINCIPLE(OCP)***

  Il principio aperto-chiuso è rispettato in quanto le unità software sono aperte all'estensione e chiuse alla modifica. Questo comportamento è possibile osservarlo dal fatto che si potrebbero aggiungere nuovi controlli in **ServizioGestioneUtenti** senza modificare **Biblioteca.**

  Inoltre è possibile cambiare l'implementazione della persistenza senza modificare **ServizioGestioneUtenti.**

  Dunque, l'OCP è rispettato.

- ***LISKOV SUBSTITUTION PRINCIPLE (LSP)***

  Il servizio impiega ii repository tramite operazioni astratte (prestitiAttiviUtente, eliminaUtente).

  Ogni implementazione dei repository può essere modificata senza alterare il comportamento del servizio. Per cui, l'LSP è soddisfatto.

- ***INTERFACE SEGREGATION PRINCIPLE (ISP)***

  Le interfacce sono ben strutturate ed eseguono mansioni ben specifiche:

  - PrestitoRepository si occupa solo delle operazioni relative ai prestiti;
  - UtenteRepository gestisce solo la persistenza degli utenti;

- ServizioGestioneUtenti presenta funzioni necessarie alla gestione utenti.

Non vi sono interfacce grandi o non necessarie. Ogni entità lavora solo su ciò che le compete. L'ISP è ben rispettato.

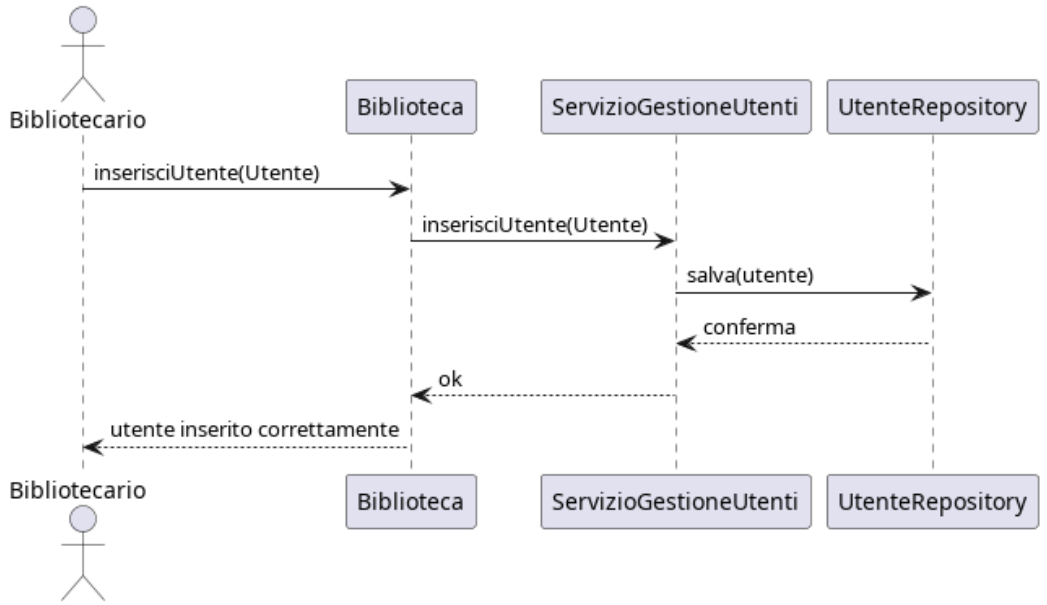
- ***DEPENDENCY INVERSION PRINCIPLE(DIP)***

  I moduli di alto livello di questo sistema dipendono da interfacce (repository), non da moduli inferiori.
  - Il servizio dipende da repository astratti.
  - La Biblioteca dipende dall'interfaccia del servizio, non dalla classe concreta.

  Ciò comporta:
  - possibilità di sostituire/modificare i repository;
  - separazione ben definita tra dominio e infrastruttura.

  Il DIP, dunque, è pienamente rispettato.

inserisciUtente



Coesione

La coesione è ben gestita e ben distribuita tra i partecipanti del diagramma:

- **Biblioteca**: ha un ruolo circoscritto, ossia ricevere la richiesta dall'attore e delegarla al servizio appropriato.Inoltre, non fa logica applicativa né persistenza . Per cui,la coesione è alta;
- **ServizioGestioneUtenti:** gestisce tutta la logica relativa alla creazione o gestione degli utenti. Tutte le operazioni necessarie passano da questo servizio(coesione alta);
- **UtenteRepository:** gestisce la persistenza.Il messaggio salva(utente) è coerente con il suo ruolo (coesione alta);.
- **attore (Bibliotecario):** interagisce solo a livello esterno(coesione elevata).

Ogni partecipante svolge un solo tipo di compito, senza contaminazioni di responsabilità. Questo favorisce chiarezza e manutenibilità.

