| **Università degli Studi di Salerno Corso di Ingegneria del Software** |
| --- |

**Avenue 814  
Object Design Document**

**Versione 3.0**

****

Data: 16/12/2021

**Coordinatore del progetto:**

| **Nome** | **Matricola** |
| --- | --- |
|  |  |
|  |  |

**Partecipanti:**

| **Nome** | **Matricola** |
| --- | --- |
| Pisu Antonio | 0512108616 |
| Renzulli Marta | 0512107377 |
| Ricco Maurizio | 0512108622 |
|  |  |
|  |  |
|  |  |

| **Scritto da:** | PISU A.,RENZULLI M.,RICCO M. |
| --- | --- |

**Revision History**

| **Data** | **Versione** | **Descrizione** | **Autore** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Indice**

1. INTRODUZIONE 5

1.1. Trade-off

1.2. Componenti off-the-shelf

1.3. Linee guida per la documentazione

2. DESIGN PATTERN 11

2.1. Singleton

3. DEFINIZIONI,ACRONIMI,ABBREVIAZIONI 12

4. PACKAGE 14

4.1. View

4.2. Model

4.3. Control

5. INTERFACCE DELLE CLASSI 18

1. **INTRODUZIONE**

L’**Object Design Document** consente di specificare in modo dettagliato le decisioni prese in fase di analisi e di design; verranno specificati i principali trade-offs, descritte le componenti off-the-shelfs utilizzate dal sistema, le linee guida per la documentazione delle interfacce e l’individuazione dei Design Patterns. Inoltre, verranno definiti i packages, le interfacce delle classi e i diagrammi delle classi che riguardano importanti decisioni implementative.

### 1.1 Trade-off

1.1.1 *Comprensibilità vs costi*

Si preferisce aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte nel progetto o le persone coinvolte che non hanno lavorato a quella sezione di progetto. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorare la comprensibilità agevola il mantenimento e anche il processo di modifica.

1.1.2 *Tempo di risposta e Affidabilità*

Il sistema sarà implementato in modo tale da preferire l’affidabilità al tempo di risposta,per garantire un controllo più accurato dei dati in input a discapito del tempo di risposta del sistema.

1.1.3 *Manutenibilità vs efficienza*

Si preferisce avere una maggiore manutenibilità del sistema facendo in modo che ogni sottosistema non acceda direttamente al DB, ma che l’accesso ai dati venga gestito da un sottosistema intermedio a discapito dell’efficienza e delle prestazioni generali.

## 

## 

## 1.2 Componenti off-the-shelf

Per il progetto software che si vuole realizzare non facciamo uso di componenti***off-the-shelf****,* che sono componenti software disponibili sul mercato per facilitare la creazione del software.

Il comparto grafico è realizzato interamente da noi, con [HTML](https://it.wikipedia.org/wiki/HTML) e [CSS](https://it.wikipedia.org/wiki/CSS), sia per la [tipografia](https://it.wikipedia.org/wiki/Tipografia), che per le varie componenti dell'interfaccia, come [moduli](https://it.wikipedia.org/wiki/Form), bottoni e navigazione, e altri componenti dell'interfaccia.

## 1.3 Linee guida per la documentazione dell’interfaccia

Nell’implementazione del sistema, i programmatori dovranno attenersi alle linee guida di seguito definite.

### 1.3.1 Classi e interfacce Java

Nella scrittura di codice per le classi Java ci si atterrà allo standard Google Java nella sua interezza. Tale standard fornisce delle regole da seguire ad esempio ogni metodo ed ogni file possono non essere preceduti da un commento. Potranno esserci, inoltre, commenti e giustificazioni in merito a particolari decisioni o calcoli. La convenzione utilizzata dai team member per quanto riguarda i nomi delle variabili, è la nota lowerCamelCase, che consiste nello scrivere parole composte o frasi unendo tutte le parole tra loro. Quando si codificano classi e interfacce Java, si dovrebbero rispettare le seguenti regole di formattazione:

