*Università degli studi di Salerno*

***Corso di Laurea in Informatica***



***Ingegneria del Software***

***FreshFood***

**Studenti:**

Anna Fulgione 0512105572

Martina Casalnuovo 0512105110

*Anno Accademico: 2019/20*

Sommario

[1 INTRODUZIONE 3](#_Toc29982092)

[1.1 Trade-Off dell’Object Design 3](#_Toc29982093)

[1.1.1 Modularità vs. Efficienza 3](#_Toc29982094)

[1.1.2 Portabilità vs. Efficienza 3](#_Toc29982095)

[1.1.3 Spazio di Memoria vs. Tempo di Risposta 3](#_Toc29982096)

[1.2 Linee guida per Documentazione delle Interfacce 3](#_Toc29982097)

[1.2.1 Package 3](#_Toc29982098)

[1.2.2 Classi 4](#_Toc29982099)

[1.2.3 Metodi 4](#_Toc29982100)

[1.2.4 Campi, Variabili e Costanti 4](#_Toc29982101)

[1.3 Definizioni, Acronimi ed Abbreviazioni 4](#_Toc29982102)

[1.3.1 Definizioni 4](#_Toc29982103)

[1.3 Riferimenti 4](#_Toc29982104)

[1.4 Abbreviazioni 4](#_Toc29982105)

[2. Package 5](#_Toc29982106)

[2.1 Model 5](#_Toc29982107)

[2.2 View 5](#_Toc29982108)

[2.3 Control 5](#_Toc29982109)

[3. Interfacce delle classi 6](#_Toc29982110)

[3.1 Classi 6](#_Toc29982111)

[3.1.1 Prodotto 6](#_Toc29982112)

[3.1.2 Utente 7](#_Toc29982113)

[3.1.3 Ordine 8](#_Toc29982114)

[3.1.4 Database Connector 10](#_Toc29982115)

[3.2 Classi Control 10](#_Toc29982116)

[3.2.1 Product Control 10](#_Toc29982117)

[3.2.2 InsertProduct 12](#_Toc29982118)

[3.2.3 UtenteControl 12](#_Toc29982119)

[3.2.4 OrdineControl 12](#_Toc29982120)

[3.2.5 RegisterAccount 13](#_Toc29982121)

[3.2.6 Login 13](#_Toc29982122)

# INTRODUZIONE

## Trade-Off dell’Object Design

### Modularità vs. Efficienza

Il sistema FreshFood si contraddistingue per una vasta modularità, grazie a cui il sistema gode di grande indipendenza nella gestione delle sue caratteristiche. In parallelo, questa modularità vieni a trovarsi in contrasto con l’indispensabile efficienza nelle elaborazioni lato server. Si è deciso di favorire la modularità, piuttosto che l’efficienza, in quanto essa facilita molto l’implementazione e la manutenzione del software e la sua comprensione sia da parte dei programmatori intenzionati a modificarlo, sia da parte degli utenti.

### Portabilità vs. Efficienza

Si è stabilito di dare molto rilievo alla portabilità del sistema FreshFood. Dunque, si è optato per l’utilizzo del linguaggio Java come linguaggio di programmazione con cui costruire il software, ma ciò penalizza l’efficienza del sistema stesso. Si è giunti, quindi, ad un compromesso indispensabile, a causa dei numerosi supporti forniti dal linguaggio Java nello sviluppo di sistemi avanzati e dei numerosi vantaggi di un sistema con grande efficienza. Affinché il linguaggio non perda le caratteristiche di portabilità, si è deciso di attenersi ad alcune regole base per rendere il codice portabile, qualità di fondamentale importanza per un programma.

### 1.1.3 Spazio di Memoria vs. Tempo di Risposta

Siccome FreshFood sarà un sistema software utilizzato per lo shopping online di frutta e verdura biologici, è parso opportuno ottimizzare il più possibile il tempo di risposta per ogni funzione, piuttosto che ottimizzare lo spazio usato. Tutto ciò causerà, inoltre, una richiesta di maggior spazio di memoria, che non rappresenta un grave problema per la piattaforma, in quanto i dispositivi hardware di memoria in commercio sono dotati di spazio di memoria maggiore di quello stimato e di cui si avrà bisogno.

