# **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 gestioneAccount conterrà LoginControl.java, LoginBean.java e LoginModeDS.java si troveranno nello stesso pacchetto perché hanno funzionalità dedicate alla gestione degli account.

**1)JSP relative all’account**

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

|  |  |
| --- | --- |
| Nome Classe | cambioPassword.jsp |
| Descrizione | Pagina che mostra all’utente la pagina per inserire i dati per cambiare la password |

|  |  |
| --- | --- |
| Nome Classe | successoRegistrazione.jsp |
| Descrizione | Pagina che mostra all’utente l’avvenuto successo della registrazione. |

|  |  |
| --- | --- |
| Nome Classe | successoCambioPassword.jsp |
| Descrizione | Pagina che mostra all’utente l’avvenuto successo del cambio password. |

|  |  |
| --- | --- |
| Nome Classe | adminDirettore.jsp |
| Descrizione | Pagina che mostra le operazioni dedicate al direttore. |

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

|  |  |
| --- | --- |
| Nome Classe | loginPersonale.jsp |
| Descrizione | Pagina che mostra al personale la pagina di login |

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

**2)JSP relative agli Acquisti**

|  |  |
| --- | --- |
| Nome Classe | spedizione.jsp |
| Descrizione | Pagina che mostra all’utente la pagina di inserimento dati di spedizione |

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

|  |  |
| --- | --- |
| Nome Classe | adminOrdini.jsp |
| Descrizione | Pagina che mostra le operazioni dedicate al gestore ordini. |

**3)JSP relative al carrello**

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

**4)JSP relative ai prodotti**

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

|  |  |
| --- | --- |
| Nome Classe | Bambini.jsp |
| Descrizione | Pagina che mostra all’utente i prodotti riguardanti la categoria bambini |

|  |  |
| --- | --- |
| Nome Classe | Donna.jsp |
| Descrizione | Pagina che mostra all’utente i prodotti riguardanti la categoria Donna |

|  |  |
| --- | --- |
| Nome Classe | Uomo.jsp |
| Descrizione | Pagina che mostra all’utente i prodotti riguardanti la categoria Uomo |

|  |  |
| --- | --- |
| Nome Classe | adminProdotti.jsp |
| Descrizione | Pagina che mostra le operazioni dedicate al gestore dei prodotti. |

**4)JSP generali**

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

|  |  |
| --- | --- |
| Nome Classe | headerNavBar.jsp |
| Descrizione | Pagina che mostra all’utente la barra fi navigazione |

|  |  |
| --- | --- |
| Nome Classe | footer.jsp |
| Descrizione | Pagina che mostra all’utente il footer in fondo alla pagine |

**Package**

1. **GestioneAccount**

|  |  |
| --- | --- |
| Nome Classe | UtenteBean.java |
| Descrizione | Questa classe rappresenta le informazioni di un utente |

|  |  |
| --- | --- |
| Nome Classe | PersonaleBean.java |
| Descrizione | Questa classe rappresenta le informazioni di un utente |

|  |  |
| --- | --- |
| Nome Classe | AccountControl.java |
| 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 OrdiniModelDAO.java per prendere gli ordini associati al cliente. |

|  |  |
| --- | --- |
| Nome Classe | CambiaPasswordControl.java |
| Descrizione | Questa classe è control (servlet) si occupa di gestire le richieste web per modificare la password dell’utente. Fa uso della RegistrazioniModelDS.java per cambiare la password. |

|  |  |
| --- | --- |
| Nome Classe | GestorePersonaleControl.java |
| Descrizione | Questa classe è control (servlet) si occupa di gestire le richieste web per la gestione del personale. Fa uso della GestorePeronaleModelDS.java per aggiungere, rimuovere o modificare i dati un personale. |

|  |  |
| --- | --- |
| Nome Classe | GestorePersonaleModelDS.java |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di validazione, aggiunta, rimozione e modifica personale. |

