*Object Design*

*Document*

*Progetto*

*OctoPlus*



**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Descrizione | Autore |
| 06/12/2023 | **0.1** | **Prima stesura** | **Tutto il team** |
| 07/12/2023 | **0.2** | **Aggiunta sezione package + revisione documento** | **Donnarumma Salvatore** |
| 08/12/2023 | **0.3** | **Aggiunti interfaccia admin e prodotto** | **Tomeo Orlando** |
| 09/12/2023 | **0.4** | **Aggiunta interfaccia Ordini** | **Donnarumma Salvatore** |
| 20/01/2023 | **0.5** | **Modifica e correzione dei Dao** | **Donnarumma Salvatore** |
| 22/01/2024 | **0.6** | **Revisione** | **Tutto il team** |
| 26/01/2024 | **0.7** | **Aggiunto Layer Storage** | **Orlando Tomeo** |
| 27/01/2024 | **0.8** | **Modificati tutti i grafici** | **Donnarumma Salvatore** |

# 1. Introduzione

## 1.1 Object design trade-offs

### 1.1.1 Robustezza vs Tempo

Nei primi tempi non sono previsti picchi di utenza elevati, pertanto si è deciso di ritardare l’implementazione di funzionalità avanzate che renderebbero il sistema più robusto e capace di gestire uno stress maggiore, questo per velocizzare il lancio della prima versione.

### 1.1.2 Sicurezza vs Tempo

## Si ritiene che la sicurezza dei dati degli utenti registrati e l’attendibilità della piattaforma siano caratteristiche necessarie che la piattaforma dovrebbe avere fin dalla prima versione al fine di tutelare i clienti. Il tutto, naturalmente, richiede tempo per lo sviluppo il quale sarà sottratto alla robustezza.

## 1.2 Linee guida

Qui di seguito sono riportate alcune linee guide per la stesura del codice:

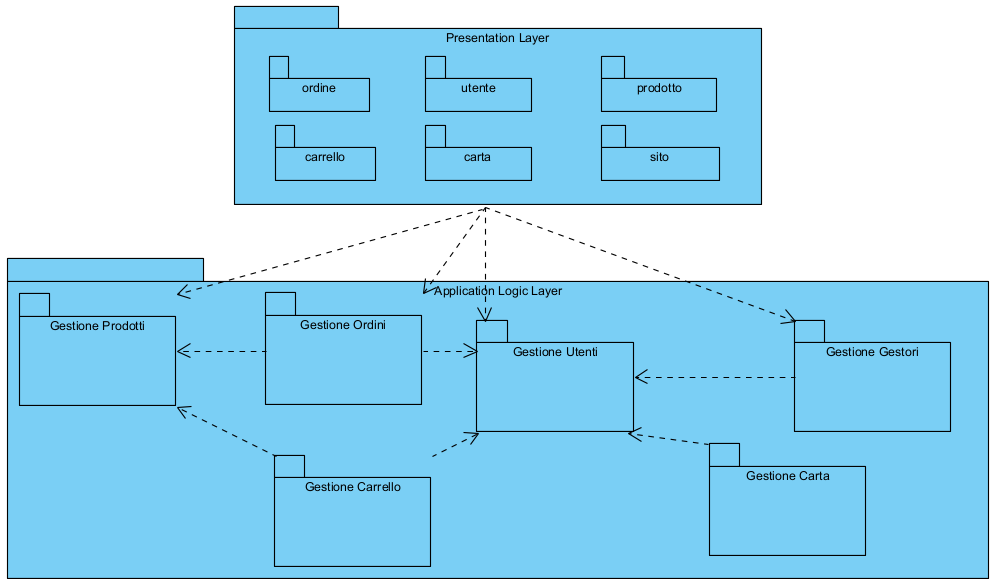
* Gli oggetti Dao dovranno avere il suffisso DaoDataSource nel nome.
* Le classi che identificano le entità devono essere chiamate con nomi singolari.
* I nomi dei file jsp devono essere totalmente in minuscolo mentre i nomi delle Servlet e classi Bean, così come i DAO dovranno iniziare con la prima lettera in maiuscolo.

