# **OBJECT DESIGN DOCUMENT**

# Manager:

|  |
| --- |
| **Professori** |
| Prof. De Lucia Andrea |

Partecipanti:

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Luigi Emanuele Sica | 0512109540 |
| Francesco Ciccone | 0512108238 |
| Emanuele Riccardi | 0512107254 |

# Revision History:

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
| 20/10/2021 | 1.0 | Prima stesura del document problem statement | Team members |
| 04/11/2021 | 2.0 | Stesura del document RAD | Luigi Emanuele Sica  Francesco Ciccone  Emanuele Riccardi |
| 20/11/2021 | 3.0 | Revisione del document RAD | Luigi Emanuele Sica  Francesco Ciccone  Emanuele Riccardi |
| 16/12/2021 | 3.1 | Ultima revisione del document | Luigi Emanuele Sica  Francesco Ciccone  Emanuele Riccardi  Emanuele Iannone |
| 20/12/2021 | 3.2 | Stesura document System Design | Luigi Emanuele Sica  Francesco Ciccone  Emanuele Riccardi  Emanuele Iannone |
| 21/12/2021 | 4 | Stesura Object design document | Luigi Emanuele Sica  Francesco Ciccone  Emanuele Riccardi |

Sommario

[1. Introduzione 3](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490419)

[1 Object Design Trade-Offs 3](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490420)

[1.1.1 Componenti Off-the-Shelf 3](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490421)

[1.1.2 Design Patterns 4](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490422)

[1.1.2.1 Data mapper 4](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490423)

[1.1.2.2 Repository pattern 4](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490424)

[1.1.2.3 Service Layer 5](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490425)

[1.2 Linee guida per la documentazione dell'interfaccia 6](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490426)

[1.2.1 Nomenclatura delle componenti 6](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490427)

[1.2.1.1 Nomi delle classi 6](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490428)

[1.2.1.2 Nomi dei metodi 6](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490429)

[1.2.1.3 Nomi delle eccezioni 6](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490430)

[1.2.1.4 Nomi degli altri sorgenti 6](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490431)

[1.2.1.5 Organizzazione delle componenti 7](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490432)

[1.2.1.5 Organizzazione del codice 7](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490433)

[1.3 Definizioni, acronimi ed abbreviazioni 7](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490434)

[1.4 Riferimenti 7](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490435)

[2. Packages 8](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490436)

[2.1 Divisione in pacchetti 8](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490437)

[2.2 Organizzazione del codice in file 8](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490438)

[3. Interfacce delle classi 9](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490439)

[4. Class Diagram 17](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490440)

[5. Glossario 17](file:///C:\Users\luigi\Downloads\ODD_urCoach.docx#_Toc30490441)

# **Object Design Trade-offs**

**● Comprensibilità vs Tempo:**

Il codice deve essere il più comprensibile possibile per poter facilitare la fase di testing ed eventuali future modifiche del codice. A tale scopo, il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Questa caratteristica incrementerà il tempo di sviluppo, ma allo stesso tempo lo renderà più comprensibile.

**● Interfaccia vs Usabilità:**

Il sistema verrà sviluppato con un’interfaccia grafica realizzata in modo semplice, chiara ed intuitiva. Nell’interfaccia saranno presenti form, menu e pulsanti, disposti in maniera tale da rendere semplice l’utilizzo del sistema da parte dell’utente finale.

**● Sicurezza vs Efficienza:**

La sicurezza, come descritto nei requisiti non funzionali, rappresenta uno degli aspetti importanti del sistema. A causa dei tempi di sviluppo molto limitati, ci limiteremo ad implementare un sistema di sicurezza basato sull’utilizzo di username e password degli utenti.

