ODD
Object Design Document

BiblioNet

Riferimento	C07_ODD_ver.1
Versione	1.0
Data	19/01/2021
Destinatario	Prof.ssa Filomena Ferrucci
Presentato da	C07 Team BK
Approvato da	Stefano Lambiase

Revision History

Data	Versione	Descrizione	Autori
30/10/2020	0.1	Prima stesura	SL
16/12/2020	0.2	Creazione package diagrams e stesura paragrafo 2	Tutto il team
17/12/2020	0.3	Stesura class interfaces per i package autenticazione, prenotazione libri e comunicazione con esperto	AC, CM, GT
17/12/2020	0.4	Stesura class interfaces per i package club del libro, gestione eventi e preferenze di lettura	AP, VP, GV
17/12/2020	0.5	Stesura class interfaces per i package registrazione e questionario di supporto	LT, NP
25/12/2020	0.6	Completamento capitolo 1	VP, GV
13/01/2021	0.7	Aggiornamento della sezione Packages	CM, GT
14/01/2021	0.8	Aggiornamento class interfaces e scrittura glossario ed eliminazione della sezione delle convenzioni.	CM, GT
18/01/2021	0.9	Aggiunto link al sito javadoc	SL
19/01/2021	1.0	Revisione	ADP

Team members

Nome	Ruolo nel progetto	Acronimo	Informazioni di contatto
Stefano Lambiase	Project Manager	SL	s.lambiase7@studenti.unisa.it
Gianmario Voria	Team Member	GV	g.voria6@studenti.unisa.it
Ciro Maiorino	Team Member	CM	c.maiorino7@studenti.unisa.it
Alessio Casolaro	Team Member	AC	a.casolaro2@studenti.unisa.it
Giulio Triggiani	Team Member	GT	g.triggiani@studenti.unisa.it
Antonio Della Porta	Team Member	ADP	a.dellaporta26@studenti.unisa.it
Viviana Pentangelo	Team Member	VP	v.pentangelo4@studenti.unisa.it
Nicola Pagliara	Team Member	NP	n.pagliara1@studenti.unisa.it
Luca Topo	Team Member	LT	l.topo@studenti.unisa.it

Sommario

- Aggiungere il capitolo del "Class Diagram" tra 3. e 4.
- Aggiungere gli "Object Trade-Off" tra 1.1. e 1.2.

1 Introduzione

BiblioNet si propone di semplificare le interazioni tra biblioteche e lettori, al fine di rinvigorire il settore bibliotecario italiano creando uno strumento di comunicazione con persone interessate alla lettura.

In questa prima sezione del documento, verranno descritti i trade-offs e le linee guida per la fase di implementazione, riguardanti la nomenclatura, la documentazione e le convenzioni sui formati.

1.1 Object design goals

Riusabilità:

Il sistema Biblionet deve basarsi sulla riusabilità, attraverso l'utilizzo di ereditarietà e design patterns.

Robustezza:

Il sistema deve risultare robusto, reagendo correttamente a situazioni impreviste attraverso il controllo degli errori e la gestione delle eccezioni.

Incapsulamento:

Il sistema garantisce la segretezza sui dettagli implementativi delle classi grazie all'utilizzo delle interfacce, rendendo possibile l'utilizzo di funzionalità offerte da diversi componenti o layer sottoforma di black-box.

1.2 Linee guida per la documentazione dell'interfaccia

Le linee guida includono una lista di regole che gli sviluppatori dovrebbero rispettare durante la progettazione delle interfacce. Per la loro costruzione si è fatto riferimento alla convenzione java nota come **Sun Java Coding Conventions** [Sun, 2009].

Link a documentazione ufficiale sulle convenzioni

Di seguito una lista di link alle convenzioni usate per definire le linee guida:

- **Java Sun**: https://checkstyle.sourceforge.io/sun_style.html
- **HTML**: https://www.w3schools.com/html/html5_syntax.asp

Aggiungere la sezione sugli "Object Trade-off"

1.3 Definizioni, acronimi, e abbreviazioni

Vengono riportati di seguito alcune definizioni presenti nel documento:

- **Package**: raggruppamento di classi, interfacce o file correlati;
- **Design pattern**: template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità;
- **Interfaccia**: insieme di signature delle operazioni offerte dalla classe;
- **View**: nel pattern MVC rappresenta ciò che viene visualizzato a schermo da un utente e che gli permette di interagire con le funzionalità offerte dalla piattaforma;

- **lowerCamelCase**: è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione nel mezzo della frase inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi;
- **UpperCamelCase**: è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi;
- **Javadoc**: sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile.

1.4 Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

- Statement Of Work;
- Business Case;
- Requirements Analysis Document;
- System Design Document;
- Object Design Document;
- Test Plan;
- Matrice di tracciabilità;
- Manuale di installazione;
- Manuale utente;

Di seguito il link al sito contenente il javadoc di BiblioNet

- Javadoc di BiblioNet

2 Packages

In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali prese e ricalca la struttura di directory standard definita da Maven.

- **.idea**
- **.mvn**, contiene tutti i file di configurazione per Maven
- **src**, contiene tutti i file sorgente
    - **main**
        - **java**, contiene le classi Java relative alle componenti Control e Model
        - **resources**, contiene i file relativi alle componenti View
            - **static**, contiene i fogli di stile CSS e gli script JS
            - **templates**, contiene i file HTML renderizzati da Thymeleaf

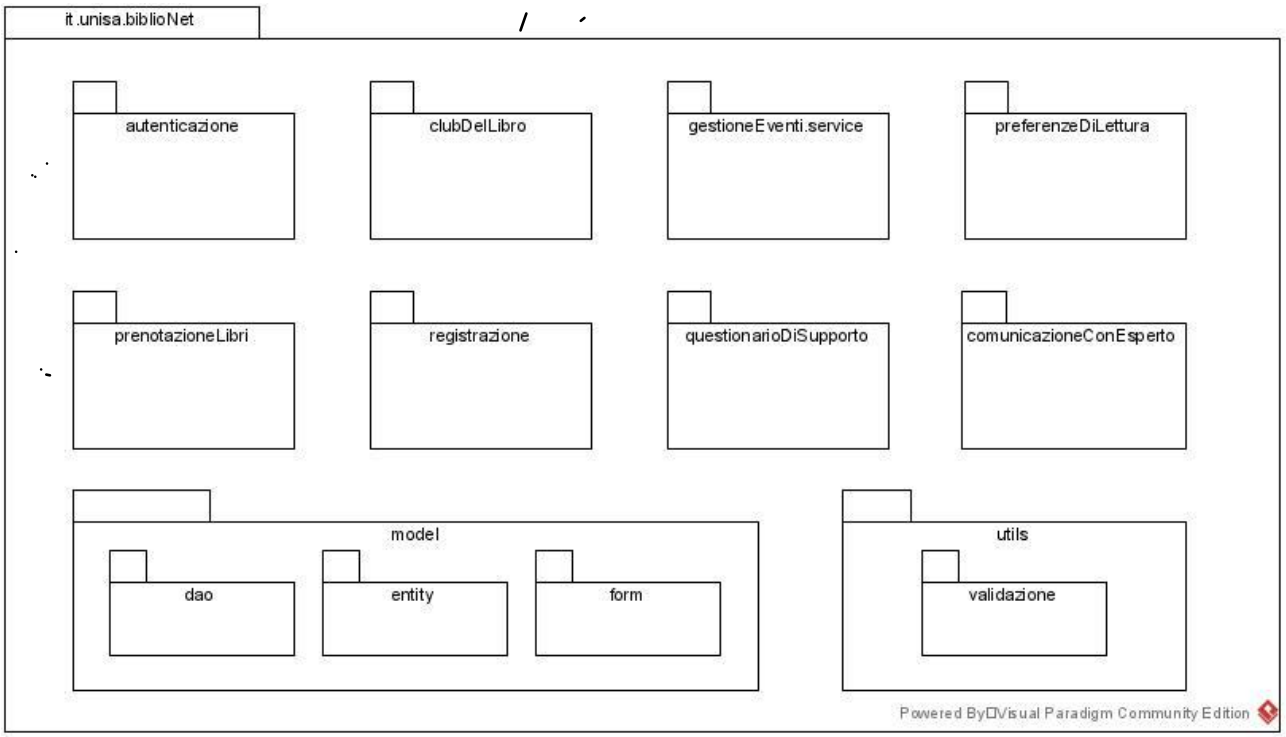
- **test**, contiene tutto il necessario per il testing
  - **java**, contiene le classi Java per l'implementazione del testing
- **target**, contiene tutti i file prodotti dal sistema di build di Maven