## 1.3 Referenze

* **R.A.D.**
* **S.D.D.**

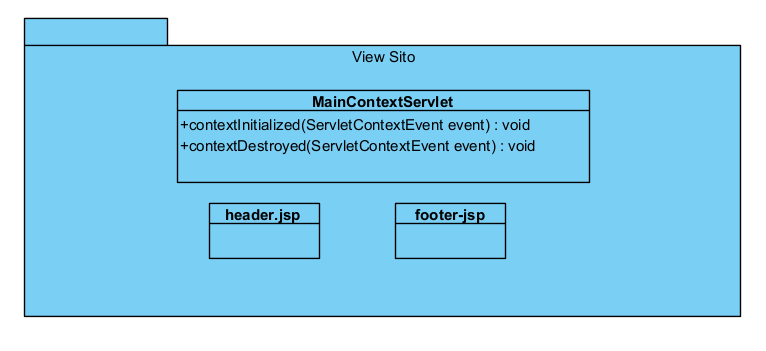
**2. Packages**

Nella sezione presente vengono mostrati nel dettaglio i packages implementati nel nostro sistema. Il pacchetto Presentation Layer contiene tutti i boundary object del sistema, il pacchetto Application Logic Layer conterrà entity objects, control objects e DAO del sistema. È stata scelta questa soluzione poiché i DAO verranno implementati sulla componente server e non database.



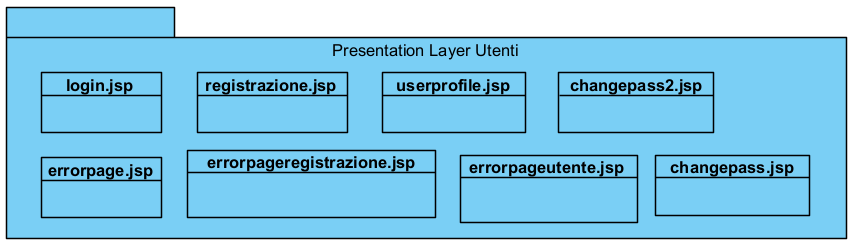
**2.1 Presentation Layer Sito**

Questo sottopacchetto è composto dai file jsp adibite alle funzioni generali del sito come footer, header, etc.



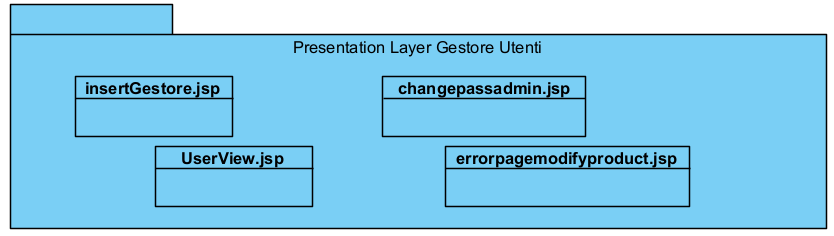
**2.2 Presentation Layer Utente**

Questo pacchetto è composto dai file jsp relativi alle funzionalità per la gestione degli utenti, come autenticazione, registrazione, cambio password, etc.



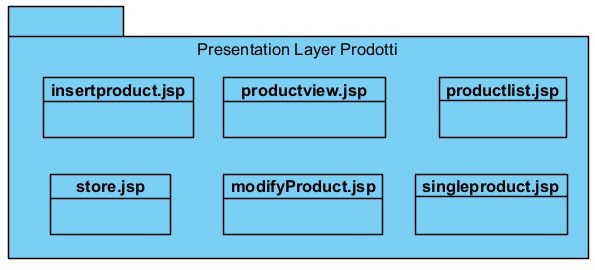
**2.3 Presentation Layer Gestore**

Questo pacchetto è composto dai file jsp relativi alle funzionalità per la gestione degli utenti da parte del Gestore Utente come creazione di nuovi gestori, cambio password degli altri gestori, ordinamento degli utenti etc.



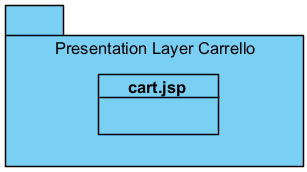
**2.4 Presentation Layer Prodotti**

Questo pacchetto è composto dai file jsp relativi alle funzionalità per la gestione dei prodotti come rimozione degli stessi dal catalogo, aggiunta, modifica, etc.



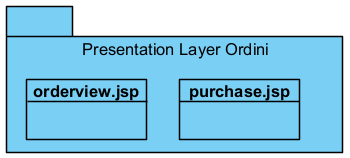
**2.5 Presentation Layer Carrello**

Questo pacchetto è composto dai file jsp relativi alle funzionalità per la gestione del carrello come rimozione dei prodotti da esso, aggiunta e visualizzazione.