Accoppiamento

L'accoppiamento è ben gestito:

- Biblioteca → Servizio . Biblioteca chiama direttamente il servizio. L'accoppiamento è moderato. La Biblioteca svolge un ruolo centrale nell'applicazione.
- Servizio → Repository. Il servizio comunica con il repository tramite un'interfaccia. Il servizio dipende da un'astrazione (non da una classe concreta). Rispetta il DIP ed elimina accoppiamenti rigidi.
- Attore → Biblioteca. L'attore dipende solo dall'API pubblica di Biblioteca (accoppiamento minimo).

Principi di buona progettazione

- ***Single Responsibility Principle (SRP).*** Il diagramma mostra una suddivisione di responsabilità coerente:
    - Bibliotecario (attore): invia la richiesta, senza logica interna.
    - Biblioteca: funge da punto di accesso ai servizi, inoltrando la richiesta.
    - ServizioGestioneUtenti: contiene la logica applicativa per la gestione degli utenti.
    - UtenteRepository: gestisce la persistenza dei dati.

  Ogni partecipante del diagramma svolge un compito specifico e distinto, senza sovrapposizioni. Questo rispetta pienamente l' SRP.

- ***Open/Closed Principle (OCP).*** Il flusso è progettato in modo da essere estendibile senza modificare i componenti esistenti:
    - nuove modalità di salvataggio (come file) possono essere aggiunte creando nuove implementazioni del repository;
    - Il servizio rimane invariato perché dipende dall'interfaccia del repository, non dalla sua implementazione.

L’interazione mostrata nel diagramma è quindi compatibile con l’OCP.

- ***Liskov Substitution Principle (LSP).*** Il comportamento del diagramma è coerente con l’ LSP:
    - il ServizioGestioneUtenti si aspetta un oggetto che rispetti il contratto di UtenteRepository.
    - qualsiasi implementazione concreta può essere sostituita (InMemoria).

  Lo schema è quindi conforme con il LSP.

- ***Interface Segregation Principle (ISP).*** L’uso di UtenteRepository come interfaccia dedicata alla gestione degli utenti è un buon esempio di ISP:

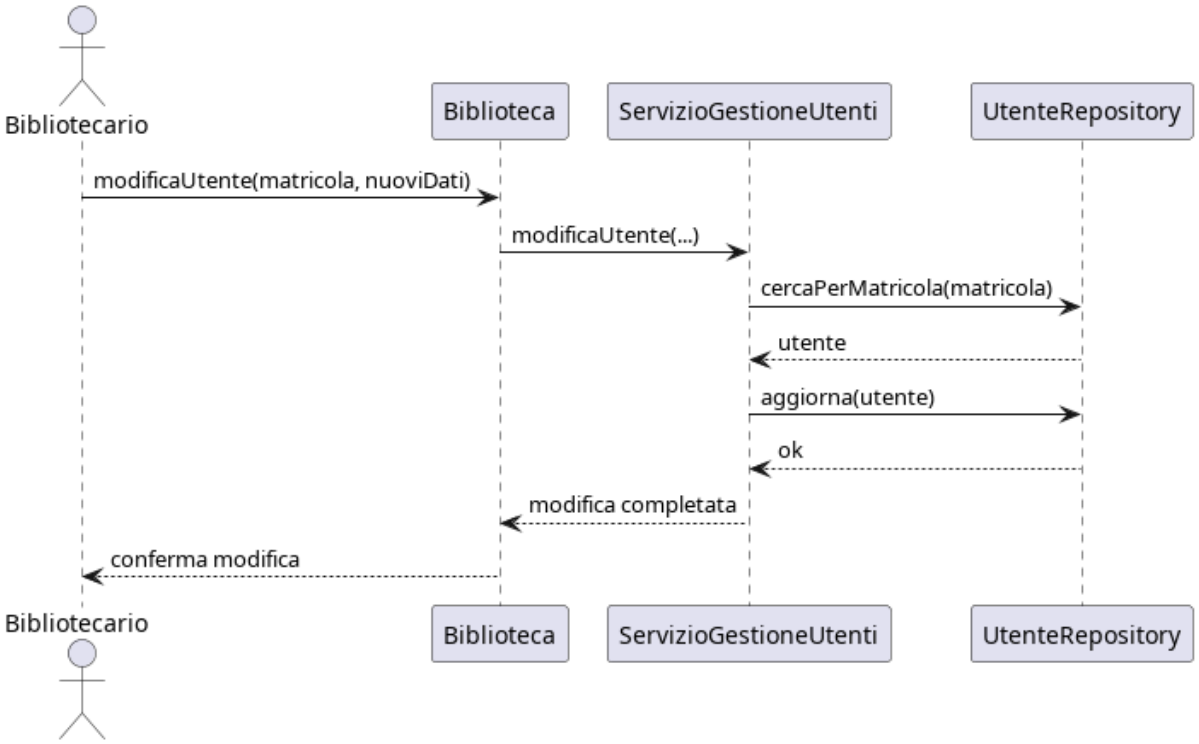
    - il servizio dipende solo da un’interfaccia specifica e non da una generica contenente metodi inutili;
    - il diagramma mostra che il servizio usa un’unica operazione, salva, evitando dipendenze da metodi non necessari.

