UNIVERSITÀ DEGLI STUDI
DI SALERNO

Ingegneria del Software

A.A. 2018/2019

innovatec
Elettronica

Object Design Document

Versione 1.0

Coordinatore del progetto:

Professore:	Andrea De Lucia

Partecipanti:

Nome	**Matricola**
Giuseppe Emanuele Pezzillo	0512103476

Revision History

Data	Versione	Descrizione	Autore
	1.0		G.E. Pezzillo

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 sistema, definendo gli 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 struttura 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, verranno implementati 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, 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.

Esempio: idProdotto, nomeProdotto
● 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.

Esempio: AGGIUNGI_PRODOTTO
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.
Esempio: GestioneBolle.jsp
Ogni file sorgente .jsp 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:
● Dichiarazione della classe pubblica
● Dichiarazioni di costanti
● Dichiarazioni di variabili di classe
● Dichiarazioni di variabili d'istanza
● Costruttore
● Commento e dichiarazione metodi e variabili

ESEMPIO:

```
package classiProgetto;

/**

*

* @author Giuseppe Emanuele Pezzillo

*

*/



public class Bolla {

private int id;

private String tipo;
```

```
private String marca;

private String modello;

private int costo;

private String difetto;

}

/**

* Questo è il costruttore di una bolla

*

* @param id è l'id

* @param tipo è il tipo di oggetto da riparare

* @param marca è la marca dell'oggetto da riparare

* @param modello è il modello dell'oggetto da riparare

* @param costo è il costo della riparazione

* @param difetto è il difetto dell'oggetto da riparare

*/

public Bolla(  ) {

id=0;

tipo=" ";

marca=" ";

modello=" ";

costo=0;

difetto=" ";

}



/**

*

* @return id ritorna l'id
```

```
*/
public int getId() {
return id;
}
/**
*
* @param id è l'id
*/
public void setId(int id) {
this.id = id;
}
    }
```

1.3 Definizioni, acronimi e abbreviazioni

Acronimi:
- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document

Abbreviazioni:
- DB: DataBase

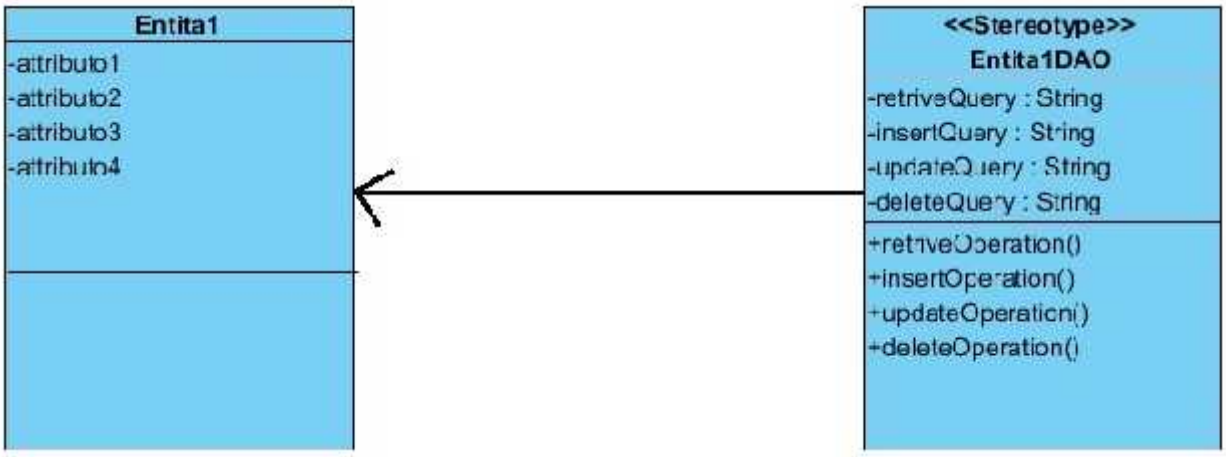
1.4 Riferimenti

- B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall, 3rd edition, 2009
- Documento RAD_Innovatec Elettronica.pdf del progetto Innovatec Elettronica
- Documento SDD_Innovatec Elettronica.pdf del progetto Innovatec Elettronica
- Documento DatiPersistenti_Innovatec Elettronica.pdf del progetto Innovatec Elettronica

2. Design Pattern

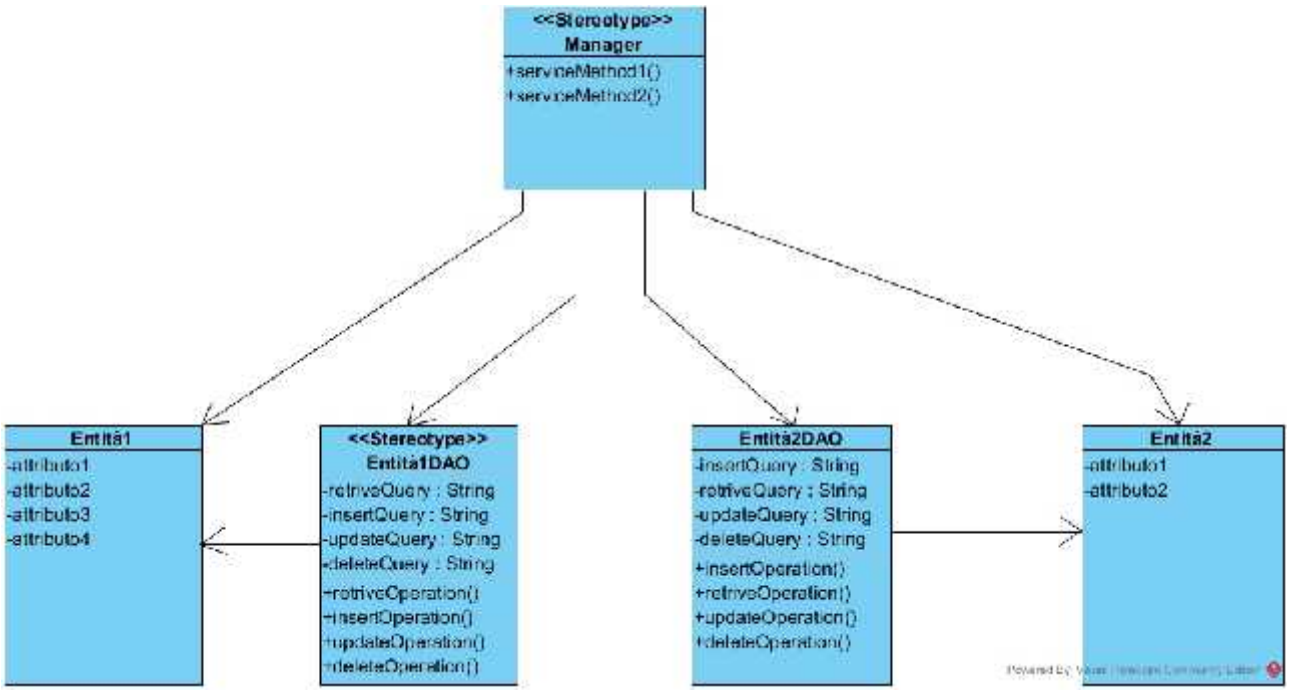
4.1 Dao Pattern

Per l'accesso al database è stato impiegato il pattern DAO (Data Access Object). Il pattern è usato per separare le API, di basso livello, di accesso ai dati, dalla logica di business di alto livello. Per ogni tabella presente nel DB esisterà una sua corrispondente classe DAO che possiederà metodi per effettuare le operazioni CRUD. Ogni classe DAO utilizzerà esclusivamente la classe Entity corrispondente alla tabella su cui effettua le operazioni. Di seguito è presentato un modello di utilizzo del pattern:



4.2 Façade Pattern

Per la realizzazione dei servizi dei sottosistemi è stato usato il pattern Façade. Il pattern si basa sull'utilizzo di un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi. Nel caso di Innovatec Elettronica, ogni sottosistema, identificato nell'SDD, possiede una classe chiamata manager. Questa implementa dei metodi che corrispondono ai servizi offerti dal sottosistema. tali metodi, nella loro implementazione, utilizzeranno le classi entity e le classi DAO per eseguire il servizio da essi offerto. In output presenteranno un oggetto che servirà alla servlet per la presentazione del risultato dell'operazione. In questo modo si svincola completamente la logica di business, implementata dai manager, e la logica di controllo, implementata dalla servlet. Viene qui presentato un modello di utilizzo del pattern:



3. Packages

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

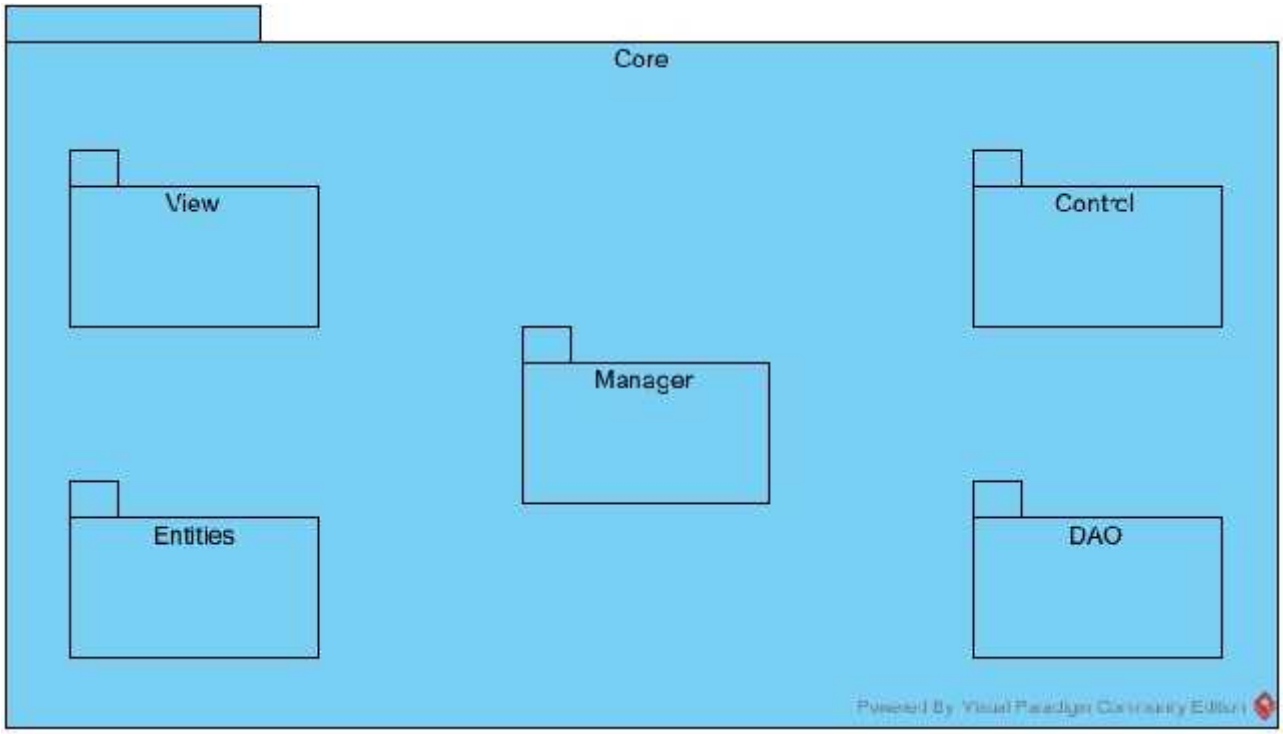
- Presentation layer
- Application layer
- Storage layer

Presentation layer	Include tutte le interfacce grafiche e in generale i boundary objects, come le form con cui interagisce l'utente. L'interfaccia verso l'utente è rappresentata da un Web server e da eventuali contenuti statici (es. pagine HTML).

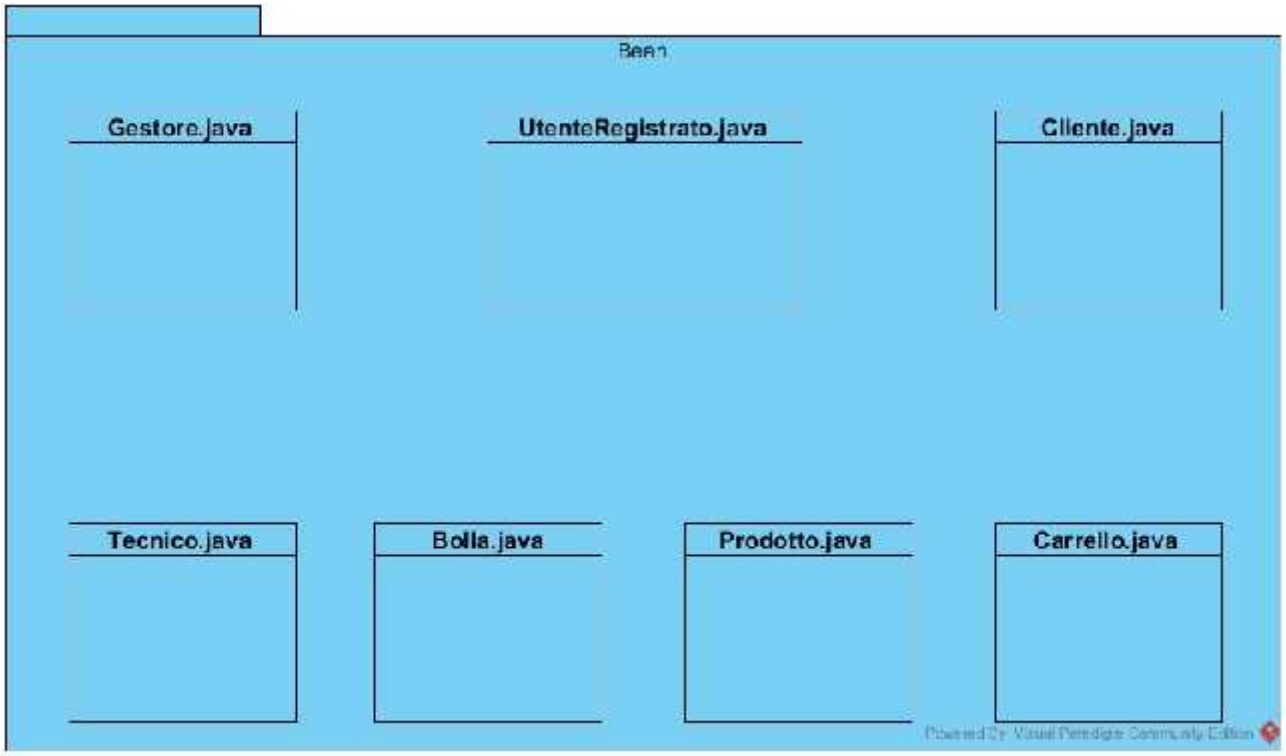
Application layer	Include tutti gli oggetti relativi al controllo e all'elaborazione dei dati. Questo avviene interrogando il database tramite lo storage layer per generare contenuti dinamici e accedere a dati persistenti. Si occupa di varie gestioni quali: ● Gestione Autenticazione ● Gestione Account ● Gestione Bolle ● Gestione Prodotti ● Gestione Vendita