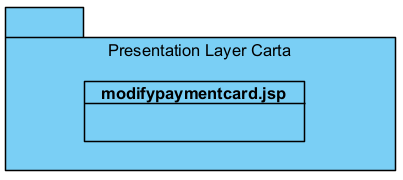
**2.6 Presentation Layer Ordini**

Questo pacchetto è composto dai file jsp relativi alle funzionalità per la gestione degli ordini come rimozione degli ordini, creazione dell’ordine (dunque acquisto di prodotti), visualizzazione, etc.



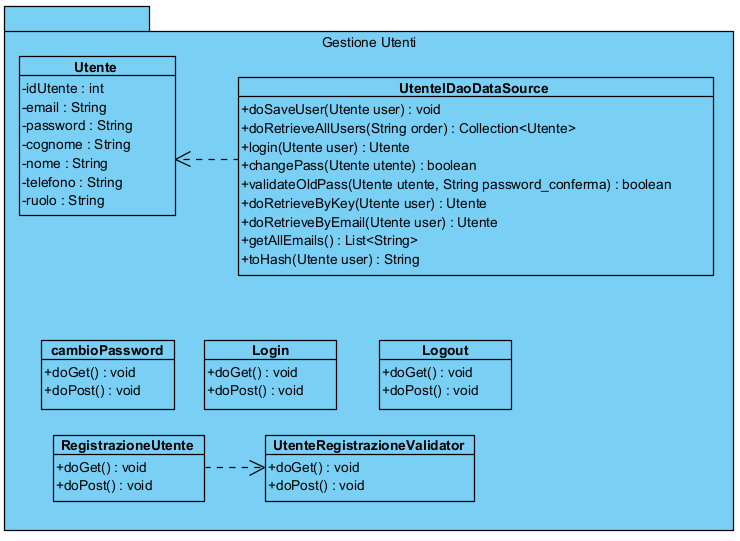
**2.7 Presentation Layer Carta**

Questo pacchetto è composto dai jsp relativi alle funzionalità per la gestione della carta come il suo salvataggio o rimozione etc.

****

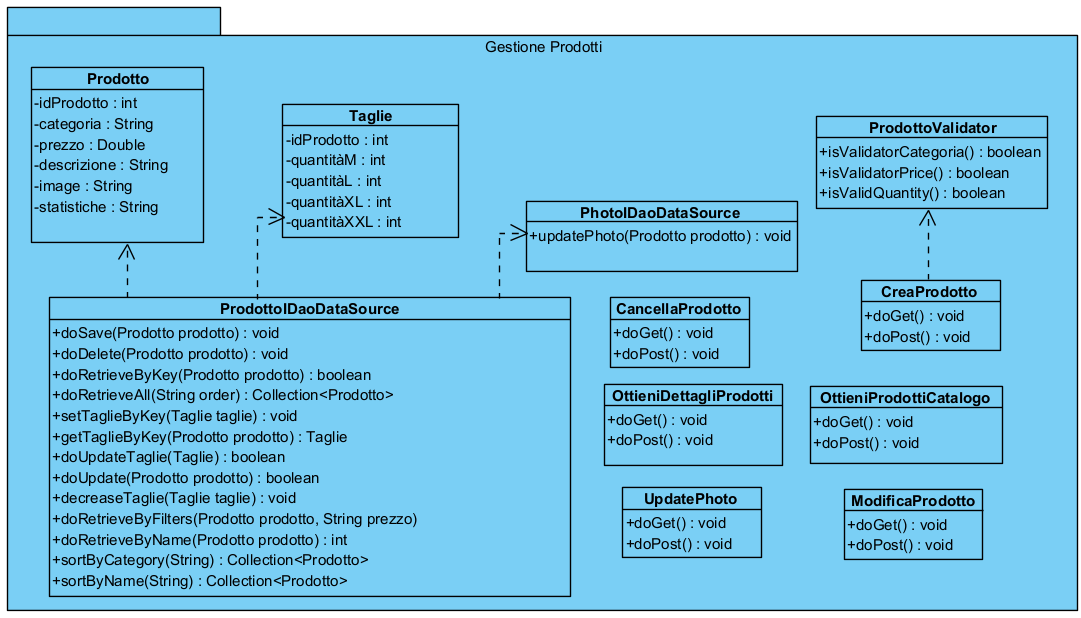
**2.8 Gestione Utenti**

Questo pacchetto contiene le classi Java (Control Objects, Entity Objects e DAO) dedicati alle funzionalità per la gestione degli utenti e informazioni relative ad essi.



