

## Diagramma delle classi

Ogni classe è impegnata alla finalizzazione di un'unica funzione: si vedano ad esempio le classi

- Libro (incaricata di cambiare il numero di copie dei libri);
- Utente (designato a variare il numero di prestiti attivi dell'utente);
- le varie classi che implementano l'interfaccia Gestore.

Grazie a tale progettazione, è possibile dunque raggiungere un livello di coesione elevato. In generale, sono state evitate dipendenze che non fossero essenziali: eccezion fatta per la classe BibliotecaController, che necessita di un gran numero di dipendenze per poter rendere funzionante il programma, presentando accoppiamenti per Dati e per Controllo (verso le classi ValidaData).

## Principi SOLID:

### S- Single Responsibility

Nelle entità (Libro, Utente, Prestito) e nei gestori, ciascuno svolge singole funzionalità (o parti che concorrono a realizzare una singola funzionalità). Fatta eccezione per BibliotecaController.

### O - Open/Closed

Abbiamo definito interfacce generiche come Gestore<T> e ValidaData<T>. Permettendo elevata modularità e testabilità nel codice.

### L - Liskov Substitution

Tutte le implementazioni di Gestore<T> (GestioneLibri, GestioneUtenti, GestionePrestiti) rispettano il contratto delle interfacce.

