DESIGN PROGETTO BIBLIOTECA

Progetto Ingegneria del Software
A.A. 2025/26

De Riggi Simone
Dell'Orfano Jacopo
Famiglietti Manuel Huy
Fantis Aya

Sommario

1. Diagramma delle classi

Per descrizione dei diagrammi delle classi abbiamo scelto di suddividere i vari diagrammi secondo l'ordine predisposto nei package. Per cui andremo prima a descrivere le relazioni tra le classi appartenenti ad uno stesso package, e infine sarà presentata una discussione sulla relazione tra tutte le classi dell'intero sistema. Questa suddivisione è stata fatta per favorire la leggibilità delle classi e una chiara visione d'insieme. Per lo stesso motivo, la rappresentazione del diagramma completo è stata posta esternamente a questo documento.

1.0. Principi di buona progettazione

L'architettura del sistema, descritta nel documento di design, dimostra una rigorosa applicazione dei principi fondamentali dell'Ingegneria del Software attraverso una decomposizione modulare strategica nei package **entity, archive, service** e **gui**.

Il design fonda la sua solidità strutturale sulla **Separazione delle Preoccupazioni (Separation of Concerns)** e sul **Single Responsibility Principle (SRP)**. Ciò è mostrato nella netta distinzione tra i package: il package **entity** modella esclusivamente lo stato dei dati (alta coesione funzionale) , **archive** gestisce isolatamente la persistenza , **service** incapsula la logica di business scomposta in gestori specializzati , e **gui** è dedicato unicamente all'interfaccia utente.

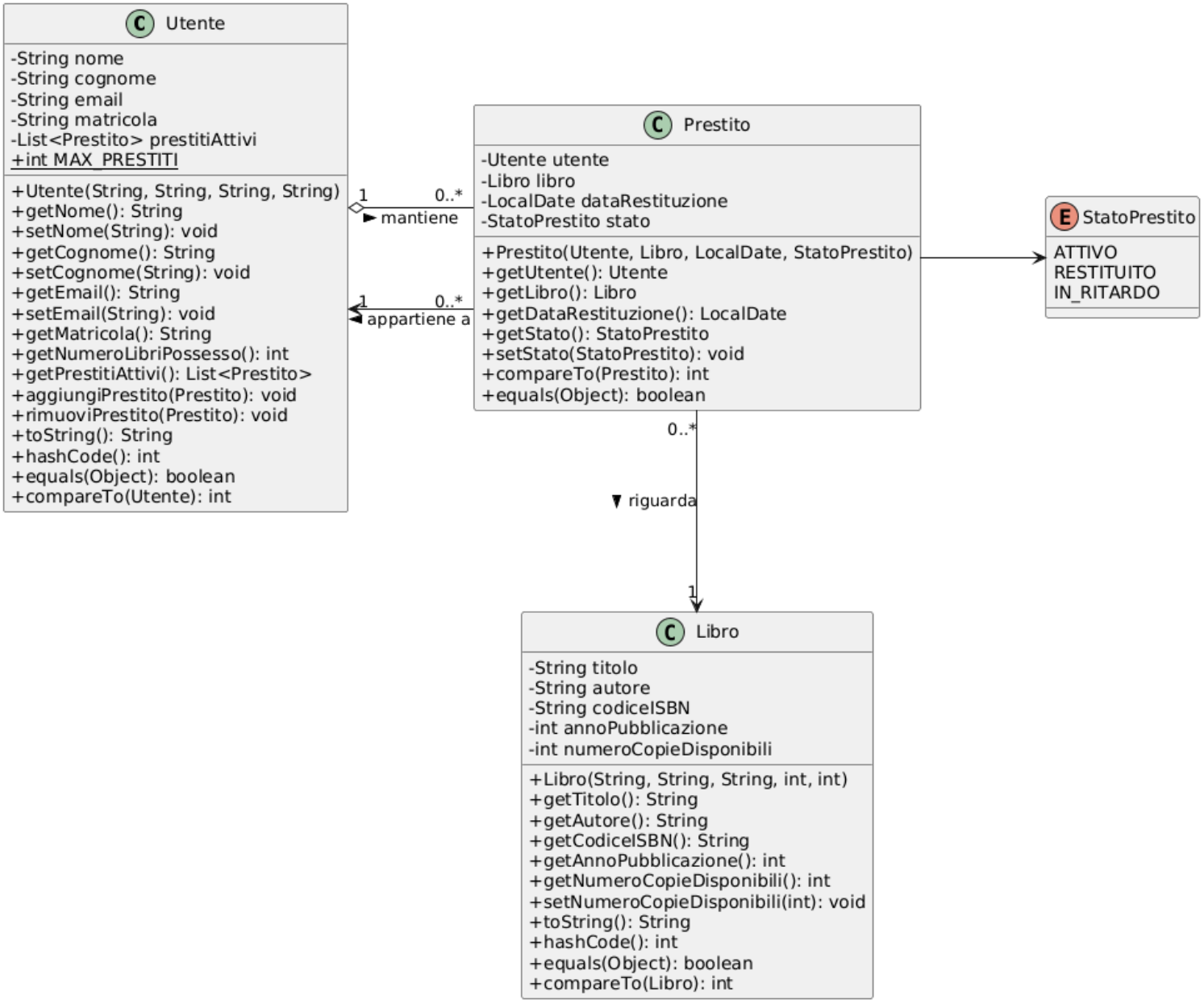
La manutenibilità del codice è garantita dall'applicazione del principio **Don't Repeat Yourself (DRY)** nel livello di persistenza. L'introduzione della classe **ArchiveAstratto** permette di centralizzare la logica ripetitiva di I/O, evitando duplicazioni nelle classi concrete e riducendo il debito tecnico.