  Questo evita interfacce troppo grandi e garantisce una dipendenza mirata.

- ***Dependency Inversion Principle (DIP).*** Il diagramma mostra chiaramente l’applicazione del DIP:
    - il ServizioGestioneUtenti comunica con UtenteRepository come astrazione (interfaccia);
    - non c’è dipendenza diretta da un’implementazione concreta;

  Il diagramma di sequenza illustra perfettamente una catena di chiamate in cui i moduli di alto livello dipendono da astrazioni, non da moduli di basso livello.

modificaUtente



Coesione

La coesione è buona in tutte le parti:

- Biblioteca: svolge un ruolo unico, ossia inoltrare le richieste ai servizi appropriati → (coesione alta);
- ServizioGestioneUtenti: contiene tutta la logica che compone l'operazione modificaUtente:
    - trova;
    - aggiorna;
    - richiede salvataggio.

  Queste operazioni fanno parte dello stesso caso d'uso (coesione molto alta).

- UtenteRepository: ha la responsabilità di svolgere operazioni di persistenza legate agli utenti (coesione alta).

Il diagramma descrive un flusso pulito, in cui ogni elemento fa una sola cosa e la fa bene.

Accoppiamento

Il livello di accoppiamento è basso e ben controllato:

- Bibliotecario→ Biblioteca: l'attore usa solo un'operazione pubblica. E' facile da sostituire o estendere;
- Biblioteca → Servizio:accoppiamento moderato.
- Servizio → Repository: è il punto più importante, poiché Il servizio dipende solo da un'interfaccia, non da una classe concreta. Questo permette test unitari, inversione delle dipendenze, sostituzione dei componenti.

Principi di buona progettazione

- ***Single Responsibility Principle (SRP).*** La divisione delle responsabilità è chiara e ben organizzata:
    - Bibliotecario (OP): attore esterno che avvia l'operazione;
    - Biblioteca (B): punto di accesso all'applicazione; delega al servizio corretto;
    - ServizioGestioneUtenti (S): contiene la logica dell'operazione modifica utente, cioè:
        - recupero dell'utente;
        - aggiornamento dei suoi dati;
        - richiesta di salvataggio.

    - UtenteRepository (R): esclusivamente dedicato alla persistenza (ricerca e aggiornamento)

  Ogni partecipante svolge un compito specifico e non sovrapposto. Per cui, il SRP è rispettato pienamente.

- ***Open/Closed Principle (OCP).*** Il flusso permette estensioni senza modifiche strutturali ai componenti:
    - nuove funzionalità di ricerca/aggiornamento possono essere aggiunte creando nuovi metodi nel repository o nuove implementazioni dello stesso;
    - il servizio dipende da un'interfaccia.

  Il sistema è quindi aperto all'estensione e chiuso alla modifica, come l' OCP richiede.

- ***Liskov Substitution Principle (LSP).*** Il servizio interagisce con UtenteRepository come interfaccia astratta:
    - qualsiasi implementazione del repository può sostituire le altre senza alterare il comportamento del servizio;

  Questo costruisce un sistema compatibile con LSP.