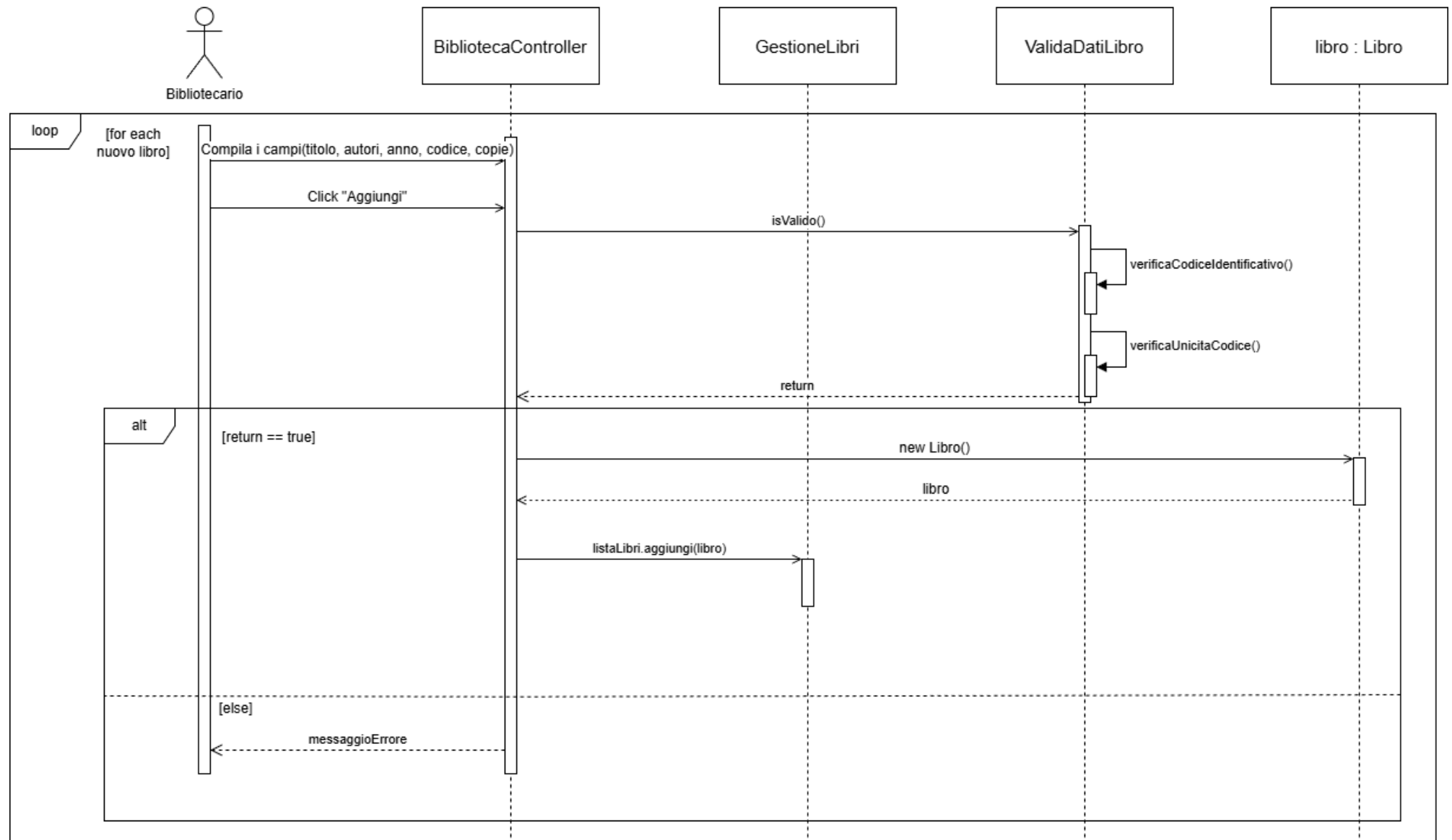
### I - Interface Segregation

ValidaData<T> presenta solo il metodo *isValido()*. Gestore<T>, invece, ha tre metodi, di cui uno potrebbe essere troppo generico (*cerca()*) e non utilizzabile da una delle classi che lo implementano (GestionePrestiti), violando di fatto il principio di segregazione delle interfacce. Quindi per migliorarlo si potrebbe creare una nuova interfaccia.

### D - Dependency Inversion

Gestori e validatori dipendono da astrazioni.

Diagramma di sequenza: Inserimento Libro

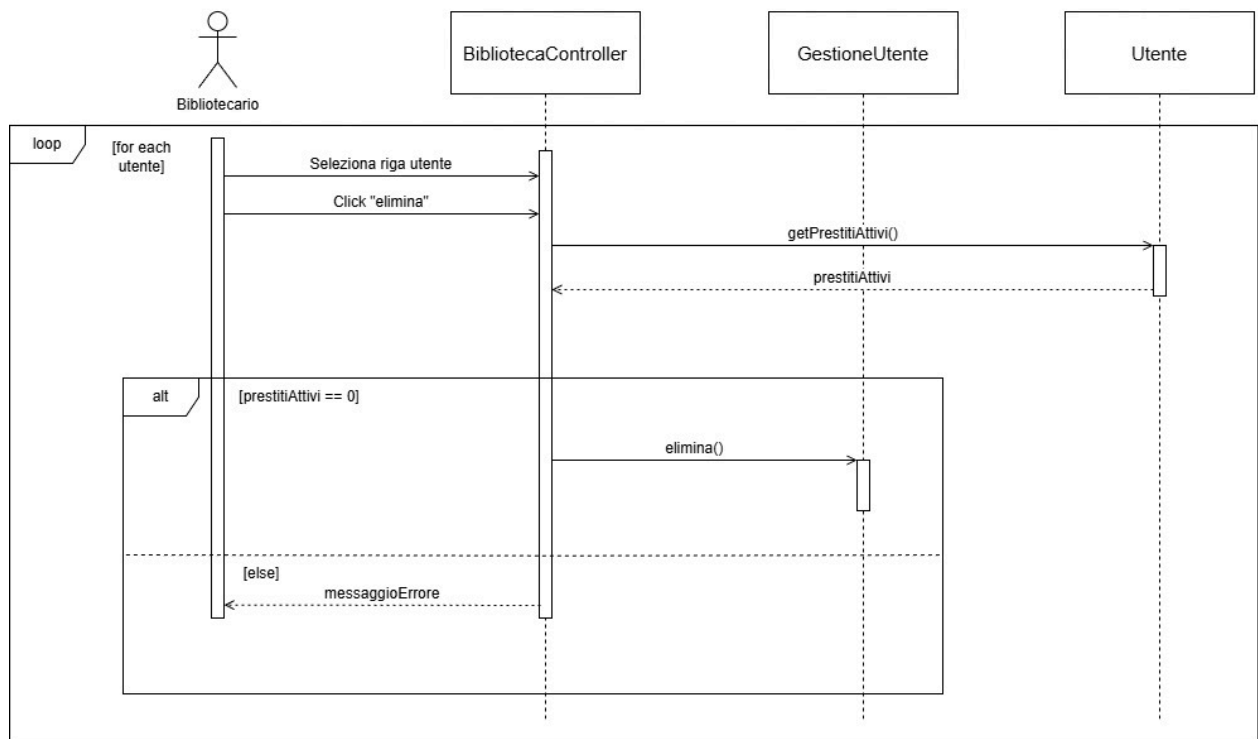


## Inserimento libro

- 1) Inizialmente il Bibliotecario compilerà i vari campi necessari all'inserimento di un libro tramite l'interfaccia utente, resa disponibile da BibliotecaController.
- 2) Il Bibliotecario cliccherà il pulsante "Aggiungi".
- 3) BibliotecaController invocherà il metodo *isValido()* di ValidaDatiLibro, che eseguirà i metodi *verificaCodiceIdentificativo()* e *verificaUnicitàCodice()*. Il valore ritornato da *isValido()* dipenderà dal ritorno di questi.
- 4) Se '*isValido()* == true', BibliotecaController procederà alla creazione di una nuova istanza di Libro, che verrà aggiunta all'elenco di libri già presente.
- 4a) In caso di '*isValido()* == false', BibliotecaController mostrerà al Bibliotecario un messaggio di errore.