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

|  |  |
| --- | --- |
| Nome Classe | LoginControl.java |
| 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 LoginModelDS.java e Sessione. |

|  |  |
| --- | --- |
| Nome Classe | LoginPersonaleControl.java |
| 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 GestorePersonaleModelDS.java e Sessione. |

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

|  |  |
| --- | --- |
| Nome Classe | RegistrazioneModelDS.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 RegistrazioneModelDS.java e di UtentiBean.java. |

**Package**

1. **Gestione Carrello**

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

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

|  |  |
| --- | --- |
| Nome Classe | CarrelloControl |
| 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, e generare un ordine. Fa uso dei Metodi di carrello DAO e della classe carrello Bean. |

|  |  |
| --- | --- |
| Nome Classe | GestioneCarrelloModelDS |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di visualizzazione, salvataggio, modifica ed eliminazione dei dati di un carrello. |

|  |  |
| --- | --- |
| Nome Classe | SessionCarrelloBean |
| Descrizione | Questa classe rappresenta il carrello in sessione |

**Package**

1. **Gestione Acquisti**

|  |  |
| --- | --- |
| Nome Classe | GestioneOrdineBean.java |
| Descrizione | Questa classe rappresenta le informazioni contenente tutte le informazioni di un ordine |

|  |  |
| --- | --- |
| Nome Classe | GestioneOrdiniModelDS.java |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni visualizzazione degli ordini, modifica, rimozione e conferma ordine. |

|  |  |
| --- | --- |
| Nome Classe | GestoreOrdiniControl.java |
| Descrizione | Questa classe è un control (servlet) si occupa gestire le richieste web ricevendo i dati degli ordini, elaborarli e decidere se confermare, eliminare e modificare i dati dell’ordini. Fa uso dei servizi di GestioneOrdiniModelDS.java per effettuare le operazioni dette in precedenza. |

|  |  |
| --- | --- |
| Nome Classe | OrdineBean.java |
| Descrizione | Questa classe rappresenta le informazioni dell’ordine. |

|  |  |
| --- | --- |
| Nome Classe | PagementoBean.java |
| Descrizione | Questa classe rappresenta le informazioni del pagamento. |

|  |  |
| --- | --- |
| Nome Classe | PagamentoModelDS.java |
| 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 PagamentoModelDS, Pagamento Bean. |

|  |  |
| --- | --- |
| Nome Classe | Spedizione Control |
| Descrizione | Questa classe è un control (servlet) si occupa di gestire le richieste web ricevendo i dati di spedizione e mettendoli nella sessione. |

**Package**

1. **Gestione Prodotti**

|  |  |
| --- | --- |
| Nome Classe | ShopBean |
| Descrizione | Questa classe rappresenta le informazioni di un prodotto |