## Linee guida per Documentazione delle Interfacce

Per l’implementazione del sistema FreshFood sono state definite varie convenzioni.

Tali convenzioni riguardano: i nomi dei pacchetti, classi, metodi, campi e variabili che faranno parte del codice e l’organizzazione dei pacchetti.

### Package

* Tutte le classi che fanno parte del sistema devono appartenere al package FreshFood o ad un suo subpackage. Tale package è, infatti, il principale, quindi risulta essere quello più generale e superiore nella gerarchia dei package del sistema.
* Ogni package deve avere un nome che rispecchi il suo scopo.
* Ogni package deve avere un nome differente rispetto agli altri.
* Il package che contiene le servlet deve avere il nome “Control”
* Il package che contiene i bean deve avere il nome “Model”

### Classi

* Ogni classe deve avere un nome che sia composto da uno o più sostantivi singolari della lingua italiana o inglese, ognuno dei quali deve avere la prima lettera in maiuscolo e le altre in minuscolo; non sono ammessi spazi, segni di punteggiatura o altri caratteri diversi da quelli alfabetici.
* Ogni classe deve avere un nome che rispecchi il suo scopo.
* Ogni classe deve avere un nome differente rispetto a tutte le altre classi, package e metodi.
* Ogni classe deve avere almeno un costruttore.

### Metodi

* Ogni metodo deve avere un nome composto da uno o più termini singolari della lingua italiana o della lingua inglese ognuno dei quali deve avere la prima lettera in maiuscolo e le altre in minuscolo.
* Ogni metodo deve avere un nome che rispecchi il suo scopo.
* Ogni metodo deve avere un nome diverso degli altri metodi della sua stessa classe e del suo stesso package.
* Ogni classe che gestisce entità deve avere metodi che permettono di restituire il valore per ogni campo ed il nome di tali metodi deve essere preceduto dalla particella “get-”.
* Ogni classe che gestisce entità deve avere metodi che permettono di modificare il valore per ogni campo ed il nome di tali metodi deve essere preceduto dalla particella “set-”.

### Campi, Variabili e Costanti

* Ogni campo, ogni variabile deve avere un nome composto da uno o più termini della lingua italiana o inglese. Il primo termine deve essere al singolare, non sono ammessi spazi, segni di punteggiatura o altri caratteri diversi da quelli alfabetici.
* Ogni campo deve avere un nome diverso dagli altri campi della stessa classe.
* Ogni variabile ed ogni costante deve avere un nome diverso da qualsiasi altra variabile o costante presente nello stesso blocco di codice della stessa classe.
* Ogni campo ed ogni variabile deve avere un nome il cui primo termine ha tutte le lettere minuscole, mentre gli altri termini devono avere la prima lettera in maiuscolo e le restanti lettere in minuscolo.

## 1.3 Definizioni, Acronimi ed Abbreviazioni

### 1.3.1 Definizioni

|  |  |
| --- | --- |
| Utente registrato | Colui che utilizza il sistema per acquistare i prodotti di FreshFood |
| Gestore dei prodotti | Colui che interagisce con il sistema al fine di manipolare i prodotti (aggiunge e/o rimuove) |

## Riferimenti

* RAD
* SDD

## Abbreviazioni

* Requirements Analysis Document (RAD)
* System Design Document (SDD)

# 2. Package

## 2.1 Model

Rappresenta il sistema di gestione dei dati. Si occupa della memorizzazione dei dati, come l’interazione con i database. Il package Model contiene i bean.

|  |  |
| --- | --- |
| DriverMaagerConnectionPool.java | Gestisce la connessione con il database |
| Ordine.java | Contiene i le variabili ed i metodi get/set dell’ordine |
| OrdineModel.java | Gestisce gli ordini |
| ProductBean.java | Contiene i le variabili ed i metodi get/set dei prodotti |
| ProductModel.java | Interfaccia dei prodotti |
| ProductModelIDS.java | Gestisce i prodotti |
| Utente.java | Contiene i le variabili ed i metodi get/set degli utenti |
| UtenteModel.java | Gestisce gli utenti |
| Cart.java | Gestisce il carrello |

