Università degli Studi di Salerno
Corso di Ingegneria del Software

RAAF-GAMING
Object Design Document
Versione 1.0

Data: 16/12/2021

Partecipanti:

Nome	Matricola
Rocco Iuliano	0512106804
Francesco Peluso	0512107194
Antonio De Lucia	0512109225
Antonio Maddaloni	0512107890

Revision History

Data	Versione	Descrizione	Autore
16/12/2021	1.0	Prima stesura del documento	Membri del team

Indice

1. Introduzione

1.1. Object Design Trade-Offs

1.1.1. Sicurezza VS Costi

A causa di tempi di sviluppo limitati e del budget ridotto, ci limiteremo a realizzare sistemi di sicurezza basati su username password, attraverso una pagina apposita di autenticazione ed in ogni operazione dove si necessita di sapere l'identità dell'utente, verificheremo se si è autenticati o meno.

1.1.2. Interfaccia VS Tempo di risposta

Il tempo di risposta tra server e interfaccia sono più che sufficienti, a soddisfare le esigenze dei vari utenti collegati al sistema

1.1.3. Persistenza VS Efficienza

Per ottenere efficienza sulla gestione della persistenza, ci affideremo ad un Database, che ci offre un DBMS per poter interagire più velocemente possibile con esso. Utilizzeremo un DataSource (Un oggetto che ci offre JNDI) per poter ottenere connessioni al database, gestione della sincronizzazione in modo veloce e semplice.

1.2. Riferimenti

- Requirements Analysis Document
- System Design Document

2. Package and Class Interfaces

2.1. Package

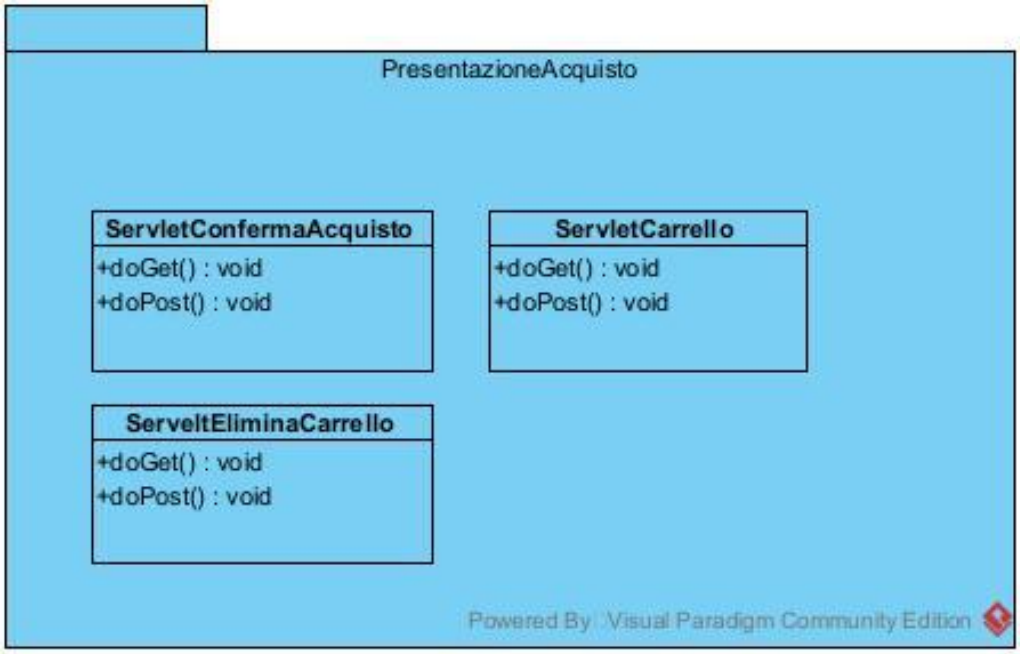
Attraverso il System Decomposition abbiamo individuato questi package per il layer view-control:

1. PresentazioneAcquisto
2. PresentazioneProdotto
3. PresentazioneProfilo
4. PresentazioneOrdini

Per il layer business logic:

1. Acquisto
2. Prodotti
3. Magazzino
4. Profilo

2.1.1. PresentazioneAcquisto

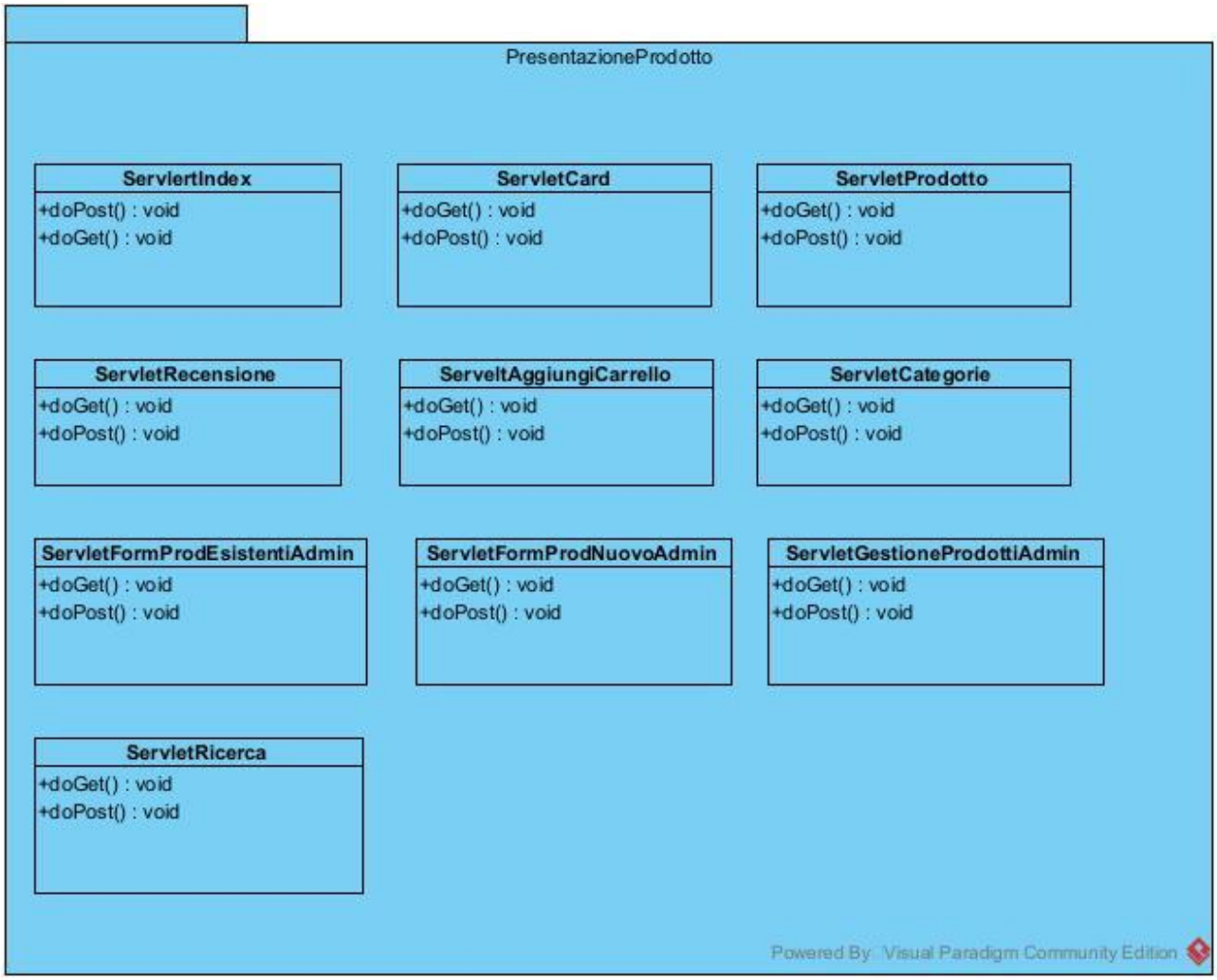


- **ServletConfermaAcquisto=** è la servlet che gestisce la realizzazione di un ordine
- **ServletCarrello=** è la servlet che consente di visualizzare il carrello di un utente
- **ServletEliminaCarrello=** è la servlet che consente di eliminare un prodotto dal carrello

In questo package troviamo anche le seguenti JSP:

- **paginaCarrello.jsp=** è la pagina che permette la visualizzazione dei prodotti aggiunti al carrello

2.1.2. PresentazioneProdotto



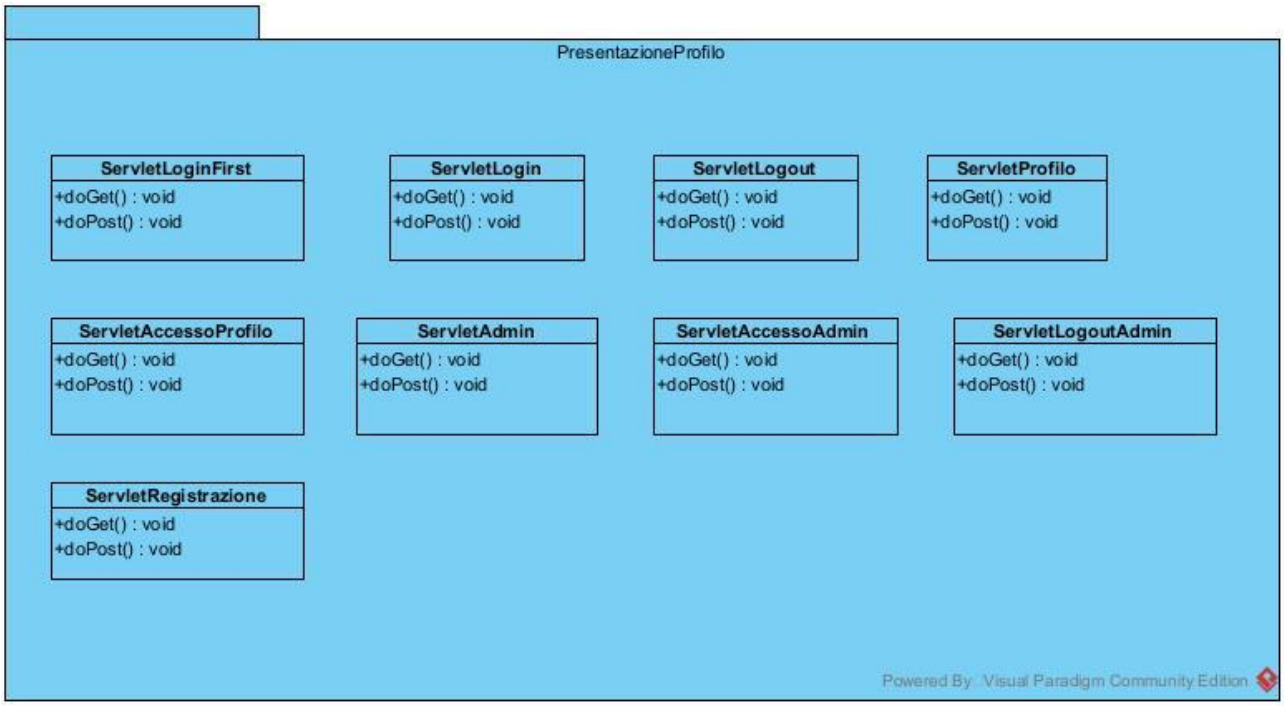
- **ServletIndex=** è la servlet che consente di visualizzare l'homepage;
- **ServletCard=** è la servlet che consente di caricare l'immagine del prodotto nella card per poterlo visualizzare;
- **ServletProdotto=** è la servlet che consente di visualizzare la pagina informativa del prodotto;

- **ServletRecensione=** è la servlet che gestisce la realizzazione di una recensione;
- **ServletAggiungiCarrello=** è la servlet che gestisce l'operazione di aggiunta di un prodotto al carrello;
- **ServletCategorie=** è la servlet che gestisce l'operazione di ricerca dei prodotti per categoria;
- **ServletFormProdEsistentiAdmin=** è la servlet che gestisce l'operazione di rifornitura di un prodotto esistente;
- **ServletFormProdNuovoAdmin=** è la servlet che gestisce l'operazione di fornitura di un nuovo prodotto;
- **ServletGestioneProdottiAdmin=** è la servlet che consente di visualizzare la pagina di fornitura dei prodotti per il gestore magazzino;
- **ServletRicerca=** è la servlet che gestisce l'operazione di ricerca per nome;

In questo package troviamo anche le seguenti JSP:

- **homepage.jsp=** è la pagina che rappresenta l'homepage del sito;
- **paginaAmministratore.jsp=** è la pagina che permette le operazioni di fornitura dei prodotti tramite l'interfaccia grafica;
- **paginaGioco.jsp=** rappresenta la pagina informativa di un prodotto ed è quindi possibile poter effettuare la recensione e l'aggiunta al carrello;
- **paginaRicerca.jsp=** rappresenta la pagina che permette di visualizzare i prodotti ottenuti da una ricerca, quindi o una ricerca per nome o una ricerca per categoria;