La modularità e l'estendibilità del sistema sono assicurate dai principi di **Inversione delle Dipendenze (DIP)** e **Aperto/Chiuso (OCP)**. Attraverso l'uso dell'interfaccia **ArchiveInterfaccia**, i moduli di alto livello (**service**) dipendono da astrazioni e non da implementazioni concrete. Questo disaccoppiamento rende il sistema chiuso alle modifiche della logica esistente ma aperto all'estensione verso nuovi meccanismi di persistenza senza impatti collaterali.

Infine, la struttura privilegia la **Composizione rispetto all'Ereditarietà** per ridurre l'accoppiamento. La classe **Biblioteca** agisce come una Facade che orchestra le operazioni delegando ai vari **Service** (che possiede come attributi privati), mantenendo una struttura flessibile e modulare anziché rigida e gerarchica.

1.1. Package it.unisa.biblioteca.gruppo21.entity

Questo package contiene gli oggetti principali sui quali il nostro sistema vuole operare, ovvero gli oggetti “Utente”, “Libro” e “Prestito”.



1.1.1 Relazioni:

1.1.1.1 **Utente - Prestito**:

Tra la classe *Utente* e la classe *Prestito* c'è una relazione di "**aggregazione**" poiché la classe Prestito rappresenta "una parte" delle caratteristiche che contraddistinguono un utente, ovvero ogni utente può richiedere un prestito. Allo stesso modo un prestito può avere solo un utente associato, da qui la relazione "uno a molti".

1.1.1.2 **Prestito - Prestito**:

In prestito è implementata la funzione "*setStato()*" la quale va a richiamare lo stato del prestito, una variabile di enumerazione (enum). Per cui c'è una relazione di "**associazione**" (unidirezionale).

1.1.1.3 **Prestito - Libro**:

Tra la classe *Prestito* e la classe *Libro* c'è una relazione di "**associazione**" poiché un prestito è definito dal libro che viene prestato. La freccia parte da Prestito poiché il prestito richiede l'oggetto libro, mentre il libro non sa da quale prestito si trova. La relazione è una molti perché ogni prestito si riferisce a un solo libro, ma ogni libro (nel tempo) può avere più prestiti.