## **1.2 Linee guida per la documentazione dell'interfaccia**

* **Naming conventions**

Gli sviluppatori seguiranno le seguenti linee guida per la definizione delle interfacce:

* + - Classi Java e Servlet:
      * I nomi dovranno iniziare con la lettera maiuscola
      * Se il nome contiene più parole, ognuna di esse dovrà iniziare con una lettera maiuscola
      * I nomi dovranno corrispondere alle informazioni e/o funzionalità che offre quella classe o Servlet

La dichiarazione di classe deve essere caratterizzata da: 1. Dichiarazione della classe pubblica 2. Dichiarazioni di costanti 3. Dichiarazioni di variabili di classe 4. Dichiarazione di variabili d’istanza 5. Costruttore 6. Commento e dichiarazione dei metodi

* + - Metodi:
      * I nomi dovranno iniziare con la lettera minuscola
      * Se il nome contiene più parole, ognuna di esse dovrà iniziare con la lettera maiuscola
      * I nomi dovranno corrispondere alle informazioni e/o funzionalità che offre quel metodo. Si utilizzerà un verbo più eventualmente aggettivi
      * I nomi dei metodi per ottenere e settare attributi seguiranno la regola di nominazione getNomeVariabile e ‘setNomeVariabile
    - Variabili:
      * I nomi delle variabili dovranno iniziare con la lettera minuscola
      * Se il nome contiene più parole, ognuna di esse dovrà iniziare con la lettera maiuscola

# 2. Packages

In questa sezione presentiamo in modo più approfondito quella che è la divisione in sottosistemi e l’organizzazione del codice in file.

## 2.1 Divisione in pacchetti

Il sistema è diviso in packages nel modo seguente:

1)Ogni package conterrà tutti i componenti dedicati a definire una funzionalità specifica del nostro sistema. Ad esempio il pacchetto login conterrà LoginControl.java, LoginBean.java e LoginModeDB.java si troveranno nello stesso pacchetto perché hanno funzionalità dedicate solo al login.

**JSP**

|  |  |
| --- | --- |
| Nome Classe | Account |
| Descrizione | Pagina che mostra all’utente il proprio account |

|  |  |
| --- | --- |
| Nome Classe | Carrello |
| Descrizione | Pagina che mostra all’utente il proprio contenuto del proprio carrello |

|  |  |
| --- | --- |
| Nome Classe | Index |
| Descrizione | Pagina che mostra all’utente la homepage del sito |

|  |  |
| --- | --- |
| Nome Classe | Login |
| Descrizione | Pagina che mostra all’utente la pagina di login |

|  |  |
| --- | --- |
| Nome Classe | Logout |
| Descrizione | Pagina che mostra all’utente la pagina di logout |

|  |  |
| --- | --- |
| Nome Classe | Pagamento |
| Descrizione | Pagina che mostra all’utente la pagina di inserimento dati di pagamento |

|  |  |
| --- | --- |
| Nome Classe | Prodotto |
| Descrizione | Pagina che mostra all’utente le specifiche di un prodotto |

|  |  |
| --- | --- |
| Nome Classe | Registrazione |
| Descrizione | Pagina che mostra all’utente la pagina che consente l’inserimento dei dati di registrazione |

|  |  |
| --- | --- |
| Nome Classe | Successo |
| Descrizione | Pagina che mostra all’utente l’avvenuta ordinazione con successo |

|  |  |
| --- | --- |
| Nome Classe | Categoria |
| Descrizione | Pagina che mostra all’utente i prodotti all’interno di una specifica categoria |

|  |  |
| --- | --- |
| Nome Classe | Personale |
| Descrizione | Pagina che mostra all’utente la propria pagina di lavoro |

**Package**

**Login**

|  |  |
| --- | --- |
| Nome Classe | Login Bean |
| Descrizione | Questa classe rappresenta il login |

|  |  |
| --- | --- |
| Nome Classe | LoginDAO.java |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di validazione e selezione dei dati dell’utente |

|  |  |
| --- | --- |
| Nome Classe | Login Control |
| Descrizione | Questa classe è un control (servlet) si occupa di ricevere i dati di login, elaborarli e decidere se consentire o meno l’accesso all’area personale utilizzando i servizi di Login DAO e Sessione. |

**Logout**

|  |  |
| --- | --- |
| Nome Classe | Logout Bean |
| Descrizione | Questa classe rappresenta il logout |

|  |  |
| --- | --- |
| Nome Classe | LogoutDAO.java(DAO) |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di invalidazione e selezione dei dati del cliente |

|  |  |
| --- | --- |
| Nome Classe | Logout Control |
| Descrizione | Questa classe è un control (servlet) si occupa di ricevere le richieste per invalidare la sessione per consentire il logout. |

