EvoCinema

“ Sistema per la gestione di un cinema “

Versione 1.0

Object Design Document

Coordinatori del progetto

Prof. Andrea De Lucia - Top Manager
Francesco Vicidomini - Project Manager
Ferdinando D'Avino - Project Manager

Partecipanti

Luca Strefezza	0512102474	strluca94@gmail.com
Angelo Stefano D'Auria	0512102630	angelodauria91@gmail.com
Gianluca Villani	0512102990	lucassalerno1995@gmail.com
Giuseppe D'Ambrosio	0512103472	giuseppe.dambrosio14@gmail.com
Giuseppe Apuzzo	0512103920	g.apuzzo94@gmail.com
Sara De Filippo	0512103430	s.defilippo93@gmail.com
Antonio Giulio	0512103098	antonio.giulio96@gmail.com
Michele Delli Paoli	0512103820	mikeledellipaoli@gmail.com
Giuseppe Del Gaudio	0512103690	ciaogiuseppe96@gmail.com
Pietro Dell'Isola	0512103866	dellisola.pietro@gmail.com
Emanuele Buono	0512102370	squareman93@gmail.com
Francesco De Feo	0512103274	francescodefeo94@gmail.com

Revision history

Data	***Versione***	***Descrizione***	***Autori***
21/12/2017	1.0	Stesura ODD	Team Members

INDICE

1. Introduzione

1.1. Object design trade-offs

Dopo aver stilato il documento di Requirements Analysis e il documento di System Design in cui vi è una descrizione sommaria di ciò che sarà il nostro sistema,

definendo i nostri obbiettivi ma tralasciando gli aspetti implementativi, andiamo ora a stilare il documento di Object Design che ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti.
In particolar modo, in tale documento si definiscono le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre sono specificati i trade-off e le linee guida.

Comprensibilità vs Tempo:
Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Per rispettare queste linee guida il codice sarà accompagnato da commenti volti a semplificarne la comprensione. Ovviamente questo comporterà un aumento del tempo di sviluppo del nostro progetto.

Prestazioni vs Costi:
Dato che il nostro progetto è sprovvisto di budget, per poter mantenere prestazioni elevate, in determinate funzionalità verranno utilizzati dei template open source esterni, in particolare Bootstrap.

Interfaccia vs Usabilità:
L'interfaccia grafica è stata realizzata in maniera molto semplice, chiara e concisa, vengono utilizzati i form e pulsanti con lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza:
La sicurezza, come descritto nei requisiti non funzionali del Requirements Analysis, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

1.2. Linee guida per la Documentazione delle Interfacce

Gli sviluppatori dovranno seguire determinate linee guida per la stesura del codice:

Naming Convention:
È buona norma utilizzare nomi:

- Descrittivi
- Pronunciabili
- Di uso comune
- Di lunghezza medio-corta
- Non abbreviati
- Evitando la notazione ungherese
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili:

- I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. La dichiarazione delle variabili deve essere effettuata ad inizio blocco; in ogni riga vi deve essere una sola dichiarazione di variabile e va effettuato l'allineamento per migliorare la leggibilità.

- In determinati casi, è possibile utilizzare il carattere underscore "_", ad esempio quando si fa uso di variabili costanti oppure quando si fa uso di proprietà statiche.

Metodi:

- I nomi dei metodi devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un'azione, seguito dal nome di un oggetto. I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Se viene dichiarata una variabile all'interno di un metodo quest'ultima deve essere dichiarata appena prima del suo utilizzo e deve servire per un solo scopo, per facilitare la leggibilità. Esempio: getId(), setId()
- Ai metodi va aggiunta una descrizione, la quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo. La descrizione del metodo deve includere anche informazioni riguardanti gli argomenti, il valore di ritorno, le eccezioni. I metodi devono essere raggruppati in base alla loro funzionalità.

Classi e pagine:

- I nomi delle classi e delle pagine devono iniziare con la lettera maiuscola, e anche le parole successive all'interno del nome devono iniziare con la lettera maiuscola. I nomi delle classi e delle pagine devono essere evocativi, in modo da fornire informazioni sullo scopo di quest'ultime.
  Ogni file sorgente .php contiene una singola classe e dev'essere strutturato in un determinato modo:
- Una breve introduzione alla classe. L'introduzione indica: l'autore, la versione e la data.

```
/**
* sommario dello scopo della classe.
*
* @author [nome dell'autore]
* @version [numero di versione della classe]
* @since [versione di partenza]
*/
```

- L'istruzione include che permette di importare all'interno della classe gli altri oggetti che la classe utilizza.
- La dichiarazione di una classe è caratterizzata da:
  1. Dichiarazione della classe pubblica
  2. Dichiarazioni di costanti
  3. Dichiarazioni di variabili di classe
  4. Dichiarazioni di variabili d'istanza
  5. Costruttore
  6. Commento e dichiarazione metodi e variabili

1.3. Definizioni acronimi e abbreviazioni

Acronimi:

- RAD: Requirements Analysis Document
- SDD: System Design Document

- ODD: Object Design Document

Abbreviazioni:

- DB: DataBase

1.4. Riferimenti

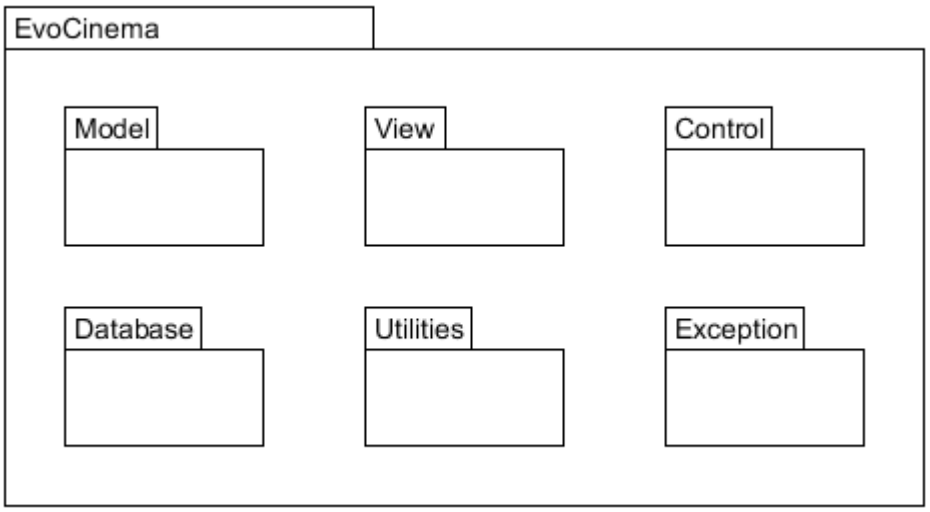
- B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall, 3rd edition, 2009
- Documento SDD del progetto Evocinema
- Documento RAD del progetto Evocinema

2. Packages

La struttura del sistema EvoCinema è strutturata secondo una divisione in package e sottopackage che raggruppano le classi che hanno il compito di gestirne la logica in base alle richieste dell'utente che ne fa uso.

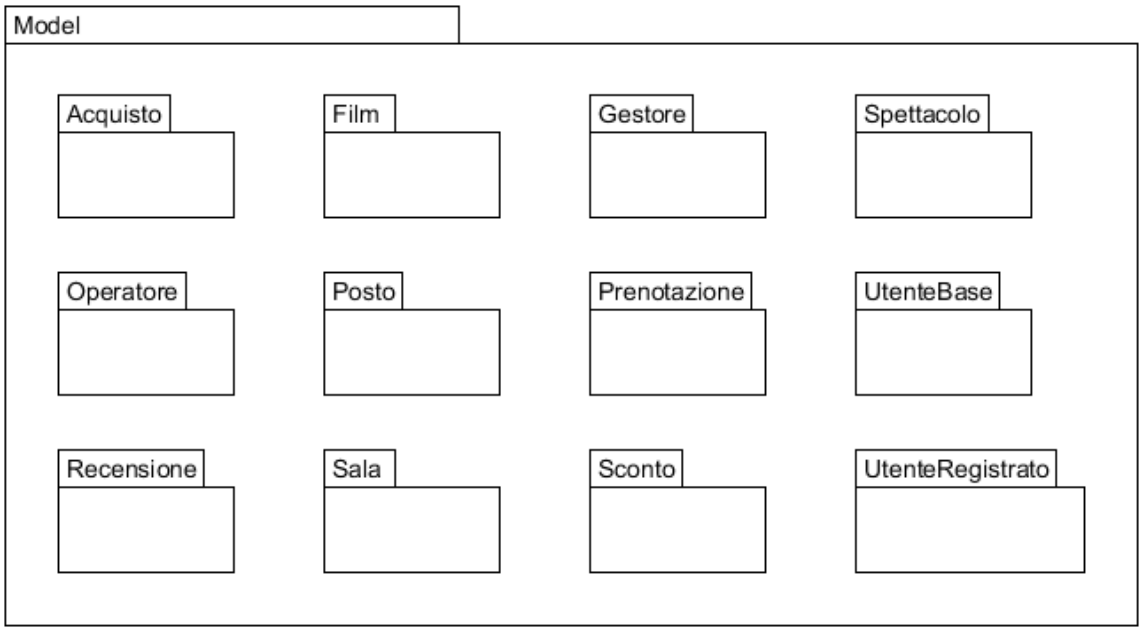
2.1. EvoCinema

Il package principale "EvoCinema" è presentato nel seguente schema.



2.1.1. Model

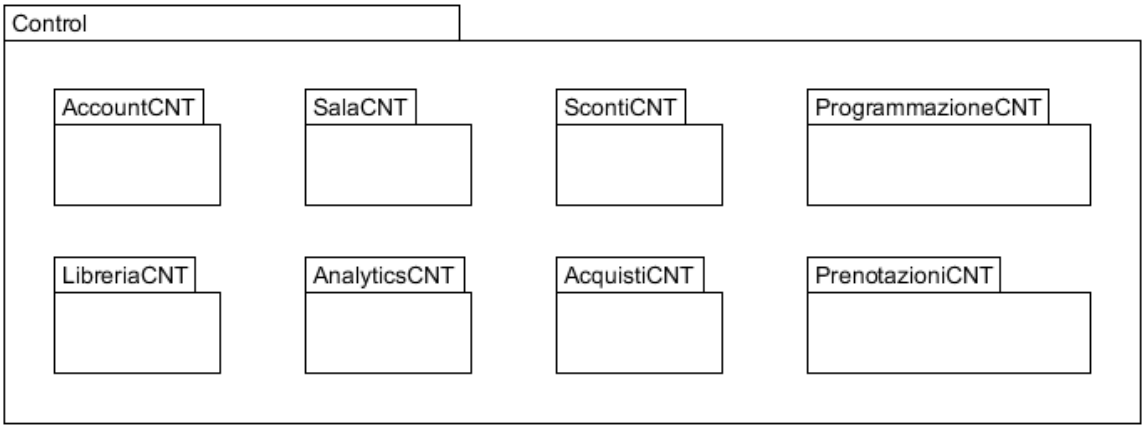
Il sottopackage “Model” è presentato nel seguente schema e contiene le classi Java rappresentanti le entità presenti all’interno del sistema.