Package biblioNet

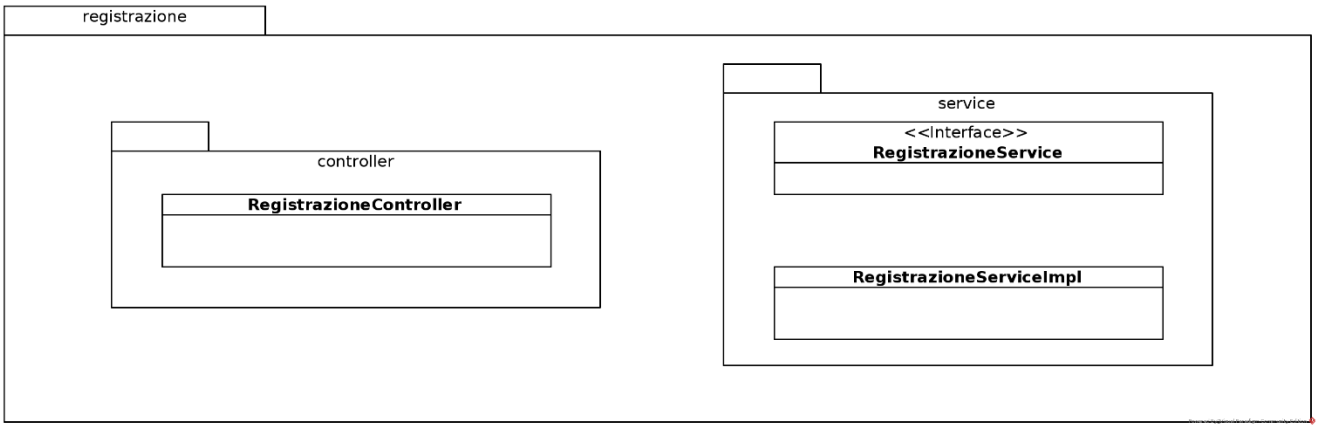
Nella presente sezione si mostra la struttura del package pricipale di biblioNet. La struttura generale è stata ottenuta a partire da tre principali scelte:

1. Creare un package separato per ogni sottosistema, contenente le classi service e controller del sottosistema, ed eventuali classi di utilità usate unicamente da esso.
2. Creare un package separato per le classi del *model*, contenente le classi entity e i DAO per l'accesso al DB. Tale scelta è stata presa vista l'elevata complessità del database di BiblioNet che prevede numerose relazioni tra le entità. Si è quindi preferito tenere tutto in un package separato e collegato a tutti gli altri package dei sottosistemi.
3. Creare un package chiamato *utils* in cui inserire eventuli classi di utlità per il sistema e usabili da più sottosistem.

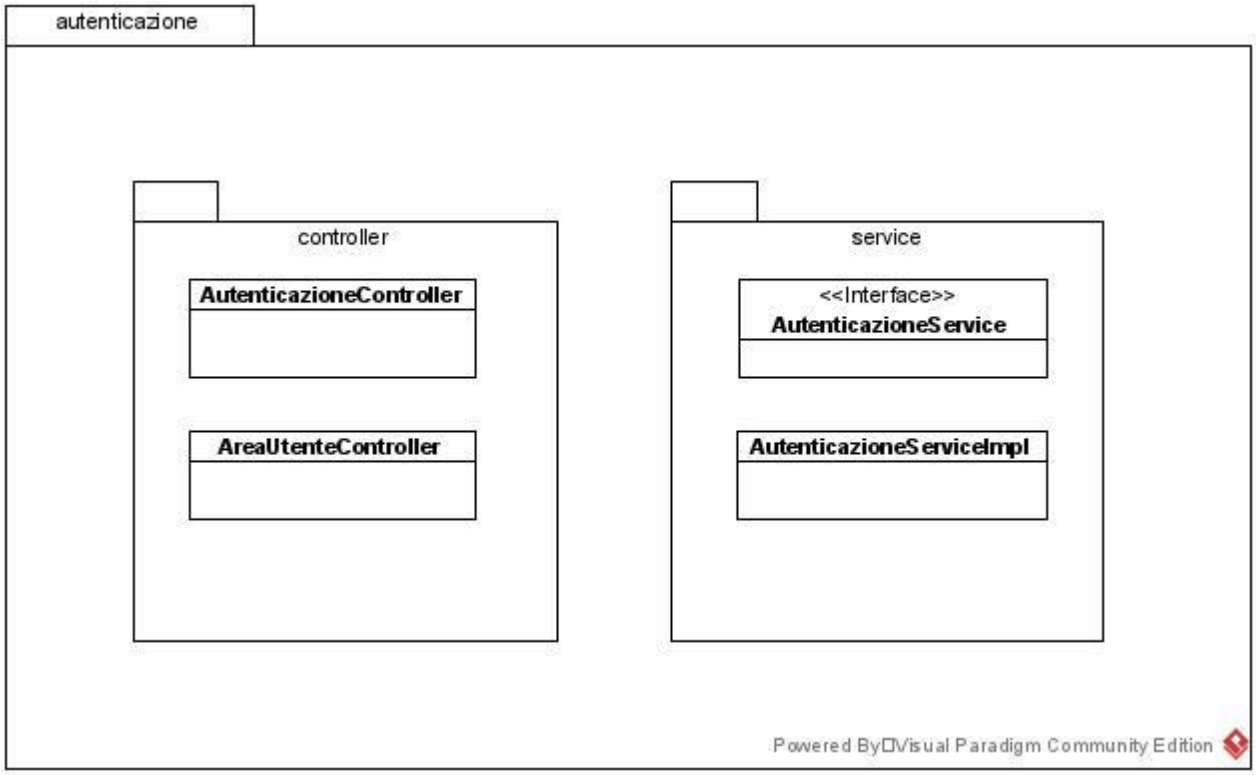
Per ciò che concerne la dipendenza tra i packages, la suddivisione precedentemente illustrata a portato alla creazione di una relazione tra il package model e tutti gli altri package del sistema.