Storage layer	Ha il compito di effettuare memorizzazione, il recupero e l'interrogazione degli oggetti persistenti. I dati, i quali possono essere acceduti dall'application layer, sono depositati in maniera persistente su un database tramite DBMS.

3.1 Packages Core

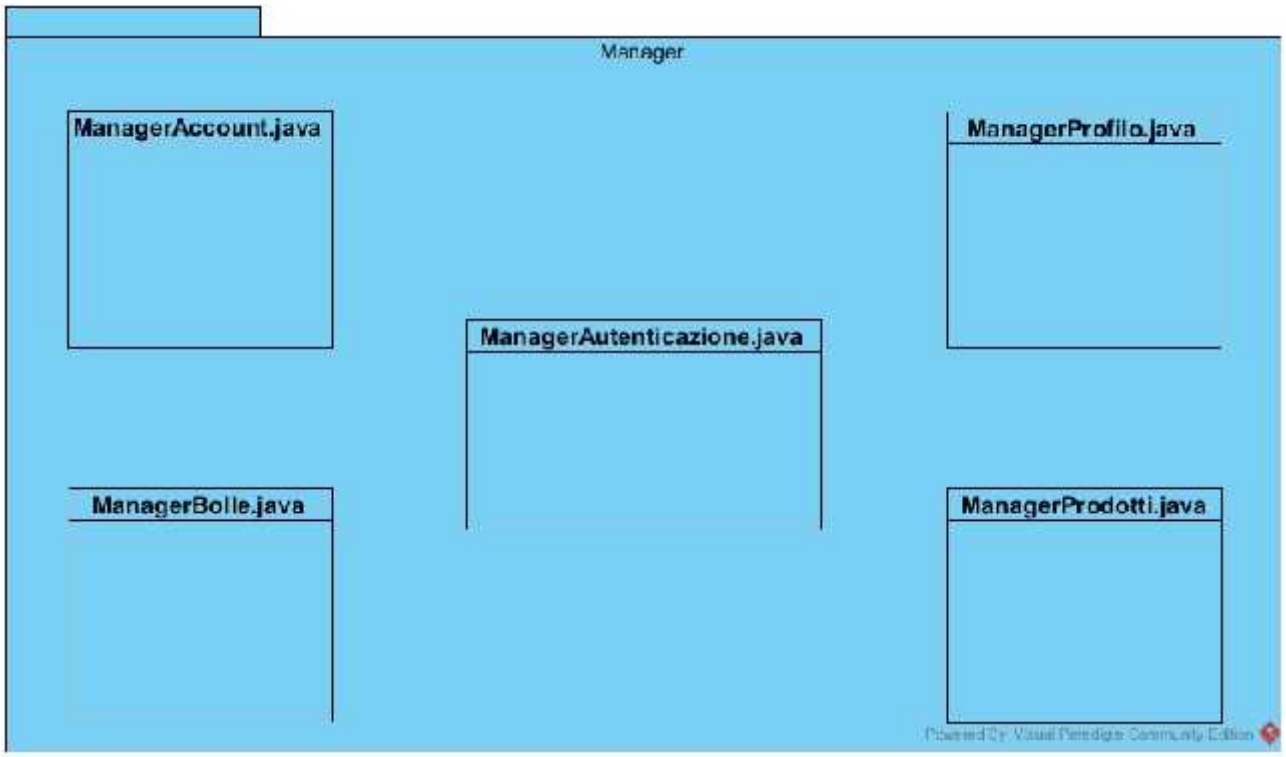


3.1.1 Packages Entities



Classe:	Descrizione:
Gestore.java	Descrive l'amministratore del sistema.
UtenteRegistrato.java	Descrive un'utente registrato che fa uso del sistema.
Cliente.java	Descrive un cliente che fa uso del sistema.
Tecnico.java	Descrive un tecnico che fa uso del sistema.
Bolla.java	Descrive una bolla del sistema.
Prodotto.java	Descrive un prodotto del sistema.
Carrelo.java	Descrive il carrello per la vendita dei prodotti nel sistema.