1. La parentesi graffa aperta “{“si trova alla fine della stessa linea dell’istruzione di dichiarazione.

2. La parentesi graffa chiusa “}” inizia su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell’interfaccia.

Return () -> {

while(condition()) {

method();

} };

Return new MyClass(){

@Override public void method(){

if(condition()){

try{

something();

} catch(ProblemException e){

recover();

}

} else if (otherCondition()){

somethingElse();

} else {

lastThing();

}

}

};

Nel caso di istruzioni semplici, ogni linea deve contenere al massimo una sola istruzione. Mentre nel caso di istruzioni composte vanno rispettate le seguenti regole:

1. Le istruzioni racchiuse all’interno di un blocco (esempio: for), devono essere indentate di

un’unità all’interno dell’istruzione composta.

2. La parentesi di apertura del blocco deve trovarsi alla fine della riga dell’istruzione composta.

3. La parentesi di chiusura del blocco deve trovarsi allo stesso livello di indentazione

dell’istruzione composta

I nomi di classe devono essere sostantivi, con lettere minuscole e, sia la prima lettera del nome della classe sia la prima lettera di ogni parola interna, deve essere maiuscola. I nomi delle classi dovrebbero essere semplici, descrittivi e che rispettino il dominio applicativo. Non dovrebbero essere usati underscore per legare nomi. I nomi dei metodi iniziano con una lettera minuscola (non sono consentiti caratteri speciali) e seguono la notazione a cammello. Dovranno essere semplici, descrittivi e che rispettino il dominio applicativo.

Organizzazione dei file

- Ogni file deve essere sviluppato e diviso in base alla categoria di appartenenza, ovvero deve essere correlato ad un’unica funzionalità che persegue.

- La convenzione per quanto riguarda i nomi dei file, delle operazioni e delle variabili è quella di avere nomi evocativi.

- Organizzare in una cartella i file delle librerie usate e le altre risorse scaricate necessarie per lo sviluppo del progetto.

### 1.3.2 Pagine lato Server (JSP)

Le pagine JSP devono, quando eseguite, produrre un documento conforme allo standard HTML 5. Il codice Java delle pagine deve aderire alle convenzioni per la codifica in Java, con le seguenti puntualizzazioni:

1. Il tag di apertura (<%) si trova all'inizio di una riga;

2. Il tag di chiusura (%>) si trova all'inizio di una riga;

3. È possibile evitare le due regole precedenti, se il corpo del codice Java consiste in una singola istruzione (<%=%>):

<!-Accettabile ->

<% for(String par: paragraphs) {%>

<p class=’item’><% out.print(par); %></p>

<%}%>

<!-Non Accettabile->

<p class=’item’><% List<String> paragraphs = getParagraphs();

out.print(paragraphs.get(i++));%></p>

Per le Servlet è necessario far terminare il nome della classe con il suffisso Servlet.

### 1.3.3 Pagine HTML

Le pagine HTML, sia in forma statica che dinamica, devono essere conformi allo standard HTML 5. Inoltre, il codice HTML statico deve utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

1. Un'indentazione consiste in una tabulazione;

2. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;

3. Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;

4. I tag di commento devono seguire le stesse regole che si applicano ai tag normali.

*Es.*

<!-Accettabile->

<div>

<span>

<ul>

<li>

Uno

</li>

<li>

Due

</li>

</ul>

</span>

</div>

<!-Non Accettabile ->

<div><span>

<nl>

<li>Uno</li>

<li>

Due

</li>

</nl></span>

</div>

### 1.3.5 Fogli di stile CSS

I fogli di stile (CSS) devono seguire le seguenti convenzioni:

Ogni regola CSS deve essere formattata come segue:

1. I selettori della regola si trovano a livello 0 di indentazione, uno per riga;

2. L'ultimo selettore della regola è seguito da parentesi graffa aperta ({);

3. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;

4. La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

## 1.3.6 Database SQL

I nomi delle tabelle devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere maiuscole;

