Università degli Studi di Salerno
Corso di Ingegneria del Software

Agriturismo "Il Miglio"
ODD – Object Design Document
Versione 2.0

Anno Accademico 2018/2019

Top Manager:

Professore
De Lucia Andrea

Partecipanti:

Nome	Matricola
Cusano Romina	0512104608
Ferrara Federica	0512104962
Garelli Luca	0512104944
Iannone Giancarlo	0512104632

Scritto da:	Membri del Team

Revision History

Data	Versione	Descrizione	Autore
05/12/2018	1.0	Stesura del documento	Membri del Team
14/12/2018	1.1	Stesura del documento	Membri del Team
03/01/2019	1.2	Revisione	Membri del Team
08/02/2019	2.0	Ultima Revisione	Membri del Team

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 obiettivi 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 software open source.

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 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
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili:

- I nomi delle variabili devono iniziare con la lettera minuscola, seguiti dal carattere underscore “_” per separare le singole parole.
  Esempio: codice_prodotto
- 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.

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()

Classi e pagine:

- I nomi delle classi devono iniziare con la lettera maiuscola, e anche le parole successive all’interno del nome devono iniziare con la lettera maiuscola.
- I nome delle pagine devono iniziare con la lettera minuscola e 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.
  Esempio: gestoreProdotti.jsp

Ogni file sorgente .class contiene una singola classe e dev’essere strutturato in un determinato modo.

L’introduzione alla classe conterrà:

```
/*
* scopo della classe.
*/
```

La dichiarazione di una classe è caratterizzata da:

- Dichiarazione della classe pubblica
- Dichiarazioni di costanti
- Dichiarazioni di variabili di classe
- Dichiarazioni di variabili d’istanza
- Costruttore
- Commento e dichiarazione metodi e variabili

```java
package ilmiglio.model;

import java.sql.Date;

public class AcquistoBean {
    private int codice_scontrino;
    private Date data_acquisto;
    private double totale;
    private String email_utente;

    /**
     * Restituisce la data di acquisto
     *
     * @return data_acquisto Date
     */
    public Date getData_acquisto() {
        return data_acquisto;
    }

    /**
     * Imposta la data di acquisto
     *
     * @param data_acquisto Date
     */
    public void setData_acquisto(Date data_acquisto) {
        this.data_acquisto = data_acquisto;
    }

    /**
     * Restituisce il totale pagato
     *
     * @return totale double
     */
    public double getTotale() {
        return totale;
    }

    /**
     * Imposta il totale pagato
     *
     * @param totale double
     */
    public void setTotale(double totale) {
        this.totale = totale;
    }

    /**
     * Restituisce l'email del cliente
     *
     * @return email_utente String
     */
    public String getEmail_utente() {
        return email_utente;
    }

}
```

Package:

I nomi dei pacchetti dovranno essere scritti in minuscolo concatenando un insieme di sostantivi separati dal punto. In generale, un nome comincia con il dominio di primo livello dell’organizzazione che lo produce, seguito dal dominio e da altri eventuali sottodomini, elencati in ordine inverso. L’organizzazione può infine scegliere un nome specifico per quel particolare package.
Non sono ammessi caratteri speciali.

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

Libro di testo: B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall
Documento RAD_AIM.pdf del progetto Agriturismo il Miglio
Documento SDD_AIM.pdf del progetto Agriturismo il Miglio

2. PACKAGES

La gestione del nostro sistema è suddivisa in tre livelli (three-tier):

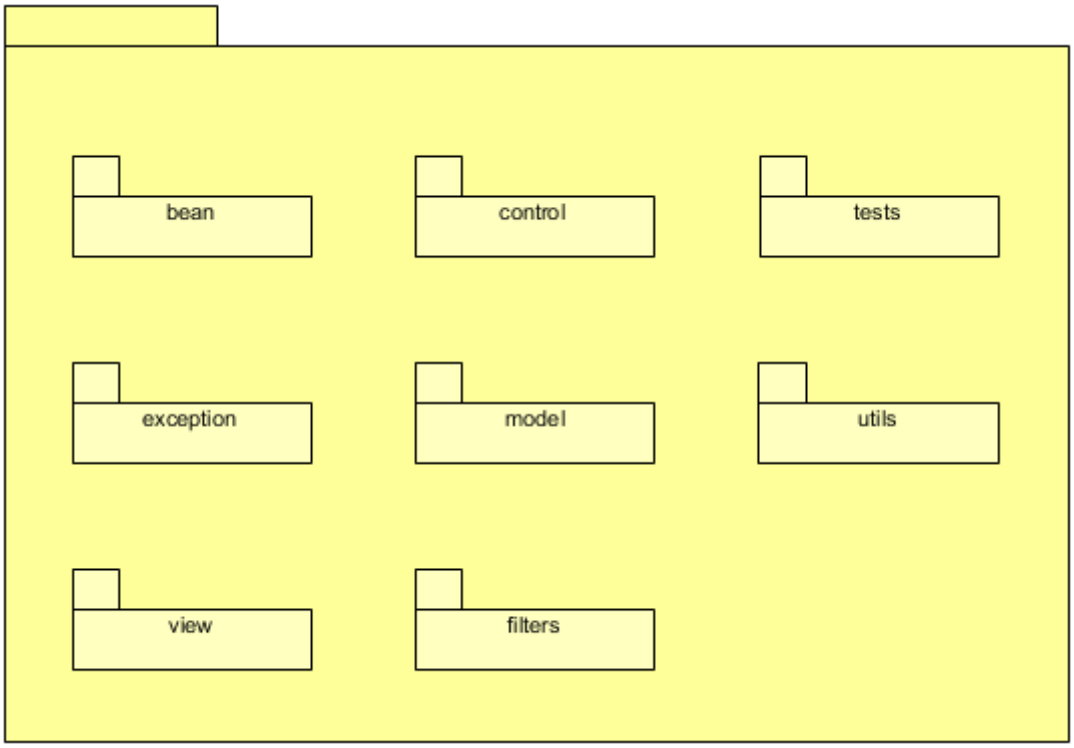
- Interface layer
- Application Logic layer
- Storage layer