2.1.3. PresentazioneProfilo

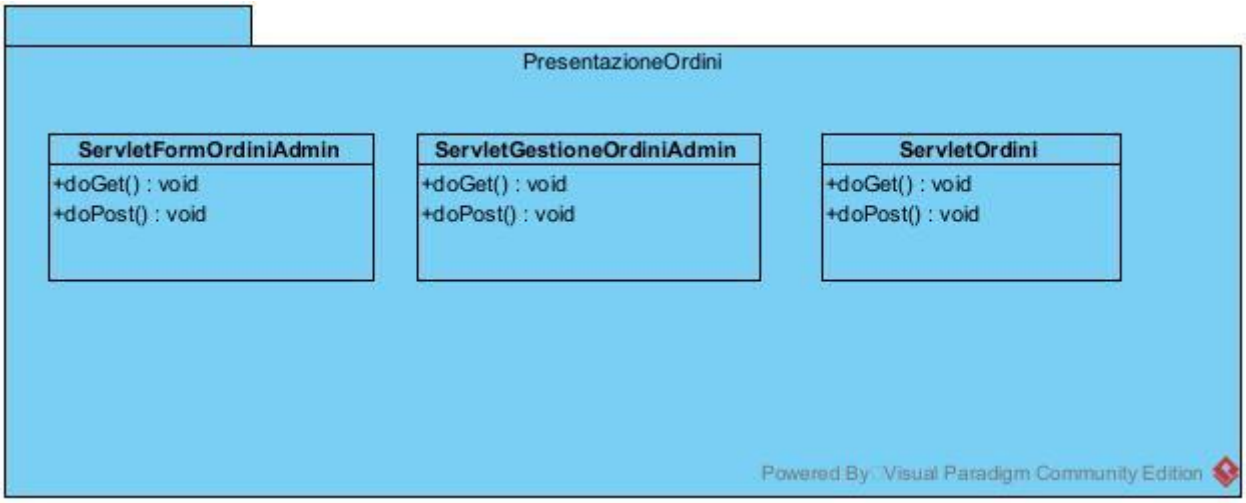


- **ServletLoginFirst=** è la servlet che consente di poter accedere alla pagina di login;
- **ServletLogin=** è la servlet che gestisce l'operazione di autenticazione;
- **ServletLogout=** è la servlet che gestisce l'operazione di logout
- **ServletProfilo=** è la servlet che gestisce l'operazione di modifica dei dati personali (sia password che i dati della carta di credito);
- **ServletAccessoProfilo=** è la servlet che consente di poter accedere alla pagina dei dati personali;
- **ServletAdmin=** è la servlet che gestisce l'operazione di autenticazione del gestore (sia gestore magazzino che gestore ordini);
- **ServletAccessoAdmin=** è la servlet che consente di poter accedre alla pagina di login per i gestori;
- **ServletLogoutAdmin=** è la servlet che gestisce l'operazione di logout dei gestori
- **ServletRegistrazione=** è la servlet che gestisce l'operazione di registrazione di un utente (non gestore) e permette di accedere alla pagina di registrazione effettuando anche determinati controlli necessari.

In questo package troviamo anche le seguenti JSP:

- **login.jsp=** rappresenta la pagina di autenticazione per gli utenti;
- **registrazione.jsp=** rappresenta la pagina di registrazione;
- **profilo.jsp=** rappresenta la pagina dei dati personali di un utente;
- **admin.jsp=** rappresenta la pagina di autenticazione per i gestori.

2.1.4. PresentazioneOrdini

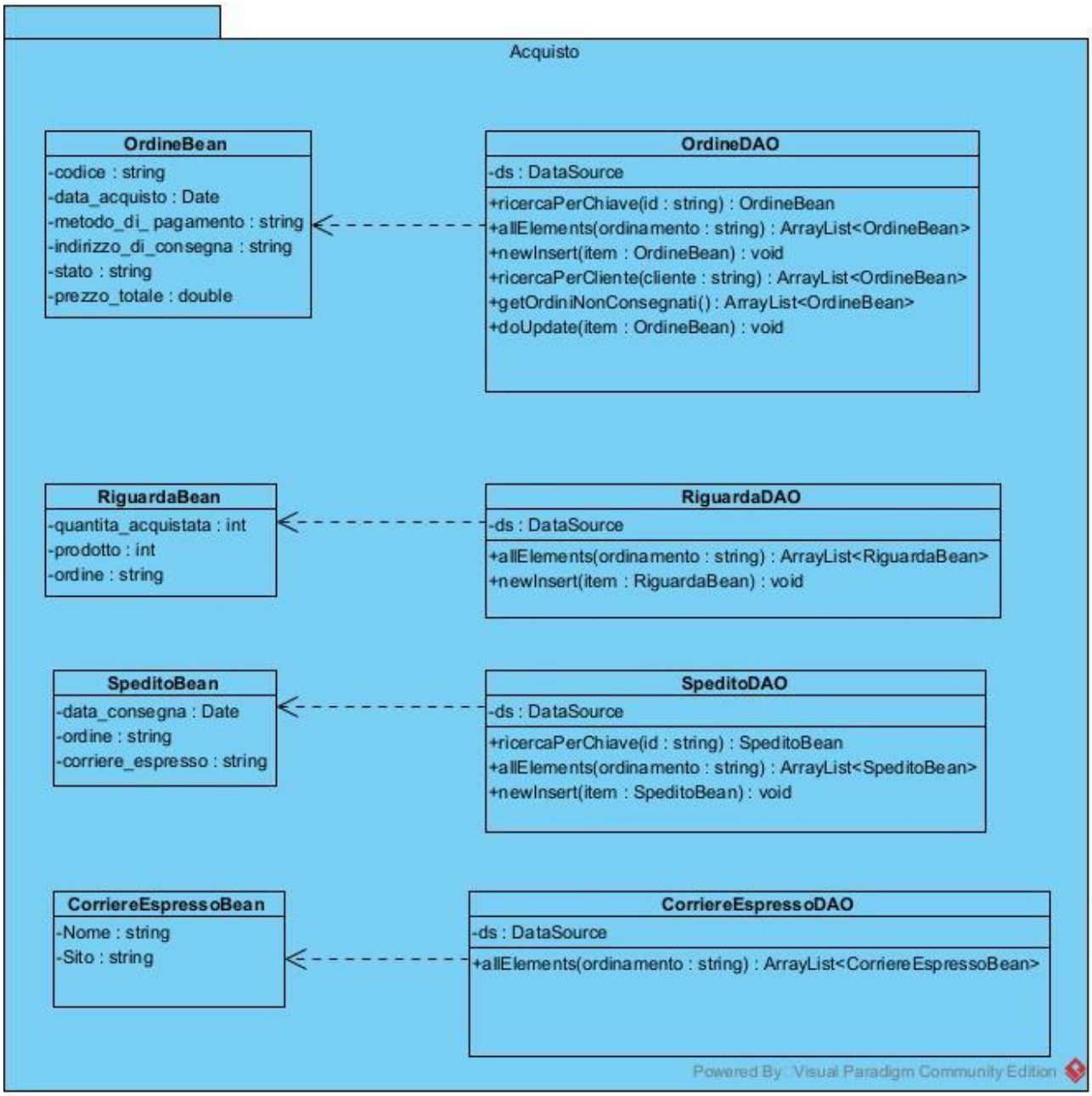


- **ServletFormOrdiniAdmin=** è la servlet che consente di visualizzare la pagina di gestione degli ordini;
- **ServletGestioneOrdiniAdmin=** è la servlet che gestisce l'operazione di spedizione di un ordine;
- **ServletOrdini=** è la servlet che permette di visualizzare la pagina degli ordini effettuati da un determinato utente.

