Università degli Studi di Salerno

Corso di Ingegneria del Software

Nome Progetto: DressMe

Titolo Documento: Object Design Document

Sommario

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
23/12/2021	3.3	Revisione document System Design	Luigi Emanuele Sica Francesco Ciccone Emanuele Riccardi Emanuele Iannone

27/12/2021	4	Stesura Object design document	Luigi Emanuele Sica Francesco Ciccone Emanuele Riccardi
29/1/2021	4.1	Ultima revisione del Object design document	Luigi Emanuele Sica Francesco Ciccone Emanuele Riccardi Emanuele Iannone

1. 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 iniziarecon una lettera maiuscola
  - I nomi dovranno corrispondere alle informazioni e/ofunzionalità

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 iniziarecon 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 è 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

1) Package 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.

2) Package 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

3) 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

4) Package 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.

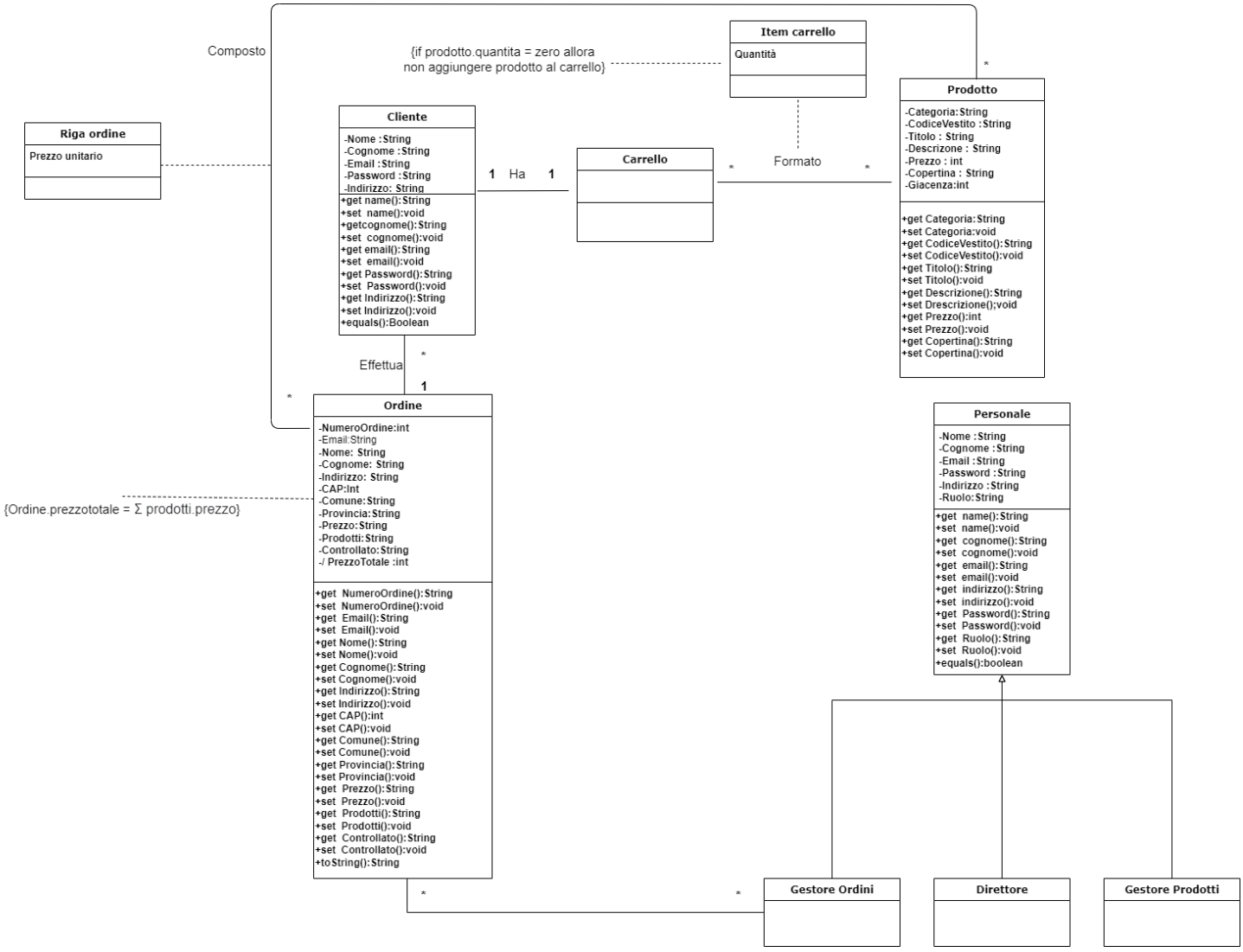
Nome Classe	PagementoBean.java
Descrizione	Questa classe rappresenta le informazioni del pagamento.

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.

3. Diagramma delle classi



4. Interfaccia delle classi

Nome Classe:	**RegistrazioneModelDS**

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 un 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 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 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:	AccountModel:doUpdatePassword(utente:Utente):Boolean
Descrizione	Questo metodo consente di modificare la password 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.
Pre-condizione:	
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

Nome Classe:	**GestioneOrdiniModelDS**

Context:	GestoreOrdiniModelDS:RitonaTuttiGliOrdiniDaControllare(): Collection<OrdineBean>
Descrizione	Questo metodo consente di restituire tutte le tuple all'interno della tabella ordine del database con il parametro controllato=false.
Pre-condizione:	
Post-condizione:	Restituisce tutte le tule della tabella ordine all'interno del dataBase.