1.1.2 Coesione: Funzionale

Ciascuna classe sono unicamente contenitori di dati (POJO), e le sue operazioni sono strettamente necessarie per la gestione del proprio stato interno, con l'unica responsabilità di rappresentare lo stato di una specifica entità del dominio. Quindi, poiché tale modulo contiene le funzionalità che lavorano insieme per realizzare un singolo compito ben definito, gli assegnamo coesione Funzionale.

1.1.3 Accoppiamento:

1.1.3.1 **Accoppiamento: per Timbro tra Prestito e Libro**

Il modulo *Prestito* riceve un intero oggetto *Libro* , con tutti i suoi campi (titolo, autore, ISBN, annoPubblicazione, numeroCopieDisponibili), anche se durante il suo funzionamento ne utilizza solamente alcuni (di solito ISBN o titolo).

1.1.3.2 **Accoppiamento: per Timbro tra Prestito e Utente**

La classe *Prestito* dipende dall'intero oggetto *Utente*, anche se spesso le operazioni di prestito richiedono la matricola.

1.1.3.3 **Accoppiamento: per Timbro tra Utente e Prestito**

La classe *Utente* contiene al suo interno una lista di oggetti di tipo *Prestito*. Questo implica che *Utente* conosce la struttura interna di *Prestito*.

1.1.3.4 **Accoppiamento: Nessuno tra Libro e Prestito**

La classe *Libro* è totalmente indipendente dalla classe *Prestito*.

1.1.3.5 **Accoppiamento: Nessuno tra Libro e Utente**

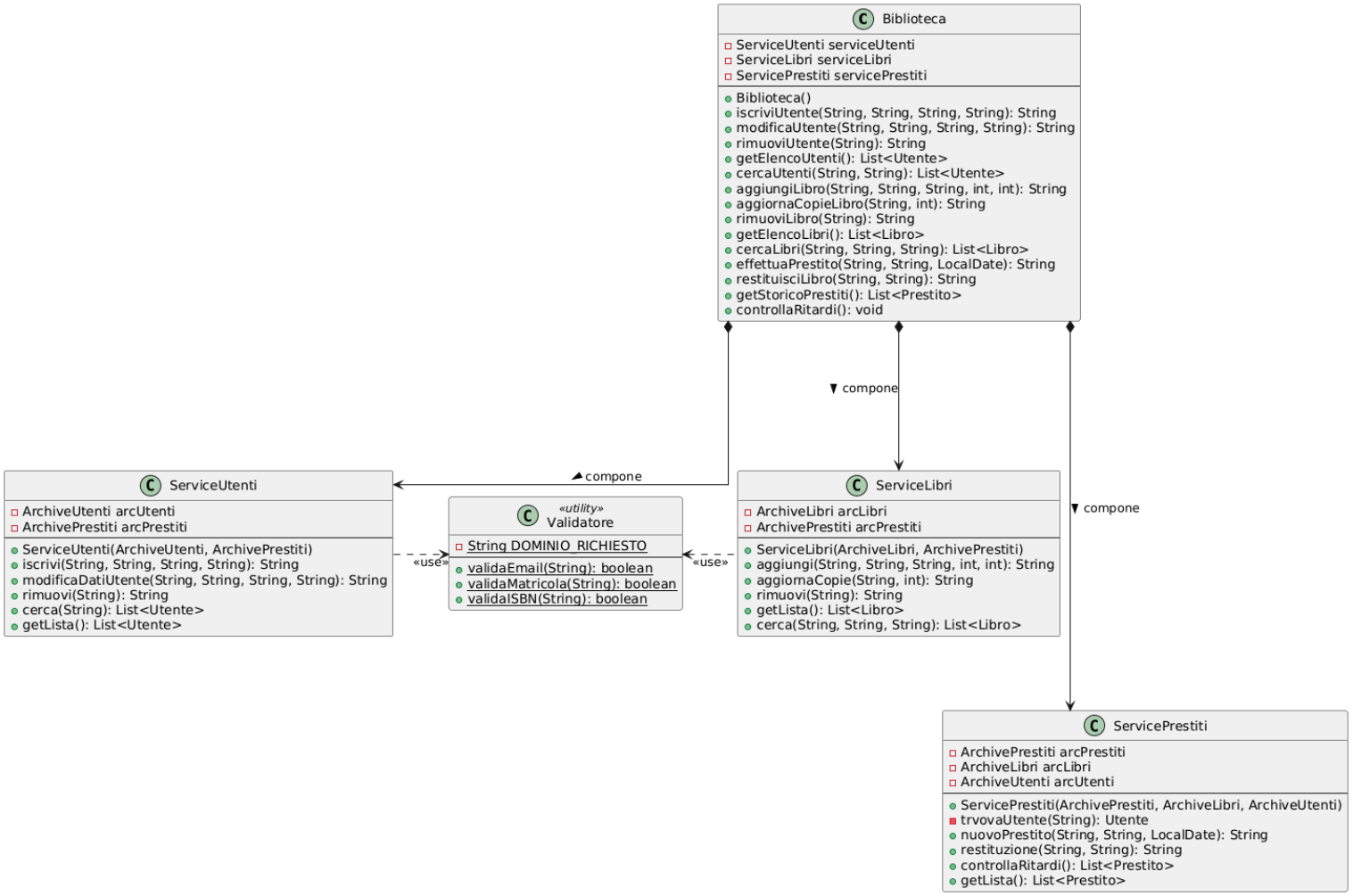
Le due classi sono totalmente indipendenti una dall'altra.