In questo package troviamo anche le seguenti JSP:

- **ordini.jsp=** rappresenta la pagina della cronologia degli ordini effettuati da un determinato utente;
- **paginaGestioneOrdini.jsp=** rappresenta la pagina che consente di visualizzare gli ordini che deve gestire il gestore degli ordini.

2.1.5. Acquisto



- **OrdineBean=** rappresenta l'entità ordine del database
- **RiguardaBean=** rappresenta l'entità ordine del database
- **SpeditoBean=** rappresenta l'entità ordine del database
- **CorriereEspressoBean=** rappresenta l'entità ordine del database

Le classi DAO sono descritte in OCL.

Class Interface delle classi DAO:

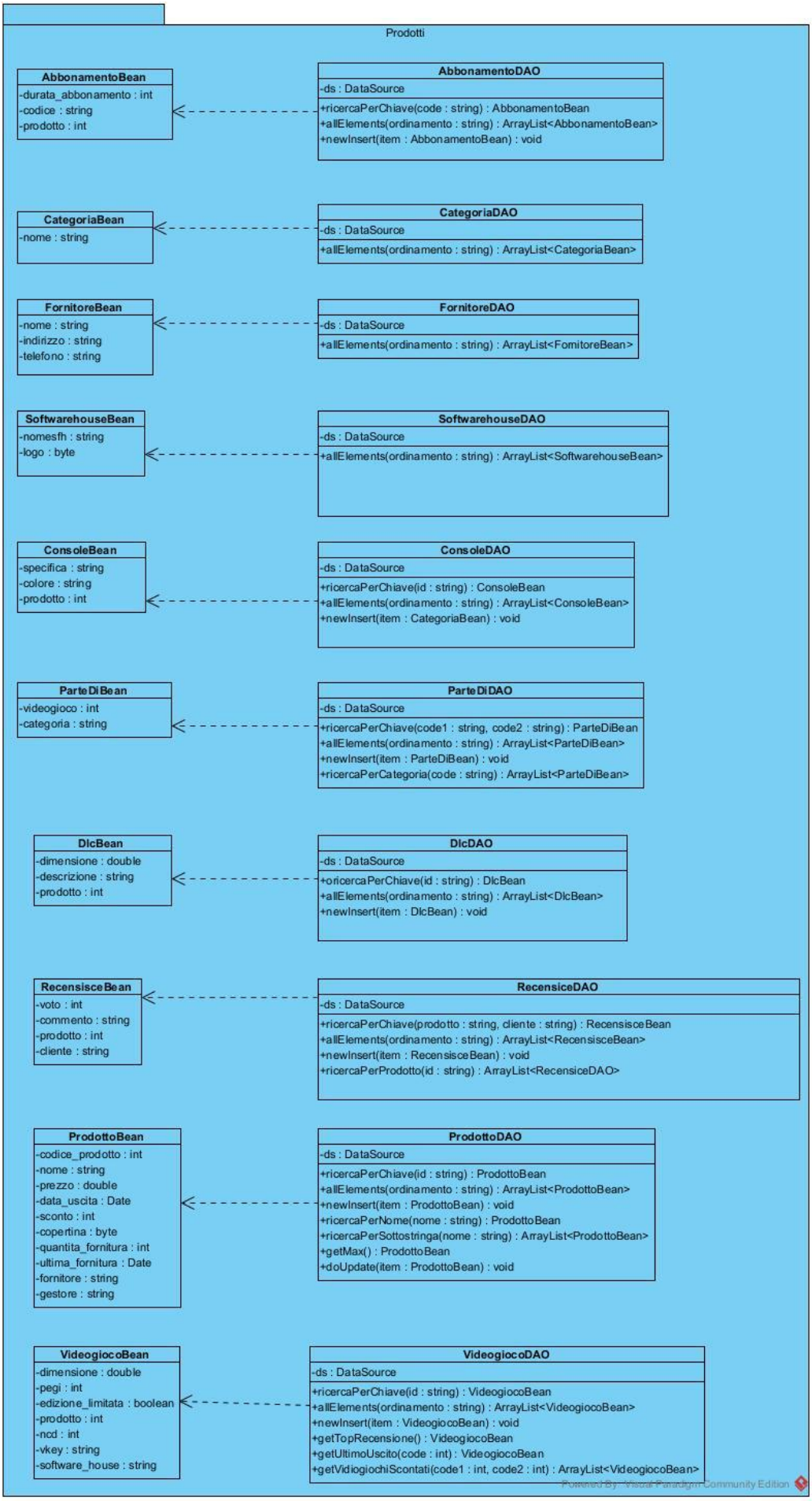
Nome Classe	**OrdineDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Ordine tramite dei metodi specifici.
Pre-condizione	**context** OrdineDAO**: :** ricercaPerChiave(String id) **pre:** id != null and id != "" **context** OrdineDAO **: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** OrdineDAO **: :** newInsert(OrdineBean item) **pre:** item != null and !dbOrdineContains(item.codice) **context** OrdineDAO **: :** ricercaPerCliente(String cliente) **pre:** cliente != null. **context** OrdineDAO **: :** getOrdiniNonConsegnati() **pre:** hasn't precondition **context** OrdineDAO : : doUpdate(OrdineBean item) **pre:** item != null and dbOrdineContains(item.codice)
Post-condizione	**context** OrdineDAO**: :** ricercaPerChiave(String id) **post:** return OrdineBean OR null se non esiste **context** OrdineDAO**: :** allElements(String ordinamento) **post:** return sequence(OrdineBean) **context** OrdineDAO**: :** newInsert(OrdineBean item) **post:** viene inserita una tupla nel database nella tabella ordine con I valori che si trovano in item **context** OrdineDAO**: :** ricercaPerCliente(String cliente) **post:** return set(OrdineBean) **context** OrdineDAO**:** :getOrdiniNonConsegnati() **post:** return set(OrdineBean) **context** OrdineDAO : : doUpdate(OrdineBean Item) **post:** viene aggiornata una tupla nel database nella tabella Ordini con i valori che si trovano in item
Invarianti	