2. Se il nome è costituito da più parole, è previsto l’uso di underscore (\_);

3. Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

I nomi dei campi devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere minuscole e, sia la prima lettera del nome del campo, sia la prima lettera di ogni parola interna, deve essere maiuscola;

2. Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

**2. Design Pattern**

### 2.1 Singleton

Abbiamo progettato una singola classe (DBConnection) che consentisse di effettuare tutte le operazioni con il database. Per evitare la perdita di efficienza dovuta alla creazione di più istanze di questa classe si è deciso di renderla un Singleton



## 3. Definizioni, acronimi e abbreviazioni

**DBMS:** DataBase Management System.

**Off-The-Shelf:** Servizi esterni di cui viene fatto utilizzo da terzi.

**Framework:** Software di supporto allo sviluppo web.

**HTML:** Linguaggio di mark-up per pagine web.

**CSS:** Linguaggio usato per definire la formattazione di pagine web.

**JavaScript:**  [Linguaggio di scripting](https://it.wikipedia.org/wiki/Linguaggio_di_scripting) [orientato agli oggetti](https://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti) e agli [eventi](https://it.wikipedia.org/wiki/Programmazione_orientata_agli_eventi), comunemente utilizzato nella [programmazione Web](https://it.wikipedia.org/wiki/Programmazione_Web) [lato client](https://it.wikipedia.org/wiki/Lato_client) per la creazione, in [siti web](https://it.wikipedia.org/wiki/Sito_web) e [applicazioni web](https://it.wikipedia.org/wiki/Applicazione_web), di effetti dinamici

[interattivi](https://it.wikipedia.org/wiki/Interattivit%C3%A0) tramite [funzioni](https://it.wikipedia.org/wiki/Funzione_(informatica)) di [script](https://it.wikipedia.org/wiki/Script) invocate da *eventi* innescati a loro volta in vari modi dall'utente sulla [pagina web](https://it.wikipedia.org/wiki/Pagina_web) in uso

**Bootstrap:** Framework che contiene librerie utili per lo sviluppo responsive di pagine web.

**Camel Notation:** Consiste nello scrivere più parole insieme delimitando la fine e l’inizio di una nuova parola con una lettera maiuscola.

**Hard Coding:** Codifica fissa.

**4.** **Packages**

**4.1 View**

**Il package View è formato a sua volta da quattro packages, PaginaAutenticazione, PaginaContatti, PaginaHome; inoltre sono presenti le pagine datipersonali, login, mieiordini, register, home.**

**Il package Cliente viene implementato con le pagine:**

* **home.jsp:** mostra la pagina home che permette di navigare nel sito web;
* **login.jsp:** pagina che permette ad un utente di accedere al sito web;
* **datipersonali.jsp:** pagina che mostra i dati personali del cliente;
* **mieiordini.jsp:** permette di visualizzare una pagina contenente gli ordini effettuati;
* **carrello.jsp:** mostra il carrello con gli articoli selezionati;
* **shop.jsp:** elenco dei prodotti acquistabili .

**Il package Amministratore viene implementato con le pagine:**

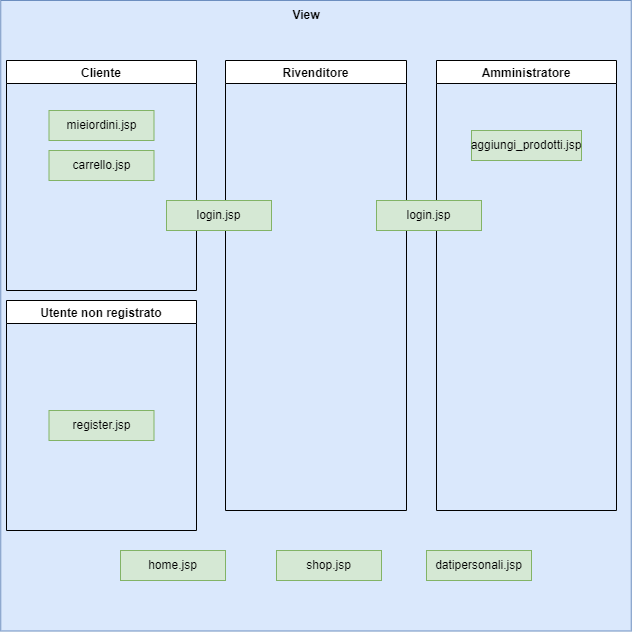
* **home.jsp:** mostra la pagina home che permette di navigare nel sito web;
* **login.jsp:** pagina che permette ad un utente di accedere al sito web;
* **datipersonali.jsp:** pagina che mostra i dati personali del cliente;
* **aggiungi\_prodotti.jsp:** pagina di aggiunta dei prodotti selezionati da depositare nel carrello;
* **shop.jsp:**elenco dei prodotti acquistabili.