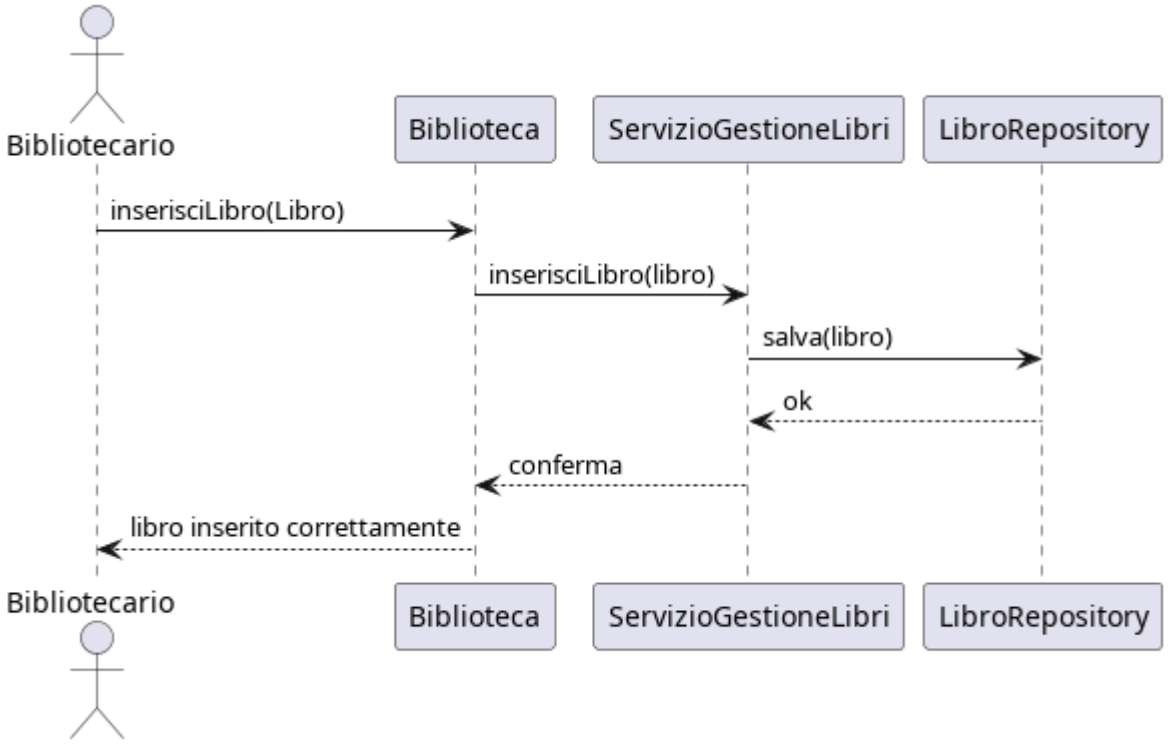
- ***Interface Segregation Principle (ISP).*** Il repository espone solo metodi rilevanti per utenti:
    - trovaUtente;
    - aggiorna.

  Non ci sono metodi superflui o interfacce inutili.
  Il servizio dipende solo dai metodi necessari e questo è proprio ciò che l' ISP richiede.

- ***Dependency Inversion Principle (DIP).*** È uno dei punti più evidenti del diagramma:
    - il servizio non dipende da una classe concreta, ma da un repository astratto (interfaccia);
    - il flusso logico va dai moduli di alto livello (Biblioteca, Servizio) verso quelli di basso livello (Repository), ma le dipendenze sono invertite:
        - il servizio dipende da un'astrazione, non da una classe specifica.

  Questo rende il sistema flessibile, testabile. Quindi il DIP è rispettato.

inserisciLibro



Coesione

La coesione è elevata in tutti i partecipanti:

- Biblioteca: svolge un compito preciso, quello di inoltrare la richiesta al servizio corretto(coesione alta);
- ServizioGestioneLibri: accetta i dati del libro, applica logica di validazione/creazione e richiama il repository (coesione alta);
- LibroRepository: è responsabile solo della persistenza del libro (coesione alta);
- Attore(Bibliotecario): lui inoltra la richiesta (coesione totale).

Accoppiamento

Il diagramma mostra un accoppiamento ottimizzato :

- Bibliotecario→ Biblioteca: qui l'accoppiamento minimo. L'attore usa solo l'interfaccia pubblica;

- Biblioteca → Servizio: l'accoppiamento è moderato. La Biblioteca funge da access point dell'applicazione;
- Servizio → Repository: rappresenta il punto principale. Il servizio dipende da un'interfaccia.

Principi di buona progettazione

- ***Single Responsibility Principle (SRP).*** La separazione dei ruoli è chiara e corretta:
    - Bibliotecario (UA): è l'attore che avvia l'operazione;
    - Biblioteca (B): è il punto unico di accesso ai servizi, inoltrando la richiesta;
    - ServizioGestioneLibri (S): contiene la logica applicativa per creare e validare un nuovo libro;
    - LibroRepository (R): è responsabile della persistenza del libro.

  Ogni partecipante svolge una sola responsabilità senza sovrapposizioni. Per cui, il SRP è rispettato.

- ***Open/Closed Principle (OCP).*** Il sistema è strutturato in modo che i componenti non debbano essere modificati per introdurre nuove funzionalità:
    - si può aggiungere una nuova implementazione del repository senza modificare il servizio;
    - si possono introdurre nuove regole nel servizio senza modificare Biblioteca o il repository;

  Le dipendenze verso interfacce rendono il sistema aperto all'estensione e chiuso alla modifica, proprio come l'OCP richiede.

- ***Liskov Substitution Principle (LSP).*** Il servizio interagisce con LibroRepository come interfaccia:
    - qualunque implementazione (in memoria, su file, su DB) può sostituirne un'altra senza cambiare il comportamento del servizio.

  Ciò garantisce il rispetto dell' LSP.