Nome Classe	**RiguardaDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Riguarda tramite dei metodi specifici.
Pre-condizione	**context** RiguardaDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** RiguardaDAO**: :** newInsert(RiguardaBean item) **pre:** item != null and !dbRiguardaContains(item.prodotto, item.ordine)
Post-condizione	**context** RiguardaDAO**: :** allElements(String ordinamento) **post:** return sequence(RiguardaBean) **context** RiguardaDAO**: :** newInsert(RiguardaBean item) **post:** viene inserita una tupla nel database nella tabella Riguarda con i valori che si trovano in item
Invarianti	

Nome Classe	**CorriereEspressoDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Corriere Espresso tramite dei metodi specifici.
Pre-condizione	**context** CorriereEspressoDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento
Post-condizione	**context** CorriereEspressoDAO**: :** allElements(String ordinamento) **post:** return sequence(CorriereEspressoBean)
Invarianti	

Nome Classe	**SpeditoDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Spedito tramite dei metodi specifici.
Pre-condizione	**context** SpeditoDAO: **:** ricercaPerChiave(String id) **pre:** id != null and id != "" **context** SpeditoDAO: **:** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** SpeditoDAO: **:** newInsert(SpeditoBean item) **pre:** item != null and !dbRiguardaContains(item.prodotto, item.ordine)
Post-condizione	**context** SpeditoDAO: **:** ricercaPerChiave(String id) **post:** return SpeditoBean OR null se non esiste **context** SpeditoDAO: **:** allElements(String ordinamento) **post:** return sequence(SpeditoBean) **context** SpeditoDAO: **:** newInsert(SpeditoBean item) **post:** viene inserita una tupla nel database nella tabella Spedito con i valori che si trovano in item
Invarianti	

2.1.6. Prodotti



- **AbbonamentoBean=** rappresenta l'entità abbonamento del database
- **CategoriaBean=** rappresenta l'entità categoria del database
- **FornitoreBean=** rappresenta l'entità fornitore del database
- **SoftwarehouseBean=** rappresenta l'entità software house del database
- **ConsoleBean=** rappresenta l'entità console del database
- **ParteDiBean=** rappresenta l'entità partedi del database
- **DlcBean=** rappresenta l'entità dlc del database
- **RecensisceBean=** rappresenta l'entità recensisce del database
- **ProdottoBean=** rappresenta l'entità prodotto del database
- **VideogiocoBean=** rappresenta l'entità videogioco del database

Le classi DAO sono descritte in OCL.

Class Interface delle classi DAO:

Nome Classe	**CategoriaDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Categoria tramite dei metodi specifici.
Pre-condizione	**context** CategoriaDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento
Post-condizione	**context** CategoriaDAO**: :** allElements(String ordinamento) **post:** return sequence(CategoriaBean)
Invarianti	

Nome Classe	**FornitoreDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Fornitore tramite dei metodi specifici.
Pre-condizione	**context** FornitoreDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento
Post-condizione	**context** FornitoreDAO**: :** allElements(String ordinamento) **post:** return sequence(FornitoreBean)

Invarianti	

Nome Classe	**SoftwarehouseDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Softwarehouse tramite dei metodi specifici.
Pre-condizione	**context** SoftwarehouseDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento
Post-condizione	**context** SoftwaregouseDAO**: :** allElements(String ordinamento) **post:** return sequence(SoftwarehouseBean)
Invarianti	

Nome Classe	**AbbonamentoDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Abbonamento tramite dei metodi specifici.
Pre-condizione	**context** AbbonamentoDAO**: :** ricercaPerChiave(String code) **pre:** code != null and code !="" **context** AbbonamentoDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** AbbonamentoDAO**: :** newInsert(AbbonamentoBean item) **pre:** item != null and !dbAbbonamentoContains(item.codice,item.prodotto)
Post-condizione	**context** AbbonamentoDAO**: :** ricercaPerChiave(String code) **post:** return AbbonamentoBean OR null se non esiste **context** AbbonamentoDAO**: :** allElements(String ordinamento) **post:** return sequence(AbbonamentoBean) **context** AbbonamentoDAO**: :** newInsert(AbbonamentoBean item) **post:** viene inserita una tupla nel database nella tabella abbonamento con I valori che si trovano in item

Invarianti	

Nome Classe	**DlcDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Dlc tramite dei metodi specifici.
Pre-condizione	**context** DlcDAO**: :** ricercaPerChiave(String code) **pre:** code != null and code !="" **context** DlcDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** DlcDAO**: :** newInsert(DlcBean item) **pre:** item != null and !dbDlcContains(item.prodotto)
Post-condizione	**context** DlcDAO**: :** ricercaPerChiave(String code) **post:** return DlcBean OR null se non esiste **context** DlcDAO**: :** allElements(String ordinamento) **post:** return sequence(DlcBean) **context** DlcDAO**: :** newInsert(DlcBean item) **post:** viene inserita una tupla nel database nella tabella dlc con I valori che si trovano in item
Invarianti	

Nome Classe	**ConsoleDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Console tramite dei metodi specifici.
Pre-condizione	**context** ConsoleDAO**: :** ricercaPerChiave(String code) **pre:** code != null and code !="" **context** ConsoleDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve

	rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** ConsoleDAO**: :** newInsert(ConsoleBean item) **pre:** item != null and !dbConsoleContains(item.prodotto)
Post-condizione	**context** ConsoleDAO**: :** ricercaPerChiave(String code) **post:** return ConsoleBean OR null se non esiste **context** ConsoleDAO**: :** allElements(String ordinamento) **post:** return sequence(ConsoleBean) **context** ConsoleDAO**: :** newInsert(ConsoleBean item) **post:** viene inserita una tupla nel database nella tabella console con I valori che si trovano in item
Invarianti	

Nome Classe	**RecensisceDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Recensisce tramite dei metodi specifici.
Pre-condizione	**context** RecensisceDAO**: :** ricercaPerChiave(int Prodotto,String Cliente) **pre:** prodotto>0 and cliente!=null and cliente=!"" **context** RecensisceDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** RecensisceDAO**: :** newInsert(RecensisceBean item) **pre:** item != null and !dbRecensisceContains(item.cliente, item. prodotto) **context** RecensisceDAO: : ricercaPerProdotto(String id) **pre:** id>0
Post-condizione	**context** RecensisceDAO**: :** ricercaPerChiave(int Prodotto,String Cliente) **post:** return RecensisceBean OR null se non esiste **context** RecensisceDAO**: :** allElements(String ordinamento) **post:** return sequence(RecensisceBean) **context** RecensisceDAO**: :** newInsert(RecensisceBean item) **post:** viene inserita una tupla nel database nella tabella recensisce con I valori che si trovano in item

	context RecensisceDAO: : ricercaPerProdoto(String id) **post:** return sequence(RecensisceBean)
Invarianti	