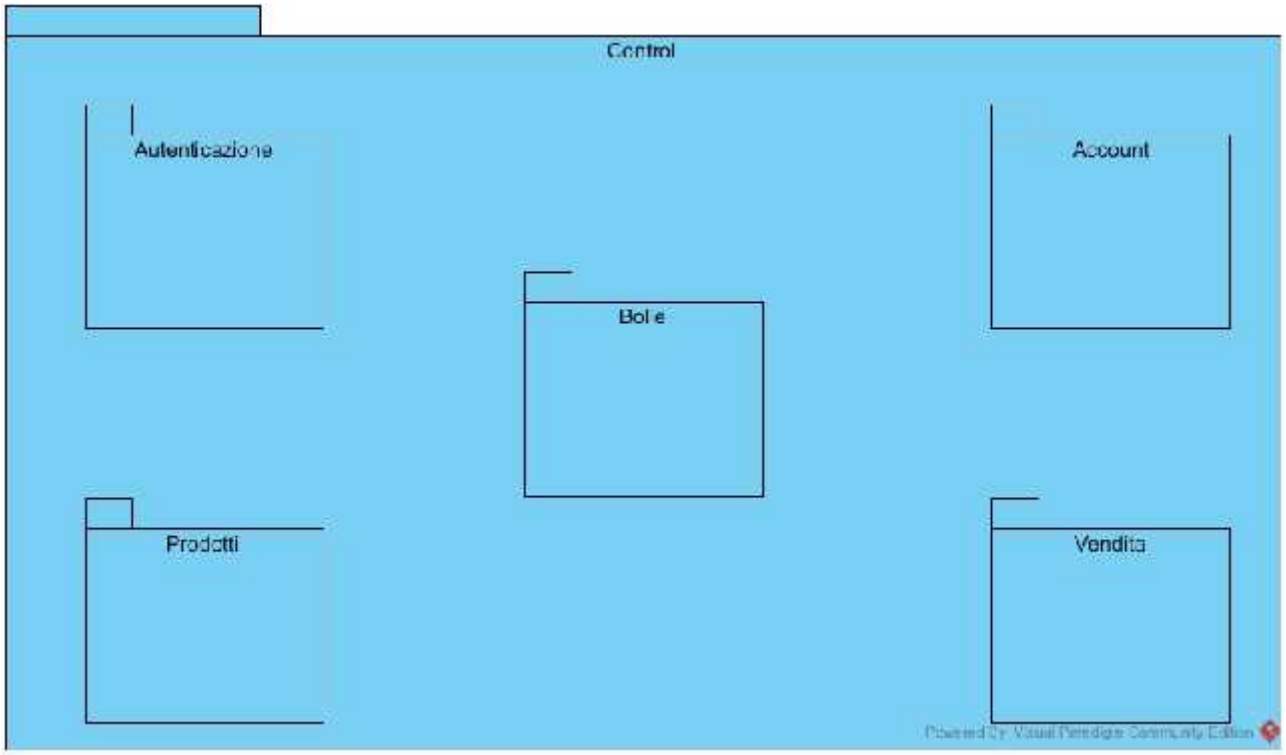
3.1.2 Packages Manager



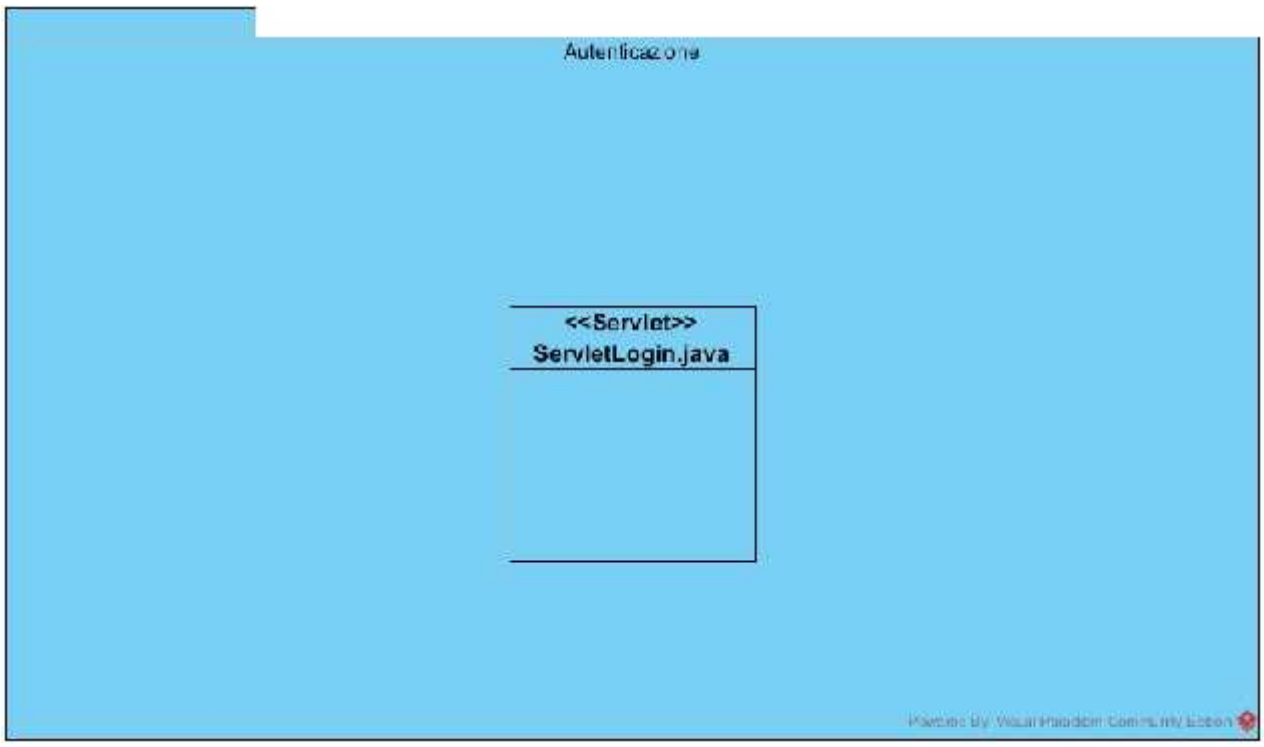
Classe:	Descrizione:
ManagerAccount.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce gli account.

ManagerAutenticazione.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'autenticazione.
MnagerProfilo.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce i profili degli utenti.
ManagerBolle.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce le bolle.
ManagerProdotti.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce i prodotti.