**Il package Utente non registrato viene implementato con le pagine:**

* **home.jsp:** mostra la pagina home che permette di navigare nel sito web;
* **datipersonali.jsp:** pagina che mostra i dati personali del cliente;
* **register.jsp:** pagina che permette di effettuare una registrazione al sito web;
* **shop.jsp:**elenco dei prodotti acquistabili.

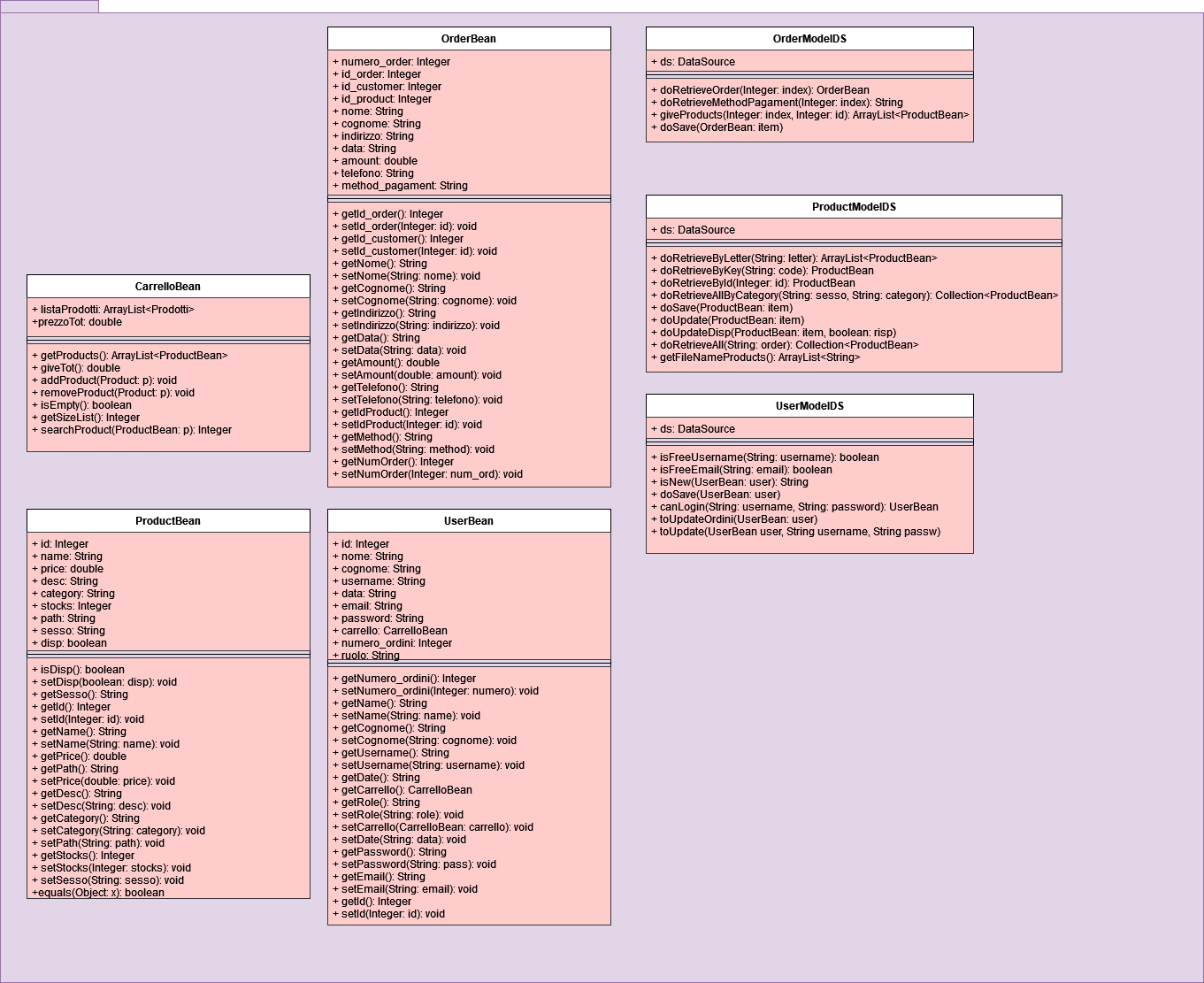
**Il package Rivenditore viene implementato con le pagine:**