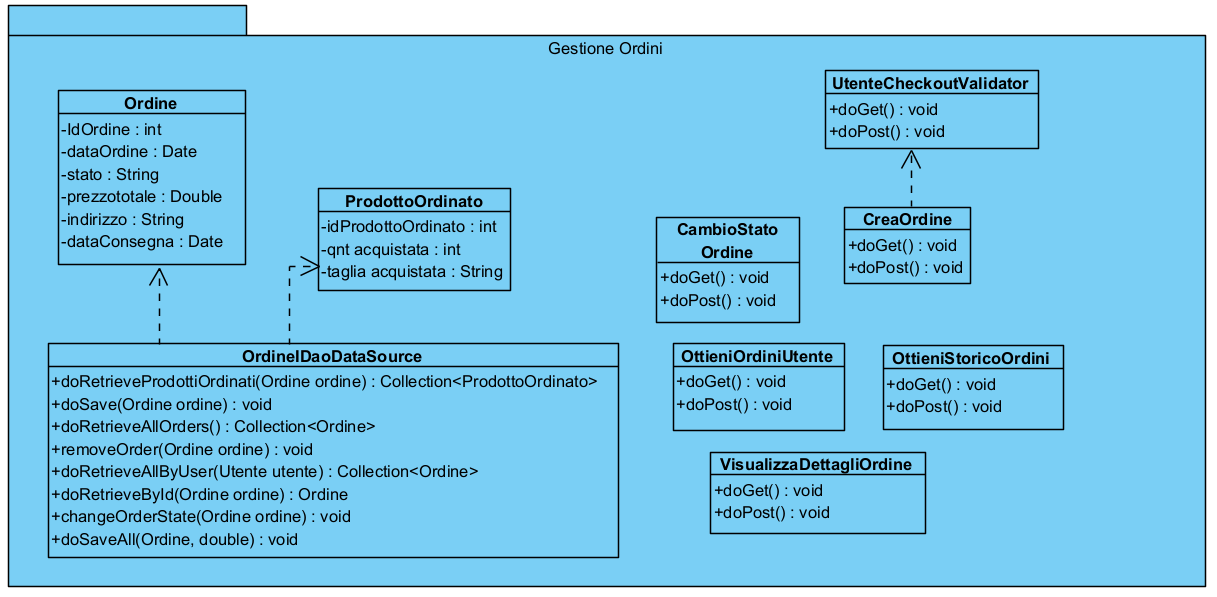
**2.9 Gestione Prodotti**

Questo pacchetto contiene le classi Java (Control Objects, Entity Objects e DAO) dedicati alle funzionalità per la gestione dei prodotti e informazioni relative ad essi.

****

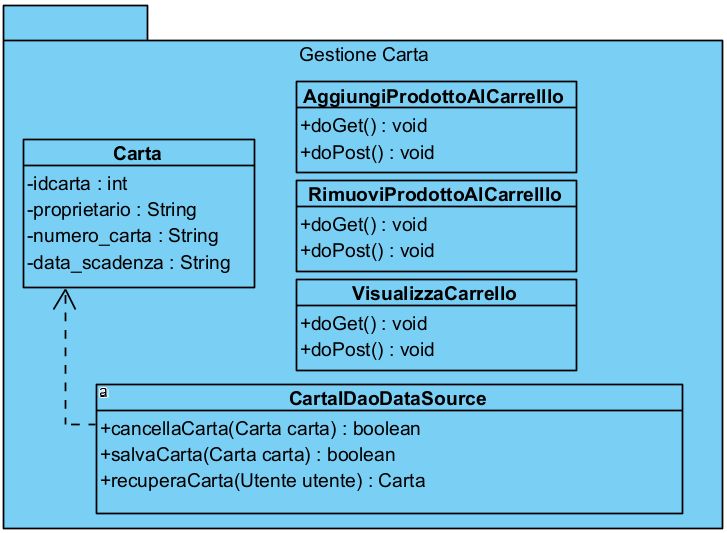
**2.10 Gestione Ordini**

Questo pacchetto contiene le classi Java (Control Objects, Entity Objects e DAO) dedicati alle funzionalità per la gestione degli ordini e informazioni relative ad essi.