3.1.3 Packages Control

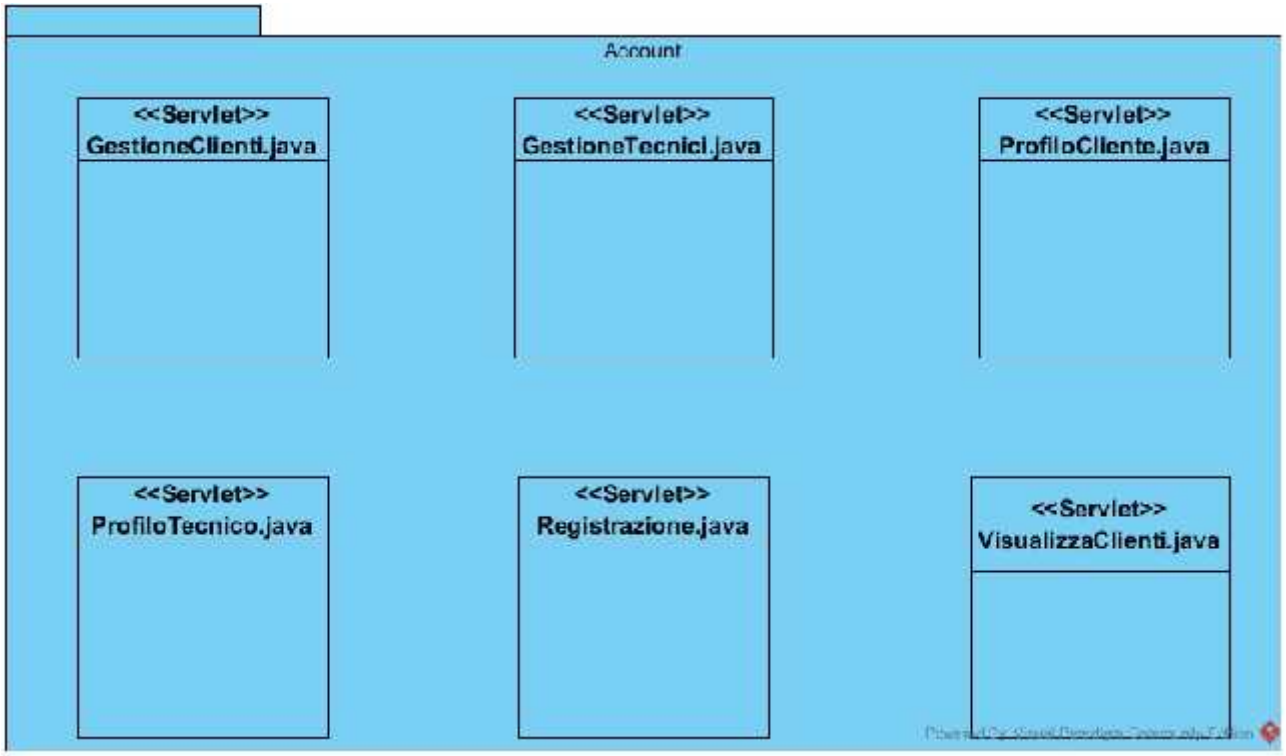


3.1.3.0 Autenticazione



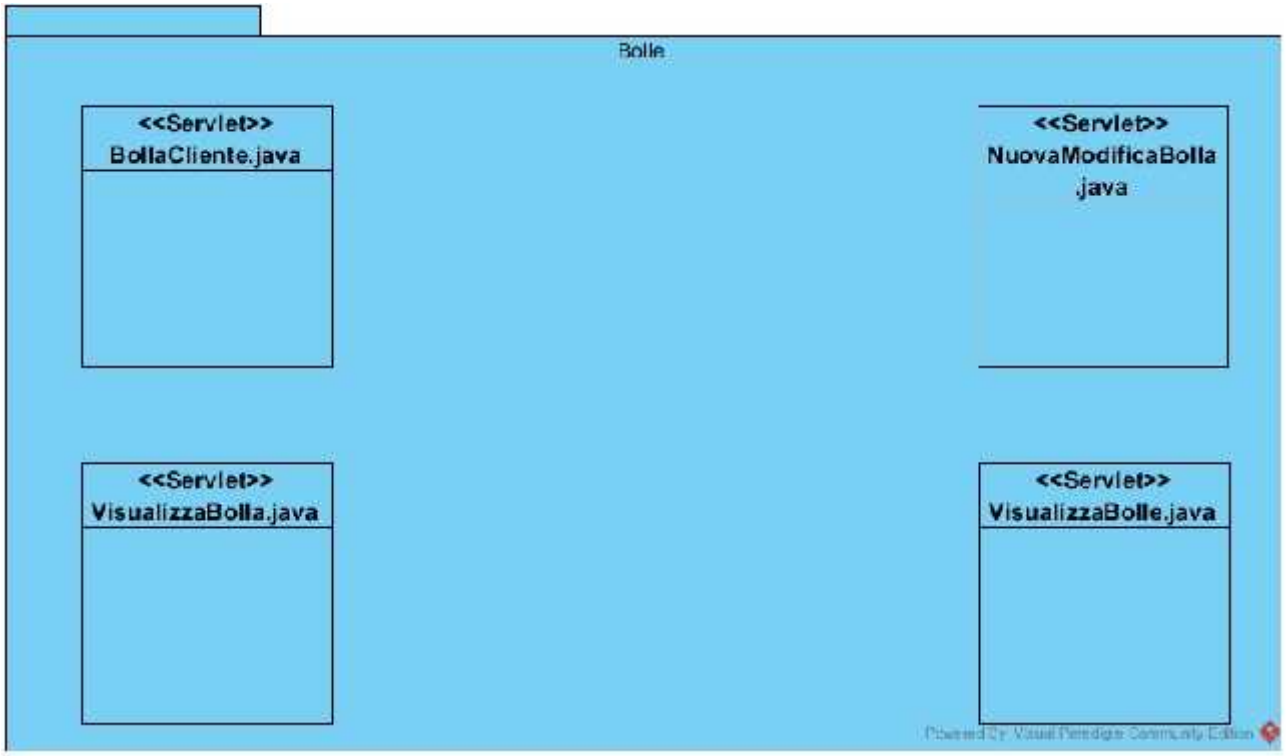
Classe:	**Descrizione:**
ServletLogin.java	Gestisce il login e il logout degli utenti.

3.1.3.1 Account



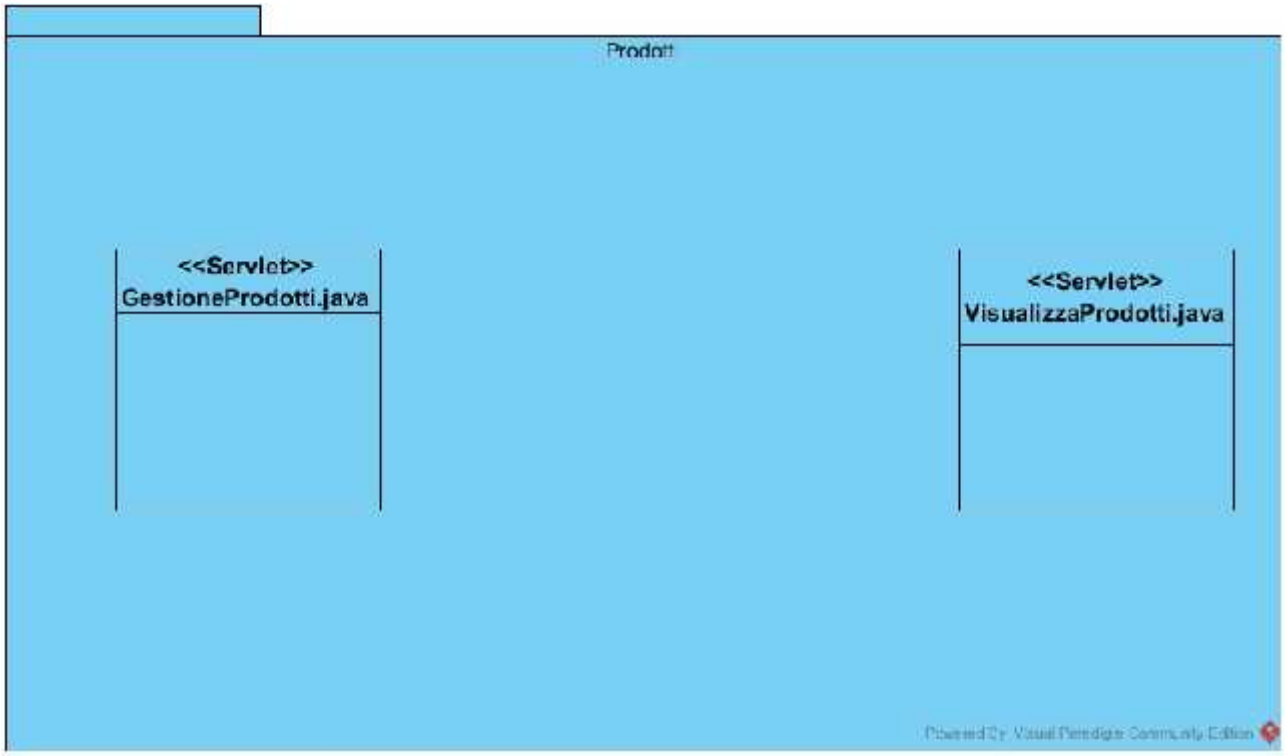
Classe:	**Descrizione:**
GestioneClienti.java	Permette la gestione dei profili dei clienti presenti nel sistema.
GestioneTecnici.java	Permette la gestione dei profili dei tecnici presenti nel sistema.
ProfiloCliente.java	Permetta la gestione del profilo personale del cliente.
ProfiloTecnico.java	Permette la gestione del profilo personale del tecnico.
Registrazione.java	Permette la registrazione dei clienti al sistema.
VisualizzaClienti.java	Permette la visualizzazione di tutti i clienti presenti nel sistema.