Context:	GestoreOrdiniModelDS:doRetrieveByKey(code:String): ordineBean
Descrizione	Questo metodo consente di ottenere l'ordine all'interno della tabella ordine del database con il codice ordine specificata in code.
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 id specificato dalla tabella ordine.

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

Context:	GestoreOrdiniModelDS:doUpdate(ordine: OrdineBean):Boolean
Descrizione	Questo metodo consente di aggiornare un ordine all'interno della tabella ordine restituendo true se è andato a buon fine, false altrimenti.
Pre-condizione:	Ordine deve essere diverso da null e il numeroOrdine dell'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 prodotto passato come parametro, false altrimenti.

Context:	GestoreOrdiniModelDS:ConfermaOdrdine(ordine:OrdineBean) :Boolean
Descrizione	Questo metodo consente di aggiornare un ordine all'interno della tabella ordine settando il parametro controllo a true. restituendo true se è andato a buon fine, false altrimenti.
Pre-condizione:	Ordine deve essere diverso da null e il numeroOrdine dell'ordine deve essere presente all'interno di una tupla nella tabella ordine
Post-condizione:	Vine aggiornato il controllo true nella tupla della tabella ordine e ritorna true.

Nome Classe:	**ProdottoModelDS**

Context:	ShopModelSD:doSave(prodotto:Prodotto)
Descrizione	Questo metodo consente di aggiungere un nuovo prodotto all'interno della tabella vestito 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 vestito.
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:doRetrieveAllBambini(order:String): Collection<Prodotto>
Descrizione	Questo metodo consente di ottenere l'insieme dei prodotti bambini all'interno della tabella Prodotto del database.
Pre-condizione:	
Post-condizione:	Return dell'insieme di tutte le tuple con categoria bambini all'interno della tabella Prodotto

Context:	ProdottoModel:doRetrieveAllUomo(order:String): Collection<Prodotto>
Descrizione	Questo metodo consente di ottenere l'insieme dei prodotti uomo all'interno della tabella Prodotto del database.
Pre-condizione:	
Post-condizione:	Return dell'insieme di tutte le tuple con categoria uomo all'interno della tabella Prodotto

Context:	ProdottoModel:doRetrieveAllDonna(): Collection<Prodotto>
Descrizione	Questo metodo consente di ottenere l'insieme dei prodotti donna all'interno della tabella Prodotto del database.
Pre-condizione:	
Post-condizione:	Return dell'insieme di tutte le tuple con categoria donna 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:	**GestorePersonaleModelDS**

Context:	GestorePersonaleModelDS:stampaTuttoIlPersonale():Collection<PersonaleBean>
Descrizione	Questo metodo consente di ottenere l'insieme del personale all'interno della tabella Personale del database.
Pre-condizione:	
Post-condizione:	Return dell'insieme di tutte le tuple all'interno della tabella Personale

Context:	GestorePersonaleModelDS:doRetrieveByKey(code:String):Personale
Descrizione	Questo metodo consente di ottenere il personale all'interno della tabella Personale del database con email specificata.
Pre-condizione:	code diverso da null e code deve essere presente all'interno di una tupla nella tabella Personale
Post-condizione:	Return del personale con email specificata all'interno della tabella Personale

Context:	GestorePersonaleModelDS:inserisciPersonale(personale:Personale)
Descrizione	Questo metodo consente di aggiungere un nuovo personale all'interno della tabella Personale del database.
Pre-condizione:	Il personale deve essere diverso da null e l'email deve essere diversa da tutte le altre email appartenenti ai personali già presenti nella tabella Personale.
Post-condizione:	Il personale è visibile nella tabella Personale e l' email è univoca.

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

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

Nome Classe:	**GestioneCarrelloModelDS**

Context:	GestioneCarrelloModelDS: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 l'idEmail deve essere diverso da tutti gli altri idEmail appartenenti ai carrelli già presenti nella tabella Carrello.
Post-condizione:	Il carrello è visibile nella tabella Carrello e il idEmail è univoco

Context:	GestioneCarrelloModelDS:trovaCarrello(code:String):Carrello
Descrizione	Questo metodo consente di ottenere il carrello all'interno della tabella Carrello del database con email specificata.
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 email specificata all'interno della tabella Carrello

Context:	GestioneCarrelloModelDS:doDelete(carrello:Carrello):Boolean
Descrizione	Questo metodo consente di rimuovere un prodotto dal 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 l'email del carrello 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.

Context:	GestioneCarrelloModelDS:deleteAll(carrello:Carrello):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 l'email del carrello 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:	**OrdineModelDS**

Context:	OrdineModelDS:SalvaOrdine(ordini: OrdineBean)
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 numerordine deve essere diverso da tutti gli altri numerordine appartenenti agli ordini già presenti nella tabella ordine.
Post-condizione:	L'ordine è visibile nella tabella ordine e il numerordine è univoco

Nome Classe:	**LoginModelDS**

Context:	LoginModelDS: doRetrieveByKeyPersonale (code:String)
Descrizione	Questo metodo consente di ottenere il personale all'interno della tabella Personale del database con email specificata.
Pre-condizione:	Personale deve essere diverso da null e il personale con email cercata deve essere presente al'interno di una tupla nella tabella Personale
Post-condizione:	Return del personale con email specificata dall'interno della tabella Personale

Context:	LoginModelDS: doRetrieveByKey(code:String)
Descrizione	Questo metodo consente di ottenere il cliente all'interno della tabella Cliente del database con email specificata.
Pre-condizione:	cliente deve essere diverso da null e il cliente con email cercata deve essere presente al'interno di una tupla nella tabella cliente.
Post-condizione:	Return del Cliente con email specificato dall'interno della tabella Cliente