**Registrazione**

|  |  |
| --- | --- |
| Nome Classe | Registrazione Bean |
| Descrizione | Questa classe rappresenta le informazioni di una registrazione |

|  |  |
| --- | --- |
| Nome Classe | RegistrazioneDAO.java |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di visualizzazione, aggiunta, cancellazione e modifica degli account utente. |

|  |  |
| --- | --- |
| Nome Classe | Registrazione Control |
| Descrizione | Questa classe è un control (servlet) si occupa di ricevere i dati di registrazione, elaborarli e decidere se consentire o meno la registrazione dell’utente. Fa uso dei servizi di Utente DAO e di Utenti Bean. |

Account

|  |  |
| --- | --- |
| Nome Classe | Account (utente)Bean |
| Descrizione | Questa classe rappresenta le informazioni di un utente |

|  |  |
| --- | --- |
| Nome Classe | Account Control |
| Descrizione | Questa classe è control (servlet) si occupa di gestire le richieste web mostrando lo storico degli ordini effettuati da un utente. Fa uso della Ordini DAO per prendere gli ordini associati al cliente. |

|  |  |
| --- | --- |
| Nome Classe | AccountDAO.java |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di visualizzazione, aggiunta, cancellazione e modifica degli ordini di un utente. |

**Carrello**

|  |  |
| --- | --- |
| Nome Classe | Carrello Bean |
| Descrizione | Questa classe rappresenta il carrello |

|  |  |
| --- | --- |
| Nome Classe | CarrelloDAO.java |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di visualizzazione dei dati di un carrello. |

|  |  |
| --- | --- |
| Nome Classe | Carrello Control |
| Descrizione | Questa classe è un control (Servlet) che si occupa di gestire le richieste web alle funzionalità del sottosistema carrello come aggiunta rimozione e procedi all’ordine. Fa uso dei Metodi di carrello DAO e della classe carrello Bean. |

**Pagamento**

|  |  |
| --- | --- |
| Nome Classe | Pagamento Bean |
| Descrizione | Questa classe rappresenta le informazioni del pagamento |

|  |  |
| --- | --- |
| Nome Classe | Pagamento DAO |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di salvataggio dei dati di un pagamento. |

|  |  |
| --- | --- |
| Nome Classe | Pagamento Control |
| Descrizione | Questa classe è un control (servlet) si occupa gestire le richieste web ricevendo i dati di Pagamento, elaborarli e decidere se consentire di concludere l’acquisto creando un ordine. Fa uso dei servizi di Pagamento DAO, Pagamento Bean e Ordine Bean. |

|  |  |
| --- | --- |
| Nome Classe | Ordine Bean |
| Descrizione | Questa classe rappresenta le informazioni dell’ordine |

|  |  |
| --- | --- |
| Nome Classe | Ordine DAO |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di salvataggio dei dati di un’ ordine. |

**Prodotto**

|  |  |
| --- | --- |
| Nome Classe | Prodotto(shop) Bean |
| Descrizione | Questa classe rappresenta le informazioni di un prodotto |

|  |  |
| --- | --- |
| Nome Classe | Prodotto DAO |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di visualizzazione di prodotti. |

|  |  |
| --- | --- |
| Nome Classe | Prodotto Control |
| Descrizione | Questa classe è un control (servlet) si occupa di gestire le richieste web per visualizzare un prodotto dal sistema. Fa uso di Prodotti DAO e Prodotto Bean |

**Spedizione**

|  |  |
| --- | --- |
| Nome Classe | Spedizione Bean |
| Descrizione | Questa classe rappresenta le informazioni di una spedizione |

|  |  |
| --- | --- |
| Nome Classe | SpedizioneDAO |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di salvataggio dei dati di una spedizione. |

|  |  |
| --- | --- |
| Nome Classe | Spedizione Control |
| Descrizione | Questa classe è un control (servlet) si occupa di gestire le richieste web ricevendo i dati di Spedizione ed elaborarli. Fa uso dei servizi di Spedizione DAO e di Spedizione Bean. |