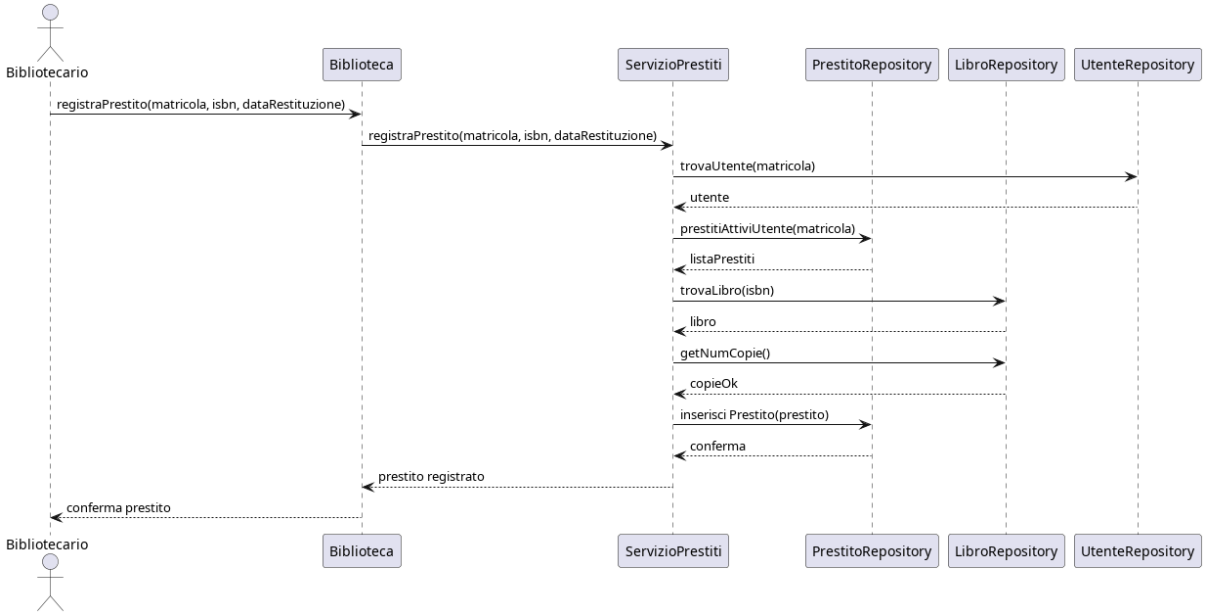
- ***Interface Segregation Principle (ISP).*** Il repository ha una funzione ben specifica:
    - rendere accessibili solo i metodi necessari per la gestione dei libri (in questo caso, salva).

  L'interfaccia non contiene metodi superflui. L'ISP è ben applicato.

- ***Dependency Inversion Principle (DIP).*** Il diagramma esprime il DIP:
    - il servizio dipende da LibroRepository, che è un'astrazione;
    - il servizio ignora l'implementazione concreta;.
    - la logica applicativa rimane indipendente dai dettagli di persistenza.

  Questa separazione garantisce un sistema flessibile e ben manutenibile. Quindi, il DIP è rispettato.

registraPrestito



Coesione

- ServizioPrestiti: questo servizio presenta una coesione funzionale. Esso svolge soltanto le azioni necessarie alla gestione del prestito;
- Repository: ogni repository ha una coesione alta, poiché racchiude tutte le operazioni relative a un unico concetto del dominio:
    - libri;
    - utenti;
    - prestiti.

  La coesione del sistema è dunque alta, con responsabilità e mansioni ben separate.

Accoppiamento

Il sistema si presenta in questo modo:

- Biblioteca si collega con i servizi;
- i servizi conoscono solo interfacce dei repository;
- i repository non conoscono nulla di superiore.

Il ServizioPrestiti dipende dai tre repository, ma questa dipendenza è inevitabile, in quanto deve controllare utente, libro e prestiti.
La dipendenza, inoltre, è funzionale. Per cui, l'accoppiamento è controllato.

Principi di buona progettazione

- ***Single Responsibility Principle(SRP).*** Ogni partecipante svolge una sola mansione:
    - Biblioteca (B): ha il compito di ricevere la richiesta dal bibliotecario e di inoltrarla al servizio correttamente specializzato;
    - ServizioPrestiti (PS): gestisce tutta la logica dei prestiti (controlli, validazioni, verifica delle copie, recupero dei dati necessari e registrazione effettiva del prestito);
    - UtenteRepository (UR): esso si occupa solo di recuperare dati degli utenti;
    - LibroRepository (LR): gestisce i dati dei libri;
    - PrestitoRepository (PR): questo repository memorizza e recupera i prestiti, senza conoscere la logica applicativa.

  Tutte queste classi sono coerenti e non si sovrappongono nelle responsabilità. Dunque, l'SRP è rispettato.

- ***Open/Closed Principle(OCP).*** Il flusso è costruito in modo che, per aggiungere funzionalità (solleciti, limiti), basterebbe estendere o migliorare i comportamenti del ServizioPrestiti, senza modificare Biblioteca o i repository.
    - I repository sono già chiusi a modifiche e aperti a nuove query;
    - PS (ServizioPrestiti) può essere ampliato senza cambiare le interfacce esterne.