****

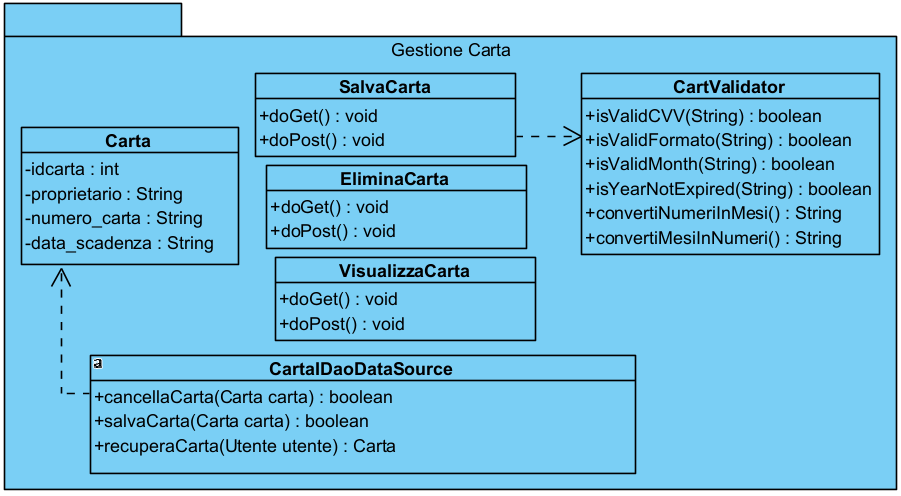
**2.11 Gestione Carrello**

Questo pacchetto contiene le classi Java (Control Objects, Entity Objects e DAO) dedicati alle funzionalità per la gestione degli ordini e informazioni relative ad essi.

****

**2.12 Gestione Carta**

Questo pacchetto contiene le classi Java (Control Objects, Entity Objects e DAO) dedicati alle funzionalità per la gestione della carta.



# 3. Interfacce di classe

## 3.1 UtenteDaoDataSource

|  |  |
| --- | --- |
| UtenteDaoDataSource | Questa classe permette di interfacciarci al DBMS relazione modificando e interrogando l’entità Utente. |
| Precondizione | **PostCondizione** |
| context *UtenteDaoDataSource::doRetrieveByKey(Utente utente)*  pre: utente.email != null | **context** *UtenteDaoDataSource::doRetrieveByKey(Utente utente)*  **post**: return Utente, se la tupla non esiste nel DB l’oggetto restituito ha tutti i valori settati a null, altrimenti contiene le informazioni ricavate dalla tupla. |
| context *UtenteDaoDataSource::doSaveUser(Utente utente)*  pre: *utente.email != null and utente.email != “”*  *and utente.email è formattata correttamente*  *and DB not contains utente.email*  *and utente.password è formattata correttamente and password conferma== utente.password*  *and utente.nome != null and utente.nome != “”*  *and utente.cognome != null and utente.cognome != “”*  *and utente.numerotelefono è formattato correttamente* | **context** *UtenteDaoDataSource::doSaveUser(Utente utente)*  **post**: viene inserita nel DB una tupla contenente le informazioni di account nella tabella Utente. |
| context *UtenteDaoDataSource::doRetrieveAllUsers(String order)*  pre: order == null or order == “Email” or order ==”Cognome” | **context** *UtenteDaoDataSource::doRetrieveAllUsers(String order)*  **post**: return Collection<Utente> contenente tutte le tuple della tabella |
| context *UtenteDaoDataSource::login(Utente utente)*  pre: utente.email != null and utente.email != “” and utente.password != null and utente.password != “” | **context** *UtenteDaoDataSource::login(Utente utente)*  **post**: return Utente, se la tupla non esiste nel DB l’oggetto restituito ha tutti i valori settati a null. |
| context *UtenteDaoDataSource::changePass(Utente utente)*  utente.email != null and utente.email != “” and utente.password != null and utente.password != “” and utente.email è formattata correttamente and utente.password è formattata correttamente | **context** *UtenteDaoDataSource::changePass(Utente utente)*  **post:** return Boolean.true or Boolean.false in base a sé la password è stata correttamente modificata oppure no. Viene aggiornata quindi la tupla corrispondente all’Utente interessato. |
| context *UtenteDaoDataSource::validateOldPassword(Utente old\_pass, Utente new\_pass)*  pre: old\_pass.password != null and old.password != “” and new\_pass.password != null and new\_pass.password != “” | **context** *UtenteDaoDataSource::validateOldPassword(Utente old\_pass, Utente new\_pass)*  **post:** return Boolean.true or Boolean.false se la password inserita corrisponde con quella presente all’interno della tupla relativa a quell’Utente. |
| context *UtenteDaoDataSource::toHash(Utente utente)*  pre: utente.password != null and utente.password != “” and utente.password è formattata correttamente | **context** *UtenteDaoDataSource::toHash(Utente utente)*  **post:**  return String password\_crittografata. |