*Il processo si ripete per ogni libro il Bibliotecario voglia inserire.*

Diagramma di sequenza: Elimina Utente



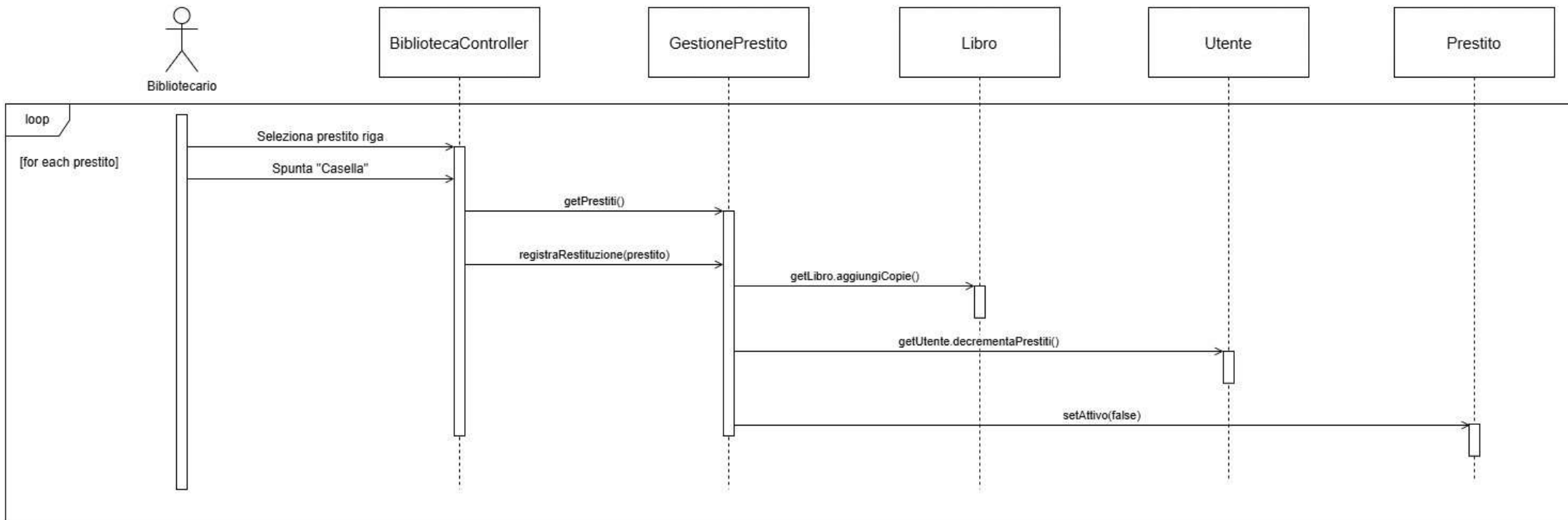
## Elimina utente

- 1) Il Bibliotecario selezionerà la riga dell'elenco in cui è contenuto l'utente da eliminare e cliccherà il pulsante "Elimina".
- 2) BibliotecaController ricaverà da Utente il numero di prestiti attivi dell'utente tramite il metodo *getPrestitiAttivi()*.
- 3) Se l'utente non ha prestiti attivi, BibliotecaController procederà con l'eliminazione dell'utente dall'elenco di utenti attraverso il metodo *elimina()*.

**3a)** Se l'utente ha almeno un prestito attivo, BibliotecaController mostrerà un messaggio di errore al Bibliotecario, bloccando l'operazione.