Classe:	**Descrizione:**
Acquisto	Descrive un acquisto effettuato.
Film	Descrive un film della libreria.
Gestore	Descrive un gestore del sistema.
Operatore	Descrive un operatore del sistema.
Posto	Descrive un posto di una sala.
Prenotazione	Descrive una prenotazione per uno spettacolo.
Recensione	Descrive una recensione di un film.
Sala	Descrive una sala del cinema.
Sconto	Descrive uno sconto applicabile.
Spettacolo	Descrive uno spettacolo in programmazione.
UtenteBase	Descrive un utente che può effettuare acquisti.
UtenteRegistrato	Descrive un utente generico che raggruppa chi è registrato al sistema.

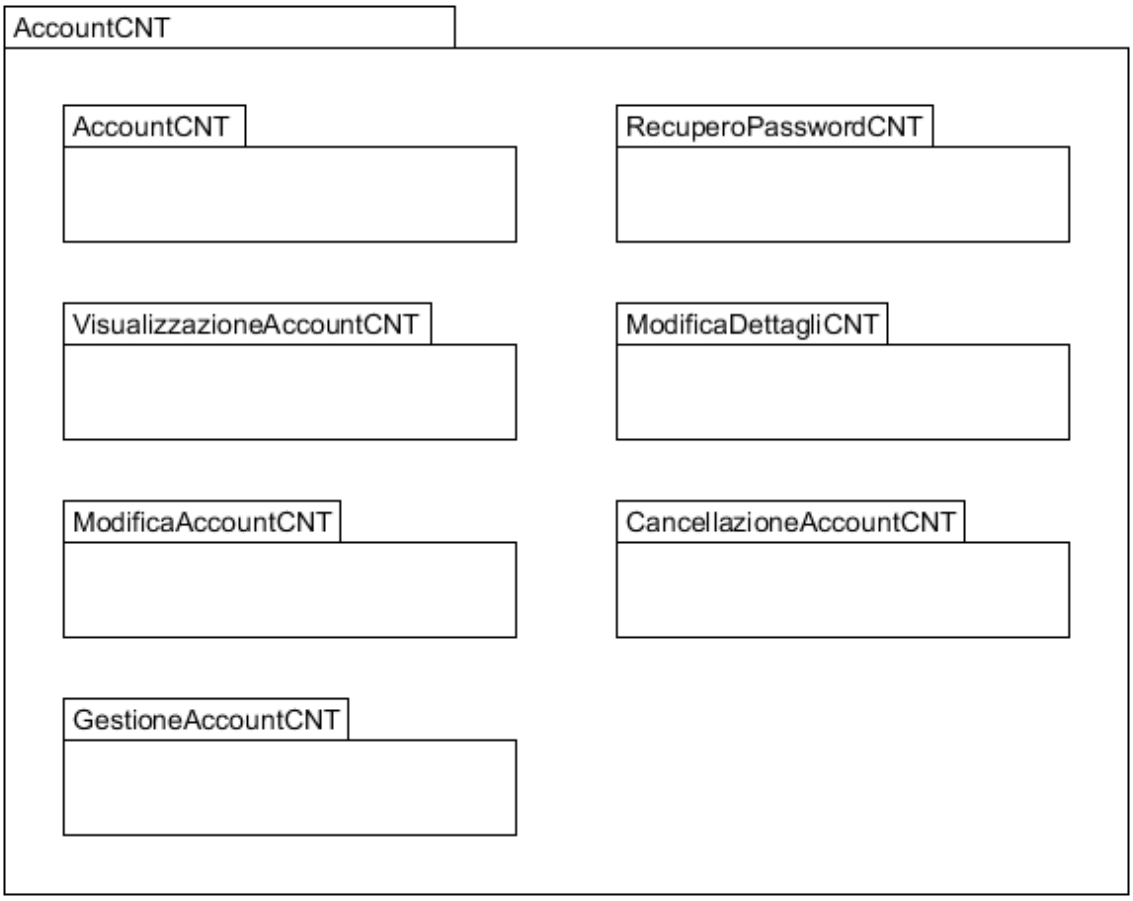
2.1.2. Control

Il sottopackage “Control” è presentato nel seguente schema e contiene le classi Java che si occupano della logica di business del sistema.



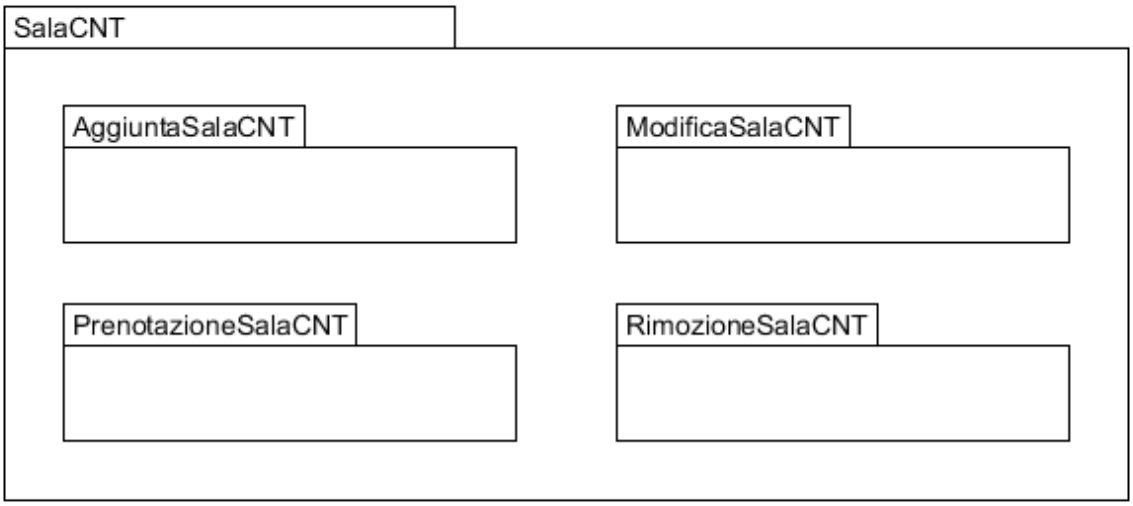
All’interno del sottopackage troviamo una ulteriore suddivisione in base alle diverse gestioni.

2.1.2.1. AccountCNT



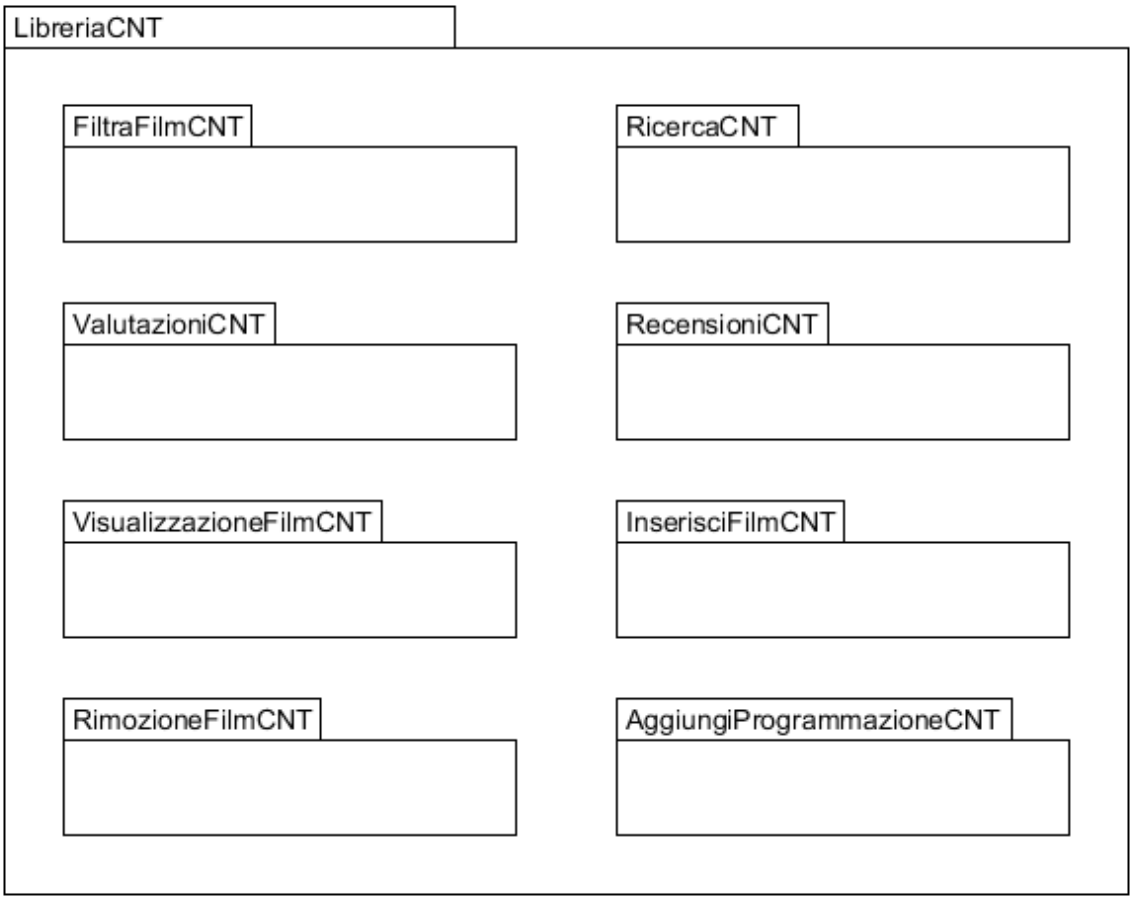
Classe:	**Descrizione:**
AccountCNT	Permette di gestire l'account dell'utente.
RecuperoPasswordCNT	Permette di recuperare la propria password.
VisualizzazioneAccountCNT	Permette di visualizzare i dettagli dell'account.
ModificaDettagliCNT	Permette di modificare i dettagli dell'account.
RegistrazioneAccountCNT	Permette di registrarsi al sistema.
CancellazioneAccountCNT	Permette di cancellare l'account.
GestioneSaldoCNT	Permette la gestione del saldo.