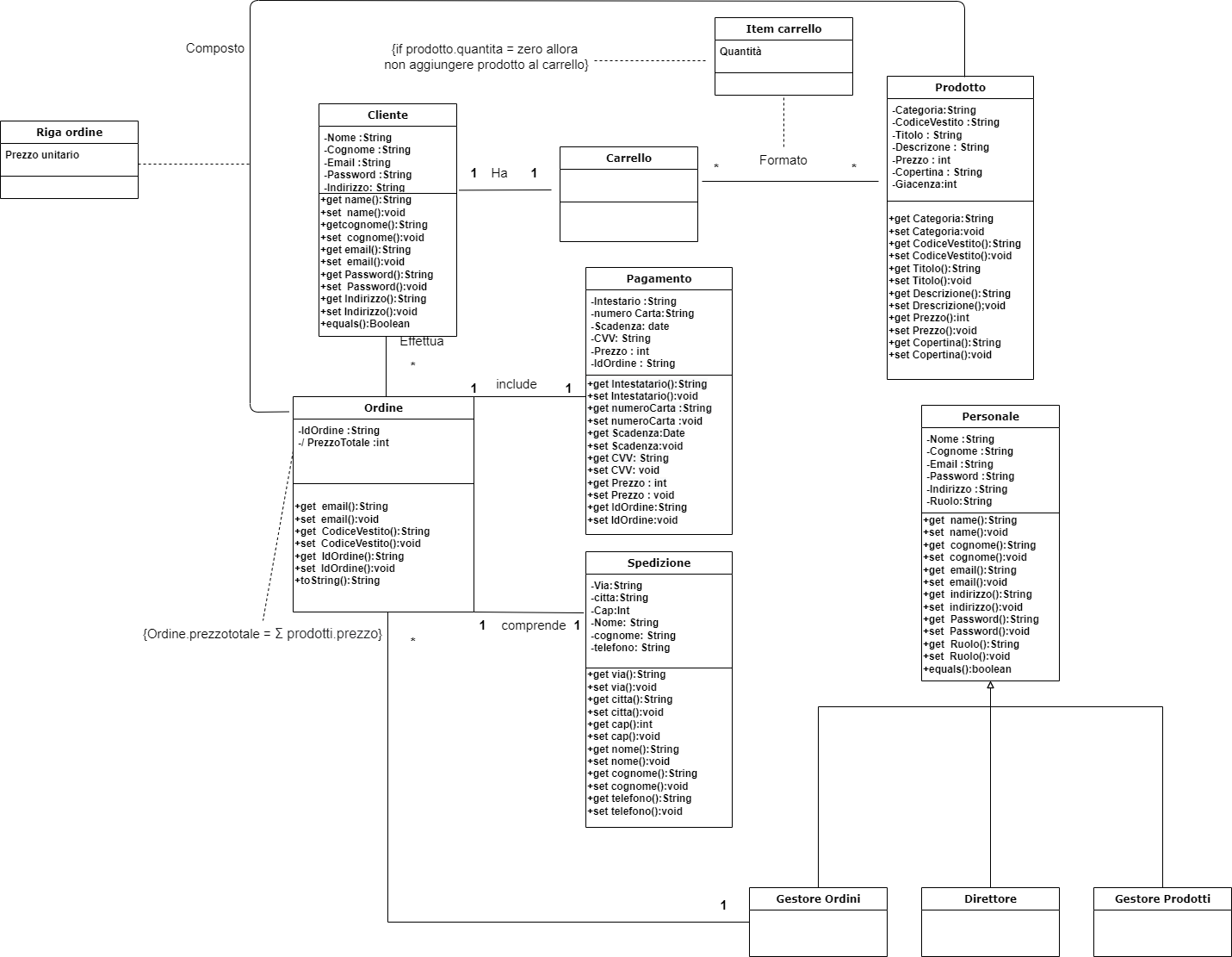
|  |  |
| --- | --- |
| Nome Classe | ShopModelDS |
| Descrizione | Questa classe contiene i metodi che permettono di effettuare le operazioni di visualizzazione di prodotti. |

|  |  |
| --- | --- |
| Nome Classe | ProdottiControl |
| Descrizione | Questa classe è un control (servlet) si occupa di gestire le richieste web per visualizzare un prodotto dal sistema(uomo, donna bambino). Fa uso di ShopModelDS e ShopBean |

|  |  |
| --- | --- |
| Nome Classe | GestoriProdottiControl |
| Descrizione | Questa classe è un control (servlet) si occupa di gestire le richieste web per visualizzare un prodotto dal sistema. Fa uso di ShopModelDS e ShopBean |

**Diagramma delle classi**



****



# Interfacce delle Classi

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

|  |  |
| --- | --- |
| Context: | ShopModelSD: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 codiciVestiti 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 id 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: | **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 id 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 id 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: | **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 non 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 univoco. |

|  |  |
| --- | --- |
| 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 Clieti è andata a buon fine ed il numero delle tuple all’interno della tabella Prodotto 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. |