1.1.3.6 **Accoppiamento: per Dati**

In tutte le classi sono presenti metodi semplici che non usano come parametri oggetti complessi, non richiedono l'intera struttura di una classe e che restituiscono valori semplici.

1.2. Package it.unisa.biblioteca.gruppo21.service

Questo package contiene le classi che si occupano della gestione dei dati sul sistema. In particolare si occupa dell'inserimento, della ricerca e della rimozione dei singoli oggetti.
La gestione dei vari servizi viene effettuata dalla classe Biblioteca, la quale si occupa solo di richiamare le azioni interessate.



1.2.1 Relazioni:

1.2.1.1 **Biblioteca - Service:**

Tra la classe *Biblioteca* e i vari service è presente una relazione di "**composizione**" dato che la biblioteca crea gli oggetti dei vari service, i quali nascono e muoiono col suo utilizzo. La molteplicità è univoca dato che la biblioteca possiede una sola istanza per ogni tipo di service.

1.2.1.2 **Service - Validatore:**

Tra i service di utente e libro e la classe *Validatore* è presente una relazione di "**dipendenza**" poiché utilizzano i metodi di quest'ultima solo al momento del bisogno del controllo di dati in ingresso.

1.2.2 Coesione: Funzionale

In quanto tutte le operazioni concorrono a un unico scopo, in questo caso la gestione del catalogo bibliografico. Inoltre, la suddivisione in tre service specializzati garantisce il Principio di Singola Responsabilità (SRP), dove ogni Service ha una coesione funzionale alta, e con la classe Biblioteca anch'essa con coesione funzionale alta in quanto , agendo da Facciata del sistema, ha l'unico compito di delegare le chiamate ai servizi appropriati , nascondendo la complessità.

1.2.3 Accoppiamento:

1.2.3.1 **Accoppiamento: per Timbro tra Biblioteca e ServiceUtenti**

La classe *Biblioteca* dipende dall'intero oggetto *ServiceUtente*. Contiene anche informazioni che non sono necessarie all'altro modulo.

1.2.3.2 **Accoppiamento: per Timbro tra Biblioteca e ServiceLibri**

La classe *Biblioteca* dipende dall'intero oggetto *ServiceLibri*. Contiene anche informazioni che non sono necessarie all'altro modulo.