2.1.2.2. SalaCNT



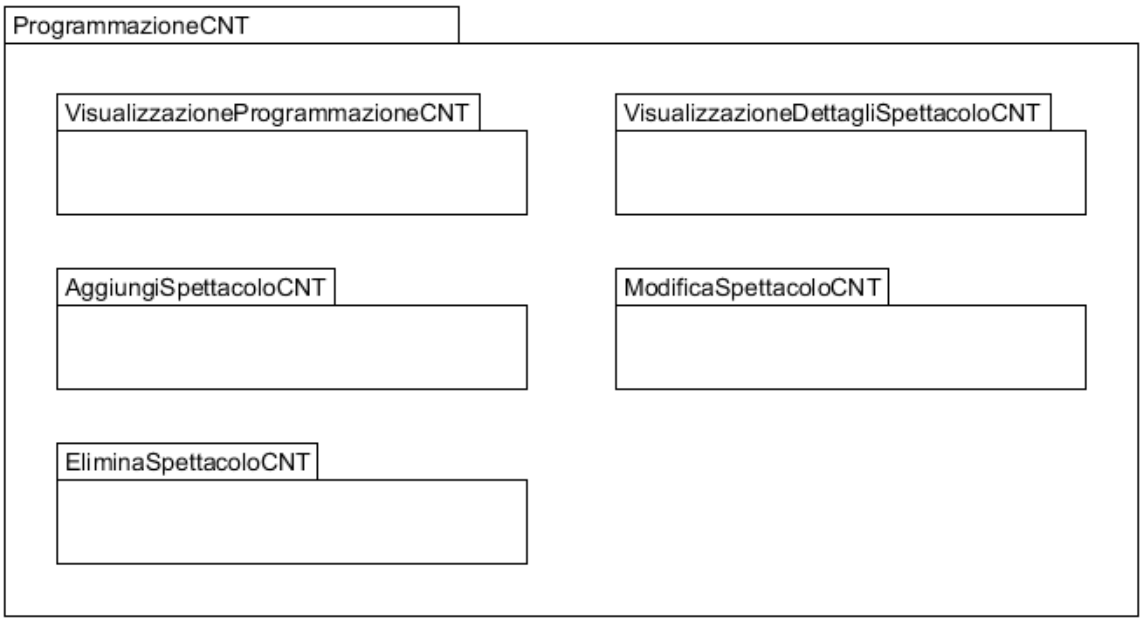
Classe:	**Descrizione:**
AggiuntaSalaCNT	Permette l'aggiunta di una sala.
ModificaSalaCNT	Permette la modifica di una sala.
PrenotazioneSalaCNT	Permette la prenotazione di una sala.
RimozioneSalaCNT	Permette la rimozione di una sala.

2.1.2.3. LibreriaCNT



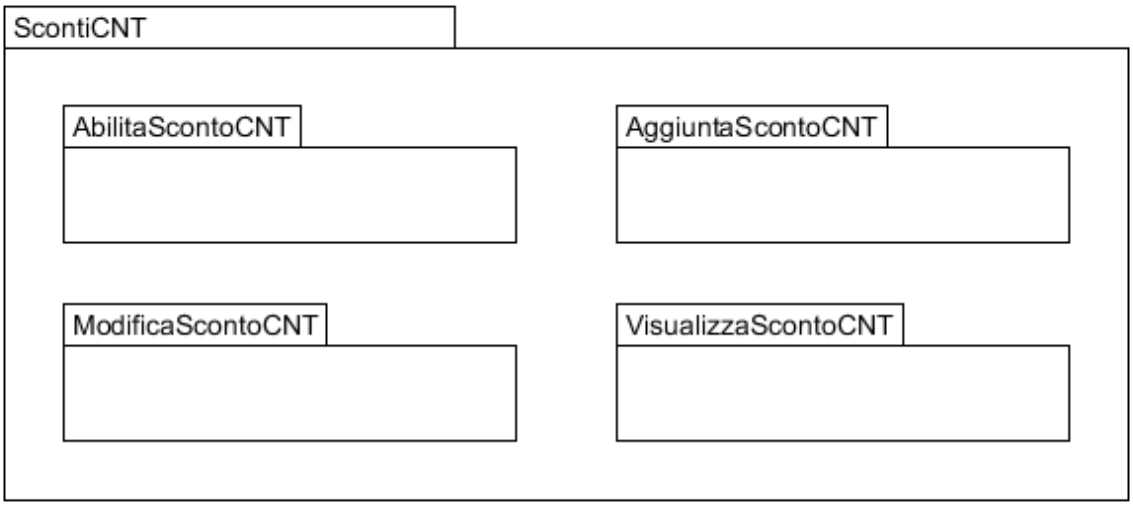
Classe:	**Descrizione:**
FiltraFilmCNT	Permette di applicare filtri alla visualizzazione dei film.
RicercaCNT	Permette di effettuare una ricerca all'interno dei film del sistema.
ValutazioniCNT	Permette di gestire le valutazioni di un film.
RecensioniCNT	Permette di gestire le recensioni di un film.
VisualizzazioneFilmCNT	Permette la visualizzazione dei film del sistema.
InserisciFilmCNT	Permette di inserire un film.
RimozioneFilmCNT	Permette la rimozione di un film.
AggiungiProgrammazioneCNT	Permette l'aggiunta di un film in programmazione.

2.1.2.4. ProgrammazioneCNT



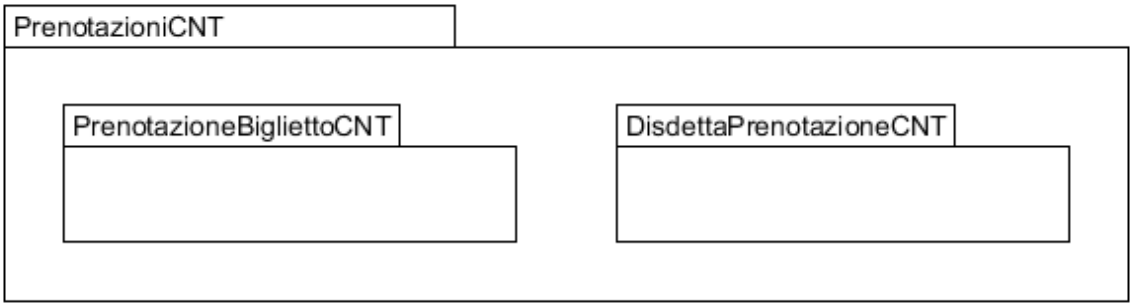
Classe:	**Descrizione:**
VisualizzazioneProgrammazioneCNT	Permette la visualizzazione della programmazione.
VisualizzazioneDettagliSpettacoloCNT	Permette la visualizzazione dei dettagli di uno spettacolo.
AggiungiSpettacoloCNT	Permette l'aggiunta di uno spettacolo.
ModificaSpettacoloCNT	Permette la modifica di uno spettacolo.
EliminaSpettacoloCNT	Permette l'eliminazione di uno spettacolo.

2.1.2.5. ScontiCNT



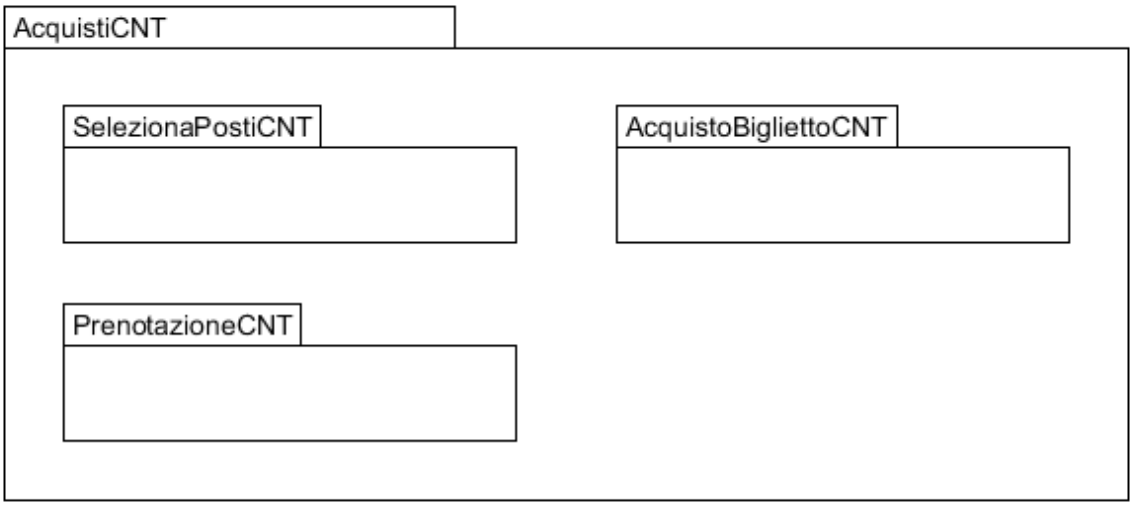
Classe:	**Descrizione:**
AbilitaScontoCNT	Permette di rendere uno sconto applicabile.
AggiuntaScontoCNT	Permette di aggiungere uno sconto.
ModificaScontoCNT	Permette di modificare uno sconto.
VisualizzaScontoCNT	Permette la visualizzazione di uno sconto.