* **home.jsp:** mostra la pagina home che permette di navigare nel sito web;
* **login.jsp:** pagina che permette ad un utente di accedere al sito web;
* **datipersonali.jsp:** pagina che mostra i dati personali del cliente;
* **shop.jsp:**elenco dei prodotti acquistabili.

****

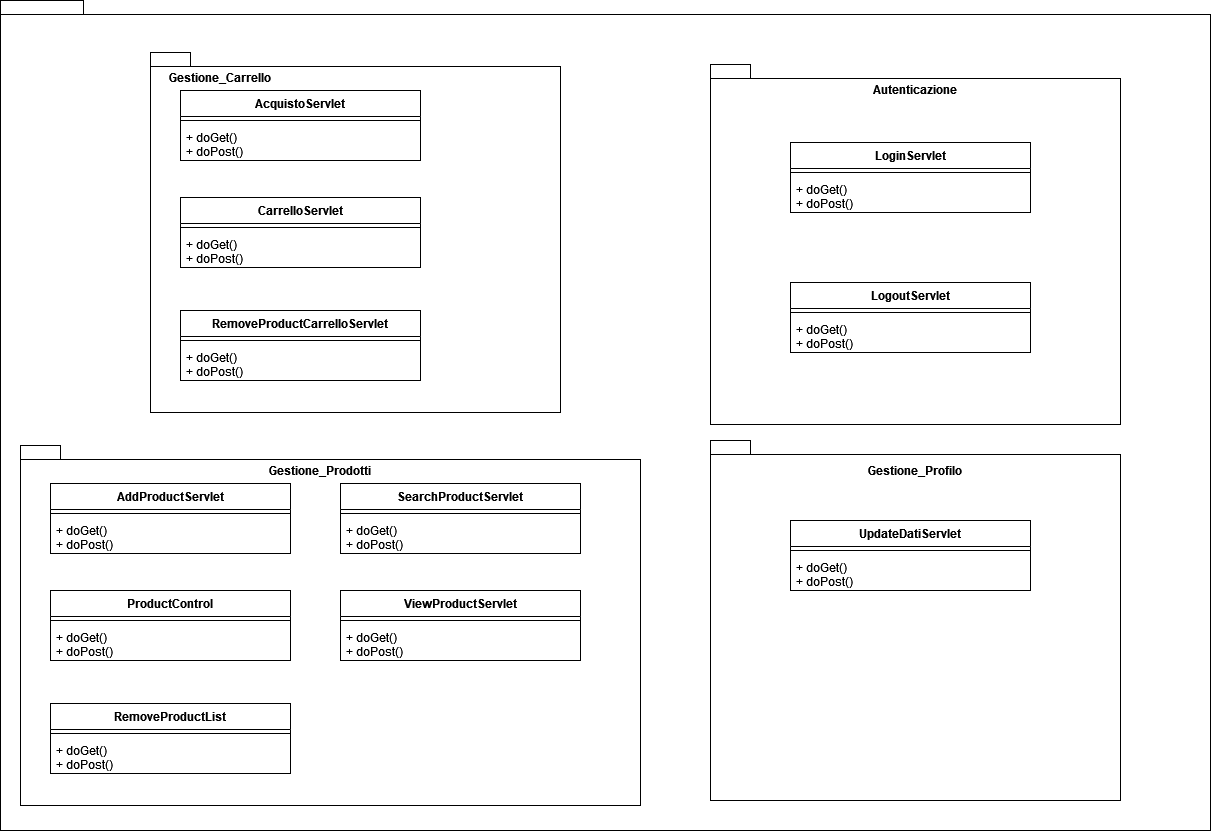
**4.2 Model**

Il package Model contiene tutte le classi dedite alla gestione dei dati persistenti. Esso si occupa di fare da tramite tra l’applicazione e il database sottostante. Ogni classe contenuta all’interno di questo pacchetto fornisce i metodi per accedere ai dati utili all’applicazione. Le classi contenute all’interno di questo package sono:

****

**4.3 Control**

Il package Control riceve, tramite il pacchetto View, i comandi dell’utente.

****

**5. Interfacce delle classi**

| **Nome Classe** | **Gestione\_Carrello** |
| --- | --- |
| **Descrizione** | **Rappresenta la gestione delle funzionalità relative al carrello.** |
| **Pre-condizione** | ***context* Gestione\_Carrello:: viewCart();**  ***pre:* CarrelloBean != null**    ***context* Gestione\_Carrello:: addProductToCart(ProductBean);**  ***pre:* CarrelloBean != null && ProductBean != null &&**  **ProductBean.isDisp()**    ***context* Gestione\_Carrello:: removeProductByCart(ProductBean);**  ***pre:* CarrelloBean != null && CarrelloBean.size() > 0 &&**  **ProductBean != null**    ***context* Gestione\_Carrello:: buyCart();**  ***pre:* CarrelloBean != null && CarrelloBean.size() > 0** |