## 2.2 View

Rappresenta il sistema di interazione diretta con l’utente; rappresenta in tutto e per tutto l’interfacciamento che il sistema ha con tutti gli utenti che possono interagire con il sistema. Il package View contiene le pagine jsp.

|  |  |
| --- | --- |
| Cancella-Prodotto.jsp | Permette di visualizzare la lista dei prodotti che possono essere eliminati |
| Carrello.jsp | Permette di visualizzare il carrello |
| Checkout.jsp | Permette di visualizzare il riepilogo dell’ordine |
| Frutta.jsp | Permette di visualizzare i prodotti della categoria “frutta” nel catalogo |
| Versura.jsp | Permette di visualizzare i prodotti della categoria “verdura” nel catalogo |
| Index.jsp | Permette di visualizzare l’homepage del sito |
| Info-Spedizione.jsp | Permette di inserire i dati per la spedizione |
| Inserisce-Prodotto.jsp | Permette di compilare il form per l’inserimento di un nuovo prodotto |
| Login.jsp | Permette di compilare il form per l’autenticazione |
| Registrazione.jsp | Permette di compilare il form per la registrazione |
| Ordini.jsp | Permette di visualizzare gli ordini effettuati da un determinato utente |
| Result-cerca | Permette di visualizzare i prodotti corrispondenti ad una determinata ricerca |
| Cancella-Utente | Permette di visualizzare la lista degli utenti registrati |

## 2.3 Control

In questo sottosistema sono presenti le componenti che utilizzano ed elaborano i dati; rappresenta il cuore del sistema, un comportamento errato da parte di una componente del sottosistema model può comportare un risultato imprevisto. Il package Control contiene le servlet.

|  |  |
| --- | --- |
| InsertProduct.java | Permette di inserire un nuovo prodotto all’interno del catalogo |
| Login.java | Permette di effettuare l’accesso agli utenti |
| OrdineControl.java | Permette di gestire i dati riguardanti gli ordini |
| ProductControl.java | Permette di gestire i prodotti |
| RegisterAccount.java | Permette agli utenti di registrarsi |
| UtenteControl.java | Permette di gestire i dati dell’utente |
| Cerca.java | Permette di cercare un determinato prodotto |

# 3. Interfacce delle classi

## 3.1 Classi

### 3.1.1 Prodotto

|  |
| --- |
| **ProductBean** |
| - code : int  - nome : String  - tipo : String  - descrizione : String  - stagionalita : String  - quantita : int  - prezzo: double  - img: String  + getCode () : int  + setNome (String unNome) : void  + getNome () : String  + setTipo (String unTipo) : void  + getTipo () : String  + setDescrizione (String unaDescrizione) : void  + getDescrizione () : String  + setStagionalita (String unaStagionalita) : void  + getStagionalita () : String  + setQuantita (int unaQuantita) : void  + getQuantita() : int  + setPrezzo (double unPrezzo) : void  + getPrezzo () : double  + setImg (String unImg) : void  + getImg () : String |

|  |  |
| --- | --- |
| **Campi** | |
| private int code | Codice univoco che identifica un prodotto |
| Private String nome | Nome del prodotto |
| Private String descrizione | Testo che descrive un prodotto |
| Private String stagionalita | Indica la stagionalità del prodotto |
| Private double prezzo | Indica il prezzo del singolo prodotto |
| Private int quantità | Indica la quantità disponibile di un prodotto |
| Private String img | Indica l’url dell’immagine del prodotto |