Nome Classe	**ParteDIDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare ParteDi tramite dei metodi specifici.
Pre-condizione	**context** ParteDiDAO**: :** ricercaPerChiave(int id1,String id2) **pre:**id1>0 and id2 != null **context** ParteDiDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** ParteDiDAO**: :** newInsert(ParteDiBean item) **pre:** item != null and !dbParteDiContains(item.videogioco, item.categoria) **context** ParteDiDAO**: :** ricercaPerCategoria(String id) **pre:** id != null and id != ""
Post-condizione	**context** ParteDiDAO**: :** ricercaPerChiave(int id1, String id2) **post:** return ParteDiBean OR null se non esiste **context** ParteDiDAO**: :** allElements(String ordinamento) **post:** return sequence(ParteDiBean) **context** ParteDiDAO**: :** newInsert(ParteDiBean item) **post:** viene inserita una tupla nel database nella tabella partedi con I valori che si trovano in item **context** ParteDiDAO**: :** ricercaPerCategoria(String id) **post:** return set(ParteDiBean)
Invarianti	

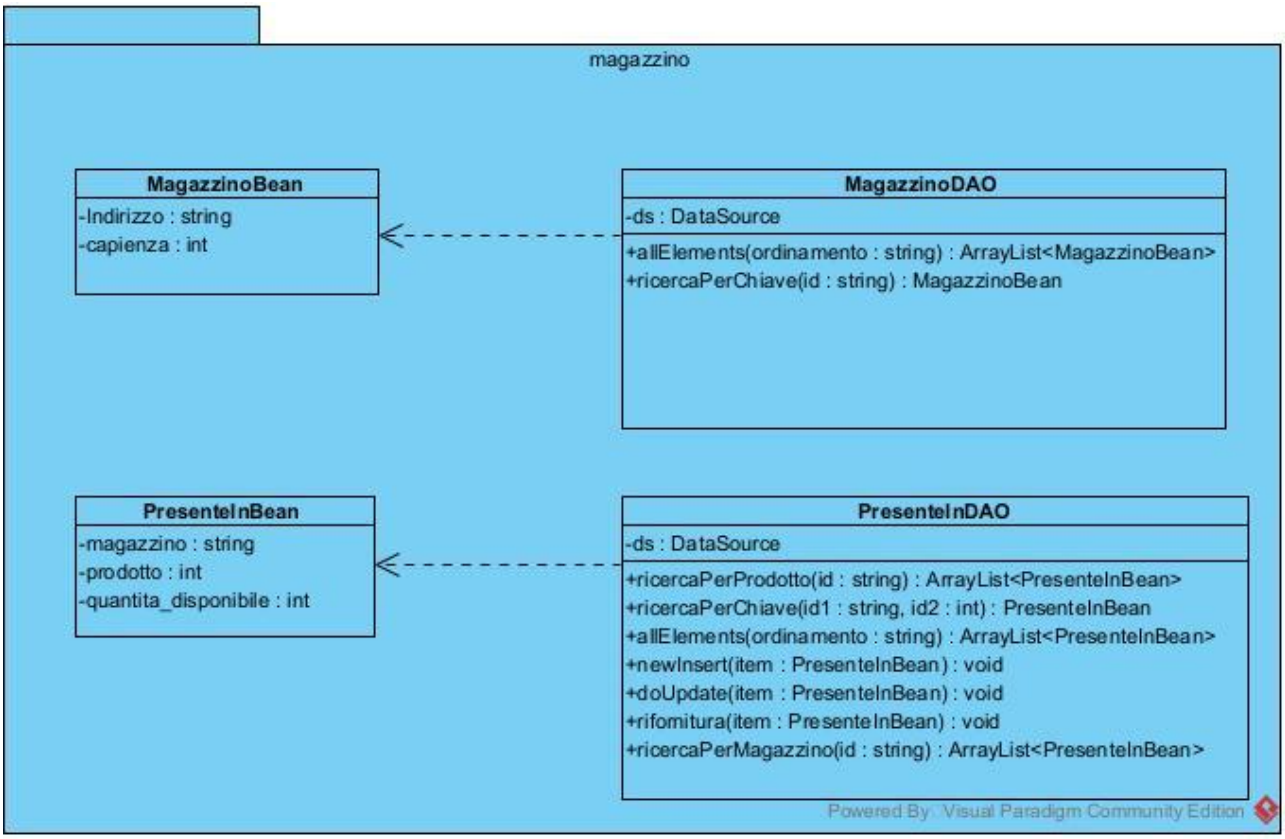
Nome Classe	**VideogiocoDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di

	interrogare l'entità tabellare Videogioco tramite dei metodi specifici.
Pre-condizione	**context** VideogiocoDAO**: :** ricercaPerChiave(String code) **pre:** code != null and code !="" **context** VideogiocoDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** VideogiocoDAO**: :** newInsert(VideogiocoBean item) **pre:** item != null and !dbVideogiocoContains(item.prodotto) and(item.ncd=null or item.vkey=null) **context** VideogiocoDAO : : getTopRecensione() **pre:** hasn't precondition **context** VideogiocoDAO : : getUltimoUscito(int code) **pre:** code>0 **context** VideogiocoDAO : : getVideogiochiScontati (int code1,int code2) **pre:** code1>0 and code2>0 and code1!=code2
Post-condizione	**context** VideogiocoDAO**: :** ricercaPerChiave(String code) **post:** return VideogiocoBean OR null se non esiste **context** VideogiocoDAO**: :** allElements(String ordinamento) **post:** return sequence(VideogiocoBean) **context** VideogiocoDAO**: :** newInsert(VideogiocoBean item) **post:** viene inserita una tupla nel database nella tabella videogioco con I valori che si trovano in item **context** VideogiocoDAO**: :** getTopRecensione(VideogiocoBean item) **post:** return VideogiocoBean **context** VideogiocoDAO**: :** getUltimoUscito(VideogiocoBean item) **post:** return VideogiocoBean **context** VideogiocoDAO**: :** getVideoGiochiScontati(ArrayList<VideogiocoBean item>) **post:** return sequence(VideogiocoBean)
Invarianti	