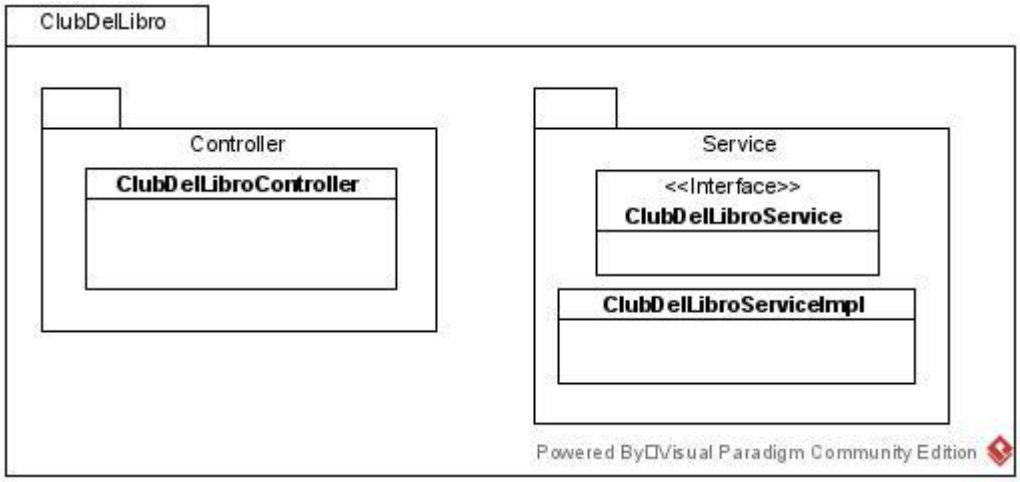
Package Registrazione



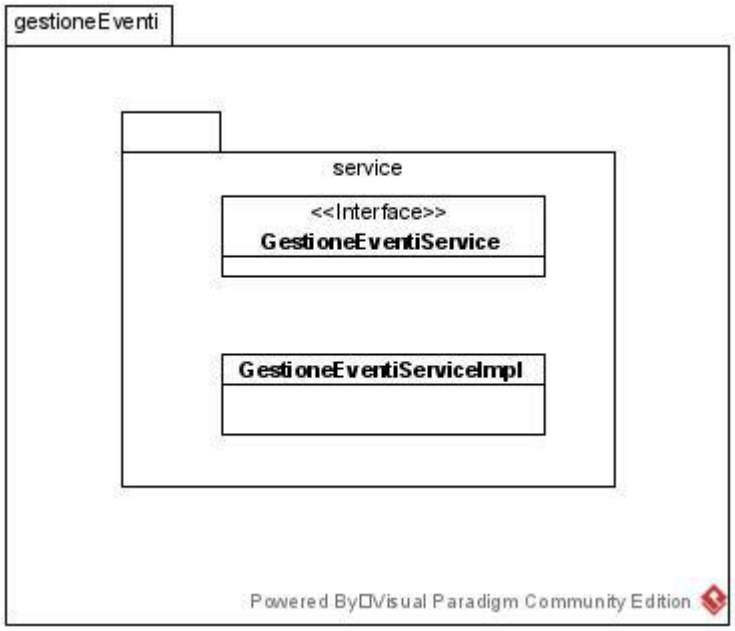
Package Autenticazione



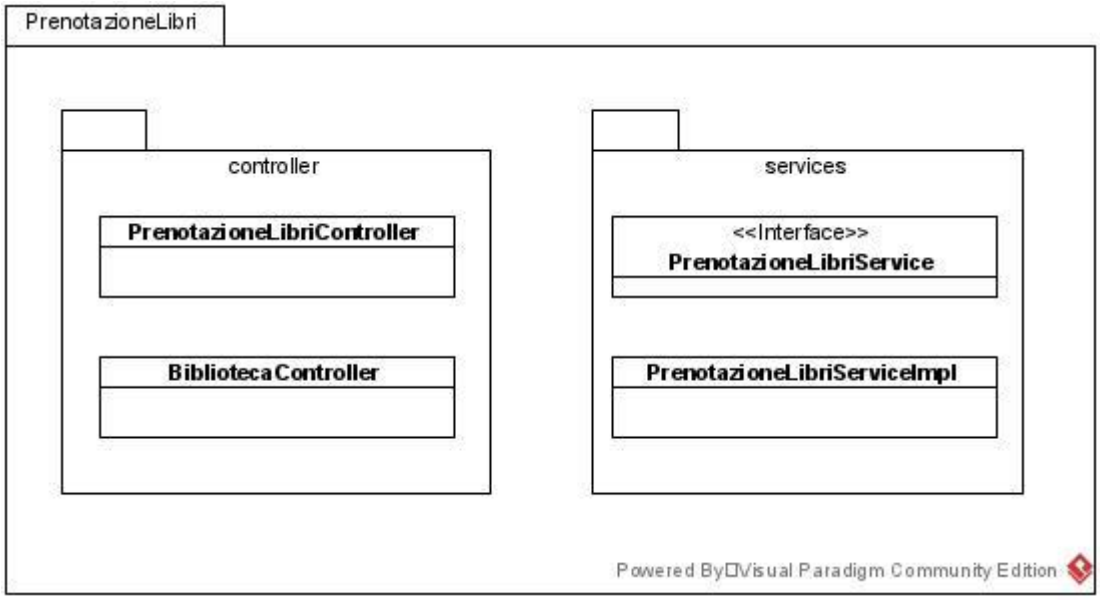
Package Club del Libro



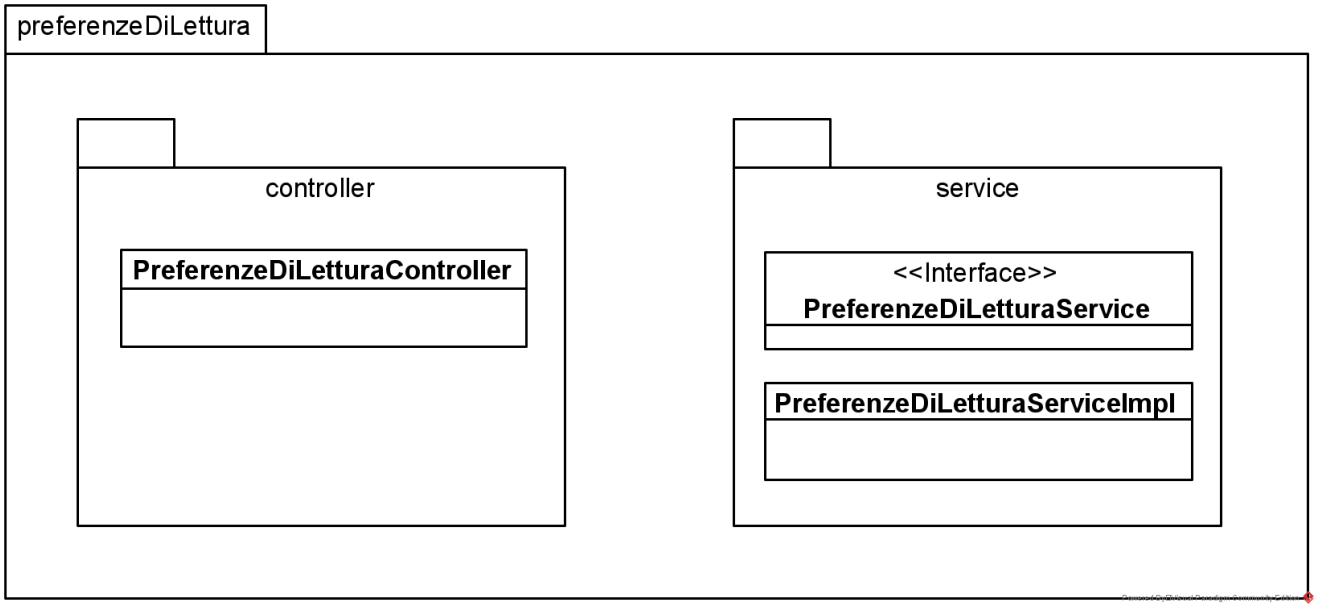
Package Gestione eventi



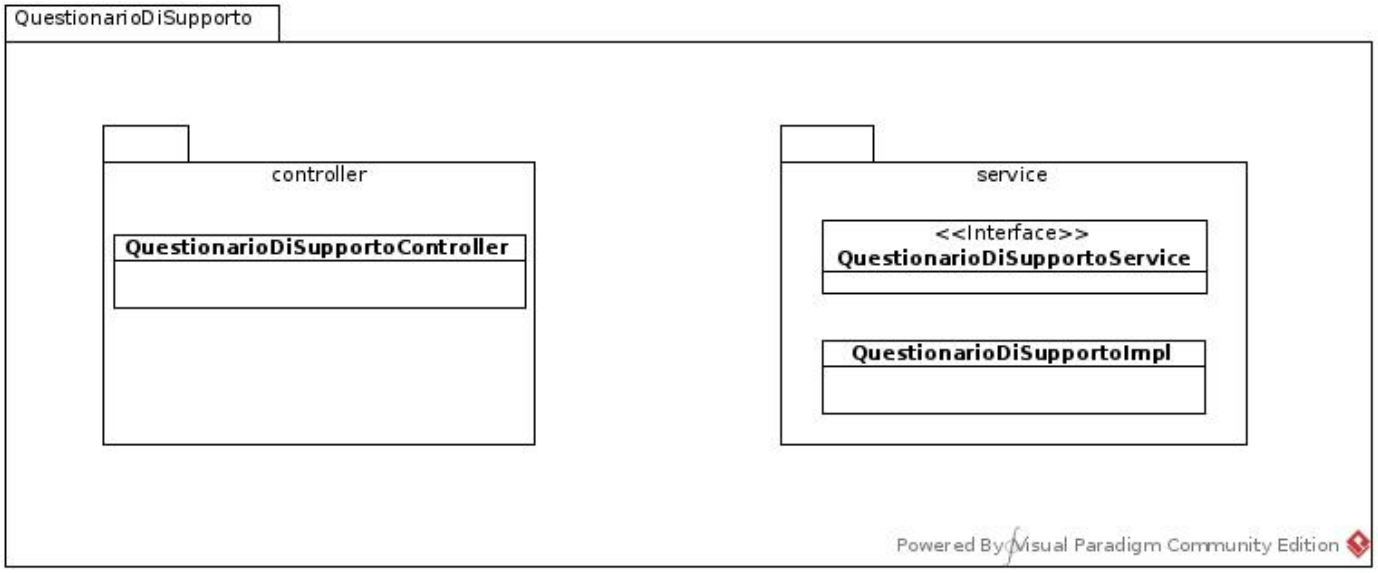
Package Prenotazione libri



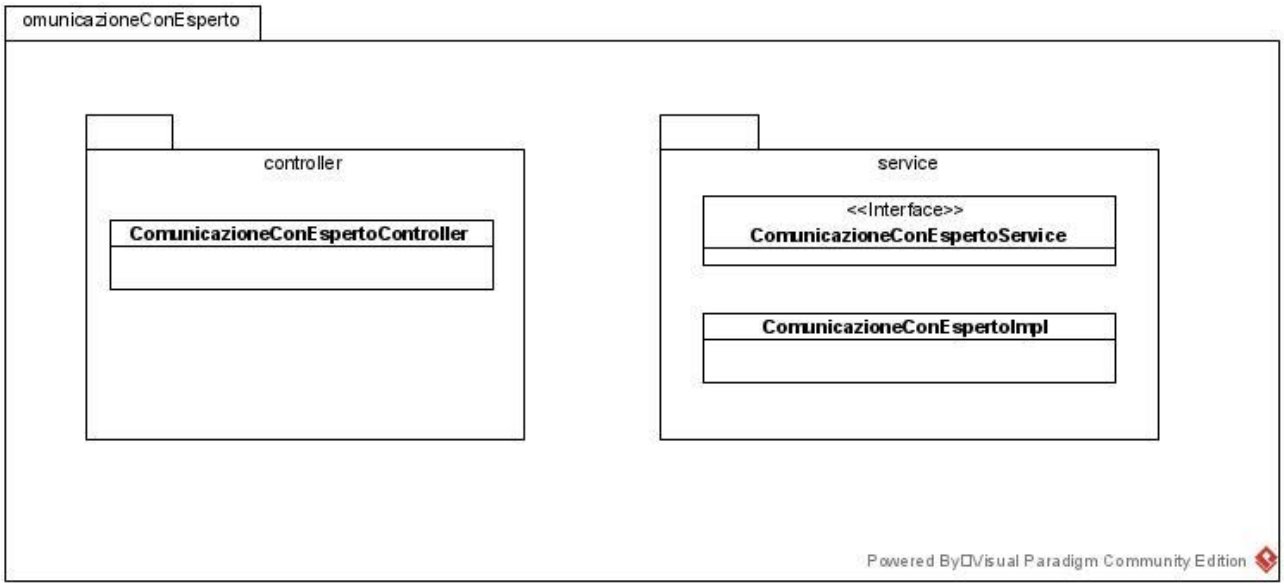
Package Preferenze di lettura



Package Questionario di supporto



Package Comunicazione con esperto



3 Class Interfaces

Di seguito saranno presentate le interfacce di ciascun package. [Non sarà il package model, le classi controller]

Javadoc di BiblioNet

IMPORTANTE

Per motivi di leggibilità si è scelto di creare un sito, hostato tramite GitHub pages, contenente la JavaDoc di BiblioNet. In tale maniera, chiunque può consultare la documentazione aggiornata dell'intero sistema.

Di seguito, il link al sito in questione: https://stefanolambiase.github.io/biblionet/

1 Package Registrazione

Nome classe	RegistrazioneService
Descrizione	Questa classe permette di gestire le operazioni relative alla registrazione.
Metodi	+registraEsperto(Esperto esperto): UtenteRegistrato +registraBiblioteca(Biblioteca biblioteca): UtenteRegistrato +registraLettore(Lettore lettore): UtenteRegistrato
Invariante di classe	/

Nome Metodo	+registraLettore(Lettore lettore)
Descrizione	Questo metodo consente di registrare un nuovo lettore.
Pre-condizione	/
Post-condizione	**context:** RegistrazioneService:: registraLettore(Lettore lettore) **post:** LettoreDAO.save(lettore) == true
Nome Metodo	+registraEsperto(Esperto esperto)
Descrizione	Questo metodo consente di registrare un nuovo esperto.
Pre-condizione	/

Post-condizione	**context:** RegistrazioneService:: registraEsperto(Esperto esperto) **post:** EspertoDAO.save(esperto) == true
Nome Metodo	+registraBiblioteca(Biblioteca biblioteca)
Descrizione	Questo metodo consente di registrare una nuova biblioteca.
Pre-condizione	/
Post-condizione	**context:** RegistrazioneService:: registraLettore(Lettore lettore) **post:** BibliotecaDAO.save(biblioteca) == true