|  |  |
| --- | --- |
| **Metodi** | |
| Public int getCode() | Restituisce il codice del prodotto |
| Public void setNome(String unNome) | Imposta come nome del prodotto la stringa unNome |
| Public String getNome() | Restituisce il nome del prodotto |
| PublicString getDescrizione() | Restituisce la descrizione del prodotto |
| Public void setDescrizione(String unaDescrizione) | Imposta come descrizione del prodotto la stringa unaDescrizione |
| Public double getPrezzo() | Restituisce il prezzo del prodotto |
| Public void setPrezzo(double unPrezzo) | Imposta come prezzo del prodotto la cifra unPrezzo |
| Public int getQuantita() | Restituisce la quantità di prodotto disponibile |
| Public void setQuantita(int unaQuantita) | Imposta come quantità di prodotto l’intero unaQuantità |
| Public String getStagionalita() | Restituisce la stagionalità di un prodotto |
| Public void setStagionalita(String unaStagionalita) | Imposta come stagionalità di un prodotto la stringa unaStagionalita |
| Public String getImg() | Restituisce l’url dell’immagine di un prodotto |
| Public void setImg(String unImg) | Imposta come url di un’immagine di un prodotto la stringa unImg |

### 3.1.2 Utente

|  |
| --- |
| **Utente** |
| - nome: String  - cognome : String  - username : String  - password : String  -tipo: String |
| + getNome : String  + setNome(String unNome) : void  + getCognome() : String  + setCognome(String unCognome) : void  + getUsername() : String  + setUsername(String unUsername) : void  + getPassword(): String  + setPasswor(String unaPassword): void  + getTipo : String  + setTipo(String unTipo) : void |

|  |  |
| --- | --- |
| **Campi** | |
| Private String nome | Nome dell’utente |
| Private String cognome | Cognome dell’utente |
| Private String username | e-mail dell’utente |
| Private String password | Password scelta dall’utente |
| Private String Tipo | Distingue l’utente dal gestore del catalogo |

|  |  |
| --- | --- |
| **Metodi** | |
| Public String getNome() | Restituisce il nome dell’utente |
| Public void setNome(String unNome) | Imposta il nome dell’utente con la stringa unNome |
| Public String getCognome() | Restituisce il cognome dell’utente |
| Public void setCognome(String unCognome) | Imposta il cognome dell’utente con la stringa unCognome |
| Public String getUsername() | Restituisce l’e-mail dell’utente |
| Public voi setUsername(String unUsername) | Imposta l’e-mail dell’utente con la stringa unUsername |
| Public String getPassword() | Restituisce la password dell’utente |
| Public void setPassword(String unaPassword) | Imposta la password dell’utente con la stringa unaPassword |
| Public String getTipo() | Restituisce il tipo di utente |
| Public void setTipo(String unTipo) | Imposta il tipo di utente con la stringa unTipo |

### 3.1.3 Ordine

|  |
| --- |
| **Ordine** |
| - codiceOrdine : int  - utente: String  - datiSpedizione: String  - infoPagamento: String  - dataOrdine:String  - spesaTotale : String  - prodotti: ArrayList<ProductBean> |
| + getCodiceOrdine() : int  +setCodiceOrdine(int unCodiceOrdine): int  + getUtente () : String  +setUtente (String unUtente) : void  + getDatiSpedizione () : String  + setDatiSpedizione (String deiDatiSpedizione) : void  + getInfoPagamento () : String  + setInfoPagamento (String delleInfoPagamento) : void  + getDataOrdine () : String  + setDataOrdine(String unaDataOrdine) : void  + getSpesaTotale () : double  + setSpesaTotale (double unaSpesaTotale) : void  + getProdotti () : ArrayList<ProductBean>  + setProdotti (ArrayList<ProductBean> prodottiDaInserire) : void |

|  |  |
| --- | --- |
| **Campi** | |
| Private int codiceOrdine | Codice dell’ordine |
| Private String utente | Utente che effettua ordine |
| Private String datiSpedizione | Dati utili per la spedizione dei prodotti |
| Private String datiPagamento | Dati utili per effettuare il pagamento |
| Private String dataOrdine | Data in cui viene effettuato l’ordine |
| Private double spesaTotale | Cifra complessiva da pagare |
| Private ArrayList<ProductBean> prodotti | Insieme dei prodotti da acquistare |