Il sistema risulta essere estensibile senza essere riscritto.

- ***Liskov Substitution Principle(LSP).*** Tutti i repository sono usati tramite comportamenti astratti e coerenti:
    - trovaUtente;
    - trovaLibro;
    - salvaPrestito;
    - prestitiAttiviUtente

Ogni implementazione concreta (file, memoria) può sostituire quella attuale senza influire sul flusso.Quindi, il LSP è rispettato.

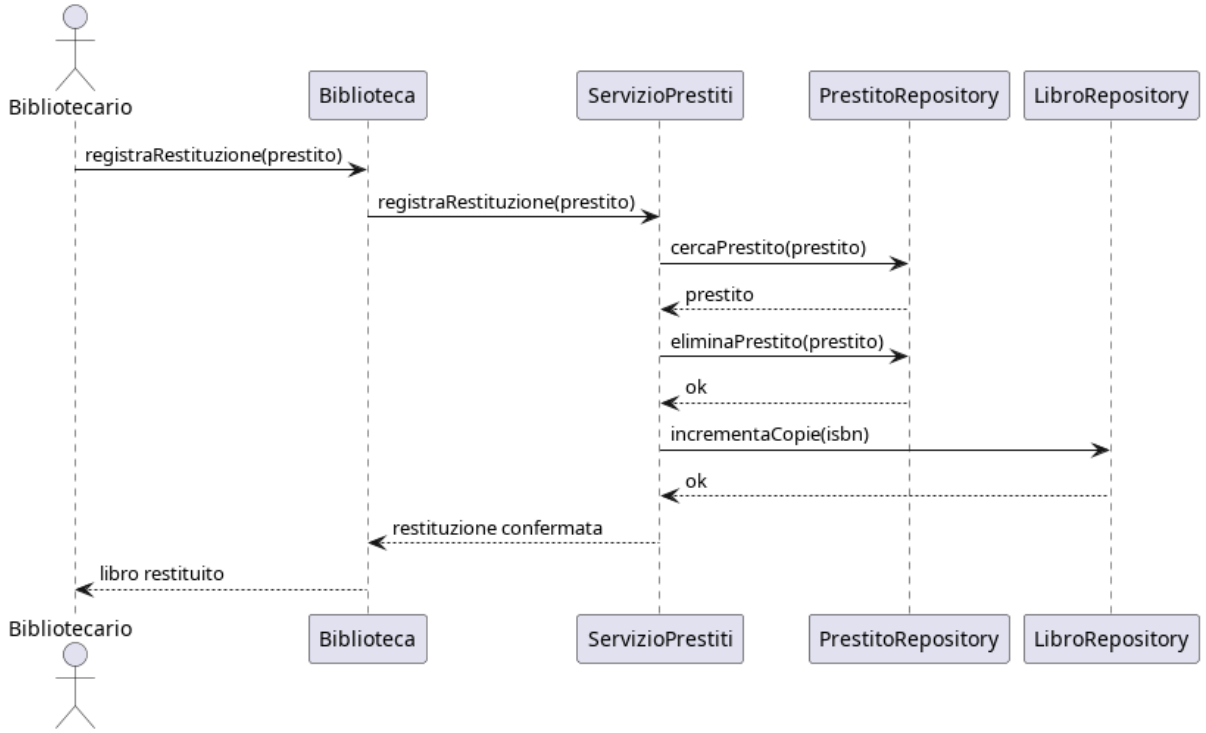
- ***Interface Segregation Principle(ISP).*** Ogni servizio o repository espone solo i metodi essenziali che servono in questo specifico contesto.Non ci sono interfacce generiche:
    - UR (UtenteRepository) gestisce solo metodi utente;
    - LR (LibroRepository) solo i metodi per i libri;
    - PR (PrestitoRepository) si occupa soltanto dei metodi per iprestiti.

  Le interfacce sono quindi pulite e prive di metodi non necessari. L'ISP è rispettato.

- ***Dependency Inversion Principle(DIP).*** La sequenza mostra la logica applicativa (ServizioPrestiti) che dipende da repository astratti, non dall'implementazione reale:
    - PS (ServizioPrestiti) dipende dalle interfacce UtenteRepository, LibroRepository, PrestitoRepository.

  Questo garantisce dei test unitari più semplici ed un minore accoppiamento. Per cui, il DIP è ben rispettato.

registraRestituzione



Coesione

- ServizioPrestiti: vi è una coesione alta, poiché il servizio gestisce un unico caso d'uso, con una sequenza logica chiara (trovare prestito → eliminarlo → incrementare copie libro). Non fa altro.
- Repository: ciascuna repository è dedicata a un'entità del dominio (c'è una buona coesione strutturale).
- Biblioteca: essa svolge una mansione importante nel sistema. Per cui, vi è una coesione appropriata al suo ruolo.