## 3.2 GestoreDaoDataSource

|  |  |
| --- | --- |
| GestoreDaoDataSource | Questa classe permette di interfacciarci al DBMS relazione modificando e interrogando l’entità Utente. |
| Precondizione | **PostCondizione** |
| context *GestoreDaoDataSource::changePassGestore(Utente utente)*  pre: utente.email != null | **context** *GestoreDaoDataSource::doRetrieveByKey(Utente utente)*  **post**: return Utente, se la tupla non esiste nel DB l’oggetto restituito ha tutti i valori settati a null, altrimenti contiene le informazioni ricavate dalla tupla. |
| context *GestoreDaoDataSource::doSaveGestore(Utente utente)*  pre: *utente.email != null and utente.email != “”*  *and utente.email è formattata correttamente*  *and DB not contains utente.email*  *and utente.password è formattata correttamente and password conferma== utente.password*  *and utente.nome != null and utente.nome != “”*  *and utente.cognome != null and utente.cognome != “”*  *and utente.numerotelefono è formattato correttamente* | **context** *GestoreDaoDataSource::doSaveGestore(Utente utente)*  **post**: viene inserita nel DB una tupla contenente le informazioni di un nuovo gestore. |
| context *GestoreDaoDataSource::doRetrieveUtentiSorted(String order)*  pre: order == null or order == “Email” or order == ”Cognome” | **context** *GestoreDaoDataSource::doRetrieveUtentiSorted(String order)*  **post**: return Collection<Utente> contenente tutte le tuple della tabella. L’ordine di restituzione delle tuple Utente può essere definito dal valore della stringa “order” con la possibilità di restituire tuple ordinate per “Email” e per “Cognome”. In alternativa verranno restituite con ordine di default. |
| context *GestoreDaoDataSource::loginGestore(Utente utente)*  pre: utente.email != null and utente.email != “” and utente.password != null and utente.password != “” | **context** *GestoreDaoDataSource::loginGestore(Utente utente)*  **post**: return Utente, se la tupla non esiste nel DB l’oggetto restituito ha tutti i valori settati a null. |
| context *GestoreDaoDataSource::changePassGestore(Utente utente)*  pre: utente.email != null and utente.email != “” and utente.password != null and utente.password != “” and utente.email è formattata correttamente and utente.password è formattata correttamente | **context** *GestoreDaoDataSource::changePass(Utente utente)*  **post:** return Boolean.true or Boolean.false in base a sé la password è stata correttamente modificata oppure no. Viene aggiornata quindi la tupla corrispondente all’Utente interessato. |
| context *GestoreDaoDataSource::validateOldPassword(Utente old\_pass, Utente new\_pass)*  pre: old\_pass.password != null and old.password != “” and new\_pass.password != null and new\_pass.password != “” | **context** *GestoreDaoDataSource::validateOldPassword(Utente old\_pass, Utente new\_pass)*  **post:** return Boolean.true or Boolean.false se la password inserita corrisponde con quella presente all’interno della tupla relativa a quell’Utente. |
| context *GestoreDaoDataSource::doDeleteGestore(Utente utente)*  pre: utente.email != null and utente.email != “” | **context** *GestoreDaoDataSource::doDeleteGestore(Utente utente)*  **post:** viene cancellata la tupla di Utente con email corrispondente all’email passata. |