2.1.2.6. PrenotazioniCNT



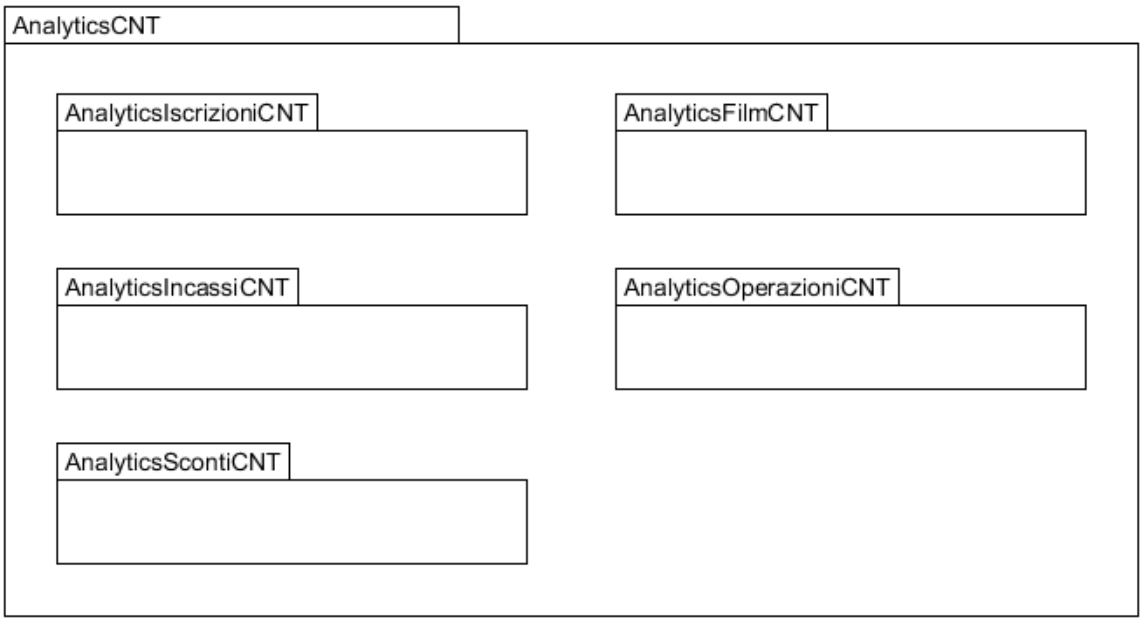
Classe:	**Descrizione:**
PrenotazioneBigliettoCNT	Permette la prenotazione di uno o più posti per uno spettacolo.
DisdettaPrenotazioneCNT	Permette di disdire una prenotazione.

2.1.2.7. AcquistiCNT



Classe:	**Descrizione:**
SelezionaPostiCNT	Permette di visualizzare la griglia dei posti e selezionare quelli desiderati.
AcquistoBigliettoCNT	Permette l'acquisto di un biglietto.
PrenotazioneCNT	Permette di acquistare un biglietto in seguito ad una prenotazione.

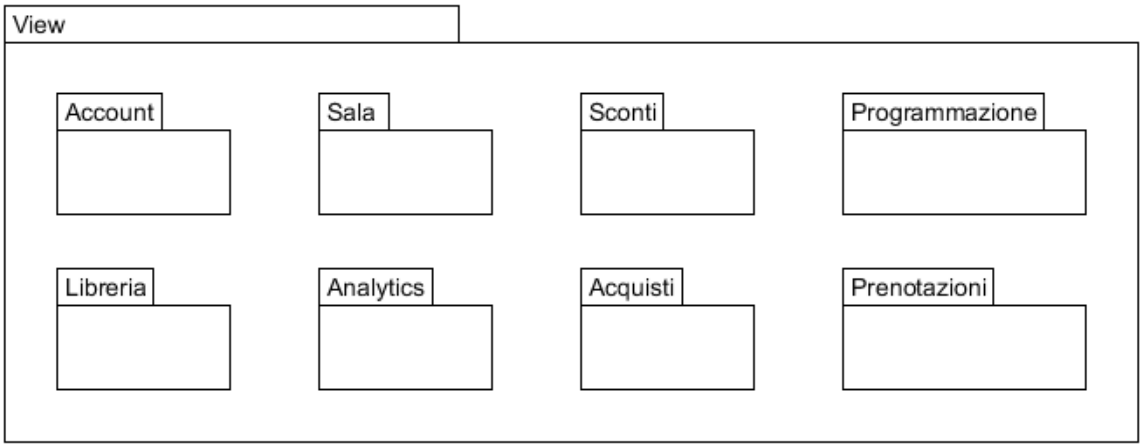
2.1.2.8. AnalyticsCNT



Classe:	**Descrizione:**
AnalyticsIscrizioniCNT	Permette di ottenere le analytics delle iscrizioni.
AnalyticsFilmCNT	Permette di ottenere le analytics dei film.
AnalyticsOperazioniCNT	Permette di ottenere le analytics delle operazioni.
AnalyticsScontiCNT	Permette di ottenere le analytics degli sconti.
AnalyticsIncassiCNT	Permette di ottenere le analytics degli incassi.

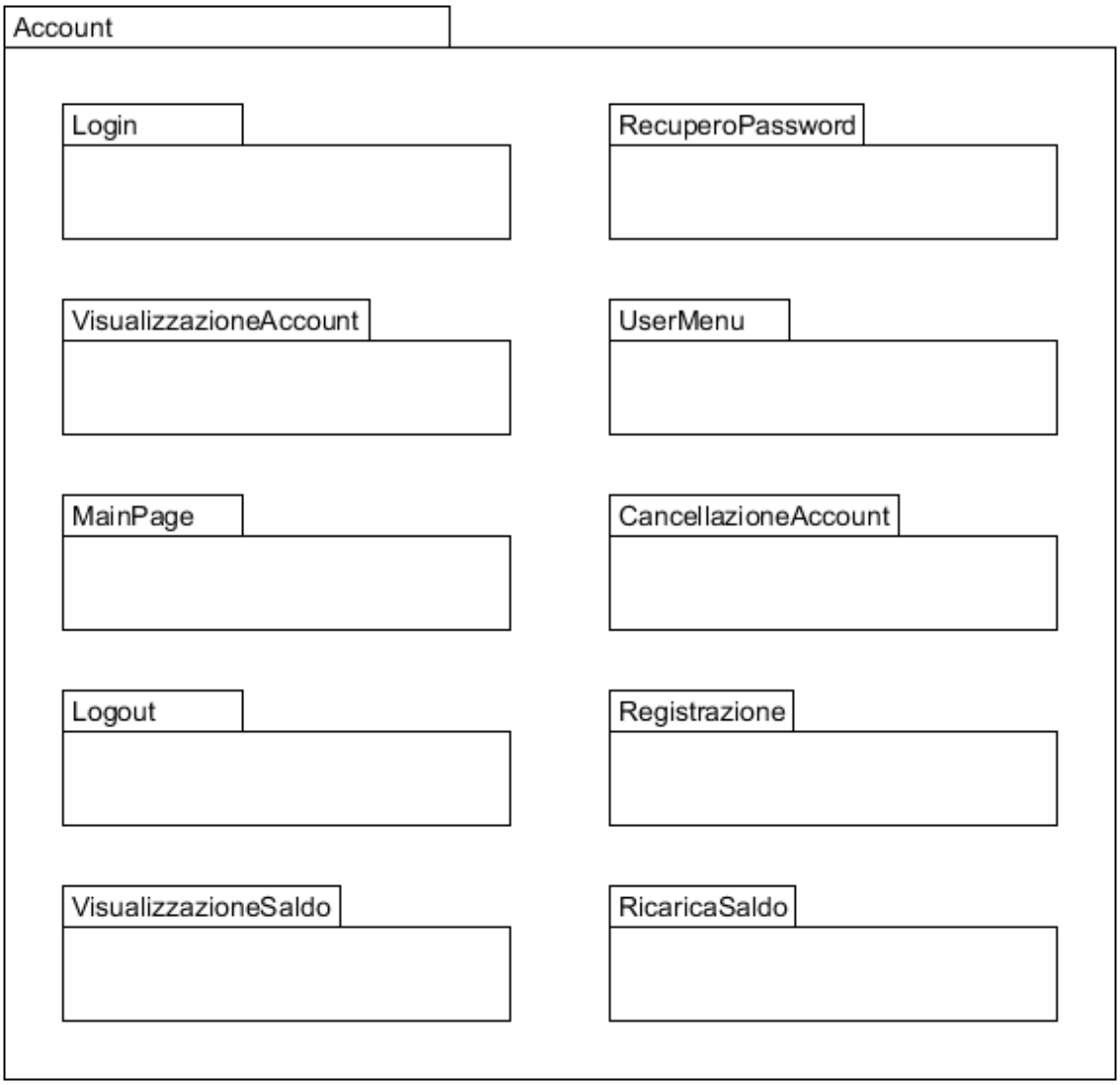
2.1.3. View

Il sottopackage “View” è presentato nel seguente schema e contiene le JSP rappresentanti le schermate del sistema che vengono visualizzate.



All’interno del sottopackage troviamo una ulteriore suddivisione in base alle diverse gestioni.

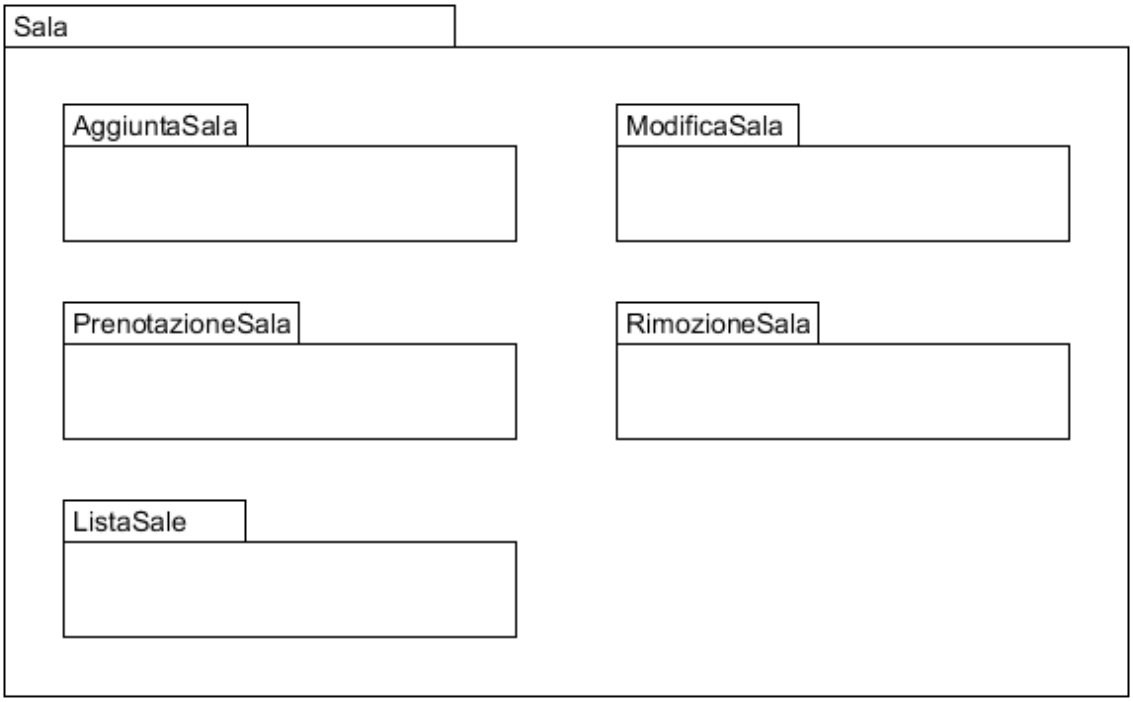
2.1.3.1. Account



Classe:	**Descrizione:**
Login	Rappresenta la schermata di accesso al sistema.
Logout	Rappresenta la schermata per l'uscita dal sistema.
UserMenu	Rappresenta la schermata delle opzioni del menu utente.
MainPage	Rappresenta la schermata principale dell'account.
RecuperoPassword	Rappresenta la schermata per il recupero della password.