|  |  |
| --- | --- |
| **Metodi** | |
| Public int getCodiceOrdine() | Restituisce il codice dell’ordine |
| Public void setCodiceOrdine(int unCodiceOrdine) | Imposta il codice dell’ordine con l’intero unCodice |
| Public String getUtente() | Restituisce l’utente che ha effettuato l’ordine |
| Public void setUtente(String unUtente) | Imposta l’utente che ha effettuato l’ordine con la stringa unUtente |
| Public String getDatiSpedizione() | Restituisce i dati utili per la spedizione |
| Public void setDatiSpedizione(String deiDatiSpedizione) | Imposta i dati utili per la spedizione con la stringa deiDatiSpedizione |
| Public String getDatiPagamento() | Restituisce i dati utili per il pagamento |
| Public voud setDatiPagamento(String deiDatiPagamento) | Imposta i dati utili per il pagamento con deiDatiPagamento |
| Public String getDataOrdine() | Restituisce la data dell’ordine |
| Public void setDataOrdine(String unaDataOrdine) | Imposta la data dell’ordine con la stringa unaDataOrdine |
| Public double getSpesaTotale() | Restituisce la cifra complessiva da pagare |
| Public void setSpesaTotale(double unaSpesaTotale) | Imposta la cifra complessiva da pagare con una cifra unaSpesaTotale |
| Public ArrayList<ProductBean> getProdottiOrdine() | Restituisce la lista dei prodotti acquistati |
| Public void setProdottiOrdine(ArrayList<ProductBean> prodottiDaInserire) | Imposta la lista dei prodotti acquistati con prodottiDaInserire |

### 3.1.4 Database Connector

|  |
| --- |
| **DriverManagerConnectionPool** |
| **-** ip : String  **-** port : String  **-** db: String  **-** username : String  - password : String |
| - createDBConnector()  + getConnection() : Connection  +releaseConnection(Connection connection): Connection |

|  |  |
| --- | --- |
| **Campi** | |
| String ip | LocalHost |
| String port | Porta alla quale è connesso il database |
| String db | Nome del database |
| String username | Username per accedere al database |
| String password | Password per accedere al database |

|  |  |
| --- | --- |
| **Metodi** | |
| private static synchronized Connection createDBConnection() | Crea una nuova connessione al database |
| public static synchronized Connection getConnection() | Restituisce la connessione al database |
| public static synchronized void releaseConnection(Connection connection) | Aggiunge una connessione |

## 3.2 Classi Control

### 3.2.1 Product Control

|  |  |
| --- | --- |
| **Classe** | **Product Control** |
| **Descrizione** | Classe che si occupa della gestione dei prodotti del sistema |
| **Pre-condizione** | **Context:** doSave(ProductBean product)  **Pre:** product != null  **Context:** doDelete(int code)  **Pre:** code != null  **Context:** doRetrieveByKey(int code)  **Pre:** code!= null  **Context:** cercaPerTipo(String tipo)  **Pre:** tipo!= null && !tipo.equals("")  **Context:** doRetrieveAll(String order)  **Pre:** order != null && !order.equals("")  **Context:** doRetrieveAll2()  **Pre:**  **Context:** doRetrieveByProd(String tipo)  **Pre:** tipo != null  **Context:** getNewCode()  **Pre:** code > newCode |
| **Post-Condizione** | **Context:** doSave(ProductBean product)  **Post:**  void  **Context:** doDelete(int code)  **Post:** true  **Context:** doRetrieveByKey(int code)  **Pre:** ProductBean  **Context:** cercaPerTipo(String tipo)  **Post:** ArrayList<ProductBean>  **Context:** doRetrieveAll(String order)  **Post:** Collection<ProductBean>  **Context:** doRetrieveAll2()  **Post:** ArrayList<ProductBean>  **Context:** doRetrieveByProd(String tipo)  **Post:** Collection<ProductBean>  **Context:** getNewCode()  **Post:** newCode |
| **Invarianti** |  |

### 3.2.2 InsertProduct

|  |  |
| --- | --- |
| **Classe** | **InsertProduct** |
| **Descrizione** | Classe che si occupa dell’inserimento di nuovi prodotti |
| **Pre-condizione** | **Context:** AddProduct(ProductBean bean)  **Pre:** !bean.getNome().equals("VUOTO") |
| **Post-Condizione** | **Context:** AddProduct(ProductBean bean)  **Post:** void |
| **Invarianti** |  |