## 3.3 OrdiniDaoDataSource

|  |  |
| --- | --- |
| OrdineDaoDataSource | Questa classe permette di interfacciarci al DBMS relazione modificando e interrogando l’entità Ordine. |
| Precondizione | **PostCondizione** |
| context *OrdineDaoDataSource::doRetrieveByKeyO(Ordine ordine)*  pre: ordine.id != null and ordine.id != “” | **context** *OrdineDaoDataSource::doRetrieveByKeyO(Ordine ordine)*  **post**: return Collection<ProdottoOrdinato>, vengono restituite le tuple di ProdottoOrdinato avente uno specifico id\_ordine. |
| context O*rdiniDaoDataSource::doSave(Ordine ordine)*  pre: ordine.id != null and ordine.dataOrdine != null and ordine.stato != null and ordine.prezzototale != null and ordine.indirizzo != null and ordine.dataConsegna != null | **context** O*rdiniDaoDataSource::doSave(Ordine ordine)*  **post**: vengono inserite nel DB una tupla contenente i dati dell’ordine appena creato e una tupla relativa al singolo prodotto ordinato nella tabella ProdottoOrdinato. |
| context O*rdiniDaoDataSource::doSaveAll(Ordine ordine, Double totp)*  pre: ordine.id != null and ordine.dataOrdine != null and ordine.stato != null and ordine.prezzototale != null and ordine.indirizzo != null and ordine.dataConsegna != null and ordine.orderedProducts != null | **context** O*rdiniDaoDataSource::doSaveAll(Ordine ordine, Double totp)*  **post**: vengono inserite nel DB una tupla contenente i dati dell’ordine appena creato e le tuple relative ai prodotti ordinati nella tabella ProdottoOrdinato. |
| context *OrdiniDaoDataSource::doRetrieveAllOrders()*  pre: | **context** *OrdiniDaoDataSource::doRetrieveAllOrders()*  **post**: return Collection<Ordine>, vengono restituite tutte le tuple presenti nella tabella Ordine. Se non sono presenti tuple, viene restituito null. |
| context *OrdiniDaoDataSource::removeOrderByEmail(Order ordine)*  pre: DB contains ordine | **context** *OrdiniDaoDataSource::removeOrderByEmail(Order ordine)*  **post**: elimina dal DB la tupla corrispondente all’ordine. |
| Context *OrdiniDaoDataSource::doRetrieveByKey(Utente utente)*  pre: utente.email != null and utente.email != “” | **context** *OrdiniDaoDataSource::doRetrieveByKey(Utente utente)*  **post:** return Collection<Ordine>, restituisce tutte le tuple di Ordine che hanno una email specifica. Se la tupla non è presente verrà restituita una istanza di Ordine con i parametri settati a null. |
| context *OrdiniDaoDataSource::doRetrieveById(Ordine ordine)*  pre: ordine.id != null and ordine.id != “” | **context** *OrdiniDaoDataSource::doRetrieveById(Ordine ordine)*  **post:** return Ordine, restituisce una tupla di Ordine che ha un id\_ordine specifico. Se la tupla non è presente verrà restituita una istanza di Ordine con i parametri settati a null. |
| context *OrdiniDaoDataSource::changeOrderState(Ordine ordine)*  pre: ordine.id != null and ordine.id = “” and ordine.stato != null and ordine.stato != “” | **context** *OrdiniDaoDataSource::changeOrderState(Ordine ordine)*  **post:** update del campo “stato” di una tupla di Ordine con id ordine.idOrdine. |

**3.4 Prodotto Model IDS**

|  |  |
| --- | --- |
| ProdottoDaoDataSource | Questa classe permette di interfacciarci al DBMS relazione modificando e interrogando l’entità Prodotto. |
| Precondizione | **PostCondizione** |
| context *ProdottoDaoDataSource::doSave(Prodotto prodotto)*  pre: *prodotto.categoria != null and prodotto.prezzo>0 and prodotto.descrizione != “”*  *and prodotto .foto != null and prodotto.statistiche != “”* | **context**  *ProdottoDaoDataSource::doSave(Prodotto prodotto)*  **post**: viene inserita nel DB una tupla contenente le informazioni di un prodotto in Prodotto |
| context Prodotto*DaoDataSource::doDelete(Prodotto prodotto)*  pre: *DB contains prodotto* | **context** Prodotto*DaoDataSource::doDelete(Prodotto prodotto)*  **post**: elimina dal DB la tupla corrispondente al prodotto |
| context Prodotto*DaoDataSource::doRetrieveAll(String order)*  pre: order == null or order == “Nome” or order ==”Categoria” | **context** Prodotto*DaoDataSource::doRetrieveAll (String order)*  **post**: return Collection<Prodotto> contenente tutte le tuple della tabella |
| context Prodotto*DaoDataSource::doRetrieveByName(Prodotto prodotto)*  pre: prodotto.nome != null and prodotto.nome != “” | **context** *ProdottoDaoDataSource::doRetrieveByFilters(Prodotto prodotto, String prezzo)*  **post**: return int id di Prodotto, il cui nome corrispondente al nome passato. Altrimenti null |
| context Prodotto*DaoDataSource::doRetrieveByKey(Prodotto prodotto)*  pre: prodotto.id != null and prodotto.id != “” | **context** *ProdottoDaoDataSource::doRetrieveByKey(Prodotto prodotto)*  **post:** return Prodotto, contenente le informazioni della tupla, se esiste. Altrimenti null. |
| context Prodotto*DaoDataSource::setTaglieByKey(Prodotto prodotto, Taglie taglie)*  pre: prodotto.id != null and taglie.id != null and DB contains prodotto | **context** *ProdottoDaoDataSource::setTaglieByKey(Prodotto prodotto, Taglie taglie)*  **post:** aggiorna le tuple delle taglie del prodotto. |
| context Prodotto*DaoDataSource::getTaglieByKey(Prodotto prodotto)*  pre: prodotto.id != null and prodotto.id != “” | **context** Prodotto*DaoDataSource::getTaglieByKey(Prodotto prodotto)*  **post**: return Taglie associate a uno specifico prodotto |
| context Prodotto*DaoDataSource::decreaseTaglie( Taglie taglie)*  pre: prodottoè nel DB and taglie è nel DB and taglie.id\_prodotto != null and taglie.id != null and (taglie.qntM != null and taglie.qntL != null and taglie.qntXL != null and taglie.qntXXL != null) | **context** *ProdottoDaoDataSource::decreaseTaglie(Prodotto prodotto, Tagli taglie)*  **post:** decrementa la disponibilità di taglie relativa ad uno specifico prodotto di una specifica quantità. |
| context Prodotto*DaoDataSource::doUpdate(Prodotto prodotto)*  pre: prodotto è nel DB and prodotto.id != null | **context** *ProdottoDaoDataSource:: doUpdate (Prodotto prodotto)*  **post:** aggiorna le tuple del prodotto con id prodotto.idProdotto and return Boolean.TRUE se l’operazione è andata a buon fine, else return Boolean.FALSE. |
| context Prodotto*DaoDataSource::doUpdateTaglie( Taglie taglie)*  pre: prodottoè nel DB and taglie è nel DB and taglie.id\_prodotto != null and taglie.id != null | **context** *ProdottoDaoDataSource::doUpdateTaglie( Taglie taglie)*  **post:** aggiorna le taglie disponibili di uno specifico prodotto. |