3.1.3.2 Bolle



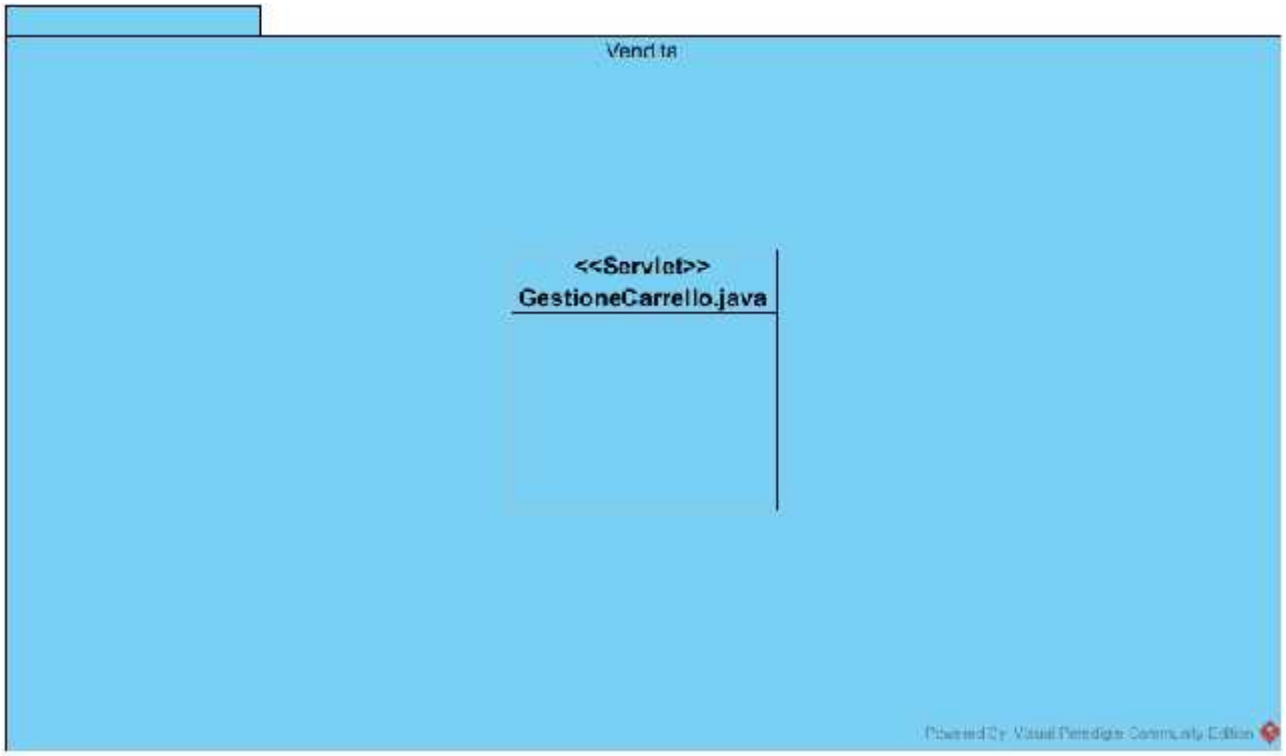
Classe:	**Descrizione:**
BollaCliente.java	Permette la visualizzazione della bolla associata al cliente.
NuovaModificaBolla.java	Permette di creare una nuova bolla modificarla ed eliminarla dal sistema.
VisualizzaBolla.java	Permette la visualizzazione della bolla.
VisualizzaBolle.java	Permette la visualizzazione delle bolle presenti all'interno del sistema.

3.1.3.3 Prodotti



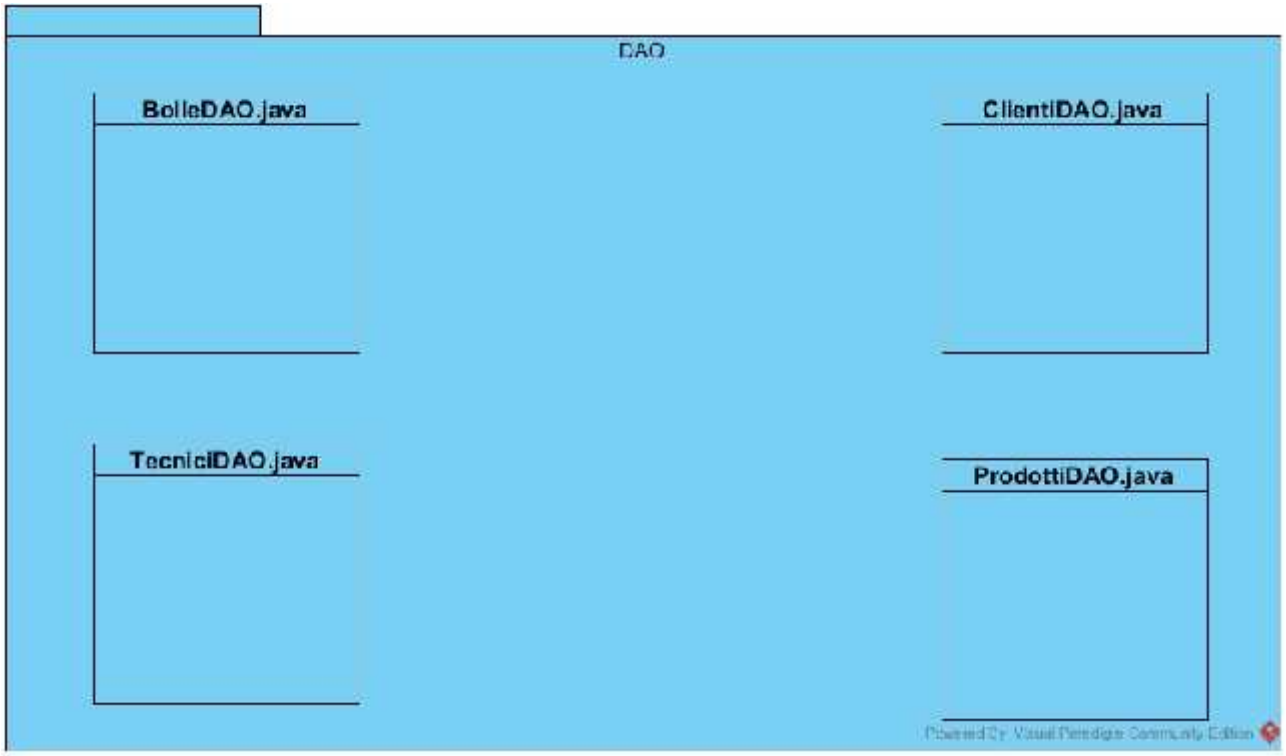
Classe:	**Descrizione:**
GestioneProdotti.java	Permette la gestione dei prodotti presenti nel sistema.
VisualizzaProdotti.java	Permette la visualizzazione dei prodotti presenti nel sistema.