Il package AIM contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

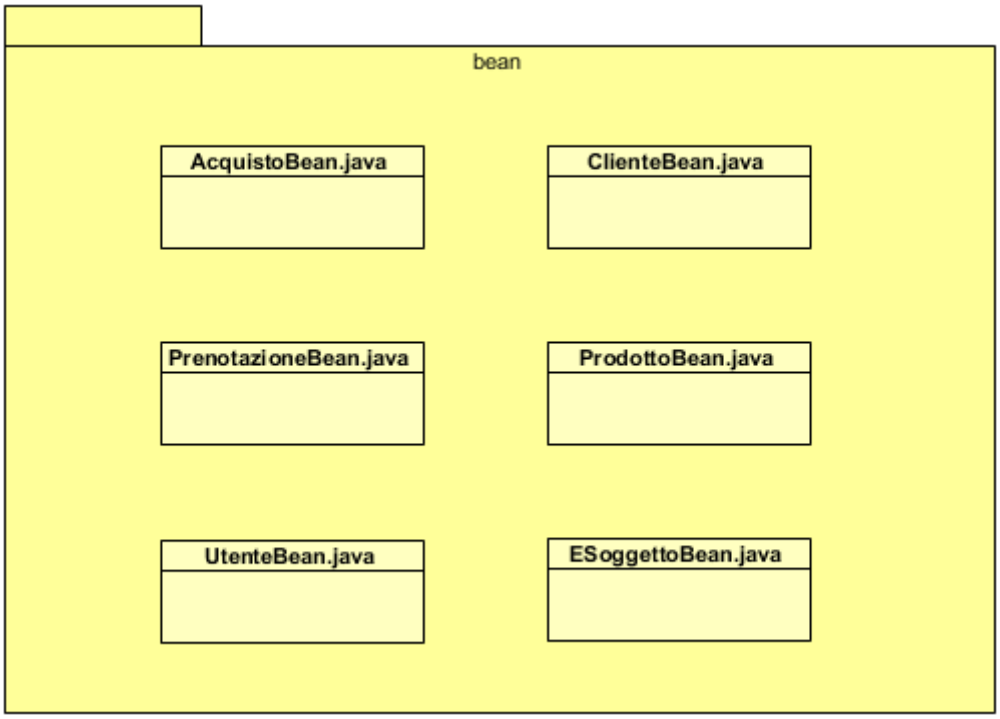
Interface layer	Rappresenta l’interfaccia del sistema, ed offre la possibilità all’utente di interagire con quest’ultimo, offrendo sia la possibilità di inviare, in input, che di visualizzare, in output, i dati.

Application Logic layer	Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite lo Storage Layer, accede ai dati persistenti. Si occupa di varie gestioni quali: 1. Gestione Registrazione 2. Gestione Autenticazione 3. Gestione Account 4. Gestione Prodotti 5. Gestione Prenotazioni

Storage layer	Ha il compito di memorizzare i dati sensibili del sistema, utilizzando un DBMS, inoltre riceve le varie richieste dall’Application Logic layer inoltrandole al DBMS e restituendo i dati richiesti.

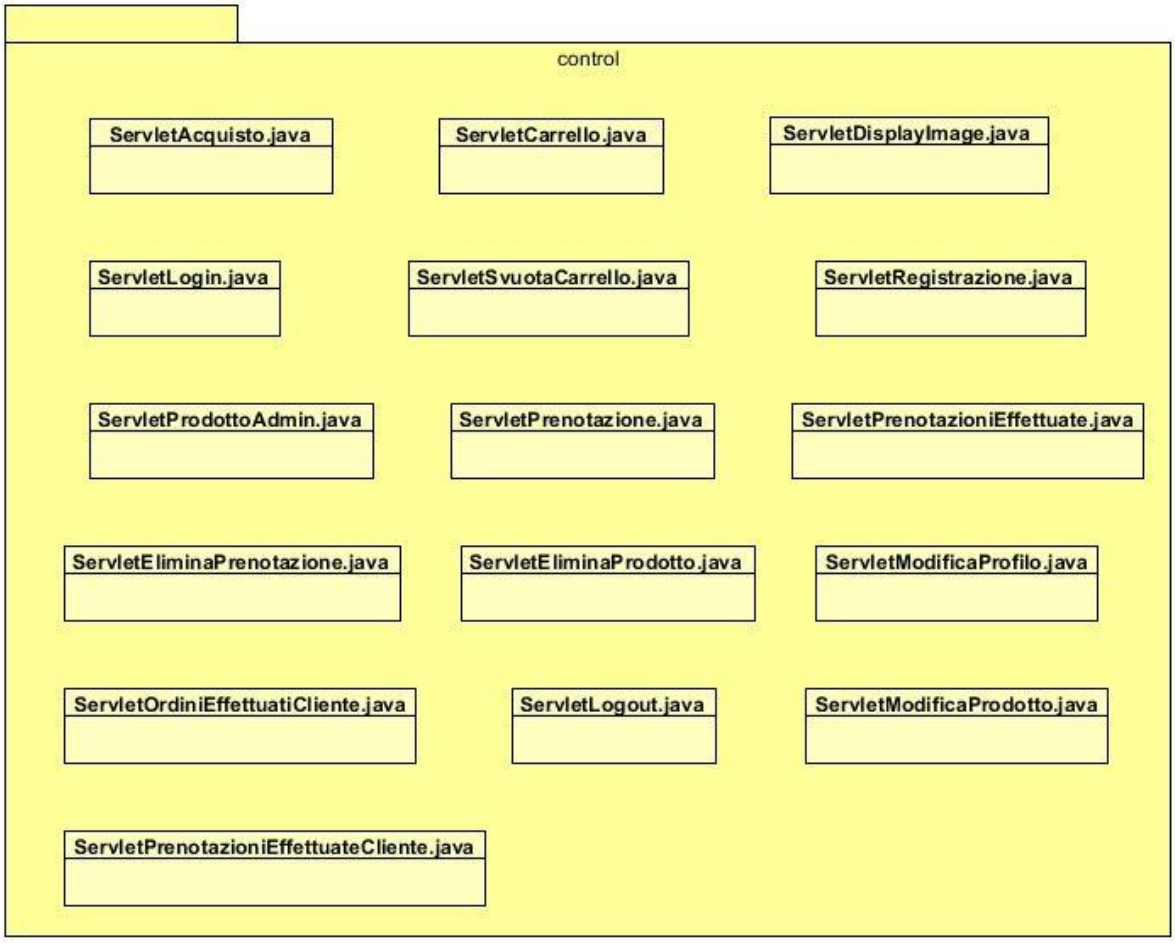

bean
control
tests
exception
model
utils
view
filters

2.1. Package bean



CLASSE	DESCRIZIONE
AcquistoBean.java	Classe che rappresenta le informazioni degli acquisti
ClienteBean.java	Classe che rappresenta le informazioni di un cliente
ESoggettoBean.java	Classe che rappresenta le informazioni degli acquisti
PrenotazioneBean.java	Classe che rappresenta le informazioni delle prenotazioni effettuata
ProdottoBean.java	Classe che rappresenta le informazioni dei prodotti presenti nello shop
UtenteBean.java	Classe che rappresenta le informazioni per effettuare il login