**Personale**

|  |  |
| --- | --- |
| Nome Classe | Personale Bean |
| Descrizione | Questa classe rappresenta le informazioni di un personale |

|  |  |
| --- | --- |
| Nome Classe | Direttore DAO |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di visualizzazione, inserimento, modifica e rimozione del personale. |

|  |  |
| --- | --- |
| Nome Classe | GestoreOrdini DAO |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di visualizzazione , inserimento, modifica e rimozione di un ordine. |

|  |  |
| --- | --- |
| Nome Classe | GestoreProdotti DAO |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di visualizzazione, inserimento, modifica e rimozione di un prodotto. |

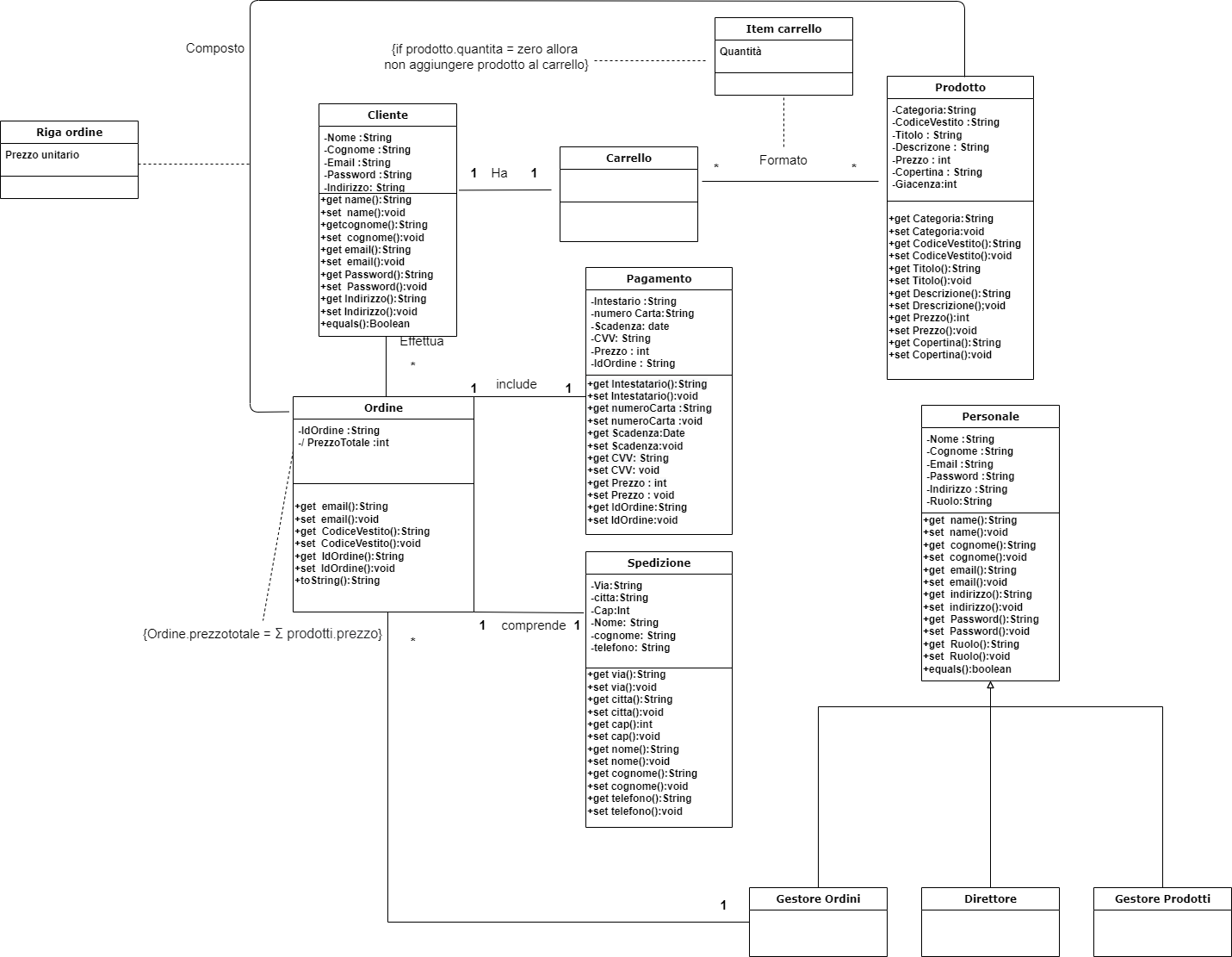
|  |  |
| --- | --- |
| Nome Classe | Direttore Control |
| Descrizione | Questa classe è un control (servlet) si occupa di ricevere i dati di un direttore, elabora e decidere di inserire nuovi personali nel sito. Fa uso dei servizi di Direttore DAO e di Personale Bean. |