2 Package Autenticazione

it.unisa.biblioNet.autenticazione.service.AutenticazioneService

Nome classe	AutenticazioneService
Descrizione	Questa classe permette di gestire le operazioni relative all'autenticazione.
Metodi	+login(String email, String password) : UtenteRegistrato +aggiornaBiblioteca(Biblioteca biblioteca) : Biblioteca +aggiornaEsperto(Esperto esperto) : Esperto +aggiornaLettore(Lettore lettore) : Lettore
Invariante di classe	/

Nome Metodo	+login(String email, String password)
Descrizione	Questo metodo consente di loggare un utente registrato.
Pre-condizione	/
Post-condizione	**context:** AutenticazioneService::login(email, password)

	post: isLettore(loggedUser) | |isEsperto(loggedUser) | |isBiblioteca(loggedUser)==true
Nome Metodo	+aggiornaBiblioteca(Biblioteca biblioteca)
Descrizione	Questo metodo consente di aggiornare i dati di un account biblioteca.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+aggiornaEsperto(Esperto esperto)
Descrizione	Questo metodo consente di aggiornare i dati di un account Esperto.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+aggiornaLettore(Lettore lettore)
Descrizione	Questo metodo consente di aggiornare i dati di un account Lettore.
Pre-condizione	/
Post-condizione	/

3 Package Club del Libro

it.unisa.biblionet.clubDelLibro.service.ClubDelLibroService

Nome classe	ClubDelLibroService
Descrizione	Questa classe offre delle funzionalità utili alla gestione di un Club del Libro.
Metodi	+creaClub(ClubDelLibro club): ClubDelLibro

	+modficaDatiClub(ClubDelLibro club): ClubDelLibro +visualizzaClubsDelLibro(Predicate<ClubDelLibro> filtro): List<ClubDelLibro> +partecipaClub(ClubDelLibro club, Lettore lettore): Boolean +findAllByLettori(Lettore lettore):List<ClubsDelLibro>
Invariante di classe	/

Nome Metodo	+creaClub(ClubDelLibro club): ClubDelLibro
Descrizione	Questo metodo consente ad un esperto di creare un Club del Libro.
Pre-condizione	**context**: ClubDelLibroService::creaClub(club) **pre:** not visualizzaClubsDelLibro (null).contains(club)
Post-condizione	**context**: ClubDelLibroService::creaClub(club) **post:** visualizzaClubsDelLibro(null).contains(club) and visualizzaClubsDelLibro (null).size = @pre. visualizzaClubsDelLibro (null).size+1
Nome Metodo	+modficaDatiClub(ClubDelLibro clubClubDelLibro)
Descrizione	Questo metodo consente di modificare i dati di un club del libro.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+visualizzaClubsDelLibro(Predicate<ClubDelLibro> filtro): List<ClubDelLibro>

Descrizione	Questo metodo restituisce una lista di tutti I club del libro che soddisfano I filtra inseriti. Se viene passato un filtro null, restituisce tutti I club del libro.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+partecipaClub(ClubDelLibro club, Lettore lettore): Boolean
Descrizione	Questo metodo permette di far partecipare un lettore ad un club del libro.
Pre-condizione	**context**: ClubDelLibro:: partecipaClub(club,lettore) **pre:** not findAllByLettori(Lettore).contains(club)
Post-condizione	**context**: ClubDelLibro:: partecipaClub(club,lettore) **post:** findAllByLettori(Lettore).contains(club)

4 Package Gestione Eventi

it.unisa.biblioNet.gestioneEventi.service.GestioneEventiService

Nome classe	GestioneEventiService
Descrizione	Questa classe fornisce I metodi per effettuare le operazioni riguardanti la creazione e la gestione degli Eventi dei Club del Libro.
Metodi	+creaEvento(Evento evento): Evento +modificaEvento(Evento evento): Evento +eliminaEvento(Evento evento): Evento
Invariante di classe	/

Nome Metodo	+creaEvento(Evento evento): Evento
Descrizione	Questo metodo permette di creare un Evento relativo a un Club del Libro.
Pre-condizione	**context:** GestioneEventoService::creaEvento(Evento evento) **pre:** not visualizzaListaEventiClub(evento.club).includes(evento)
Post-condizione	**context:** GestioneEventoService::creaEvento(Evento evento) **post:** visualizzaListaEventiClub(evento.club).includes(evento) and visualizzaListaEventiClub(evento.club).size == @pre.visualizzaListaEventiClub(evento.club).size+1
Nome Metodo	+modificaEvento(Evento evento): Evento
Descrizione	Questo metodo permette di modificare un Evento già esistente relativo a un Club del Libro.
Pre-condizione	**context:** GestioneEventoService::modificaEvento(Evento evento) **pre:** visualizzaListaEventiClub(evento.club). includes(evento)
Post-condizione	**context:** GestioneEventoService::modificaEvento(Evento evento) **post:** visualizzaListaEventiClub(evento.club).size == @pre.visualizzaListaEventiClub(evento.club).size
Nome Metodo	+cancellaEvento(Evento evento): Evento
Descrizione	Questo metodo permette la cancellazione di un Evento già esistente relativo a un Club del Libro.

Pre-condizione	**context:** GestioneEventoService::cancellaEvento(Evento evento) **pre:** visualizzaListaEventiClub(evento.club). includes(evento)
Post-condizione	**context:** GestioneEventoService::cancellaEvento(Evento evento) **post:** not visualizzaListaEventiClub(evento.club).includes(evento) and visualizzaListaEventiClub(evento.club).size == @pre.visualizzaListaEventiClub(evento.club).size-1

5 Package Prenotazione libri

it.unisa.biblioNet.prenotazioneLibri.service.PrenotazioneService

Nome classe	PrenotazioneService
Descrizione	Questa classe permette di gestire le operazioni relative alla prenotazione di libri.
Metodi	+aggiungiLibro(Libro libro): Libro +getTicketsByBiblioteca(Utente biblioteca): List<TicketPrestito> +richiediPrestito(Lettore lettore, String idBiblioteca, int idLibro) +accettaRichiesta(TicketPrestito ticket): TicketPrestito +rifiutaRichiesta(TicketPrestito ticket): TicketPrestito +visualizzaListaLibriCompleta(): List<Libro>
Invariante di classe	/