*Il processo si ripete per ogni utente il Bibliotecario voglia eliminare.*

Diagramma di sequenza: Registra Restituzione

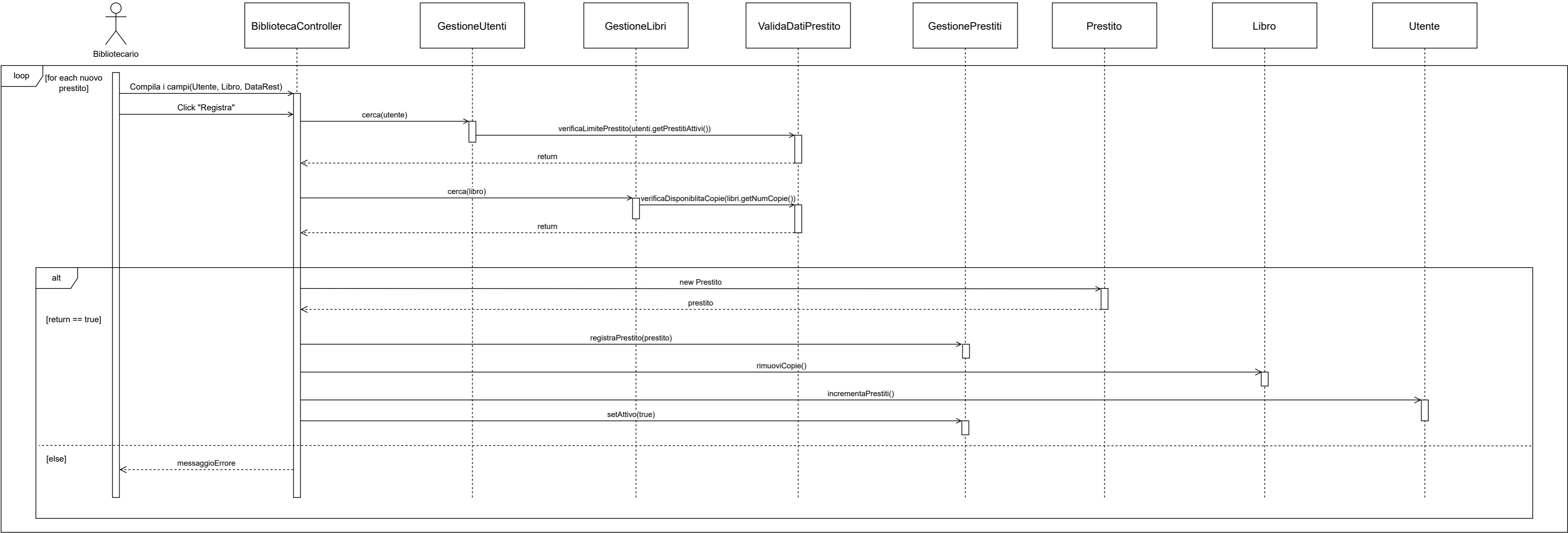


## Registra restituzione

- 1) Il Bibliotecario interagirà con il checkbox relativo al prestito da contrassegnare come restituito.
- 2) BibliotecaController ricaverà da GestionePrestiti la lista di prestiti attivi in modo da poterne invocare il metodo *registraRestituzione()* sul singolo prestito.
- 3) GestionePrestiti eseguirà il metodo, sfruttando *getLibro()* per conoscere il libro di cui incrementare il numero di copie tramite *aggiungiCopie()*, e *getUtente()* per conoscere l'utente di cui decrementare il numero di prestiti attivi tramite *decrementaPrestiti()*.
- 4) GestionePrestiti completerà il processo di restituzione impostando il prestito in questione come "non attivo".

*Il processo si ripete per ogni restituzione avvenuta*

Diagramma di sequenza: Registra Prestito



## Registra prestito

- 1) Inizialmente il Bibliotecario compilerà i vari campi necessari alla registrazione di un prestito tramite l'interfaccia utente, resa disponibile da BibliotecaController, e cliccherà il pulsante "Registra".
- 2) BibliotecaController invocherà il metodo *cerca()* di GestioneUtenti per accertarsi che l'utente in questione esista e GestioneUtenti ne verificherà il numero di prestiti attivi tramite *verificaLimitePrestito()*.
- 3) BibliotecaController eseguirà lo stesso processo anche per il libro da prestare e GestioneLibri ne verificherà il numero di copie disponibili tramite *verificaDisponibilitaCopie()*.
- 4) Se i due controlli restituiranno 'true', BibliotecaController procederà alla registrazione del prestito, decrementerà il numero di copie disponibili per quel libro, incrementerà il numero di prestiti attivi dell'utente e imposterà lo stato del prestito come "attivo".
- 4a) Nel caso in cui uno dei due controlli dovesse restituire 'false', BibliotecaController mostrerà al Bibliotecario un messaggio di errore.

*Il processo si ripete per ogni prestito il Bibliotecario voglia registrare.*