|  |  |
| --- | --- |
| Nome Classe | GestoreOrdini Control |
| Descrizione | Questa classe è un control (servlet) si occupa di ricevere i dati di un direttore, elabora e decide di gestire gli ordini effettuati nel sito. Fa uso dei servizi di GestoreOrdiniDAO e di Personale Bean. |

|  |  |
| --- | --- |
| Nome Classe | GestoreProdotti Control |
| Descrizione | Questa classe è un control (servlet) si occupa di ricevere i dati di un Gestore prodotti, elabora e decide di gestire i prodotti nel sito. Fa uso dei servizi di GestoreProdottiDAO e di Personale Bean. |

**Diagramma delle classi**



****



# Interfacce delle Classi

|  |  |
| --- | --- |
| Nome Classe: | **ProdottoModel** |

|  |  |
| --- | --- |
| Context: | ProdottoModel:doSave(prodotto:Prodotto) |
| Descrizione | Questo metodo consente di aggiungere un nuovo prodotto all’interno della tabella Prodotto del database. |
| Pre-condizione: | Il prodotto deve essere diverso da null e il codiceVestito deve essere diverso da tutti gli altri codiceVestito appartenenti ai prodotti già presenti nella tabella Prodotto |
| Post-condizione: | Il prodotto è visibile nella tabella Prodotto e il codiceVestito è univoco |

|  |  |
| --- | --- |
| Context: | ProdottoModel:doRetrieveAll(order:String):Collection<Prodotto> |
| Descrizione | Questo metodo consente di ottenere l’insieme dei prodotti all’interno della tabella Prodotto del database in base ad un certo ordine. |
| Pre-condizione: | order diverso da null e deve essere uguale ad “ASC” , “DESC” o ad una stringa vuota |
| Post-condizione: | Return dell’insieme di tutte le tuple all’interno della tabella Prodotto |

|  |  |
| --- | --- |
| Context: | ProdottoModel:doRetrieveByKey(code:String):Prodotto |
| Descrizione | Questo metodo consente di ottenere il prodotto all’interno della tabella Prodotto del database con codiceVestito specificato. |
| Pre-condizione: | code diverso da null e code deve essere presente all’interno di una tupla nella tabella Prodotto |
| Post-condizione: | Return del prodotto con codiceVestito specificato dall’interno della tabella Prodotto |

|  |  |
| --- | --- |
| Context: | ProdottoModel:doDelete(prodotto:Prodotto):Boolean |
| Descrizione | Questo metodo consente di rimuovere il prodotto all’interno della tabella Prodotto del database restituendo true se è andato a buon fine, false altrimenti. |
| Pre-condizione: | prodotto deve essere diverso da null e il codiceVestito del prodotto deve essere presente all’interno di una tupla nella tabella Prodotto |
| Post-condizione: | Return true se la rimozione all’interno della tabella Prodotto è andata a buon fine ed il numero delle tuple all’interno della tabella Prodotto sarà ridotto di uno, false altrimenti. |

|  |  |
| --- | --- |
| Context: | ProdottoModel:doUpdate(prodotto:Prodotto):Boolean |
| Descrizione | Questo metodo consente di aggiornare il prodotto all’interno della tabella Prodotto del database restituendo true se è andato a buon fine, false altrimenti. |
| Pre-condizione: | prodotto deve essere diverso da null e il codiceVestito del prodotto deve essere presente all’interno di una tupla nella tabella Prodotto |
| Post-condizione: | Return true se la modifica all’interno della tabella Prodotto è andata a buon fine e aggiorna i campi del prodotto passato come parametro, false altrimenti. |

|  |  |
| --- | --- |
| Nome Classe: | **AcquistiModel** |

**//SAREBBE PAGAMENTOMODEL(CREDO)**

|  |  |
| --- | --- |
| Context: | AcquistiModel:doSave(ordine:Ordine) |
| Descrizione | Questo metodo consente di aggiungere un nuovo ordine all’interno della tabella Ordine del database. |
| Pre-condizione: | L’ordine deve essere diverso da null e il codiceOrdine deve essere diverso da tutti gli altri codiceOrdine appartenenti agli ordini già presenti nella tabella Ordine |
| Post-condizione: | L’ordine è visibile nella tabella Ordine e il codiceOrdine è univoco |