**3.5 Carrello Model IDS**

|  |  |
| --- | --- |
| CarrelloDaoDataSource | Questa classe permette di interfacciarci al DBMS relazione modificando e interrogando l’entità Carrello. |
| Precondizione | **PostCondizione** |
| context *CarrelloDaoDataSource::carrelloEsistente(Utente utente)*  pre: utente.id != “” and utente.id != null | **context** *CarrelloDaoDataSource::doSaveAdmin(Utente utente)*  **post**: return Boolean.FALSE se non esiste nel DB un carrello con id passato. Altrimenti return Boolean.TRUE |
| context *CarrelloDaoDataSource::eliminaCarrello(Utente utente)*  pre: utente.id != “” and utente.id != null | **context** *CarrelloDaoDataSource::eliminaCarrello(Utente utente)*  **post**: elimina nel DB le tuple in ProdottoCarrello con id corrispondenti all’id passato. |
| context *CarrelloDaoDataSource::recuperaCarrello(Utente utente)*  pre: utente.id != “” and utente.id != null | **context** *CarrelloDaoDataSource::recuperaCarrello(Utente utente)*  **post**: return List<Prodotto>, restituisce le tuple in ProdottoCarrello con id corrispondenti all’idUtente passato. |
| context *CarrelloDaoDataSource::salvaCarrello(Utente utente)*  pre: utente.id != “” and utente.id != null | **context** *CarrelloDaoDataSource::salvaCarrello(Utente utente)*  **post**: salva nel DB, nella tabella ProdottoCarrello, le tuple relative ai prodotti di uno specifico Utente con id id\_utente. |

**3.6 Carta Model IDS**

|  |  |
| --- | --- |
| CartaDaoDataSource | Questa classe permette di interfacciarci al DBMS relazione modificando e interrogando l’entità Carta. |
| Precondizione | **PostCondizione** |
| context *CarrelloDaoDataSource::cancellaCarta(Carta carta*  pre: carta.idcarta != “” and carta.idcarta != null | **context** *CarrelloDaoDataSource::cancellaCarta(Carta carta)*  **post**: return Boolean.TRUE se la cancellazione della tupla con chiave idcarta avviene con successo. Altrimenti return Boolean.FALSE |
| context *CarrelloDaoDataSource::salvaCarta(Carta carta)*  pre: carta.idcarta != “” and carta.idcarta != null and carta.numero\_carta != “” and carta.numero\_carta != null and carta.data\_scadenza != null and carta.data\_scadenza != “” and carta.proprietario != null and carta.proprietario != “” | **context** *CarrelloDaoDataSource::salvaCarta(Carta carta)*  **post**: inserisce nella tabella Carta una nuova tupla con le informazioni passate. |
| context *CarrelloDaoDataSource::recuperaCarta(Utente utente)*  pre: utente.idutente != “” and utente.idutente != null | **context** *CarrelloDaoDataSource::recuperaCarta(Utente utente)*  **post**: return Carta, restituisce la tupla di Carta con idcarta corrispondente a utente.idutente passato. |