VisualizzazioneAccount	Rappresenta la schermata per la visualizzazione dei dettagli dell'account.
Registrazione	Rappresenta la schermata per la registrazione.
CancellazioneAccount	Rappresenta la schermata per la cancellazione di un account.
VisualizzazioneSaldo	Rappresenta la schermata di visualizzazione del saldo.
RicaricaSaldo	Rappresenta la schermata che permette la ricarica del saldo.

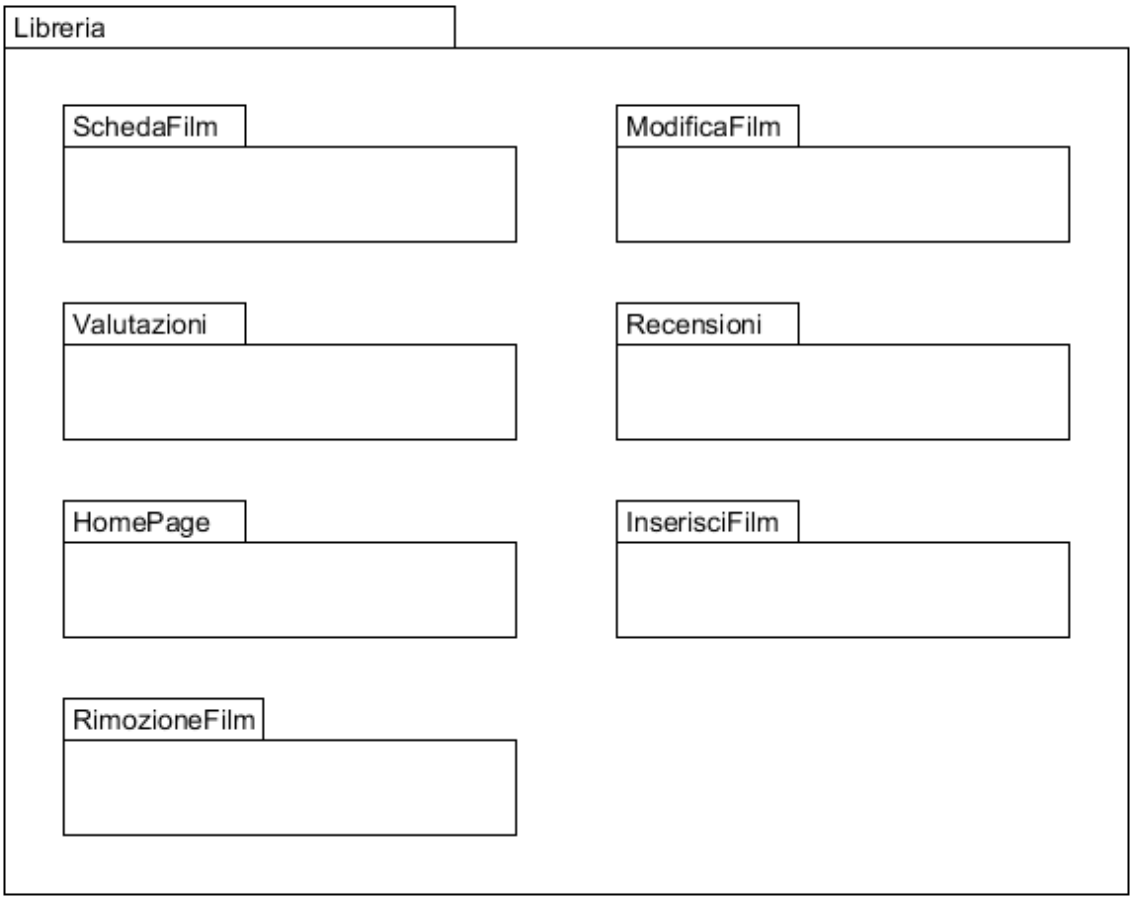
2.1.3.2. Sala



Classe:	**Descrizione:**
AggiuntaSala	Rappresenta la schermata che permette l'aggiunta di una sala.
ListaSale	Rappresenta la schermata che mostra la lista delle sale inserite.
ModificaSala	Rappresenta la schermata che permette la modifica di una sala.
PrenotazioneSala	Rappresenta la schermata che permette la prenotazione di una sala.

RimozioneSala	Rappresenta la schermata che permette la rimozione di una sala.

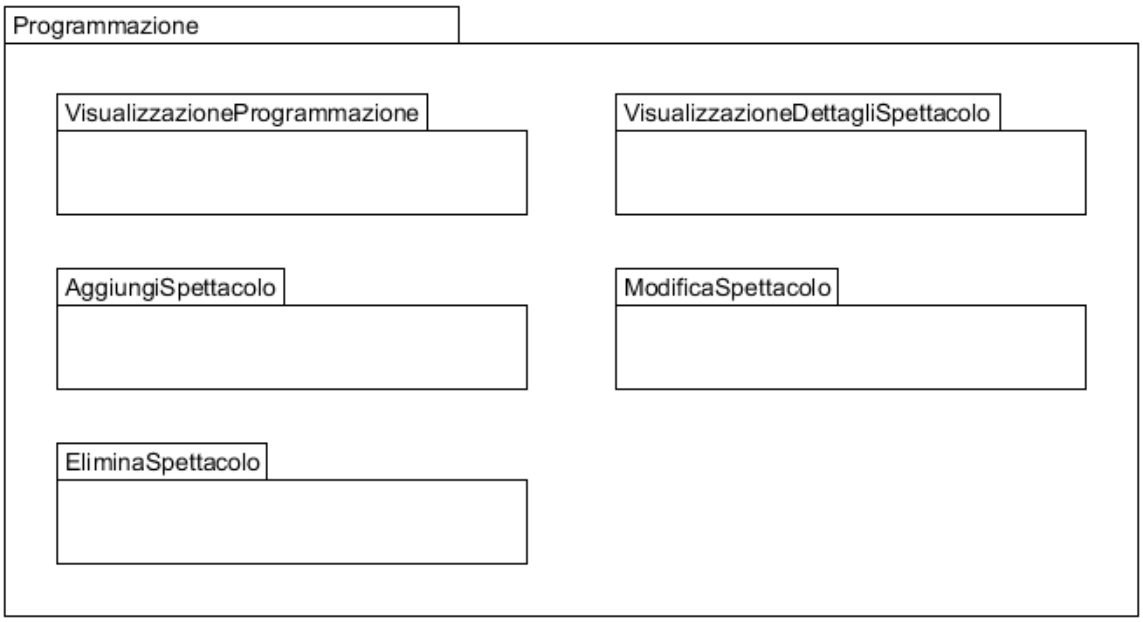
2.1.3.3. Libreria



Classe:	**Descrizione:**
SchedaFilm	Rappresenta la schermata che mostra la scheda di un film.
Valutazioni	Rappresenta la schermata che permette di inserire una valutazione.
InserisciFilm	Rappresenta la schermata che permette di inserire un film.
ModificaFilm	Rappresenta la schermata che permette di modificare un film.
RimozioneFilm	Rappresenta la schermata che permette di rimuovere un film.
Recensioni	Rappresenta la schermata che permette di inserire una valutazione.

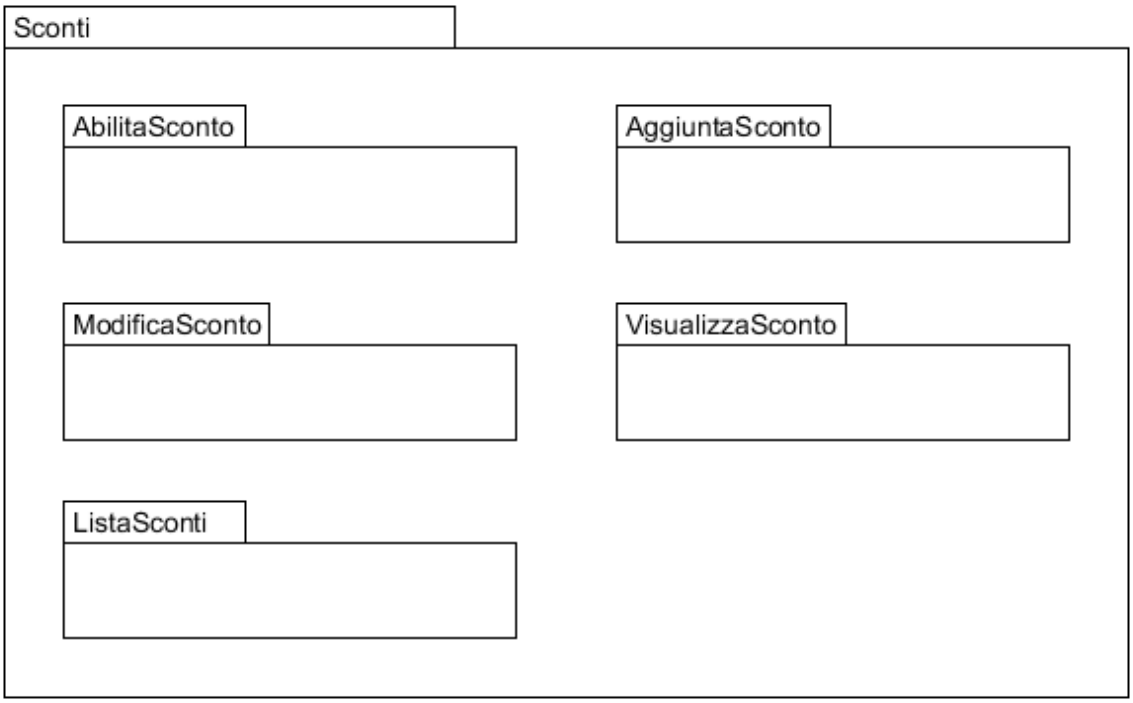
HomePage	Rappresenta la schermata principale mostrata all'accesso alla sezione libreria.