**//QUESTO PERMETTE AL GESTORE ORDINE DI PRENDERE TUTTI GLI ORDINI**

|  |  |
| --- | --- |
| Context: | AcquistiModel:doRetrieveAll(order:String):Collection<Ordine> |
| Descrizione | Questo metodo consente di ottenere l’insieme degli ordini all’interno della tabella Ordine del database in base ad un certo ordine. |
| Pre-condizione: | order diverso da null e deve essere uguale ad “ASC” , “DESC” o ad una stringa vuota |
| Post-condizione: | Return dell’insieme di tutte le tuple all’interno della tabella Ordine |

|  |  |
| --- | --- |
| Context: | AcquistiModel:doRetrieveByKey(code:String):Ordine |
| Descrizione | Questo metodo consente di ottenere l’ordine all’interno della tabella Ordine del database con codiceOrdine specificato. |
| Pre-condizione: | code diverso da null e code deve essere presente all’interno di una tupla nella tabella Ordine |
| Post-condizione: | Return dell’ordine con codiceOrdine specificato dall’interno della tabella Ordine |

|  |  |
| --- | --- |
| Context: | AcquistiModel:doDelete(ordine:Ordine):Boolean |
| Descrizione | Questo metodo consente di rimuovere l’ ordine all’interno della tabella Ordine del database restituendo true se è andato a buon fine, false altrimenti. |
| Pre-condizione: | ordine deve essere diverso da null e il codiceOrdine dell’ordine deve essere presente all’interno di una tupla nella tabella Ordine |
| Post-condizione: | Return true se la rimozione all’interno della tabella Ordine è andata a buon fine ed il numero delle tuple all’interno della tabella Ordine sarà ridotto di uno, false altrimenti. |

|  |  |
| --- | --- |
| Context: | AcquistiModel:doUpdate(ordine:Ordine):Boolean |
| Descrizione | Questo metodo consente di aggiornare l’ordine all’interno della tabella Ordine del database restituendo true se è andato a buon fine, false altrimenti. |
| Pre-condizione: | ordine deve essere diverso da null e il codiceOrdine del ordine deve essere presente all’interno di una tupla nella tabella Ordine |
| Post-condizione: | Return true se la modifica all’interno della tabella Ordine è andata a buon fine e aggiorna i campi del ordine passato come parametro, false altrimenti. |

|  |  |
| --- | --- |
| Nome Classe: | **CarrelloModel** |

|  |  |
| --- | --- |
| Context: | CarrelloModel:doSave(carrello:Carrello) |
| Descrizione | Questo metodo consente di aggiungere un nuovo carrello all’interno della tabella Carrello del database. |
| Pre-condizione: | Il carrello deve essere diverso da null e il codiceCarrello deve essere diverso da tutti gli altri codiceCarrello appartenenti ai carrelli già presenti nella tabella Carrello |
| Post-condizione: | Il carrello è visibile nella tabella Carrello e il codiceCarrello è univoco |

|  |  |
| --- | --- |
| Context: | CarrelloModel:doRetrieveAll(order:String):Collection<Carrello> |
| Descrizione | Questo metodo consente di ottenere l’insieme dei carrelli all’interno della tabella Carrello del database in base ad un certo ordine. |
| Pre-condizione: | order diverso da null e deve essere uguale ad “ASC” , “DESC” o ad una stringa vuota |
| Post-condizione: | Return dell’insieme di tutte le tuple all’interno della tabella Carrello |