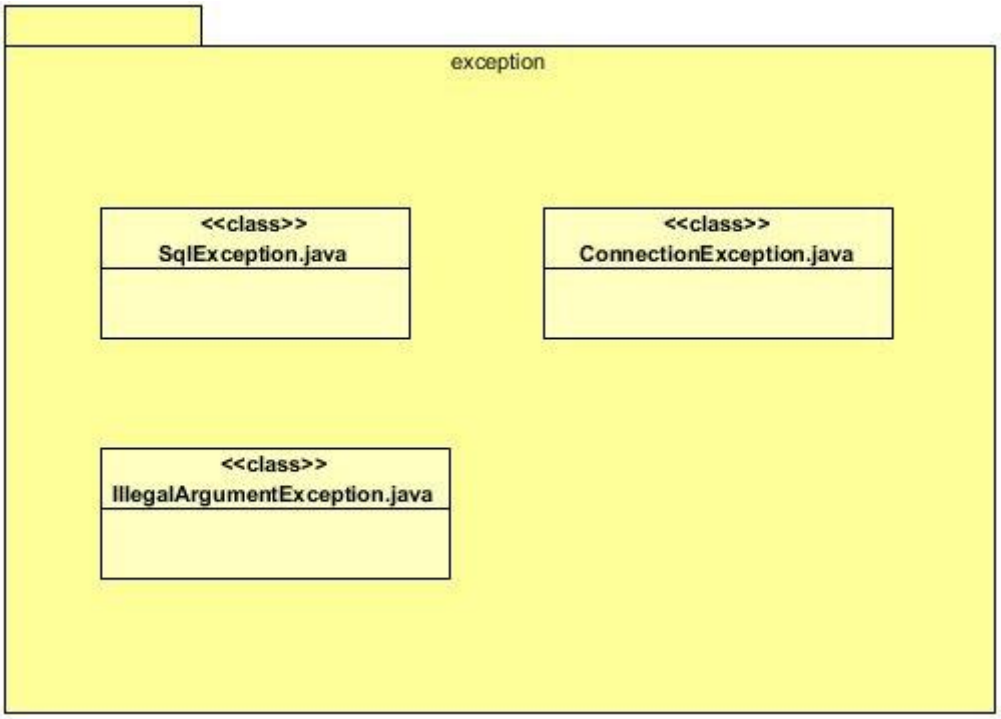
2.2. Package control



CLASSE	DESCRIZIONE
ServletAcquisto.java	Servlet che gestisce l’acquisto
ServletCarrello.java	Servlet che gestisce il carrello
ServletDisplayImage.java	Servlet che gestisce le immagini
ServletEliminaProdotto.java	Servlet che gestisce l’eliminazione di un prodotto
ServletLogin.java	Servlet che gestisce il login al sistema
ServletLogout.java	Servlet che gestisce il logout dal sistema
ServletPrenotazione.java	Servlet che gestisce una prenotazione
ServletPrenotazioniEffettuate.java	Servlet che gestisce le prenotazioni effettuate
ServletPrenotazioniEffettuateCliente.java	Servlet che gestisce le prenotazioni effettuate dai clienti
ServletProdottoAdmin.java	Servlet che gestisce i prodotti
ServletRegistrazione.java	Servlet che gestisce la registrazione

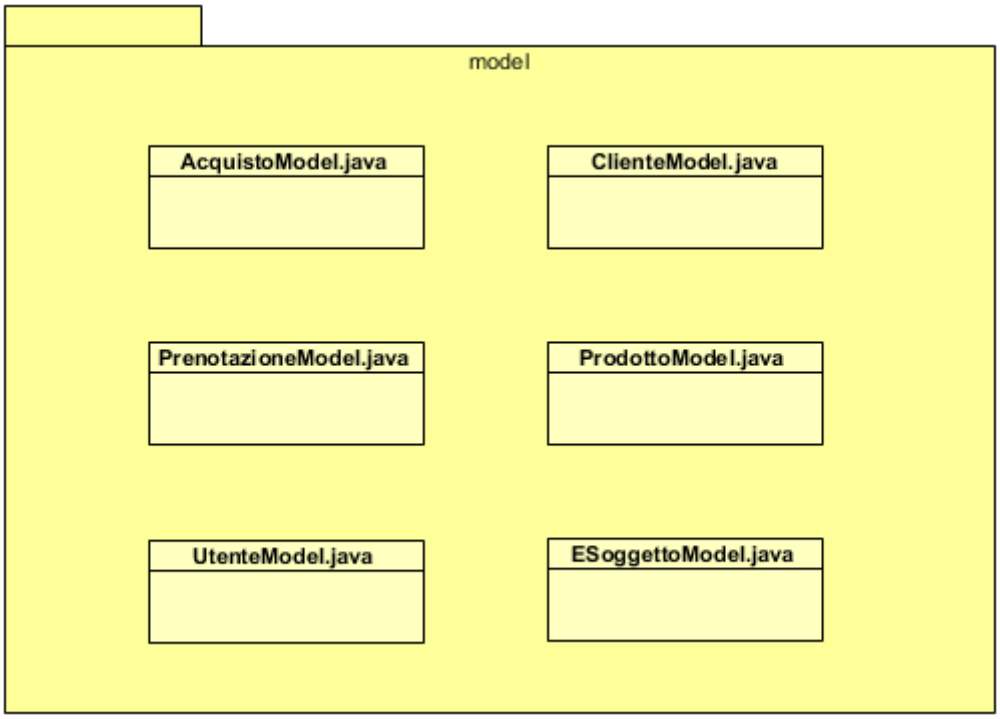
ServletUploadImmagine.java	Servlet che gestisce il caricamento delle immagini
ServletModificaProdotto.java	Servlet che gestisce la modifica dei prodotti
ServletOrdiniEffettuatiCliente.java	Servlet che gestisce gli ordini effettuati dal cliente
ServletModificaProfilo.java	Ervlet che gestisce la modifica del profilo
ServletEliminaPrenotazione.java	Servlet che gestisce l’annullamento della prenotazione

2.3. Package exception



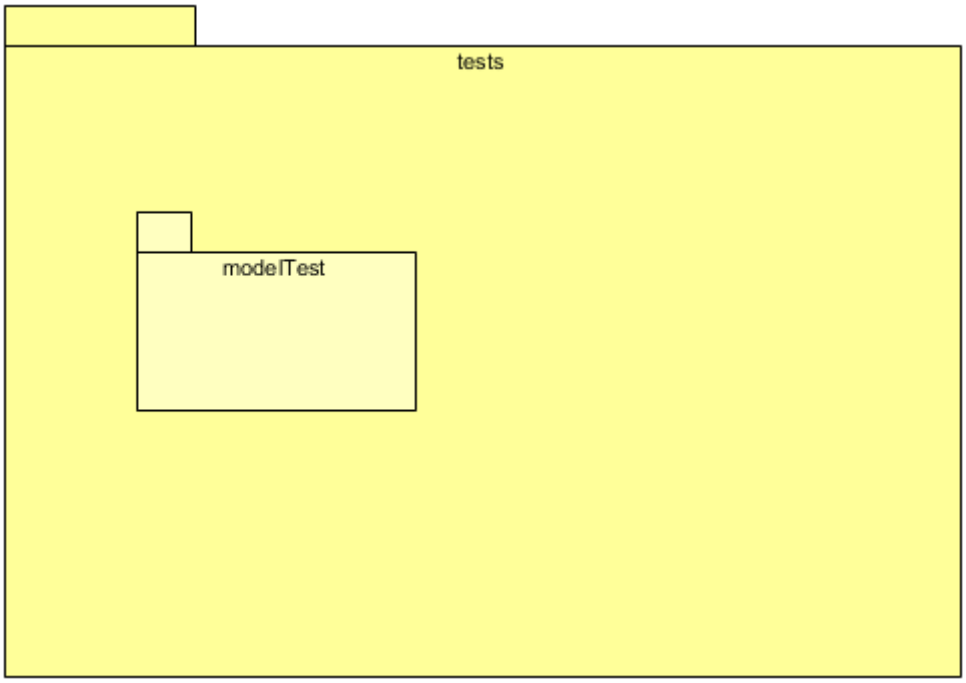
CLASSE	DESCRIZIONE
SqlException.java	L’eccezione che viene lanciata quando non viene eseguita una query
ConnectionException.java	L’eccezione che viene lanciata quando non c’è connessione al database
IllegalArgumentException.java	L’eccezione che viene lanciata quando i parametri passati ad un metodo non sono corretti