3.1.3.4 Vendita



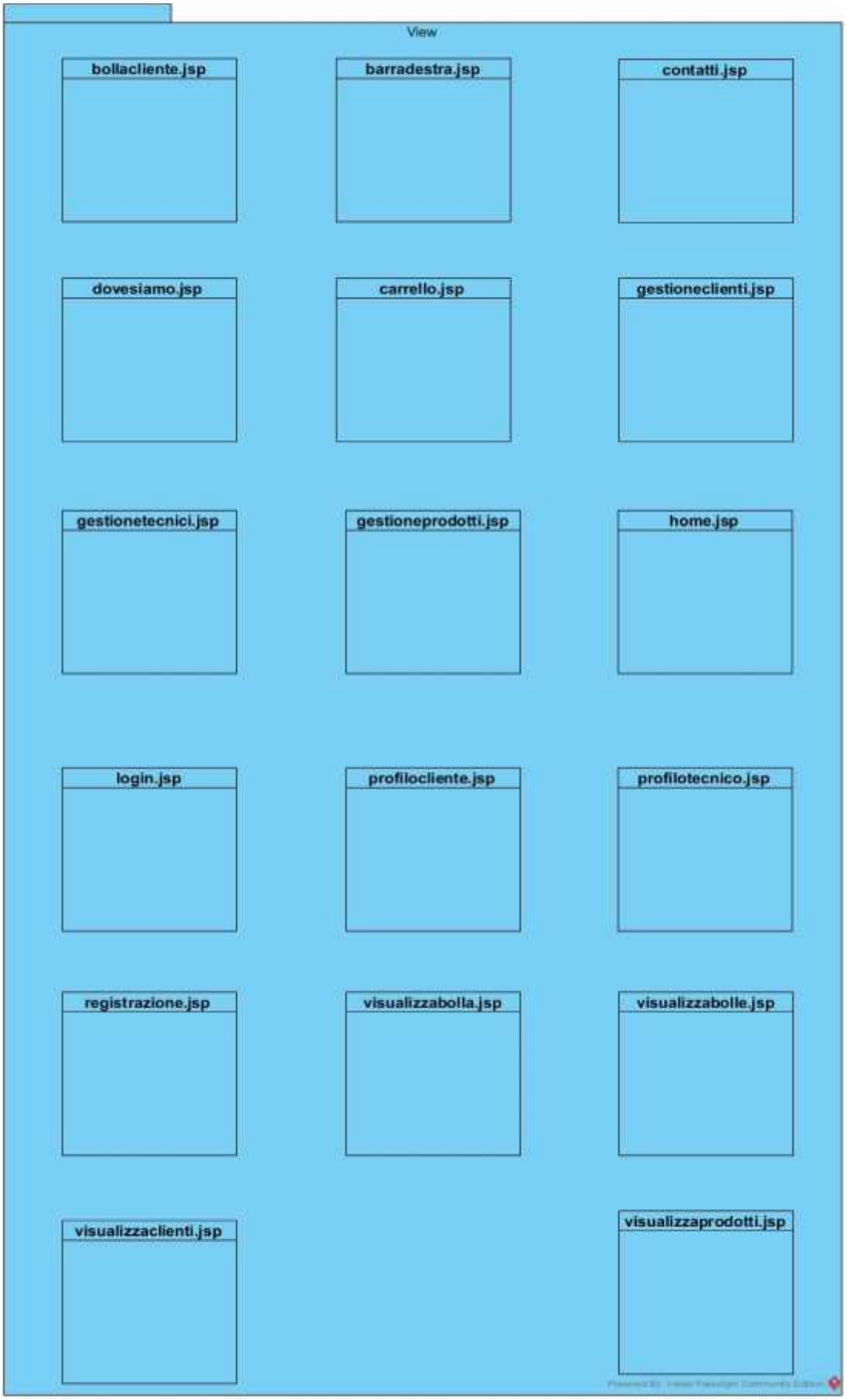
Classe:	**Descrizione:**
GestioneCarrello.java	Permette la gestione del carrello del sistema.

3.1.4 Packages DAO



Classe:	**Descrizione:**
BolleDAO.java	Permette di interagire con la tabella Bolle all'interno del database.
ClientiDAO.java	Permette di interagire con la tabella Clienti all'interno del database.
TecniciDAO.java	Permette di interagire con la tabella Tecnici all'interno del database.
ProdottiDAO.java	Permette di interagire con la tabella Prodotti all'interno del database.

3.1.5 Packages View



Classe:	**Descrizione:**
bollacliente.jsp	Visualizza la pagina che mostra i dati della bolla del cliente.
barradestra.jsp	Visualizza i prodotti messi in vendita dall'azienda nella parte destra di tutte le pagine del sistema.
contatti.jsp	Visualizza la pagina che mostra i contatti del sistema.
dovesiamo.jsp	Visualizza la pagina che mostra la localizzazione dell'azienda.
carrello.jsp	Visualizza la pagina che mostra i prodotti aggiunti al carrello.
gestioneclienti.jsp	Visualizza la pagina che permette di gestire i clienti presenti nel sistema.
gestionetecnici.jsp	Visualizza la pagina che permette di gestire i tecnici presenti nel sistema.
gestioneprodotti.jsp	Visualizza la pagina che permette di gestire i prodotti.
home.jsp	Visualizza la Home Page del sistema.
login.jsp	Visualizza la pagina che permette di effettuare il login al sistema.
profilocliente.jsp	Visualizza la pagina che permette di gestire il profilo personale del cliente.
profilotecnico.jsp	Visualizza la pagina che permette di gestire il profilo personale del tecnico.
registrazione.jsp	Visualizza la pagina che permette la registrazione del cliente al sistema.
visualizzabolla.jsp	Visualizza la pagina che permette la visualizzazione della bolla associata al cliente.
visualizzabolle.jsp	Visualizza la pagina che permette di visualizzare le bolle presenti nel sistema.
visualizzaclienti.jsp	Visualizza la pagina che permette la visualizzazione dei clienti presenti nel sistema.
visualizzaprodotti.jsp	Visualizza la pagina che permette la visualizzazione dei prodotti presenti nel sistema.