|  |  |
| --- | --- |
| Context: | CarrelloModel:doRetrieveByKey(code:String): Carrello |
| Descrizione | Questo metodo consente di ottenere il carrello all’interno della tabella Carrello del database con codiceCarrello specificato. |
| Pre-condizione: | code diverso da null e code deve essere presente all’interno di una tupla nella tabella Carrello |
| Post-condizione: | Return del carrello con codiceCarrello specificato dall’interno della tabella Carrello |

|  |  |
| --- | --- |
| Context: | CarrelloModel:doDelete(prodotto:Prodotto):Boolean |
| Descrizione | Questo metodo consente di rimuovere il carrello all’interno della tabella Carrello del database restituendo true se è andato a buon fine, false altrimenti. |
| Pre-condizione: | carrello deve essere diverso da null e il codiceCarrello del prodotto deve essere presente all’interno di una tupla nella tabella Carrello |
| Post-condizione: | Return true se la rimozione all’interno della tabella Carrello è andata a buon fine ed il numero delle tuple all’interno della tabella Carrello sarà ridotto di uno, false altrimenti. |

|  |  |
| --- | --- |
| Nome Classe: | **AccountModel** |

|  |  |
| --- | --- |
| Context: | AccountModel:doSave(utente:Utente) |
| Descrizione | Questo metodo consente di aggiungere un nuovo Utente all’interno della tabella Clienti del database. |
| Pre-condizione: | L’utente deve essere diverso da null e l’email deve essere diversa da tutte le altre email appartenenti agli utenti già presenti nella tabella Clienti. |
| Post-condizione: | L’utente è visibile nella tabella Cliente e la sua e-mail è univoca. |

|  |  |
| --- | --- |
| Context: | AccountModel:doDelete(utente:Utente):Boolean |
| Descrizione | Questo metodo consente di rimuovere il prodotto all’interno della tabella Clienti del database restituendo true se è andato a buon fine, false altrimenti. |
| Pre-condizione: | L’utente deve essere diverso da null e la sua email deve essere presente all’interno di una tupla nella tabella Clienti. |
| Post-condizione: | Return true se la rimozione all’interno della tabella Clienti è andata a buon fine ed il numero delle tuple all’interno della tabella Cliente sarà ridotto di uno, false altrimenti. |

|  |  |
| --- | --- |
| Context: | AccountModel:doUpdate(utente:Utente):Boolean |
| Descrizione | Questo metodo consente di aggiornare i dati dell’utente all’interno della tabella Clienti del database restituendo true se è andato a buon fine, false altrimenti. |
| Pre-condizione: | L’utente deve essere diverso da null e l’email del prodotto deve essere presente all’interno di una tupla nella tabella Cliente. |
| Post-condizione: | Return true se la modifica all’interno della tabella Cliente è andata a buon fine e aggiorna i campi del dell’utente passato come parametro, false altrimenti. |

|  |  |
| --- | --- |
| Context: | AccoutModel:doRetrieveAll(order:String):Collection<Utente> |
| Descrizione | Questo metodo consente di ottenere l’insieme degli utenti all’interno della tabella Clienti del database in base ad un certo ordine. |
| Pre-condizione: | order diverso da null e deve essere uguale ad “ASC” , “DESC” o ad una stringa vuota |
| Post-condizione: | Return dell’insieme di tutte le tuple all’interno della tabella Clienti. |

|  |  |
| --- | --- |
| Context: | AccountModel:doRetrieveByKey(email:String):Utente |
| Descrizione | Questo metodo consente di ottenere l’utente all’interno della tabella Clienti del database con l’email specificata. |
| Pre-condizione: | email deve essere diverso da null e deve essere presente all’interno di una tupla nella tabella Clienti. |
| Post-condizione: | Return dell’utente con email specificato dall’interno della tabella Clienti |

|  |  |
| --- | --- |
| Context: | AccountModel:Validazione(email:String,password:String):Utente |
| Descrizione | Questo metodo consente di ottenere l’utente all’interno della tabella Clienti del database con l’email specificata per permettere l’accesso. |
| Pre-condizione: | i paramtri email e password devono essere diverso da null e devono essere presente all’interno di una tupla nella tabella Clienti. |
| Post-condizione: | Return dell’utente con email specificato dall’interno della tabella Clienti per poter permettere l’acceso. |