2.4. Package model

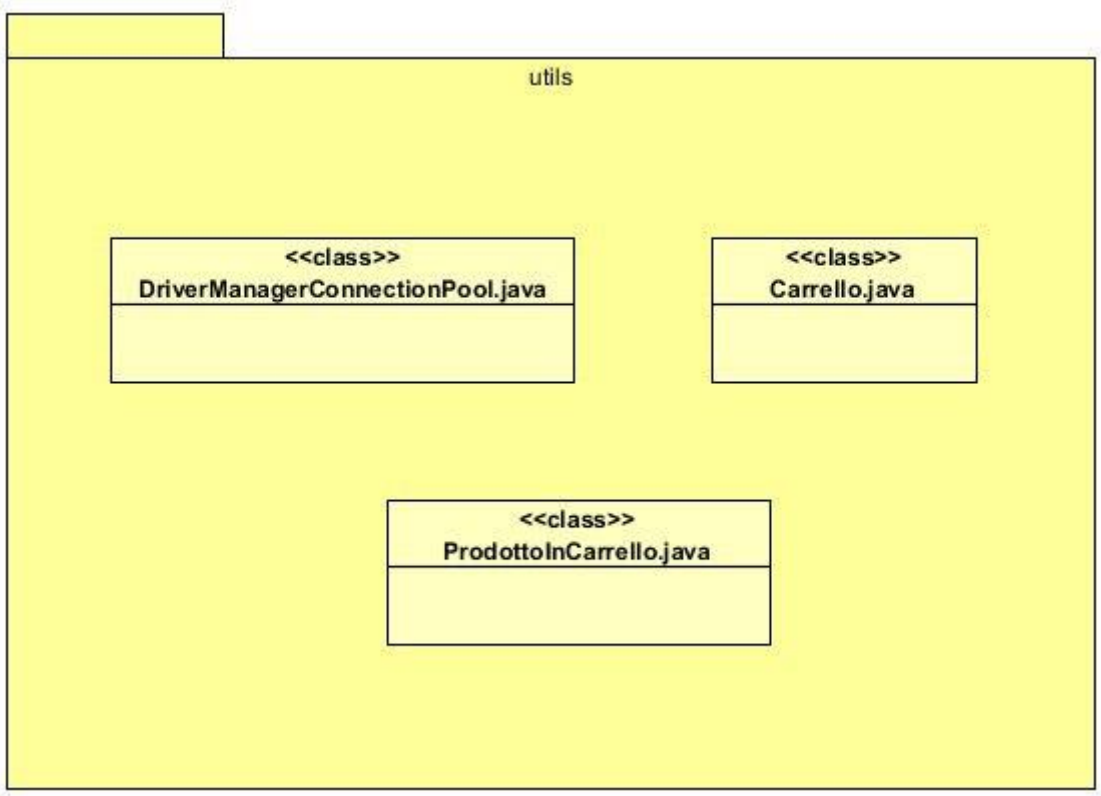


CLASSE	DESCRIZIONE
AcquistoModel.java	Il model che effettua tutte le query riguardanti l’acquisto, interfacciandosi al database al quale è connesso
ClienteModel.java	Il model che effettua tutte le query riguardanti il cliente, interfacciandosi al database al quale è connesso
ESoggettoModel.java	Il model che effettua tutte le query riguardanti le informazioni dell’acquisto, interfacciandosi al database al quale è connesso
PrenotazioneModel.java	Il model che effettua tutte le query riguardanti le prenotazioni, interfacciandosi al database al quale è connesso
ProdottoModel.java	Il model che effettua tutte le query riguardanti i prodotti, interfacciandosi al database al quale è connesso
UtenteModel.java	Il model che effettua tutte le query riguardanti gli utenti che accedono, interfacciandosi al database al quale è connesso