1.2.3.3 **Accoppiamento: per Timbro tra Biblioteca e ServicePrestiti**

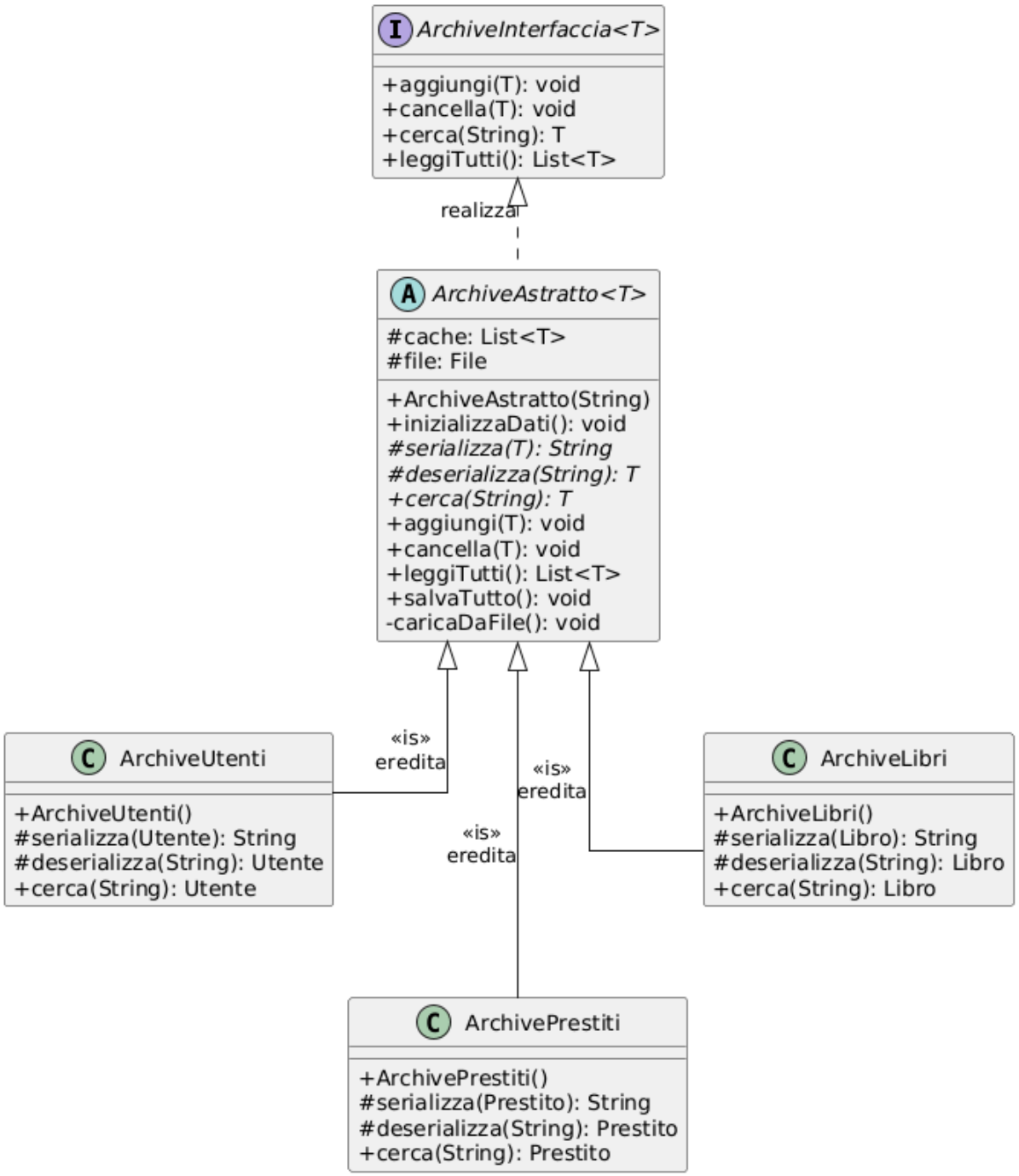
La classe *Biblioteca* dipende dall'intero oggetto *ServicePrestiti*. Contiene anche informazioni che non sono necessarie all'altro modulo.