Nome Metodo	+inserimentoPerIsbn(String isbn,String idBiblioteca, int numCopie, List<String> generi): Libro
Descrizione	Questo metodo consente di aggiungere un libro ad una lista di libri tramite ISBN.
Pre-condizione	**context:**PrestitoLibriService::inserimentoPerIsbn(String isbn, String idBiblioteca, int numCopie, List<String> generi) **pre**:not visualizzaListaLibriCompleta(null).contains(libro)
Post-condizione	**context:**PrestitoLibriService::inserimentoPerIsbn(String isbn, String idBiblioteca, int numCopie, List<String> generi) **post**: visualizzaListaLibriCompleta (null).contains(libro) and visualizzaListaLibriCompleta(null).size=@pre.visualizzaListaLibri Completa(null).size+1
Nome Metodo	+inserimentoDalDatabase(int idLibro,String idBiblioteca,int numCopie
Descrizione	Questo metodo consente di aggiungere un libro di quelli già presenti nel database.
Pre-condizione	**context:**PrestitoLibriService::inserimentoDalDatabase(int idLibro,String idBiblioteca,int numCopie) **pre**: visualizzaListaLibriCompleta (null).contains(libro)
Post-condizione	**context:**PrestitoLibriService::inserimentoDalDatabase(int idLibro,String idBiblioteca,int numCopie) **post**: visualizzaListaLibriCompleta (null).contains(libro) and visualizzaListaLibriCompleta(null).size=@pre.visualizzaListaLibri Completa(null).size+1
Nome Metodo	+inserimentoManuale(Libro libro,String idBiblioteca, int numCopie, List<String> generi): Libro
Descrizione	Questo metodo consente di aggiungere un libro ad una lista di libri manualmente.
Pre-condizione	**context:**PrestitoLibriService::inserimentoManuale(Libro libro, String idBiblioteca, int numCopie, List<String> generi) **pre**:not visualizzaListaLibriCompleta(null).contains(libro)
Post-condizione	**context:**PrestitoLibriService::inserimentoManuale(Libro libro, String idBiblioteca, int numCopie, List<String> generi)

	post: visualizzaListaLibriCompleta (null).contains(libro) and visualizzaListaLibriCompleta(null).size=@pre.visualizzaListaLibriCompleta(null).size+1
Nome Metodo	+richiediPrestito(Lettore lettore, String idBiblioteca, int idLibro)
Descrizione	Questo metodo consente di richiedere il prestito di un libro.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+accettaRichiesta(TicketPrestito ticket, int giorni): TicketPrestito
Descrizione	Questo metodo consente di accettare una richiesta di prenotazione.
Pre-condizione	**context:**PrestitoLibriService::accettaRichiesta(ticket, giorni) **pre**:visualizzaTicketsByBiblioteca().contains(ticket)
Post-condizione	**context:**PrestitoLibriService::accettaRichiesta(ticket) **post**: not visualizzaTicketsByBiblioteca().contains(ticket) and visualizzaRichiestePrestiti().size=@pre.visualizzaRichiestePrestiti(null).size-1
Nome Metodo	+rifiutaRichiesta(Ticket ticket): Boolean
Descrizione	Questo metodo consente di rifiutare una richiesta di prenotazione.
Pre-condizione	**context:**PrestitoLibriService::accettaRichiesta(ticket) **pre**:visualizzaTicketsByBiblioteca().contains(ticket)
Post-condizione	**context:**PrestitoLibriService::accettaRichiesta(ticket) **post**: not visualizzaTicketsByBiblioteca().contains(ticket) and visualizzaTicketsByBiblioteca().size=@pre.visualizzaTicketsByBiblioteca(null).size-1
Nome Metodo	+visualizzaListaLibriCompleta(): List<Libro>
Descrizione	Questo metodo consente di visualizzare la lista dei libri prenotabile completa.

Pre-condizione	/
Post-condizione	/

6 Package Preferenze di lettura

it.unisa.biblioNet.preferenzeDiLettura.service.PreferenzeDiLetturaService

Nome classe	PreferenzeDiLetturaService
Descrizione	Questa classe offre i metodi necessari per la gestione delle preferenze di lettura selezionabili da Lettore ed Esperto.
Metodi	+addGeneriEsperto(List<Genere> generi, Esperto esperto): void +addGeneriLettore(List<Genere> generi, Lettore lettore). void
Invariante di classe	/

Nome Metodo	+addGeneriEsperto(List<Genere> generi, Esperto esperto): void
Descrizione	Questo metodo si occupa di permettere ad un esperto di inserire un genere alla lista di quelli che conosce o preferisce.
Pre-condizione	/
Post-condizione	**context:** PreferenzeDiLetturaService::addGeneriEsperto(List<Genere> generi, Esperto esperto) **post:** Arrays.compare(generi, esperto.getGeneri) == 0
Nome Metodo	+addGeneriLettore(List>Genere> generi, Lettore lettore): void

Descrizione	Questo metodo si occupa di permettere ad un lettore di inserire un genere alla lista di quelli che conosce o preferisce.
Pre-condizione	/
Post-condizione	**context:** PreferenzeDiLetturaService::addGeneriLettore(List<Genere> generi, Lettore lettore) **post:** Arrays.compare(generi, lettore.getGeneri) == 0

7 Package Questionario di supporto

it.unisa.biblioNet.QuestionarioDiSupporto.service.QuestionarioDiSupportoService

Nome classe	QuestionarioDiSupportoService
Descrizione	Questa classe offre i metodi per la compilazione da parte di un lettore e della gestione da parte dell'Amministratore del Questionario di Supporto. Queste funzionalità saranno implementate in una futura release.
Metodi	+ConfermaQuestionario():boolean
Invariante di classe	/

8 Package Comunicazione con esperto

it.unisa.biblioNet.comunicazioneconEsperto.service.ComunicazioneService

Nome classe	ComunicazioneService
Descrizione	Questa classe permette di gestire le operazioni relative alla comunicazione con un esperto.

Metodi	+visualizzaEspertiPerGenere():List<Esperto>
Invariante di classe	/

Nome Metodo	+visualizzaEspertiPerGenere():List<Esperto>
Descrizione	Questo metodo permette di visualizzare la lista di esperti per genere con cui è possibile iniziare una comunicazione.
Pre-condizione	/
Post-condizione	/

4 Design Patterns

Nella presente sezione si andranno a descrivere e dettagliare i design patterns utilizzati nello sviluppo dell'applicativo BiblioNet. Per ogni pattern si darà:

- Una brevissima introduzione teorica.
- Il problema che doveva risolvere all'interno di BiblioNet.
- Una brevissima spiegazione di come si è risolto il problema in BiblioNet.
- Un grafico della struttura delle classi che implementano il pattern.

Singleton e Observer

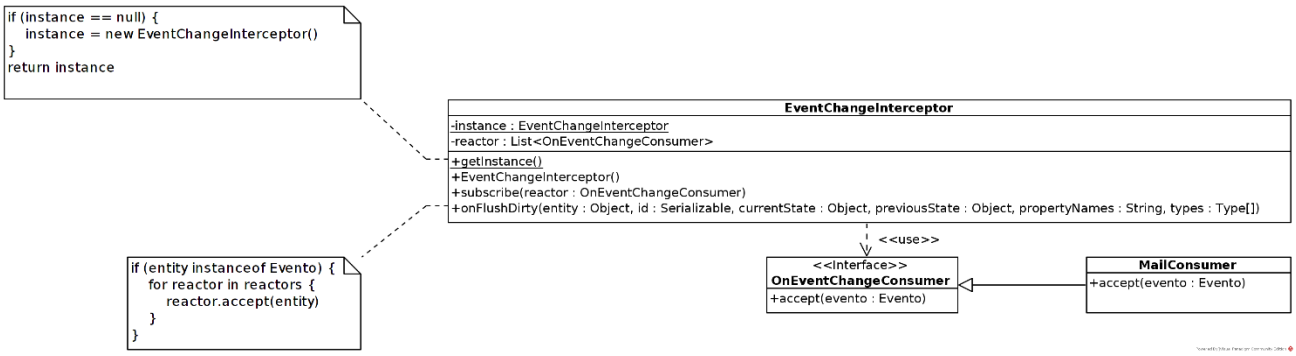
L'Observer è un design pattern comportamentale, ossia un design pattern che fornisce una soluzione a un problema di interazione tra più oggetti. Observer, nello specifico, permette a un oggetto "osservato" di generare (pubblicare) eventi ai quali una lista di oggetti "osservatori" (iscritti) possa poi reagire.