2.5. Package tests

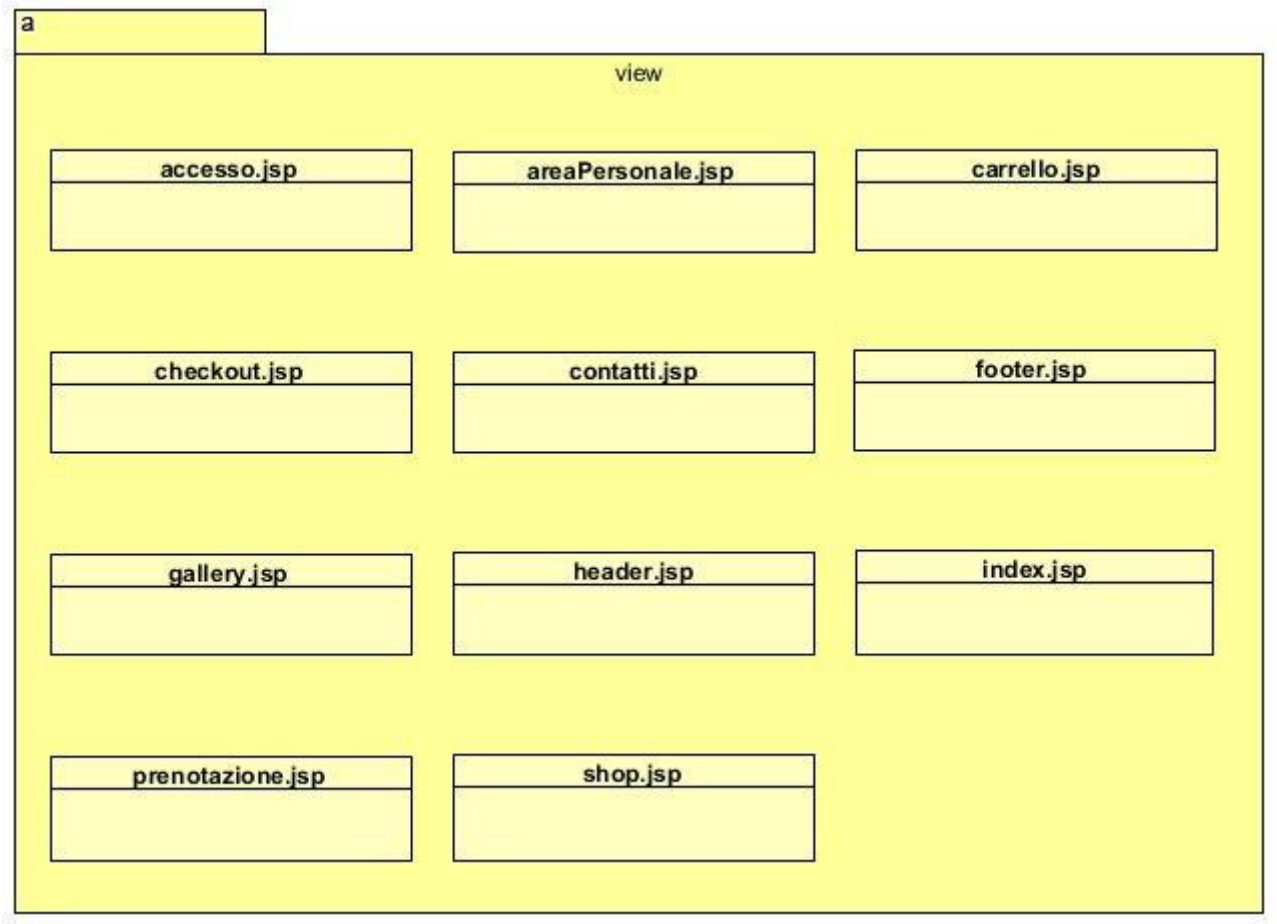


2.6. Package utils



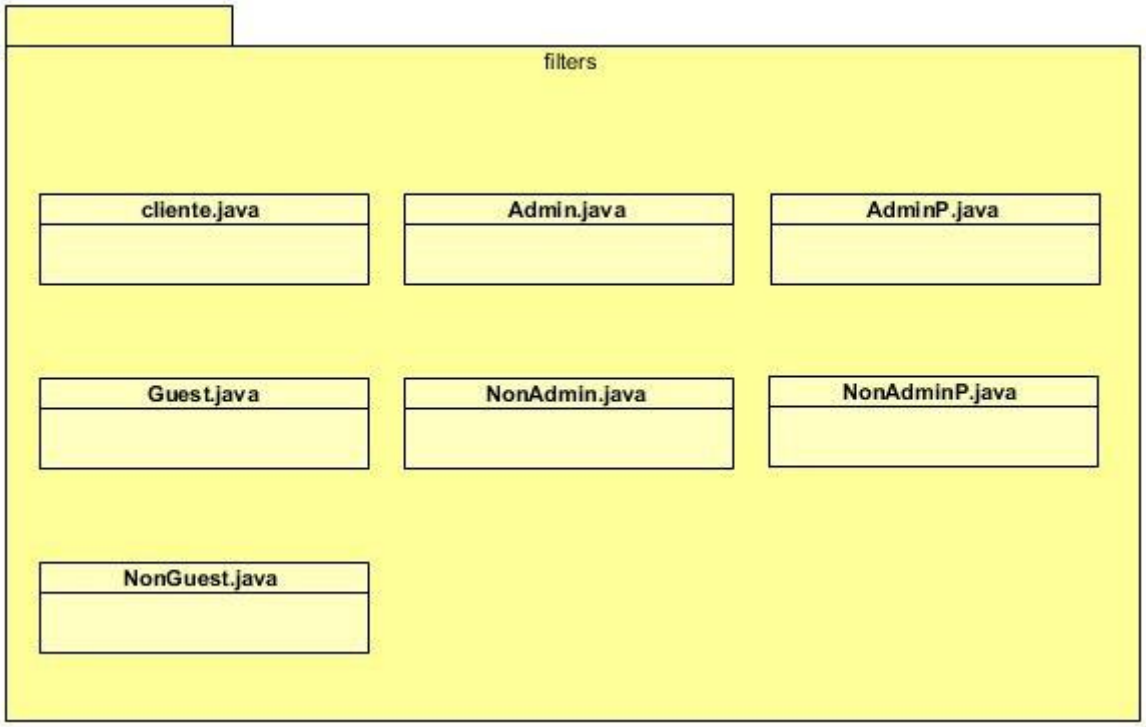
CLASSE	DESCRIZIONE
DriverManagerConnectionPool.java	Classe che gestisce la creazione di una serie di connessioni al database e il mantenimento delle stesse per la realizzazione di una connection pool
Carrello.java	Classe che contiene tutti i prodotti presenti nel carrello
ProdottoInCarrello.java	Classe che contiene la creazione di ogni prodotto

2.7. Package view



CLASSE	DESCRIZIONE
accesso.jsp	Contiene il login e la registrazione
areaPersonale.jsp	Contiene l’area personale del cliente
carrello.jsp	Contiene tutti i prodotti che si desiderano acquistare
checkout.jsp	Contiene le ultime modifiche prima dell’acquisto
contatti.jsp	Contiene le informazioni di contatto
footer.jsp	Contiene il footer comune a tutte le pagine
gallery.jsp	Contiene le immagini
header.jsp	Contiene l’header comune a tutte le pagine
index.jsp	Contiene la pagina principale
prenotazione.jsp	Contiene il form di prenotazione
shop.jsp	Contiene i prodotti in vendita