1.2.3.4 **Accoppiamento: per Dati, Validatore**

I metodi ricevono e restituiscono solo informazioni semplici, senza richiedere l'intera struttura di un oggetto o un servizio.

1.3. Package it.unina.biblioteca.gruppo 21.archive

Questo package contiene le classi che si occupano della gestione dei dati nell'archivio. In particolare si occupa del salvataggio, della ricerca e della modifica dei singoli oggetti. L'interfaccia definisce il contratto per la gestione delle operazioni nei vari service, mentre l'archivio astratto fornisce il meccanismo base dei singoli archivi.



1.3.1 Relazioni:

1.3.1.1 ArchiveAstratto - ArchiveInterfaccia:

Tra la classe *ArchiveAstratto* e *ArchiveInterfaccia* è presente una relazione di "**implementazione**" poiché la classe astratta si occupa di implementare le funzioni definite nell'interfaccia.

1.3.1.2 ArchiveAstratto - Archive:

Tra la classe *Astratta* e le sue specializzazioni (*ArchiveUtenti, ArchiveLibri, ArchivePrestiti*) è presente una relazione di "**ereditarietà**".

1.3.2 Coesione: Funzionale

In quanto tutte le operazioni concorrono alla realizzazione di un singolo compito, ovvero la Persistenza dei dati (I/O). In particolar modo, la funzione ArchiveAstratto presenta un alto livello di coesione in quanto implementa il principio ***DRY(Dont Repeat Yourself)***, centralizzando la logica comune di I/O.