2.1.3.4. Programmazione



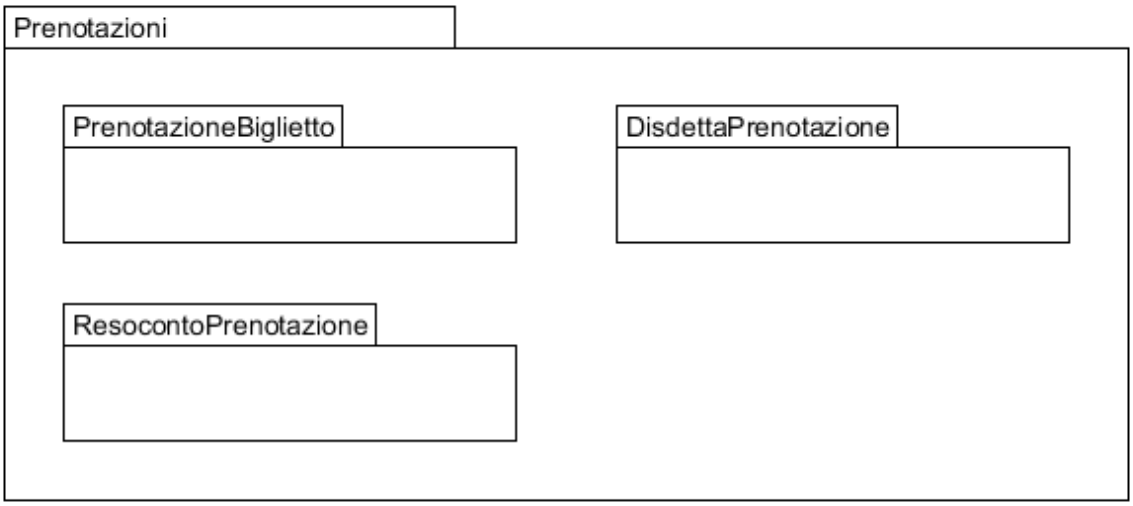
Classe:	**Descrizione:**
VisualizzazioneProgrammazione	Rappresenta la schermata che mostra l'intera programmazione.
VisualizzazioneDettagliSpettacolo	Rappresenta la schermata che mostra i dettagli di uno spettacolo.
AggiungiProgrammazioneSpettacolo	Rappresenta la schermata che permette l'aggiunta di uno spettacolo.
ModificaSpettacolo	Rappresenta la schermata che permette la modifica di uno spettacolo.
EliminazioneSpettacolo	Rappresenta la schermata che permette l'eliminazione di uno spettacolo.

2.1.3.5. Sconti



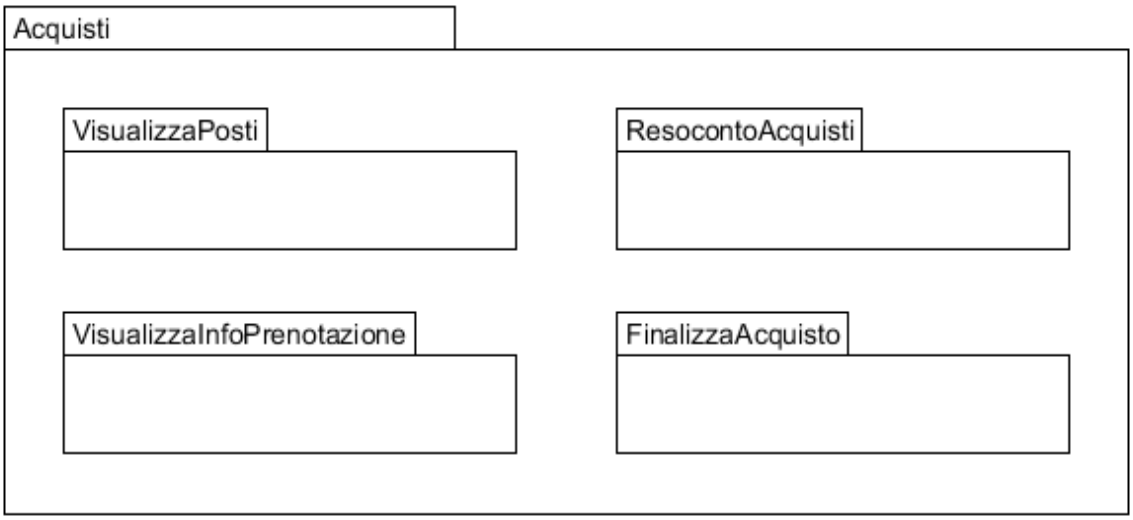
Classe:	**Descrizione:**
AbilitaSconto	Rappresenta la schermata che permette di attivare uno sconto.
AggiuntaSconto	Rappresenta la schermata che permette di aggiungere uno sconto.
ListaSconti	Rappresenta la schermata che mostra la lista di tutti gli sconti.
ModificaSconto	Rappresenta la schermata che permette la modifica di uno sconto.
VisualizzaSconto	Rappresenta la schermata che mostra i dettagli di uno sconto.

2.1.3.6. Prenotazioni



Classe:	**Descrizione:**
PrenotazioneBiglietto	Rappresenta la schermata che permette di effettuare una prenotazione.
ResocontoPrenotazione	Rappresenta la schermata che riassume i dettagli di una prenotazione.
DisdettaPrenotazione	Rappresenta la schermata che permette di disdire una prenotazione.

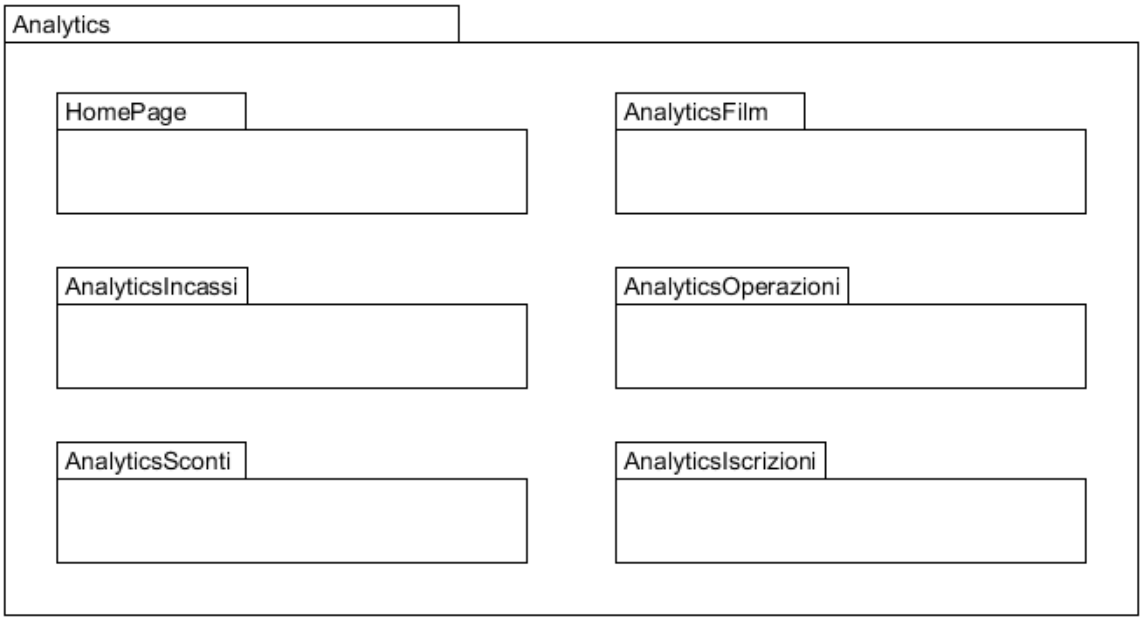
2.1.3.7. Acquisti



Classe:	**Descrizione:**

VisualizzaPosti	Rappresenta la schermata che mostra i posti della sala in cui si intende effettuare l'acquisto.
ResocontoAcquisti	Rappresenta la schermata che mostra il resoconto degli acquisti effettuati.
FinalizzaAcquisto	Rappresenta la schermata che permette di effettuare l'acquisto in modo definitivo.
VisualizzaInfoPrenotazione	Rappresenta la schermata che mostra i dati della prenotazione che si desidera utilizzare per l'acquisto.