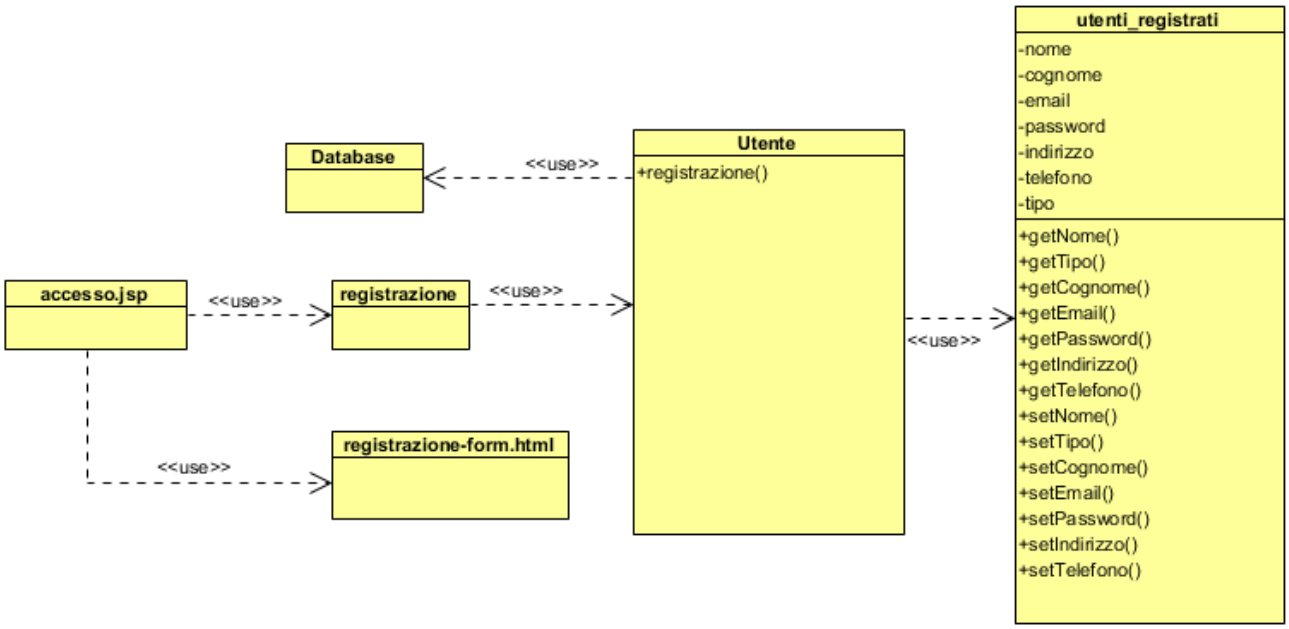
2.8. Package filters



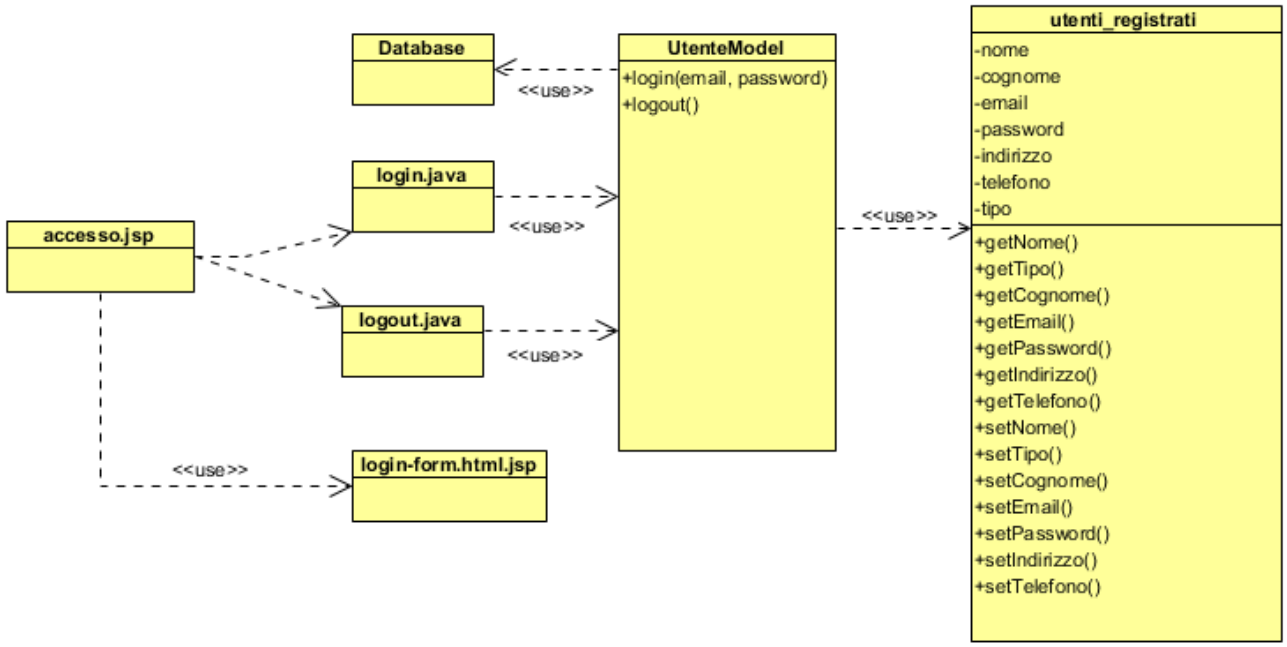
CLASSE	DESCRIZIONE
Cliente.java	Filtro che concede l’accesso alla risorsa solo se l’Utente è Cliente
AdminP.java	Filtro che concede l’accesso alla risorsa solo se l’Utente è Gestore Prenotazioni
Admin.java	Filtro che concede l’accesso alla risorsa solo se l’Utente è Gestore Prodotti
Guest.java	Filtro che consente l’accesso alla risorsa solo se l’Utente è loggato
NonAdminP.java	Filtro che nega l’accesso alla risorsa se l’Utente è Gestore Prenotazioni
NonAdmin.java	Filtro che nega l’accesso alla risorsa se l’Utente è Gestore Prodotti
NonGuest.java	Filtro che concede l’accesso alla risorsa solo se l’Utente è loggato