1.3.3 Accoppiamento:

1.3.3.1 Accoppiamento: per Contenuti tra Astratto e Archive Concreti

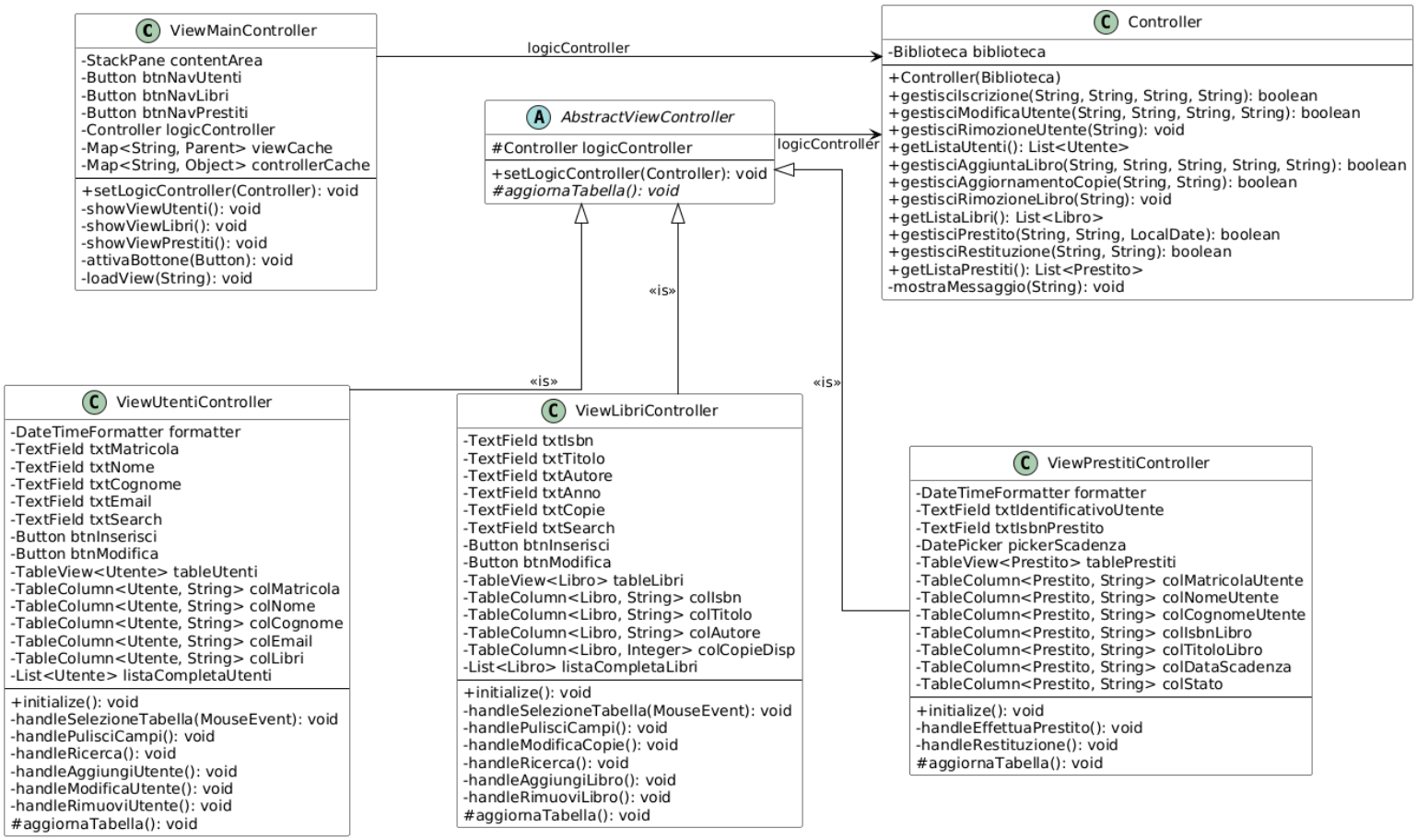
Le classi concrete (*ArchiveUtenti, ArchivePrestiti, ArchiveLibri*) sono fortemente accoppiate alla classe padre *ArchiveAstratto* attraverso una relazione di ereditarietà ("is"). Questo rappresenta un accoppiamento per contenuti poiché le sottoclassi accedono direttamente agli aspetti implementativi interni e agli attributi protetti del padre (come la lista *cache* e il *file* visibili nel diagramma), violando l'incapsulamento.

1.3.3.2 Accoppiamento: per Timbro nei metodi tra Interfaccia e Astratto

I metodi definiti in *ArchiveInterfaccia* e implementati in *ArchiveAstratto* (come *aggiungi(T elemento)* o *serializza(T t)*) presentano un accoppiamento per timbro. Essi non ricevono singoli dati primitivi, ma intere strutture dati complesse (gli oggetti generici *T* che diventeranno *Utente*, *Libro*, ecc.) per svolgere le operazioni di persistenza.

1.4. Package it.unisa.biblioteca.gruppo21.gui

Questo package contiene le classi che si occupano dell'interfaccia grafica con cui il bibliotecario interagisce per operare sul sistema.



1.4.1 Relazioni:

1.4.1.1 **SubView - AbstractViewController:**

Tra le classi *ViewUtentiController, ViewLibriController, ViewPrestitiController* e la classe *AbstractViewController* è presente una relazione di "**ereditarietà**" poiché le *SubView* si occupano di implementare i metodi ereditati dalla classe *abstract*.

1.4.1.2 **ViewMain - Controller / AbstractViewController - Controller:**

La relazione tra queste classi è di "**associazione**" poiché hanno un riferimento alla classe controller per delegare le operazioni. La relazione è di tipo unidirezionale: la GUI conosce il controller ma il controller non conosce la GUI specifica.

1.4.2 Coesione:

1.4.2.1 **ViewUtenti, ViewLibri, ViewPrestiti: Funzionale**

Le funzionalità all'interno di queste classi lavorano sugli stessi dati e contribuiscono ad un singolo compito, ovvero gestire l'interfaccia corrispondente.

1.4.2.2 **Controller: Funzionale**

Raggruppa i metodi necessari per gestire le richieste della GUI verso il sistema *Biblioteca*. Ha il solo compito di intermediario verso il modello.