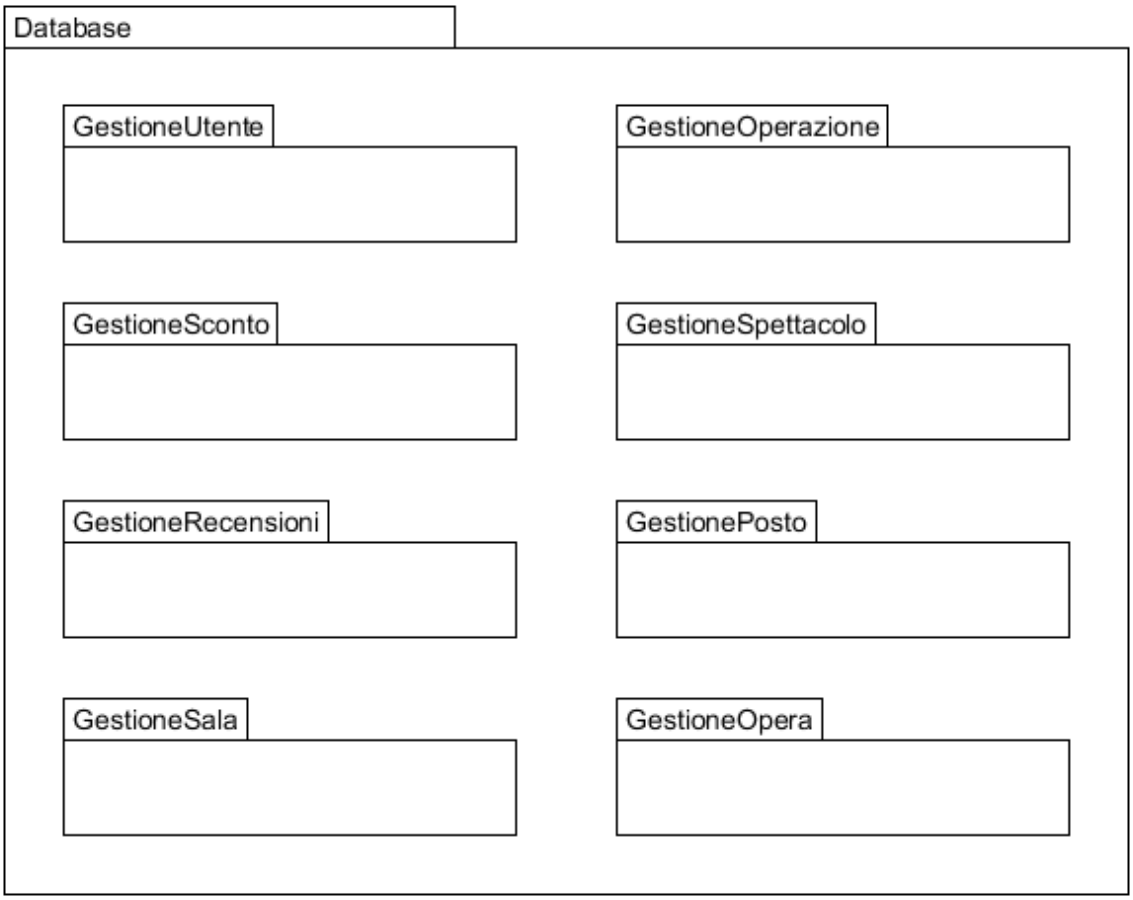
2.1.3.8. Analytics



Classe:	**Descrizione:**
HomePageAnalytics	Rappresenta la schermata principale della sezione Analytics.
AnalyticsIscrizioni	Rappresenta la schermata che mostra la sezione analytics dedicata alle iscrizioni.
AnalyticsFilm	Rappresenta la schermata che mostra la sezione analytics dedicata ai film.
AnalyticsOperazioni	Rappresenta la schermata che mostra la sezione analytics dedicata alle operazioni.
AnalyticsIncassi	Rappresenta la schermata che mostra la sezione analytics dedicata agli incassi.

AnalyticsSconti	Rappresenta la schermata che mostra la sezione analytics dedicata agli sconti.

2.1.4. Database

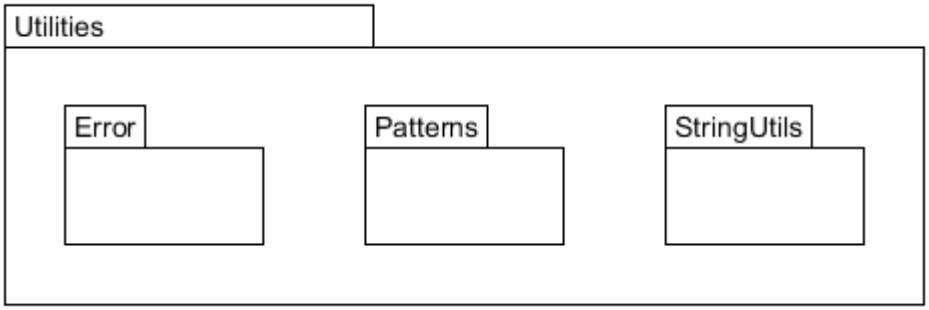


Il sottopackage “Database” è presentato nel seguente schema e contiene le classi Java che si occupano delle interazioni con il Database.

Classe:	**Descrizione:**
GestioneUtente	Gestisce le operazioni che interessano l’entità utente all’interno del database.
GestioneOperazione	Gestisce le operazioni che interessano l’entità operazione all’interno del database.
GestioneSconto	Gestisce le operazioni che interessano l’entità sconto all’interno del database.
GestioneSpettacolo	Gestisce le operazioni che interessano l’entità spettacolo all’interno del database.
GestioneRecensioni	Gestisce le operazioni che interessano

	l'entità recensione all'interno del database.
GestionePosto	Gestisce le operazioni che interessano l'entità posto all'interno del database.
GestioneSala	Gestisce le operazioni che interessano l'entità sala all'interno del database.
GestioneOpera	Gestisce le operazioni che interessano l'entità opera all'interno del database.

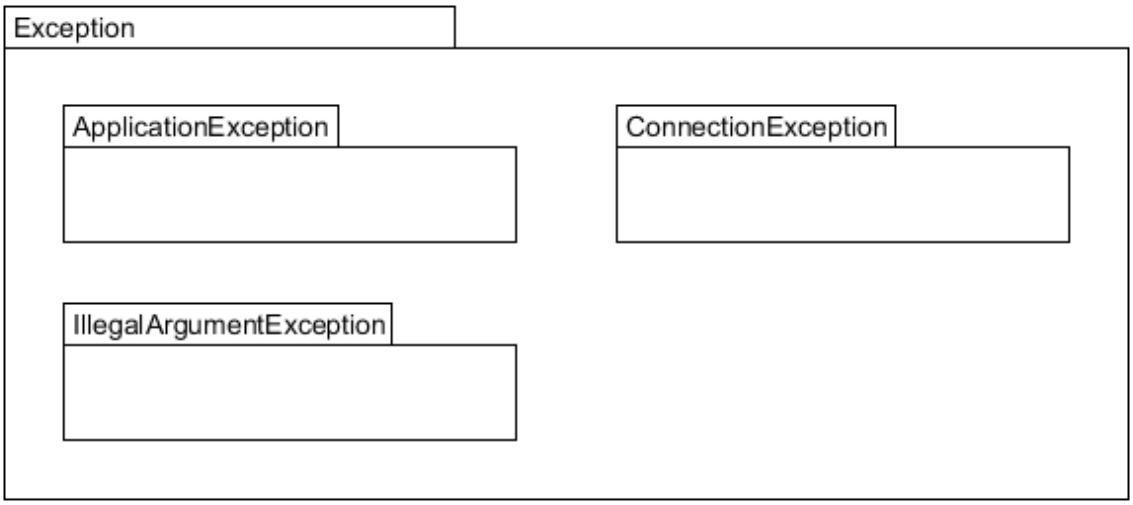
2.1.5. Utilities



Il sottopackage "Utilities" è presentato nel seguente schema e contiene le classi Java che si occupano della logica trasversale del sistema.

Classe:	**Descrizione:**
Error	Contiene le costanti stringhe utilizzate nei model per quanto attiene alle eccezioni.
Patterns	Contiene le espressioni regolari sotto forma di costanti stringhe.
StringUtils	Contiene metodi statici necessari per elaborare le stringhe.

2.1.6. Exception



Il sottopackage “Exception” è presentato nel seguente schema e contiene le classi Java che si occupano della gestione delle eccezioni.

Classe:	**Descrizione:**
ApplicationException	Eccezione lanciata in caso di esecuzione di una query non riuscita.
ConnectionException	Eccezione lanciata in caso di connessione con il database non riuscita.
IllegalArgumentException	Eccezione lanciata in caso i parametri passati ad un metodo non siano corretti.

3. Interfaccia delle classi

3.1. Gestione Account

- public void login (String email, String password)
  Effettua il login di un utente.

- public void logout (String email)
  Effettua il logout di un utente.

- public void registrazioneOperatore (String nomeUtente, String password, String nome, String cognome, String email, String cellulare, String indirizzo, GregorianCalendar dataNascita, String sesso)
  Registra un nuovo account operatore.

- public void registrazioneUtente (String idUtenteBase, float saldo, String nomeUtente, String password, String nome, String cognome, String email, String cellulare, String indirizzo, GregorianCalendar dataNascita, String sesso)
  Registra un nuovo account utente base.

- public void cancellaAccount (String email, Gestore richiedente)
  Permette al gestore di cancellare un account di un utente.

- public void recuperoPassword (String email)
  Recupera la password di un utente.

- public String visualizzaAccount (String nomeUtente, String password, String nome, String cognome, String email, String cellulare, String indirizzo, GregorianCalendar dataNascita, String sesso)
  Visualizza le informazioni dell'account di un utente.

- public void modificaAccount (String nomeUtente, String password, String nome, String cognome, String email, String cellulare, String indirizzo, GregorianCalendar dataNascita, String sesso)
  Modifica le informazioni dell'account di un utente.

- public float visualizzaSaldo (float saldo)
  Visualizza il saldo di un utente.

- public float ricaricaSaldo (float saldo, String idUtenteBase)
  Ricarica il saldo di un utente

3.2. Gestione Sala

- public void eliminaSala (String idSala)
  Elimina una sala.

- public void insertSala (String idSala, int numeroPosti, string statoSala)
  Crea una nuova sala.

- public void modificaSala (String idSala, int numeroPosti, string statoSala)
  Modifica i dettagli di una sala.

- public void occupazioneSala (String idSala)
  Occupa un'intera sala.

3.3. Gestione Libreria

- public ArrayList getLibreria ()

Visualizza la libreria completa.

- public String insertRecensioneFilm (String idFilm, String email, String testo)
  Recensisce un film.

- public void visualizzaSchedaFilm (String idFilm, String titolo, String descrizione, String genere, GregorianCalendar annoUscita, String locandina)
  Visualizza i dettagli di un film.

- public float valutazioneFilm (String idFilm, String email, float valutazione)
  Inserisce una valutazione ad un film.

- public void insertFilm (String idFilm, String titolo, String descrizione, String genere, GregorianCalendar annoUscita, String locandina)
  Inserisce un nuovo film in libreria.

- public void modificaFilm (String idFilm, String titolo, String descrizione, String genere, GregorianCalendar annoUscita, String locandina)
  Modifica un film in libreria.

- public void rimozioneFilm (String idFilm)
  Rimuove un film dalla libreria.

- public ArrayList cercaFilmByTitolo (String titolo)
  Cerca il titolo di un film in libreria.

- public Film cercaFilmById (String idFilm)
  Cerca l'ID di un film in libreria.

- public ArrayList cercaFilmByFiltro ()
  Cerca un film in libreria filtrandoli per genere, anno di uscita, regia, cast, durata,
  distribuzione o produzione.

3.4. Gestione Programmazione

- public ArrayList getProgrammazione ()
  Visualizza tutti i film in programmazione.

- public void getDettagliSpettacolo (String idSpettacolo, GregorianCalendar orario, GregorianCalendar dataInizio, GregorianCalendar dataFine, float prezzo, Sala sala)
  Visualizza i dettagli di uno spettacolo in programmazione.

- public void insertSpettacolo (String idSpettacolo, GregorianCalendar orario, GregorianCalendar dataInizio, GregorianCalendar dataFine, float prezzo, Sala sala)
  Inserisce un nuovo spettacolo in programmazione.