### 3.2.3 UtenteControl

|  |  |
| --- | --- |
| **Classe** | **UtenteControl** |
| **Descrizione** | Classe che si occupa della gestione degli account del sistema |
| **Pre-condizione** | **Context:** doDeleteAccount(String username)  **Pre:** username!= null  **Context:** showAccount()  **Pre:** account != null  **Context:** Login(Utente utente)  **Pre:** username != null && password != null |
| **Post-Condizione** | **Context:** doDeleteAccount(String username)  **Post:** void  **Context:** showAccount()  **Post:** Collection<Utente>  **Context:** Login(Utente utente)  **Post:** temp(intero che vale 1 se gestore del catalogo; 2 se cliente) |
| **Invarianti** |  |

### 3.2.4 OrdineControl

|  |  |
| --- | --- |
| **Classe** | **OrdineControl** |
| **Descrizione** | Classe che si occupa della gestione degli ordini del sistema |
| **Pre-condizione** | **Context:** mostraOrdiniEffettuati(String username )  **Pre:** username != null  **Context:** piazzaOrdine(String username,Cart cart,String infoSpedizione,String infoPagamento)  **Pre:**  cart != null  **Context:** caricaCarrelloDB(Cart cart,String nomeUtente)  **Pre:**  cart != null && nomeUtente != null  **Context:** aggiornaCarrelloDB(Cart cart,String nomeUtente)  **Pre:** nomeutente != null  **Context:** getOrdineByCodice(int codiceOrdine)  **Pre:** codiceOrdine != null |
| **Post-Condizione** | **Context:** mostraOrdiniEffettuati(String username )  **Post:** Collection<Ordine>  **Context:** piazzaOrdine(String username,Cart cart,String infoSpedizione,String infoPagamento)  **Post:** void  **Context:** caricaCarrelloDB(Cart cart,String nomeUtente)  **Post:** Cart  **Context:** aggiornaCarrelloDB(Cart cart,String nomeUtente)  **Post:** void  **Context:** getOrdineByCodice(int codiceOrdine)  **Post:** mioOrdine |
| **Invarianti** |  |

### 3.2.5 RegisterAccount

|  |  |
| --- | --- |
| **Classe** | **RegisterAccount** |
| **Descrizione** | Classe che si occupa della registrazione dei nuovi account del sistema |
| **Pre-condizione** | **Context:** doSaveNewAccount(Utente utente)  **Pre:** username.equals("") || password.equals("") || username == null || password == null  **Context:** checkUsername(String username)  **Pre:** username != null |
| **Post-Condizione** | **Context:** doSaveNewAccount(Utente utente)  **Post:**  **Context:** checkUsername(String username)  **Post:** true |
| **Invarianti** |  |

### 

### 3.2.6 Login

|  |  |
| --- | --- |
| **Classe** | **Login** |
| **Descrizione** | Classe che si occupa della gestione dell’autenticazione degli account del sistema |
| **Pre-condizione** | **Context:**  **Pre:** |
| **Post-Condizione** |  |
| **Invarianti** |  |

### 3.2.7 Cerca

|  |  |
| --- | --- |
| **Classe** | **Cerca** |
| **Descrizione** | Classe che si occupa della gestione dell’autenticazione degli account del sistema |
| **Pre-condizione** | **Context:**  **Pre:** |
| **Post-Condizione** |  |
| **Invarianti** |  |

## 3.3 Object Pool Pattern

Ha lo scopo di utilizzare un set di oggetti inizializzati, pronti ad essere usati (un “pool”) anziché allocarli e distruggerli su richiesta. Un client del pool richiederà un oggetto da esso ed eseguirà operazioni sull’oggetto restituito. Quando il client ha terminato, restituisce l’oggetto al pool invece di distruggerlo. Tutto ciò può essere fatto manualmente oppure automaticamente.

Immagine che contiene screenshot, computer, portatile, monitor

Descrizione generata automaticamente