1.4.3 Accoppiamento:

1.4.3.1 **Accoppiamento: per Dati tra SubView e Controller**

Le classi di *View* e la classe *Controller* si passano solo i parametri necessari per le operazioni (stringhe o booleani), senza passare alla struttura dati complesse.

1.4.3.2 **Accoppiamento: per Controllo tra ViewMainController e SubView**

La classe *ViewMainController* invoca esplicitamente i metodi di caricamento e visualizzazione controllando l'esecuzione delle classi dipendenti.
Inoltre la classe *ViewMainController* influenza direttamente la logica interna attraverso il metodo *setLogicController( )*.

1.5 Biblioteca_Project

Da qui descriveremo i collegamenti presenti tra le classi esternamente ai loro package, concentrandoci sulle relazioni e gli accoppiamenti non ancora descritti.

1.5.1 Relazioni:

1.5.1.1 **Biblioteca - Controller**:

Tra la classe *Controller* e la classe *Biblioteca* è presente una relazione di "**associazione**" unidirezionale. Il controller possiede un riferimento all'istanza della Biblioteca per poter invocare i metodi. La molteplicità è uno a uno poiché una specifica istanza dell'interfaccia grafica gestisce una sola istanza della biblioteca.

1.5.1.2 **Service - Archive**:

Tra le classi dei service (es. *ServiceUtenti*) e quelle dell'archive (es. *ArchiveUtenti*) è presente una relazione di "**associazione**". Ogni service possiede un riferimento all'archive corrispondente (o a più archive come nel caso di *ServicePrestiti*). La molteplicità è uno a uno poiché il service ha bisogno di accedere a quella istanza specifica dell'archivio per funzionare.

1.5.1.3 **Service - Entity**:

Tra le classi service e le classi entità(*Libro, Prestito, Utente*) c'è una relazione di "**dipendenza**". I service utilizzano solo momentaneamente le entità per la validazione o il passaggio e il ritorno dei parametri nei loro metodi.

1.5.1.4 **Archive - Entity:**

Tra le classi archive e le classi entità(*Libro, Prestito, Utente*) c'è una relazione di "**associazione**". Le entità vengono usate per istanziare nuovi oggetti durante la lettura del file (deserializzazione) e per leggere i dati dell'oggetto durante la scrittura su file (serializzazione). Il riferimento all'oggetto quindi viene mantenuto.

1.5.2 Accoppiamento:

1.5.2.1 **Accoppiamento: per Dati tra Controller e Biblioteca**:

La classe *Controller* interagisce con la classe *Biblioteca* scambiandosi esclusivamente parametri necessari per l'esecuzione delle operazioni descritte(es. stringhe per matricola, ISBN, dati anagrafici). Non vengono passate strutture dati complesse, mantenendo l'accoppiamento al livello più basso.

1.5.2.2 **Accoppiamento: per Timbro tra Service e Archive**:

I vari service invocano i metodi degli archive passando come parametri intere strutture dati con entità come istanze (es. "aggiungi(Utente u)"). Poiché viene passato l'intero oggetto "strutturato", questo rientra nella definizione di accoppiamento per timbro.

1.5.2.3 **Accoppiamento: per Timbro tra Service ed Entità**:

Le classi Service dipendono dalla struttura delle classi Entità (Libro, Prestito, Utente) poiché le istanziano direttamente (es. new Utente(...)) e le restituiscono sotto forma di liste (es. List<Utente>). Poiché i moduli si scambiano strutture dati complete, l'accoppiamento è per timbro.

1.5.2.4 **Accoppiamento: per Timbro tra Archive ed Entità**:

Le classi Archive ricevono in input e restituisce in output intere istanze delle Entità (es. metodi aggiungi(T elemento) o T deserializza(String riga)). L'archivio opera sulla struttura dati completa dell'entità per effettuare la serializzazione e il salvataggio su file, creando un accoppiamento per timbro.