| **Post-condizione** |  |
| **Invarianti** |  |

| **Nome Classe** | **Gestione\_Prodotti** |
| --- | --- |
| **Descrizione** | **Rappresenta la gestione delle funzionalità relative ai prodotti.** |
| **Pre-condizione** | ***context* Gestione\_Prodotti:: viewSingleProduct(ProductBean);**  ***pre:* ProductBean != null && ProductBean.isDisp()**    ***context* Gestione\_Carrello:: addProduct (ProductBean);**  ***pre:*  ProductBean != null && isValid()**    ***context* Gestione\_Carrello:: removeProduct(ProductBean, UserBean);**  ***pre:* ProductBean.isDisp() && ProductBean != null && ( ProductBean.getIdProprietario() == UserBean.getId()) || (if UserBean.getRole == ‘Amministratore’ then: TRUE)** |
| **Post-condizione** | ***context* Gestione\_Carrello:: removeProduct(ProductBean);**  ***post:* !ProductBean.isDisp()** |
| **Invarianti** |  |

| **Nome Classe** | **Gestione\_Profilo** |
| --- | --- |
| **Descrizione** | **Rappresenta la gestione delle funzionalità relative ai profili.** |
| **Pre-condizione** | ***context* Gestione\_Prodotti:: viewProfilo(UserBean);**  ***pre:* UserBean != null**    ***context* Gestione\_Carrello:: removeProfilo(UserBean, UserBean1);**  ***pre:*  UserBean != null && UserBean1.getRole() == ‘Amministratore’**    ***context* Gestione\_Carrello:: modifyProfilo(UserBean);**  ***pre:*  UserBean != null && isValid()**      ***context* Gestione\_Carrello:: changeRole(UserBean);**  ***pre:* UserBean != null** |
| **Post-condizione** |  |
| **Invarianti** |  |