Nome Classe	**ProdottoDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Prodotto tramite dei metodi specifici.
Pre-condizione	**context** ProdottoDAO**: :** ricercaPerChiave(String code) **pre:** code != null and code !="" **context** ProdottoDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** ProdottoDAO**: :** newInsert(ProdottoBean item) **pre:** item != null and !dbProdottoContains(item.prodotto) **context** ProdottoDAO : : ricercaPerNome(String nome) **pre:** nome!= null **context** ProdottoDAO : : ricercaPerSottostringa(String nome) **pre:** nome!= null and nome!= "" **context** ProdottoDAO : : getMax() **pre:** hasn't precondition **context** ProdottoDAO : : doUpdate(ProdottoBean item) **pre:** item != null and dbProdottoContains(item.codice_prodotto)
Post-condizione	**context** ProdottoDAO**: :** ricercaPerChiave(String code) **post:** return ProdottoBean OR null se non esiste **context** ProdottoDAO**: :** allElements(String ordinamento) **post:** return sequence(ProdottoBean) **context** ProdottoDAO**: :** newInsert(ProdottoBean item) **post:** viene inserita una tupla nel database nella tabella console con I valori che si trovano in item **context** ProdottoDAO**: :** ricercaPerNome(ProdottoBean item) **post:** return ProdottoBean **context** ProdottoDAO**: :** ricercaPerSottostringa(ArrayList <ProdottoBean item>) **post:** return sequence(ProdottoBean) **context** ProdottoDAO**: :** getMax(ProdottoBean item)

	post: return ProdottoBean **context** ProdottoDAO**: :** doUpdate(ProdottoBean item) **post:** viene aggiornata una tupla nel database nella tabella prodotto con I valori che si trovano in item
Invarianti	

2.1.7. Magazzino



- **MagazzinoBean=** rappresenta l'entità magazzino del database
- **PresenteInBean=** rappresenta l'entità presentein del database

Le classi DAO sono descritte in OCL.

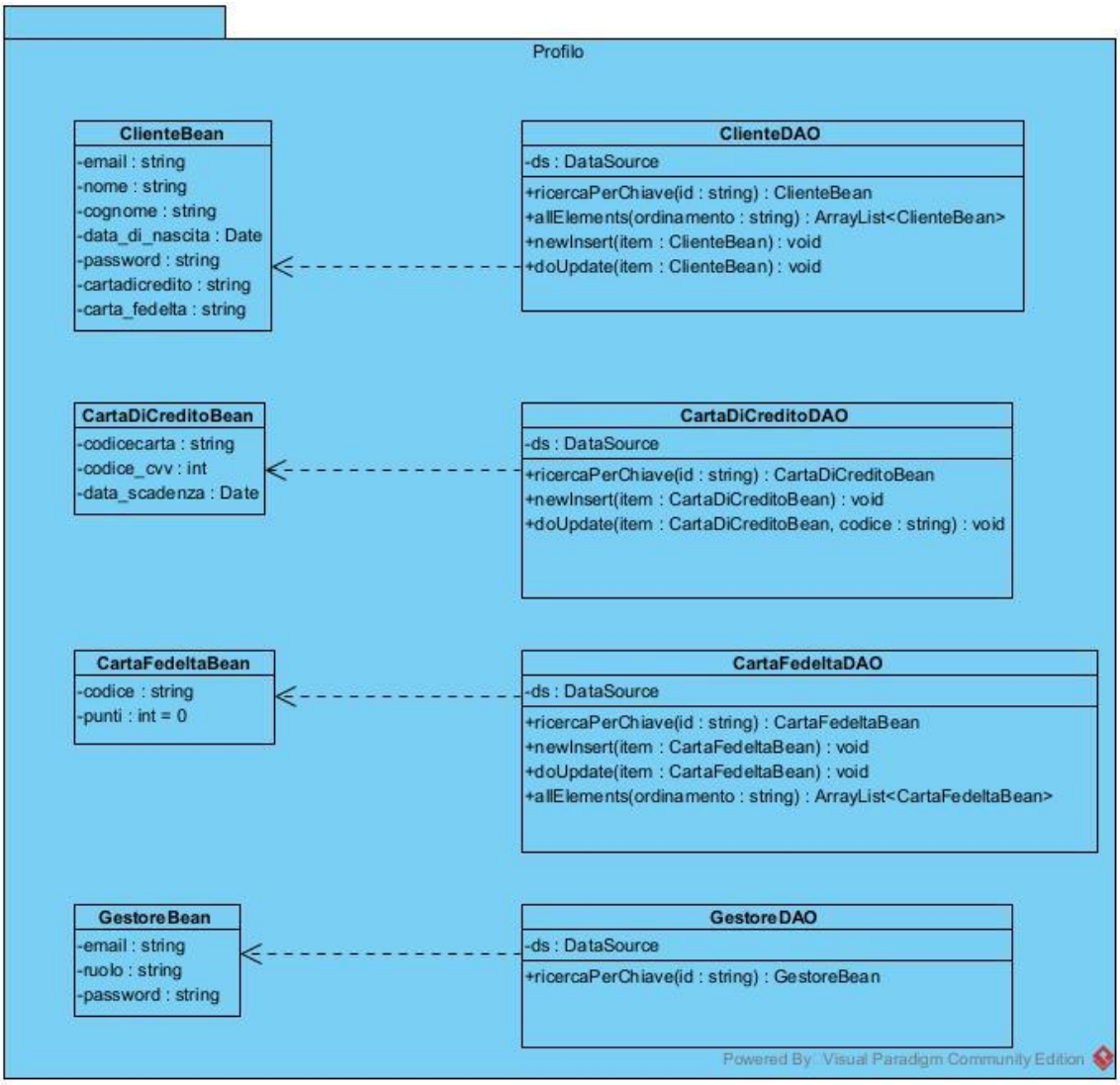
Class Interface delle classi DAO:

Nome Classe	**MagazzinoDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Magazzino tramite dei metodi specifici.
Pre-condizione	**context** MagazzinoDAO**: :** ricercaPerChiave(String id) **pre:** id != null and id != "” **context** MagazzinoDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != "”and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento
Post-condizione	**context** MagazzinoDAO**: :** ricercaPerChiave(String id) **post:** return MagazzinoBean OR null se non esiste **context** MagazzinoDAO**: :** allElements(String ordinamento) **post:** return sequence(MagazzinoBean)
Invarianti	