Singleton è un design pattern creazionale, ossia un design pattern che si occupa dell'istanziazione degli oggetti, che ha lo scopo di garantire che di una determinata classe venga strutturata una sola istanza e di fornire un punto di accesso globale a tale istanza.

BiblioNet prevede un sistema di notifica via email per la modifica degli eventi.

Per gestire ciò a livello implementativo senza accoppiare strettamente il sistema con un dato servizio di mailing o sistema di notifiche, viene creato un oggetto Singleton che conserva una propria lista di osservatori dotati di un metodo "accept" che accetti un singolo Evento come argomento. Ogni volta che viene modificato un evento sul database, l'oggetto, che è anche un intercettore di hibernate, "notifica" così la lista degli osservatori tramite il metodo onFlushDirty.

Ciò rende possibile la creazione in un momento successivo di multipli osservatori legati agli eventi (ad esempio per le notifiche push per un eventuale applicazione mobile).

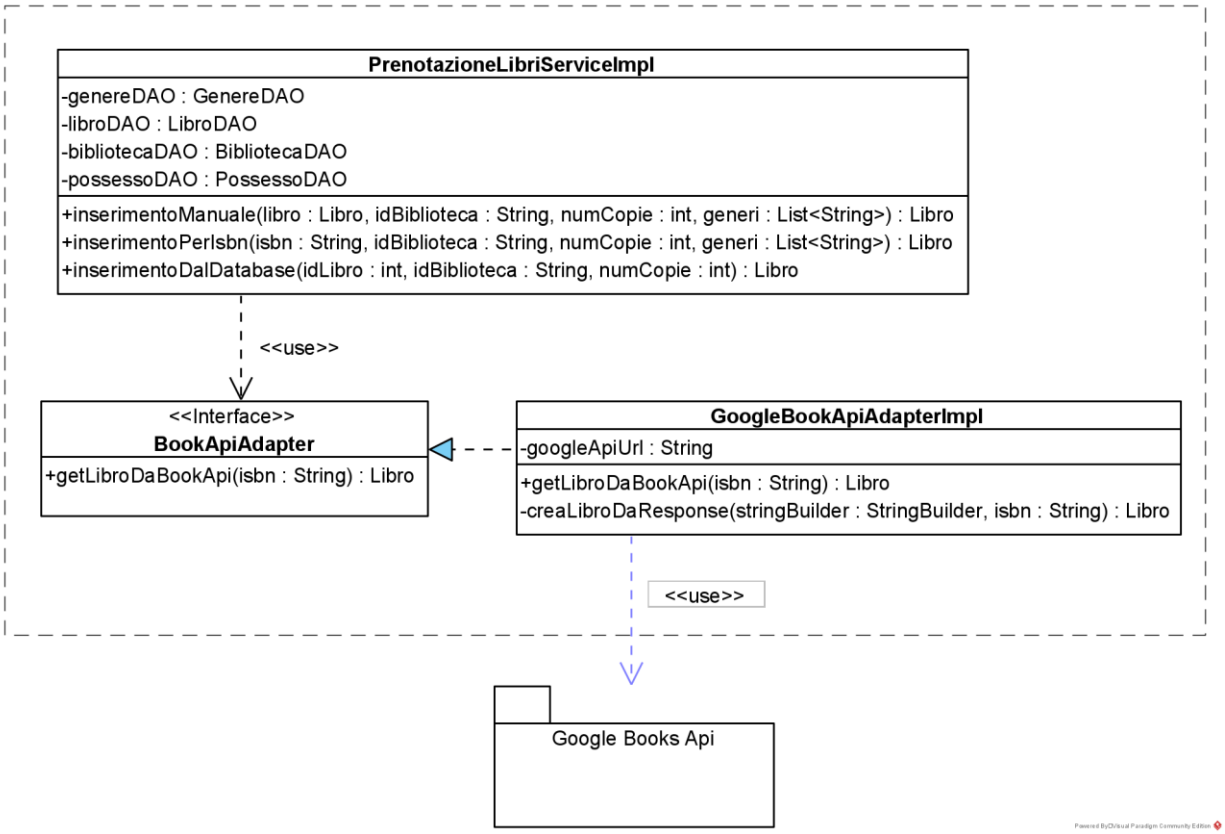


Adapter

L'Adapter è un design pattern strutturale, ovvero quei design pattern che facilitano la progettazione attraverso la semplificazione delle relazioni tra entità. Adapter, in particolare, permette ad oggetti con differenti interfacce di collaborare. Si implementa attraverso una classe "adapter", che si occupa di convertire i dati in oggetti comprensibili dal sistema.

Biblionet si pone l'obiettivo di semplificare e digitalizzare l'interazione tra biblioteche e lettori. In questo ambito, offre anche la funzionalità di prenotazione di libri attraverso la piattaforma, il che presuppone che le biblioteche debbano rendere disponibile una lista di libri prenotabili dal sito stesso. Per l'inserimento dei libri all'interno di questo catalogo virtuale, oltre all'inserimento manuale, viene offerta ai gestori delle biblioteche la possibilità di inserire un libro tramite l'ISBN, senza doversi preoccupare di compilarne tutti i campi. Questa ricerca viene effettuata tramite un servizio esterno al sistema, in particolare il database di Google Books, attraverso una API offerta al pubblico sul web. Ovviamente l'API di ricerca dei libri, in quanto accessibile da chiunque, risponde alle query in un formato standard ed ampiamente utilizzato, ovvero il JSON.

Il design pattern Adapter si occupa della conversione dei dati di un libro da formato JSON ad oggetto della classe Libro, in modo da rendere le informazioni comprensibili dal sistema. La ricerca dei libri avviene in due fasi: la query all'API passando l'ISBN, e la traduzione del risultato grazie all'adapter.