Accoppiamento

Il diagramma presenta vari accoppiamenti:

- Biblioteca conosce solo PS(ServizioPrestiti);
- PS è collegato con interfacce/ruoli dei repository;
- i repository non dipendono da nessuna entità superiore;

- PS dipende da PR (PrestitiRepository) e LR (LibriRepository), poiché la logica della restituzione ne richiede l'uso, ma si tratta di dipendenze necessarie. Qui, l'accoppiamento è funzionale e ben gestito.

Principi di buona progettazione

- ***Single Responsibility Principle (SRP).*** Le responsabilità sono ben distribuite:
    - Biblioteca (B): essa si occupa di comunicare con il bibliotecario. B riceve la richiesta e la inoltra al servizio;
    - ServizioPrestiti (PS): il servizio racchiude la logica della restituzione (verificare l'esistenza del prestito, eliminarlo, aggiornare le copie del libro);
    - PrestitoRepository (PR): l'interfaccia recupera ed elimina un prestito. Essa ha la responsabilità dei dati dei prestiti;
    - LibroRepository (LR): essa aggiorna il numero di copie disponibili del libro.

  Nessuna entità svolge più di un compito. Dunque, l'SRP è rispettato.

- ***Open/Closed Principle (OCP).*** La restituzione è modellata in modo estensibile:
    - si potrà intervenire direttamente su ServizioPrestiti, per aggiungere o eliminare qualcosa, senza toccare Biblioteca o i repository;

    - i repository possono essere ampliati con nuovi metodi, senza intaccare il funzionamento complessivo.

  Il flusso è quindi aperto a estensioni e chiuso a modifiche non necessarie.

- ***Liskov Substitution Principle(LSP).*** Il sistema usa i repository tramite operazioni astratte (cercaPrestito, eliminaPrestito, incrementaCopie). Qualsiasi implementazione concreta (file) può essere sostituita senza cambiare il comportamento del servizio.

  Ciò assicura una corretta sostituibilità e manutenibilità.

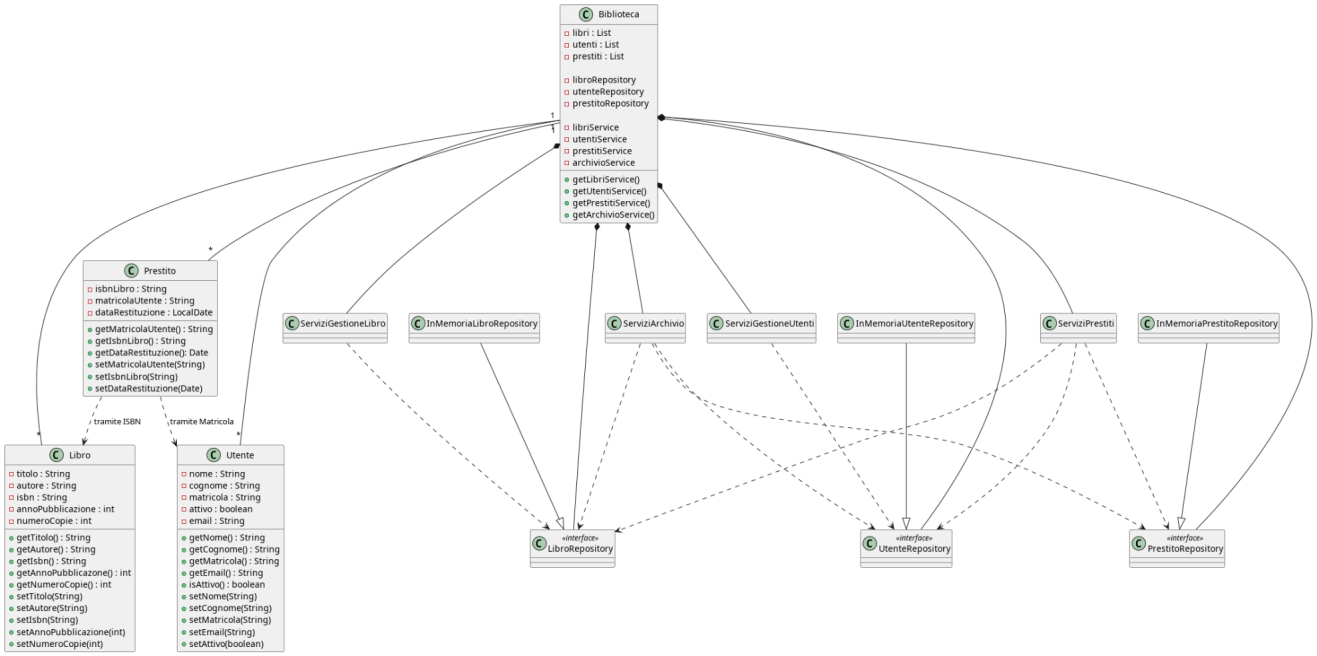
- ***Interface Segregation Principle(ISP).*** Ogni componente espone solo i metodi strettamente utili al proprio ruolo:
    - PR (PrestitiRepository) : gestisce metodi relativi ai prestiti (cerca, elimina);
    - LR (LibroRepository) : presenta metodi riguardanti i libri (come, aggiornamento copie libro);
    - PS (ServizioPrestiti): svolge un'unica operazione ad alto livello (ossia, gestione della restituzione del prestito);
    - B (Biblioteca) → funge da punto d'ingresso, senza avere metodi inutili.