3. CLASS INTERFACES

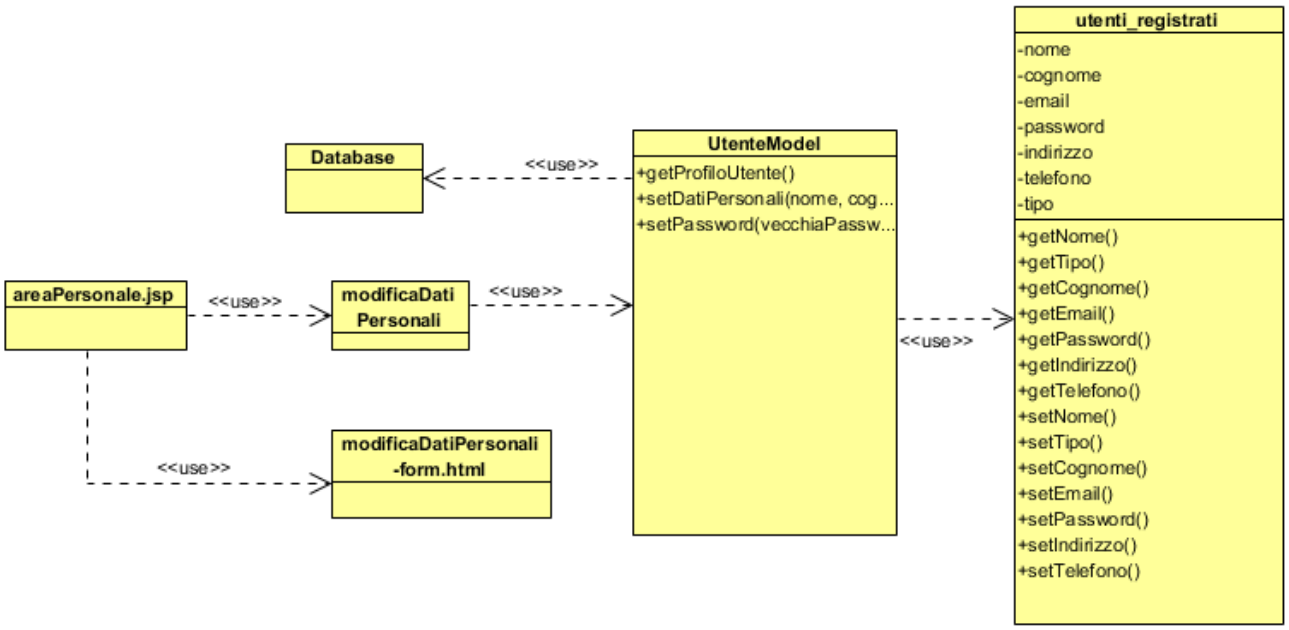
3.1. Gestione registrazione



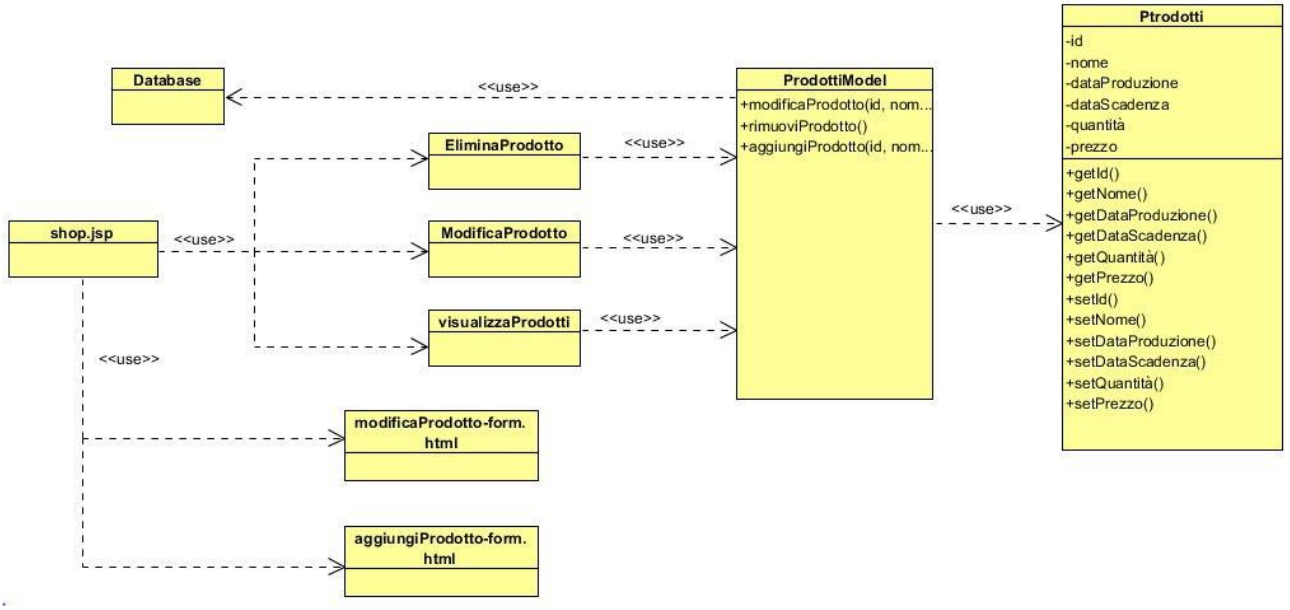
3.2. Gestione autenticazione



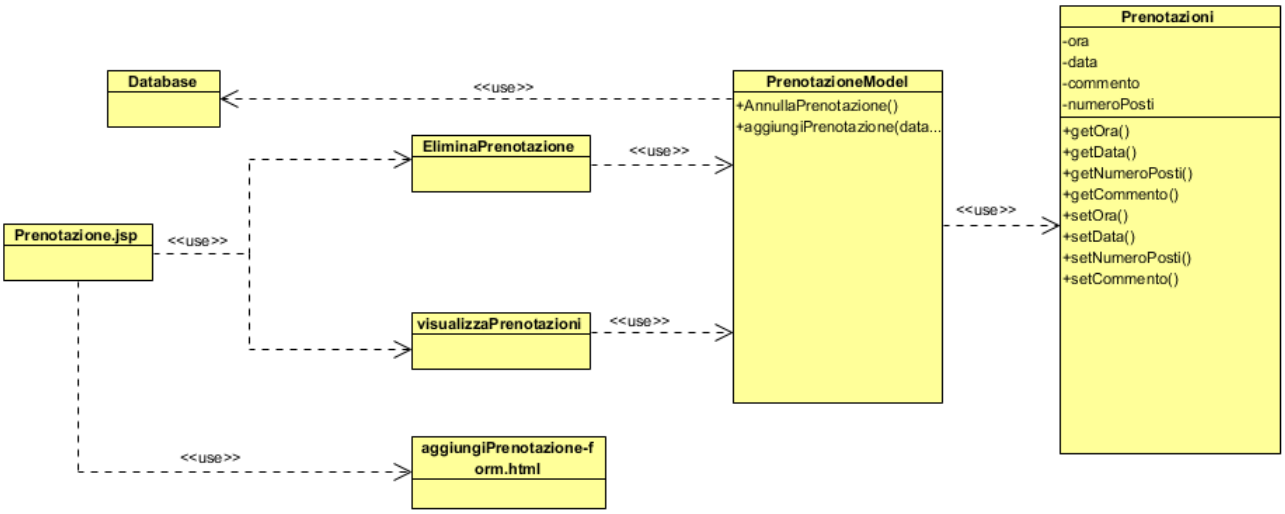
3.3. Gestione account



3.4. Gestione prodotti



3.5. Gestione prenotazioni



BEAN

Nome classe	AcquistoBean
Descrizione	Rappresenta le informazioni relative ad un acquisto
Invarianti	-

Metodi	Context AcquistoBean::getData_acquisto():Date
	Context AcquistoBean::setData_acquisto(data_acquisto:Date):void
	pre: data_acquisto != null;
	
	Context AcquistoBean::getTotale():double
	Context AcquistoBean::setTotale(totale:double):void
	pre: totale != null;
	
	Context AcquistoBean::getEmail_utente():String
	ContextAcquistoBean::setEmail_utente(email_utente:String):void
	pre: email_utente != null;
	
	Context AcquistoBean::getCodice():Int
	Context AcquistoBean::setCodice(codice:int):void
	pre: codice != null;

Nome classe	ClienteBean
Descrizione	Rappresenta le informazioni relative ad un cliente
Invarianti	-

Metodi	Context ClienteBean::getNome_cliente(): String
	Context ClienteBean::setNome_cliente(nome_cliente: String):void
	pre: nome_cliente != null;
	
	Context ClienteBean::getCognome_cliente(): String
	Context ClienteBean::setCognome_cliente(cognome_cliente: String):void
	pre: cognome_cliente != null;
	
	Context ClienteBean::getEmail_cliente():String
	Context ClienteBean::setEmail_cliente(email_cliente:String):void
	pre: email_cliente != null;
	
	Context ClienteBean::getTelefono():String
	Context ClienteBean::setCodice(telefono:String):void
	pre: telefono != null;
	
	Context ClienteBean::getIndirizzo():String
	Context ClienteBean::setIndirizzo(indirizzo:String):void
	pre: indirizzo != null;

Nome classe	ESoggettoBean
Descrizione	Rappresenta le informazioni relative ai prodotti soggetti ad acquisto
Invarianti	-

Metodi	Context ESoggettoBean::getCodice_prodotto():Int
	Context ESoggettoBean::setCodice_prodotto(codice_prodotto:int):void
	pre: codice_prodotto != null;
	
	Context ESoggettoBean::getCodice_scontrino():Int
	Context ESoggettoBean::setCodice_scontrino(codice_scontrino:int):void
	pre: codice_scontrino != null;
	
	Context ESoggettoBean::getQuantita():Int
	Context ESoggettoBean::setQuantita(quantita:int):void
	pre: quantita != null;

Nome classe	PrenotazioneBean
Descrizione	Rappresenta le informazioni relative alle prenotazioni
Invarianti	-

Metodi	Context PrenotazioneBean::getEmail():String
	Context PrenotazioneBean::setEmail(email:String):void
	pre: email != null;
	