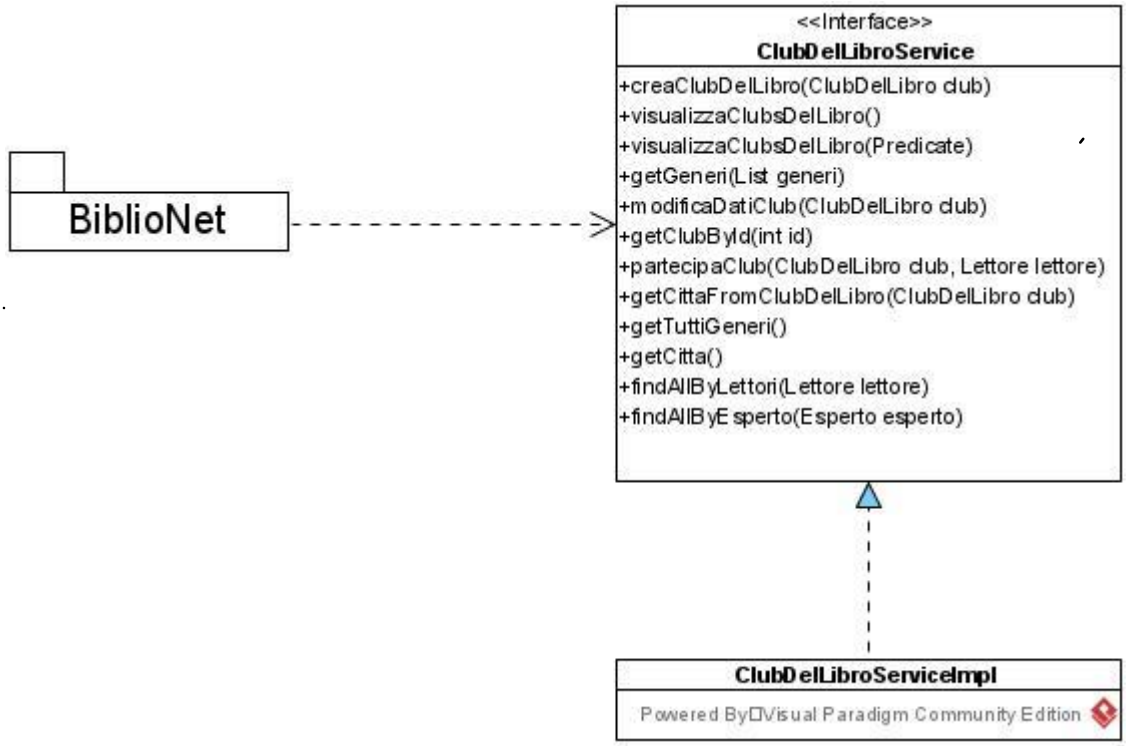


Facade

Il Facade è un design pattern che permette, implementando una interfaccia semplificata di accedere a sottosistemi più complessi. In questo modo si può nascondere al sistema la complessità delle librerie, dei framework o dei set di classi che si stanno usando. Si garantisce così un alto disaccoppiamento e si rende la piattaforma più manutenibile e più aggiornabile, poiché basterà cambiare l'implementazione dei metodi dell'interfaccia per implementare le modifiche.

BiblioNet, essendo un sistema molto complesso, sfrutta il design pattern Facade per implementare tutta la sua logica di business e rendere più facile l'interfacciarsi con essa. Nello specifico BiblioNet utilizza il Facade per ogni suo sottosistema, implementandolo attraverso delle interfacce che sono usate per accedere ai metodi interni.

Di seguito un esempio di Facade nel sistema BiblioNet:

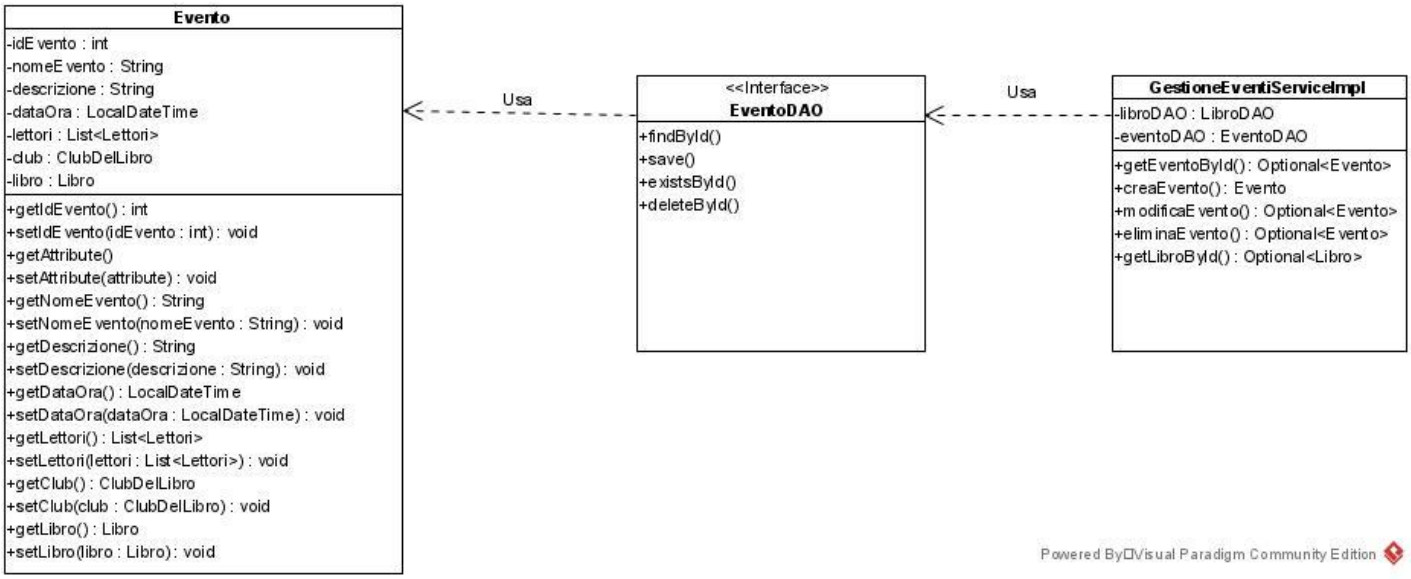


DAO

Un DAO (Data Access Object) è un pattern che offre un'interfaccia astratta per alcuni tipi di database. Mappando le chiamate dell'applicazione allo stato persistente, il DAO fornisce alcune operazioni specifiche sui dati senza esporre i dettagli del database. I DAO sono utilizzabili nella maggior parte dei linguaggi e la maggior parte dei software con bisogni di persistenza, principalmente viene associato con applicazioni JavaEE che utilizzano database relazionali.

Essendo biblionet una web application che punta di gestire sia prenotazione di libri che numerosi club del libro presenta un database molto vasto, quindi ha bisogno di poter interagire con database in modo rapido e sicuro con numerosi query per quella che è la moltitudine di dati da gestire. Per questo motivo abbiamo usato varie interfacce DAO all'interno del nostro sistema le quali, grazie all'utilizzo del framework di Spring (SpringJPARepository), vengono implementate in maniera del tutto automatica e trasparente per il programmatore. Attraverso l'utilizzo di Spring JPA Repository viene quindi rimossa la necessità di avere una classe che implementi tutte le eventuali query continuando a poter usufruire a pieno dei DAO per accedere al database nel resto dell'applicazione.

Qui è riportato un esempio di DAO utilizzato in Biblionet e delle relazioni che ha con altre classi dell'applicazione.



5 Glossario

Sigla/Termine	Definizione
Package	Raggruppamento di classi ed interfacce.
DAO	Data Access Object, implementazione dell'omonimo pattern architetturale che si occupa di fornire un accesso in modo astratto ai dati persistenti.
Controller	Classe che si occupa di gestire le richieste effettuate dal client.
Service	Classe che implementa la logica di business, viene utilizzata dal controller o da un altro sottosistema.
Model	Parte del design architetturale MVC che fornisce al sistema i metodi per accedere ai dati utili al sistema.
MVC	Model-View-Controller: design architetturale che permette di separare la logica di presentazione dalla logica di business alla base del sistema.
Facade	Un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro.

Adapter	È un pattern strutturale che può essere basato sia su classi che su oggetti il cui fine è fornire una soluzione astratta al problema dell'interoperabilità tra interfacce differenti.
Singleton	È un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga struttura una solo istanza e di fornire un punto di accesso globale a tale istanza.
Observer Pattern	Design pattern usato come base per la gestione di eventi.
Hibernate	Piattaforma open source per la gestione della persistenza dei dati sul database.
SRP	Principio di singola responsabilià: ogni element del programma deve avere una singola responsabilità