- public void modificaSpettacolo (String idSpettacolo, GregorianCalendar orario, GregorianCalendar dataInizio, GregorianCalendar dataFine, float prezzo, Sala sala)
  Modifica uno spettacolo in programmazione.

- public void eliminaSpettacolo(String idSpettacolo)
  Elimina uno spettacolo dalla programmazione

3.5. Gestione Sconti

- public void insertPoliticaSconto (String idSconto, String nome, GregorianCalendar dataInizio, GregorianCalendar dataFine, boolean attivazione, GregorianCalendar dataAttivazione, GregorianCalendar dataDisattivazione, String politica)
  Inserisce una nuova politica di sconto.

- public void modificaPoliticaSconto (String idSconto, String nome, GregorianCalendar dataInizio, GregorianCalendar dataFine, boolean attivazione, GregorianCalendar dataAttivazione, GregorianCalendar dataDisattivazione, String politica)
  Modifica una politica di sconto.

- public void abilitaPoliticaSconto ()
  Abilita una politica di sconto.

- public void disabilitaPoliticaSconto ()
  Disabilita una politica di sconto.

- public ArrayList getPoliticaSconto ()
  Visualizza tutte le politiche di sconto.

3.6. Gestione Prenotazioni

- public String prenotazioneBiglietto (String idSala, String idPosto, String fila, String numero, utenteRegistrato utenterichiedente)
  Prenota un biglietto per uno spettacolo.

- public void disdettaPrenotazione (String idPrenotazione, UtenteRegistrato utenterichiedente)
  Disdice una prenotazione effettuata.

3.7. Gestione Acquisti

- public void acquistoBiglietti (String idSpettacolo, GregorianCalendar orario, GregorianCalendar dataInizio, GregorianCalendar dataFine, float prezzo, Sala sala, UtenteRegistrato utenterichiedente)
  Acquista un biglietto di uno spettacolo.

- public void venditaBiglietti (String idSpettacolo, GregorianCalendar orario, GregorianCalendar dataInizio, GregorianCalendar dataFine, float prezzo, Sala sala, UtenteRegistrato utenterichiedente, Operatore nomeUtente)
  Vende un biglietto di uno spettacolo.

- public void acquistaBigliettoDaPrenotazione (String idPrenotazione)
  Finalizza l'acquisto di un biglietto prenotato in precedenza.

- public void venditaBigliettoDaPrenotazione (String idPrenotazione)
  Finalizza la vendita di un biglietto prenotato in precedenza.

3.8. Gestione Analytics

- public float getListaIncassi ()
  Visualizza tutti gli incassi.

- public float getIncassiSala (String sala)
  Visualizza tutti gli incassi di una sala.

- public ArrayList getListaIscrizioni ()
  Visualizza tutte le informazioni degli utenti.

- public ArrayList getListaFilm ()
  Visualizza tutte le informazioni riguardanti i film.

- public ArrayList requestListaSconti ()
  Visualizza tutte le informazioni sugli sconti.

- public ArrayList requestListaOperazioni ()
  Visualizza tutte le operazioni effettuate dagli operatori.

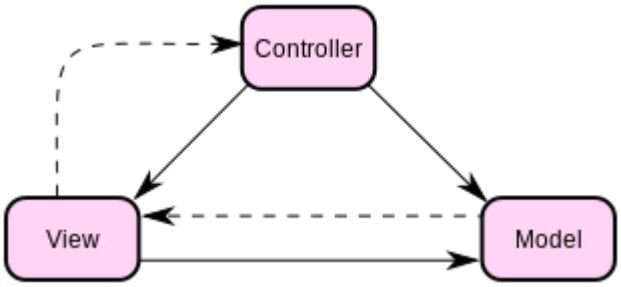
4. Design Patterns

4.1. Model-View-Controller

Il Model-View-Controller (MVC, talvolta tradotto in italiano con la dicitura Modello-Vista-Controllo),Questo pattern si posiziona nel livello di presentazione in una Architettura multi-tier.

Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- il model fornisce i metodi per accedere ai dati utili all'applicazione;
- il view visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- il controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

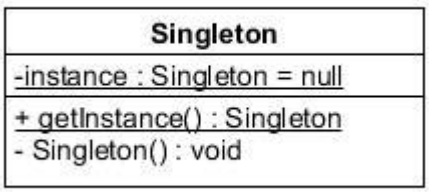


Questo schema, implica anche la tradizionale separazione fra la logica applicativa (in questo contesto spesso chiamata "logica di business"), a carico del *controller* e del *model*, e l'interfaccia utente a carico del *view*.

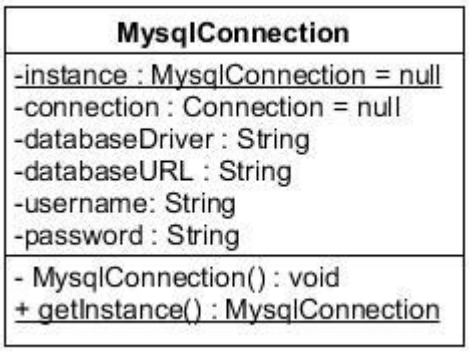
4.2. Singleton

Per aumentare l'efficienza e garantire la sicurezza del sistema è stato deciso di mantenere una sola istanza di connessione al database tramite l'utilizzo del design pattern Singleton.

Il Singleton è un pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.
L'implementazione di questo pattern prevede che la classe singleton abbia un unico costruttore privato, in modo da impedire l'istanziazione diretta della classe. La classe fornisce inoltre un metodo "getter" statico che restituisce l'istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata del metodo, e memorizzandone il riferimento in un attributo privato anch'esso statico. Il secondo approccio si può classificare come basato sul principio della lazy initialization (letteralmente "inizializzazione pigra") in quanto la creazione dell'istanza della classe viene rimandata nel tempo e messa in atto solo quando ciò diventa strettamente necessario (al primo tentativo di uso).
Di seguito il diagramma UML di una classe singleton:

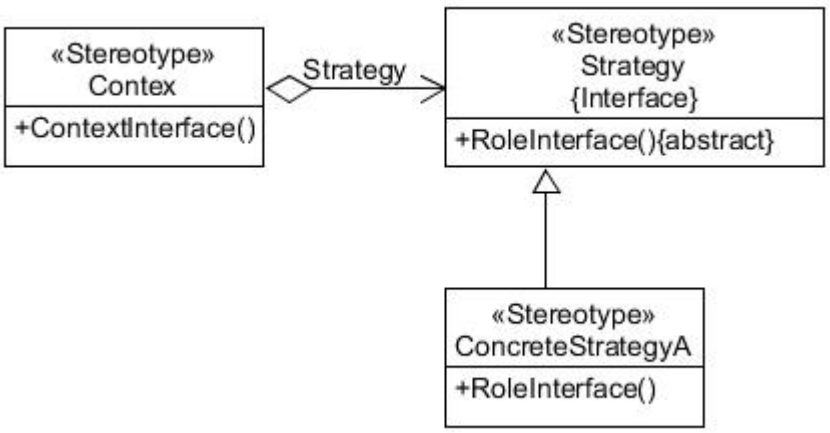


Un esempio di singleton il nostro sistema:



4.3. Strategy

Il pattern Strategy permette di definire una famiglia di algoritmi, di incapsularli e renderli intercambiabili fra loro. Questo pattern consente agli algoritmi di variare in modo indipendente rispetto al loro contesto di utilizzo, fornendo un basso accoppiamento tra le classi partecipanti del pattern e un'alta coesione funzionale delle diverse strategie di implementazione.
Questo pattern prevede che gli algoritmi siano intercambiabili tra loro, in base ad una specificata condizione, in modalità trasparente al client che ne fa uso. In altre parole, data una famiglia di algoritmi che implementa una certa funzionalità, come può essere ad esempio un algoritmo di visita oppure di ordinamento, essi dovranno esportare sempre la medesima interfaccia, così il client dell'algoritmo non dovrà fare nessuna assunzione su quale sia la strategia istanziata in un particolare istante.

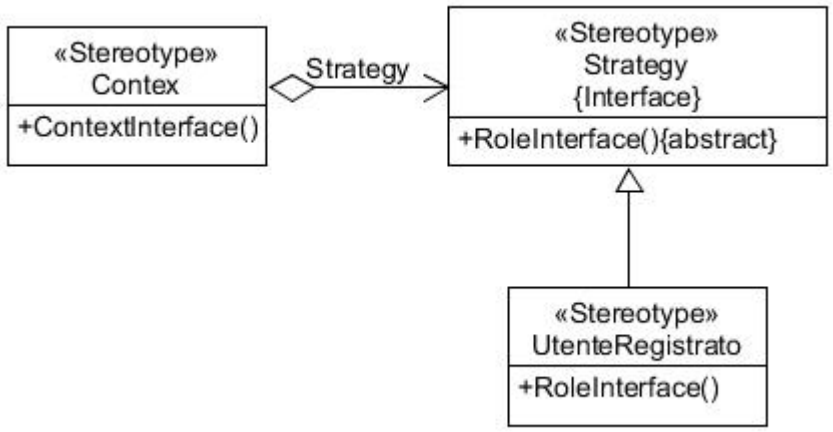


I partecipanti di questo pattern sono:

- ***Strategy***
- Dichiara l'interfaccia di riferimento per tutti gli algoritmi concreti.

- ***ConcreteStrategy***
- Implementa un particolare algoritmo utilizzando l'interfaccia definita da *Strategy*.
- ***Context***
- Carica un oggetto *ConcreteStrategy* e utilizza un riferimento a *Strategy* per eseguire l'algoritmo concreto. Definisce l'interfaccia per accedere ai membri dell'algoritmo caricato.

Nel nostro sistema questo pattern sarà utilizzato per riconoscere i vari ruoli degli utenti presenti nel sistema.



4.4. Data-Access-Object

Il *DAO* (*Data Access Object*) è un pattern architetturale per la gestione della persistenza. Si tratta di una classe usata principalmente in applicazioni web sia di tipo Java EE sia di tipo EJB, per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) ovvero al *data layer* da parte della *business logic* creando un maggiore livello di astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic.
Il vantaggio relativo all'uso del DAO è dunque il mantenimento di una rigida separazione tra le componenti di un'applicazione, le quali potrebbero essere il "Modello" e il "Controllo" in un'applicazione basata sul paradigma MVC.