4. Class Interfaces

4.1 Package Manager

4.1.0 Manager Account

Nome Classe	Manager Account
Descrizione	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'account
Metodi	+ aggiungiCliente (Cliente cliente, Tecnico tecnico) : Integer + modificaCliente (Cliente cliente) : Integer + rimuoviCliente (String rem_id) : Integer + aggiungiTecnico (Tecnico tecnico) : Integer + modificaTecnico (Tecnico tecnico) : Integer + rimuoviTecnico (String rem_id) : Integer + registrazioneCliente (HttpServletRequest request) : Integer

Nome Metodo	aggiungiCliente(Cliente cliente, Tecnico tecnico) : Integer
Descrizione Metodo	Questo metodo permette di aggiungere un nuovo cliente al sistema e associarlo a quel tecnico specificato
Pre-Condizioni	cliente != null , tecnico != null
Post-Condizioni	/

Nome Metodo	modificaCliente(Cliente cliente) : Integer
Descrizione Metodo	Questo metodo permette la modifica dei dati del cliente
Pre-Condizioni	cliente != null
Post-Condizioni	/

Nome Metodo	rimuoviCliente(String rem_id) : Integer
Descrizione Metodo	Questo metodo permette la rimozione del profilo del cliente dal sistema
Pre-Condizioni	rem_id != “ ”
Post-Condizioni	/

Nome Metodo	aggiungiTecnico(Tecnico tecnico) : Integer
Descrizione Metodo	Questo metodo permette l’aggiunta di un nuovo tecnico al sistema
Pre-Condizioni	tecnico != null
Post-Condizioni	/

Nome Metodo	modificaTecnico(Tecnico tecnico) : Integer
Descrizione Metodo	Questo metodo permette la modifica del profilo del tecnico
Pre-Condizioni	tecnico != null
Post-Condizioni	/

Nome Metodo	rimuoviTecnico(String rem_id) : Integer
Descrizione Metodo	Questo metodo permette la rimozione del tecnico dal sistema
Pre-Condizioni	rem_id != “ ”
Post-Condizioni	/

Nome Metodo	registrazioneCliente(HttpServletRequest request) : Integer
Descrizione Metodo	Questo metodo permette la registrazione del cliente al sistema

Pre-Condizioni	request != null
Post-Condizioni	/

4.1.1 Manager Autenticazione

Nome Classe	Manager Autenticazione
Descrizione	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce l'autenticazione
Metodi	+ loginCheck (String codfis, String pwd) : UtenteRegistrato + logout (HttpSession session) : void

Nome Metodo	loginCheck (String codfis, String pwd) : UtenteRegistrato
Descrizione Metodo	Questo metodo permette di effettuare l'accesso al sistema
Pre-Condizioni	codfis != " " , pwd != " "
Post-Condizioni	/

Nome Metodo	logout (HttpSession session) : void
Descrizione Metodo	Questo metodo permette di effettuare il logout dal sistema
Pre-Condizioni	session != null
Post-Condizioni	/

4.1.2 Manager Profilo

Nome Classe	Manager Profilo
Descrizione	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce i profili degli utenti del sistema
Metodi	+ modificaProfiloCliente(Cliente cliente) : Integer + modificaProfiloTecnico(Tecnico tecnico) : Integer

Nome Metodo	modificaProfiloCliente(Cliente cliente) : Integer
Descrizione Metodo	Questo metodo permette la modifica del profilo personale del cliente
Pre-Condizioni	cliente != null
Post-Condizioni	/

Nome Metodo	modificaProfiloTecnico(Tecnico tecnico) : Integer
Descrizione Metodo	Questo metodo permette la modifica del profilo personale del tecnico
Pre-Condizioni	tecnico != null
Post-Condizioni	/

4.1.3 Manager Bolle

Nome Classe	Manager Bolle
Descrizione	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce le bolle
Metodi	+ rimuoviBolla(String id) : Integer + nuovaModificaBolla(Bolla bolla, Tecnico tecnico) : Integer

Nome Metodo	rimuoviBolla(String id) : Integer
Descrizione Metodo	Questo metodo permette la rimozione di una bolla
Pre-Condizioni	id != “ ”
Post-Condizioni	/

Nome Metodo	nuovaModificaBolla(Bolla bolla, Tecnico tecnico) : Integer
Descrizione Metodo	Questo metodo permette di modificare e inserire una nuova bolla nel sistema
Pre-Condizioni	bolla != null , tecnico != null
Post-Condizioni	/

4.1.4 Manager Prodotti

Nome Classe	Manager Prodotti
Descrizione	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce i prodotti
Metodi	+ nuovoProdotto(Prodotto prodotto) : Integer + nuovoModificaProdotto(HttpServletRequest request) : Integer + rimuoviProdotto(String rim) : Integer

Nome Metodo	nuovoProdotto(Prodotto prodotto) : Integer
Descrizione Metodo	Questo metodo permette di aggiungere un nuovo prodotto al sistema
Pre-Condizioni	prodotto != null
Post-Condizioni	/

Nome Metodo	nuovoModificaProdotto(HttpServletRequest request) : Integer
Descrizione Metodo	Questo metodo permette di modificare i dati dei prodotti del sistema
Pre-Condizioni	request != null
Post-Condizioni	/

Nome Metodo	rimuoviProdotto(String rim) : Integer
Descrizione Metodo	Questo metodo permette di rimuovere un prodotto dal sistema
Pre-Condizioni	rim != “ ”
Post-Condizioni	/

5.Glossario

● **Innovatec Elettronica:** Nome del sistema che verrà sviluppato.

● **web-based:** il termine identifica un sistema basato sul web, quindi accessibile simultaneamente da più postazioni.

●**Utente Loggato:** il termine identifica un utente che ha eseguito il login correttamente.

●**Utente Registrato:** il termine identifica utente che ha effettuato la registrazione sul sistema.

● **Utente:** il termine identifica un attore del sistema che può usufruire dei servizi offerti.

● **Amministratore:** il termine identifica il creatore del sito che ha accesso al

codice sorgente.

●**Gestore:** il termine identifica il gestore dell'azienda Innovatec Elettronica che può usufruire di alcune funzionalità.

●**Tecnico:** il termine identifica i tecnici dell'azienda Innovatec Elettronica.

●**Prodotti:** il termine identifica i prodotti messi in vendita dall'azienda.

●**RAD:** Documento di Analisi dei Requisiti.

●**SDD:** Documento di System Design.

●**DBMS:** Sistema di gestione di basi di dati.

●**Database:** Insieme organizzato di dati.

●**Query:** Termine utilizzato per indicare l'interrogazione da parte di un utente di un database.