	Context PrenotazioneBean::getData():Date
	Context PrenotazioneBean::setData(data:Date):void
	pre: data!= null;
	
	Context PrenotazioneBean::getOrario():String
	Context PrenotazioneBean::setOrario(orario:String):void
	pre: orario != null;
	
	Context PrenotazioneBean::getCommento():String
	Context PrenotazioneBean::setCommento(commento:String):void
	
	Context PrenotazioneBean::getPosti():Int
	Context PrenotazioneBean::setPosti(posti:int):void
	pre: posti != null;

Nome classe	ProdottoBean
Descrizione	Rappresenta le informazioni relative ai prodotti
Invarianti	parzialPath:String

Metodi	Context ProdottoBean::getCodice():int
	Context ProdottoBean::setCodice(codice:int):void
	pre: codice != null;
	
	Context ProdottoBean::getNome_tipo():String
	Context ProdottoBean::setNome_tipo(nome_tipo:String):void
	pre: nome_tipo != null;
	
	Context ProdottoBean::getQuantità_disponibile():int
	Context ProdottoBean::setQuantità_disponibile(quanità_disponibile:int):void
	pre: quantità_disponibile != null;
	
	Context ProdottoBean::getData_produzione():Date
	Context ProdottoBean::setData_produzione(data_produzione:Date):void
	pre: data_produzione!= null;
	
	Context ProdottoBean::getData_scadenza():Date
	Context ProdottoBean::setData_scadenza(data_scadenza:Date):void
	pre: data_scadenza!= null;
	
	Context ProdottoBean::getPrezzo():double
	Context ProdottoBean::setPrezzo(prezzo:double):void
	pre: orario != null;

Nome classe	UtenteBean
Descrizione	Rappresenta le informazioni relative agli utenti
Invarianti	-

Metodi	Context UtenteBean::getEmail():String
	Context UtenteBean::setEmail(email:String):void
	pre: email != null;
	
	Context UtenteBean::getPass():String
	Context UtenteBean::setPass (pass:String):void
	pre: pass != null;
	
	Context UtenteBean::getTipo():String
	Context UtenteBean::setTipo(tipo:String):void
	pre: tipo != null;

MODEL

Nome classe	AcquistoModel
Descrizione	Gestisce le informazioni persistenti nel database attraverso la definizione di metodi di interrogazione, relativi all’acquisto.
Invarianti	-

Metodi	***Context*** AcquistoModel::doRetrieveByEmail(email:String):ArrayList<AcquistoBean>
	pre: email != null;
	post:result-> forAll(r.emailAcquistoBean.email_utente.equals(email));
	
	Context AcquistoModel::insert(acquisto:AcquistoBean):void
	Pre:acquisto!=null;
	post: self.retrieveAll()-> includes(acquisto):

Nome classe	ClienteModel
Descrizione	Definisce i metodi per gestire i dati relativi ai clienti.
Invarianti	-
Metodi	**context** ClienteMode**l**::insert(cliente: ClienteBean): void **pre**: cliente!=null; **post:** self.retrieveAll() -> includes(cliente); **context** ClienteModel::update(cliente: ClienteBean): void **pre:** (cliente!=null); **post:** self.retrieveAll() -> includes(cliente); **context** ClienteMode::retrieveByKey(email_cliente: String): ClienteBean **pre:** email_cliente !=null; **post:** result.String== email_cliente;

Nome classe	ESoggettoModel
Descrizione	Gestisce le informazioni persistenti nel database attraverso la definizione di metodi di interrogazione, inserimento, aggiornamento e cancellazione.
Invarianti	-
Metodi	**context** ESoggettoModel::insert(è_soggetto: ESoggettoBean): void **pre**: è_soggetto!=null; **post:** self.retrieveAll() -> includes(è_soggetto); **context** ESoggettoModel delete(codice:int): void **pre:** (codice!=null); **post:** self.retrieveByKey(codice) == null; **context** ESoggettoModel::retrieveByKey(cod_scontrino:int, cod_prodotto: int): ESoggettoBean **pre:** cod_scontrino&&cod_prodotto !=null; **post:** result.int== cod_scontrino && cod_prodotto;

Nome classe	PrenotazioneModel
Descrizione	Gestisce le informazioni persistenti nel database attraverso la definizione di metodi di interrogazione, inserimento,cancellazione.
Invarianti	-
Metodi	**context** PrenotazioneModel::insert(prenotazione: PrenotazioneBean): void **pre: prenotazione**!=null; **post:** self.retrieveAll() -> includes(prenotazione); **context** PrenotazioneModel::retrieveByKey(data:String ,email:String): PrenotazioneBean **pre: data && email** !=null; **post:** result.String,String == data && email; **context** PrenotazioneModel::retrieveAll(): List<PrenotazioneBean>; **context** PrenotazioneModel::delete(email:String,date:String): void **pre: email && date** !=null; **post:** self.retrieveByKey(email,date) == null;

Nome classe	ProdottoModel
Descrizione	Gestisce le informazioni persistenti nel database attraverso la definizione di metodi di interrogazione, inserimento, aggiornamento e cancellazione.
Invarianti	-
Metodi	**context** ProdottoModel::insert(prodotto: ProdottoBean): void **pre:** prodotto !=null; **post:** self.retrieveAll() -> includes(prodotto); **context** ProdottoModel::update(prodotto : ProdottoBean): void **pre:** (prodotto!=null); **post:** self.retrieveAll() -> includes(prodotto); **context** ProdottoModel::retrieveByKey(codice:int): ProdottoBean **pre: codice** !=null; **post:** result.int == codice; **context** ProdottoModel::retrieveAll(): List<ProdottoBean>; **context** ProdottoModel::delete(codice:int): void **pre: codice**!=null; **post:** self.retrieveByKey(codice) == null;

Nome classe	UtenteModel
Descrizione	Gestisce le informazioni persistenti nel database attraverso la definizione di metodi di interrogazione, inserimento.
Invarianti	-
Metodi	**context** UtenteModel::insert(utente: UtenteBean): void **pre**: utente!=null; **post:** self.retrieveAll() -> includes(utente); **context** UtenteModel::retrieveByKey(email: String): UtenteBean **pre:** email !=null; **post:** result.String==email;

4. GLOSSARIO

DB:	Database
DBMS:	Database Management System
JSP:	JavaServer Pages
MVC:	Model View Controller
AIM:	Nome del sistema che verrà sviluppato
RAD:	Requirements Analysis Document
SDD:	System Design Document;
Gestore Prodotti:	Attore del sistema che si occupa della gestione dei prodotti;
Gestore Prenotazioni:	Attore del sistema che si occupa della gestione delle prenotazioni;
Login:	Attività di accesso all’account;
Logout:	Attività di uscita dell’account connesso;
Utente Registrato:	Il termina identifica un utente che ha effettuato la registrazione sul sistema
Database:	Archivio di dati che permette di razionalizzare la gestione e l’aggiornamento delle informazioni
DBMS:	È un sistema software progettato per consentire la creazione, la manipolazione e l’interrogazione efficiente di database