Nome Classe	**PresenteInDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare PresenteIn tramite dei metodi specifici.
Pre-condizione	**context** PresenteInDAO**: :** ricercaPerProdotto(String id) **pre:** id != null and id != "” **context** PresenteInDAO**: :** ricercaPerChiave(int id1, String id2) **pre:** id1>0 and id2 !=null and id2!=""” **context** PresenteInDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != "”and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** PresenteInDAO**: :** newInsert(PresenteInBean item) **pre:** item != null **context** PresenteInDAO**: :** doUpdate(PresenteInBean item) **pre:** item != null and dbPresenteInContains(item)

	context PresenteInDAO: : rifornitura(PresenteInBean item) **pre:** item != null and dbPresenteInContains(item) **context** PresenteInDAO: : ricercaPerMagazzino(String id) **pre:** id != null and id !=“”
Post-condizione	**context** PresenteInDAO: : ricercaPerProdotto(String id) **post:** return set(PresenteInBean) **context** PresenteInDAO: : ricercaPerChiave(int id1, String id2) **pre:** return PresenteInBean OR null se non esiste **context** PresenteInDAO: : allElements(String ordinamento) **post:** return sequence(PresenteInBean) **context** PresenteInDAO: : newInsert(PresenteInBean item) **post:** viene inserita una tupla nel database nella tabella presentein con I valori che si trovano in item **context** PresenteInDAO: : doUpdate(PresenteInBean item) **post:** viene decrementato di uno la quantità disponibile di un determinato prodotto in un determinato magazzino **context** PresenteInDAO: : rifornitura(PresenteInBean item) **post:** viene aggiornata la quantità disponibile di un determinato prodotto in un dato magazzino in quanto viene effettuata una fornitura di quel prodotto. **context** PresenteInDAO: : ricercaPerMagazzino(String id) **post:** return set(PresenteInBean)
Invarianti	

2.1.8. Profilo



- **ClienteBean=** rappresenta l'entità cliente del database
- **CartaDiCreditoBean=** rappresenta l'entità carta di credito del database
- **CartaFedeltaBean=** rappresenta l'entità carta fedeltà del database
- **GestoreBean=** rappresenta l'entità gestore del database

Le classi DAO sono descritte in OCL.

Class Interface delle classi DAO:

Nome Classe	**GestoreDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Gestore tramite dei metodi specifici.
Pre-condizione	**context** GestoreDAO**: :** ricercaPerChiave(String id) **pre:** id != null and id != ""
Post-condizione	**context** GestoreDAO**: :** ricercaPerChiave(String id) **post:** return GestoreBean OR null se non esiste
Invarianti	

Nome Classe	**CartaDiCreditoDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare CartaDiCredito tramite dei metodi specifici.
Pre-condizione	**context** CartaDiCreditoDAO**: :** ricercaPerChiave(String id) **pre:** id != null and id != "" **context** CartaDiCreditoDAO**: :** newInsert(CartaDiCreditoBean item) **pre:** item != null and !dbCartaDiCreditoContains(item.codice) **context** CartaDiCreditoDAO**: :** doUpdate(CartaDiCreditoBean item,string codice) **pre:** item != null and codice != null and dbCartaDiCreditoContains(ClienteBean.CartaDiCredito)
Post-condizione	**context** CartaDiCredito DAO**: :** ricercaPerChiave(String id) **post:** return CartaDiCreditoBean OR null se non esiste **context** CartaDiCreditoDAO**: :** newInsert(CartaDiCreditoBean item) **post:** viene inserita una tupla nel database nella tabella CartaDiCredito con I valori che si trovano in item **context** CartaDiCreditoDAO**: :** doUpdate(CartaDiCreditoBean item) **post:** aggiorna la tupla nel database nella tabella CartaDiCredito corrispondente
Invarianti	

Nome Classe	**ClienteDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di interrogare l'entità tabellare Cliente tramite dei metodi specifici.
Pre-condizione	**context** ClienteDAO**: :** ricercaPerChiave(String id) **pre:** id != null and id != "” **context** ClienteDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != "”and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento **context** ClienteDAO**: :** newInsert(ClienteBean item) **pre:** item != null and !dbClienteContains(item.codice) **context** ClienteDAO**: :** doUpdate(ClienteBean item) **pre:** cliente != null.
Post-condizione	**context** ClienteDAO**: :** ricercaPerChiave(String id) **post:** return ClienteBean OR null se non esiste **context** ClienteDAO**: :** allElements(String ordinamento) **post:** return sequence(ClienteBean) **context** ClienteDAO**: :** newInsert(ClienteBean item) **post:** viene inserita una tupla nel database nella tabella Cliente con I valori che si trovano in item **context** ClienteDAO**: :** doUpdate(ClienteBean item) **post:** aggiorna la tupla nel database nella tabella Cliente corrispondente
Invarianti	

Nome Classe	**CartaFedeltaDAO**
Descrizione	Questa classe ci permette di interfacciarci con il DBMS relazionale e di

	interrogare l'entità tabellare CartaFedelta tramite dei metodi specifici.
Pre-condizione	**context** CartaFedeltaDAO**: :** ricercaPerChiave(String id) **pre:** id != null and id != "". **context** CartaFedeltaDAO**: :** newInsert(CartaFedeltaBean item) **pre:** item != null and !dbCartaFedeltaContains(item.codice) **context** CartaFedeltaDAO**: :** doUpdate(CartaFedeltaBean item) **pre:** item.codice != null and item.codice != "" and dbCartaFedeltaContains(item.codice). **context** CartaFedeltaDAO**: :** allElements(String ordinamento) **pre:** ordinamento != null and ordinamento != ""and ordinamento deve rispettare il formato di ORDER BY di SQL ovvero: "parametro asc/desc" e il parametro deve corrispondere a un attributo della tabella su cui facciamo l'ordinamento
Post-condizione	**context** CartaFedeltaDAO**: :** ricercaPerChiave(String id) **post:** return CartaFedeltaBean OR null se non esiste **context** CartaFedeltaDAO**: :** newInsert(CartaFedeltaBean item) **post:** viene inserita una tupla nel database nella tabella CartaFedelta con I valori che si trovano in item **context** CartaFedeltaDAO**: :** doUpdate(String id) **post:** aggiorna la quantita nel database nella tabella CartaFedelta corrispondente con quell id. **context** CartaFedeltaDAO**: :** allElements(String ordinamento) **post:** return sequence(CartaFedeltaBean)
Invarianti	

3. Design Pattern con Class Diagram

Nel nostro sistema utilizziamo un pattern DAO perché ci permette di separare gli oggetti che si occupano di attività di controllo e gestione dalla logica di acceso ai

dati. Infatti, i componenti che si occupano di attività di controllo e gestione non dovrebbero mai accedere direttamente al database: questo comporterebbe scarsa manutenibilità. Solo gli oggetti previsti dal pattern Dao possono accedervi.