  Le interfacce risultano coerenti e prive di operazioni non richieste. Per cui, l' ISP è rispettato.

- ***Dependency Inversion Principle(DIP).*** Il servizio centrale (PS) dipende da astrazioni, ossia dalle interfacce PrestitoRepository e LibroRepository. Ciò comporta:
    - dipendenza dalle regole del dominio;
    - sostituibilità dei repository senza intaccare la logica applicativa.

  Quindi, il DIP è ben rispettato.

Diagramma di classi



Coesione

Le classi principali (Libro, Utente, Prestito) hanno una coesione alta:

ognuna incapsula in modo pulito i dati e i metodi relativi al proprio dominio.

Sono semplici entità, senza logica, e questo segue il principio di separazione tra dati e comportamento applicativo.

Anche i servizi mostrano buona coesione:

- ServiziGestioneLibri si occupa dei libri;
- ServiziGestioneUtenti si occupa degli utenti;
- ServiziPrestiti gestisce prestiti e dipende sia da libri sia da utenti;;
- ServiziArchivio funge da servizio trasversale, aggregando dati da più repository.

La presenza di repository separati per ciascuna entità rafforza la coesione orizzontale: ogni archivio gestisce un solo tipo di oggetto.

Accoppiamento

Il livello di accoppiamento è ,generalmente, controllato. I servizi dipendono dalle interfacce dei repository, non dalle implementazioni. Questo riduce l'accoppiamento concreto e migliora la sostituibilità. La classe Biblioteca rappresenta un "contenitore" e crea un legame forte (ossia la composizione) con servizi e repository.
Qui, l'accoppiamento è alto e intenzionale: la biblioteca funge da root object del sistema.La relazione Prestito → Libro/Utente è solo indiretta, basata su chiave, e questo mantiene l'accoppiamento molto basso: il prestito non contiene riferimento fisico all'oggetto libro, solo al suo identificativo.

Principi di buona progettazione

- **Single Responsibility Principle (SRP).** Questo principio è rispettato, poiché ogni classe ha un unico motivo per cambiare. Le entità sono semplici, i repository gestiscono persistenza, i servizi gestiscono logica.

- ***Open/Closed Principle (OCP)***. La presenza di repository come interfacce permette di aggiungere nuove implementazioni (come file), senza modificare i servizi.

- ***Liskov Substitution Principle (LSP).*** In base a questo principio, non emergono rischi: le classi InMemoria implementano le interfacce in modo coerente, senza alterare la correttezza del programma.
- ***Interface Segregation Principle (ISP).*** Il principio richiede che le interfacce siano specifiche e non costringano le classi, che le implementano, ad avere metodi inutili. Nel diagramma questo principio è rispettato:
    - le interfacce **LibroRepository**, **UtenteRepository** e **PrestitoRepository** rappresentano responsabilità distinte.
    - ogni repository è dedicato a un solo tipo di entità, evitando un'interfaccia unica e generica troppo ampia che violerebbel' ISP.
    - Le classi **InMemoria** implementano solo i metodi effettivamente necessari alle operazioni del rispettivo tipo.

  In questa maniera, ogni interfaccia rimane coerente e minimale, favorendo una progettazione modulare e facilitando sostituzioni o estensioni future senza introdurre accoppiamento indesiderato.

- ***Dependency Inversion Principle (DIP)***

  Il principio afferma che i moduli di alto livello non devono dipendere da moduli di basso livello ed entrambi devono dipendere da astrazioni (ossia interfacce). Nel diagramma, il principio è applicato correttamente:
    - i servizi (ServizioGestioneLibri, ServizioGestioneUtenti, ServizioPrestiti, ServizioArchivio) dipendono dalle interfacce dei repository, non dalle classi concrete. Questo riduce l'accoppiamento e permette di sostituire le implementazioni senza modificare la logica applicativa.

- le classi InMemoria implementano le interfacce, ma nessun servizio o componente dipende direttamente da esse.Il sistema è quindi aperto alla futura introduzione di repository, rispettando il principio di inversione della dipendenza.