ODD
Object Design Document

Versione	1.2
Data	22/12/2019
Destinatario	Prof. Andrea De Lucia
Presentato da	Riccardo Martiniello Davide Cresci Alessio Rizzolo Giuseppe Caiazzo

Revision History

data	versione	descrizione	autori
20/12/2019	1.0	Introduzione,	Tutti i membri
20/12/2019	1.1	Package	Tutti i membri
21/12/2019	1.2	Class interface	Tutti i membri

Sommario

1. Introduzione

Il presente documento illustra l'Object Design per l'E-Commerce Dress-Store. Questo documento, usato come supporto dell'implementazione, ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutti i servizi individuati nelle fasi precedenti. In particolare, definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre, nei paragrafi successivi sono specificati i trade-off, le linee guida e i design pattern per l'implementazione.

1.1 Object Design Trade-off

Nella fase dell'Object Design sorgono diversi compromessi di progettazione ed è necessario stabilire quali punti rispettare e quali rendere opzionali.

Per quanto riguarda la realizzazione dell'E-Commerce Dress-Store, sono stati individuati i seguenti trade-off:

- **Comprensibilità vs Tempo**: il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione.
- **Interfaccia vs Easy-use**: l'interfaccia dovrà presentarsi semplice ed intuitiva per garantire il successo del sistema. Infatti, l'effettivo utilizzo del sistema sarà determinato dall'utilità e le semplificazioni che saranno concretamente apportate agli utenti Potrà essere richiesta una penalizzazione in termini di interfaccia per permettere il raggiungimento delle varie funzionalità in pochi passi.
- **Efficienza vs Memoria:** dal momento che il sistema coinvolge una grande quantità di dati, si è scelto di preferire la memoria a discapito della efficienza, eliminando ridondanze dalla base di dati.
- **Sicurezza vs Prestazioni**: nel nostro sistema ogni richiesta del client viene validata attraverso l'uso di sessioni ed un controllo del livello di utenza. Inoltre, si intende utilizzare una libreria per la cifratura della password, in modo tale da poterla inserire nell'URL inviato per e-mail, permettendo quindi l'inserimento dell'utente nel database solo dopo che siano stati validati i dati

Questo trade-off resta comunque piuttosto bilanciato: le caratteristiche descritte potrebbero far aumentare il tempo di risposta del sistema, tuttavia tale aumento è completamente trascurabile.

1.2 Linee guida per l'implementazione

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

1.2.1 Naming Convention

I nomi utilizzati per la rappresentazione dei concetti principali, delle funzionalità e delle componenti generiche del sistema devono rispettare le seguenti condizioni

1. I **nomi** devono essere:
   - Appartenenti alla lingua italiana, se possibile
   - Di lunghezza medio-breve
   - Non sostituiti da acronimi o abbreviazioni di alcun genere

2. Le **variabili** devono:
   - Rispettare la Camel Notation
   - Iniziare con la lettera minuscola
   - Tutte le variabili dovrebbero essere private
   - Tutte le variabili che non vengono mai modificate dovrebbero essere dichiarate come costanti (final).

È possibile far iniziare le variabili statiche con "_" cosi da contraddistinguerle dal resto delle variabili presenti all'interno del codice. Questa distinzione è utile per evitare errori sull'uso improprio di tali forme di variabili.

3. Le **classi** e le **interfacce** devono:
   - Rispettare la Camel Notation
   - Iniziare con la lettera maiuscola
   - Concludersi con il tipo di elemento che rappresentano:

   Esempio: public class ProvaBean {} /* Per una classe di tipo Bean public class ProvaClass {} /* Per una classe generica */
   public interface ProvaInterface /* Per un'interfaccia */

4. Le **variabili costanti** devono:
   - Utilizzare solamente caratteri maiuscoli
   - Separare i vari nomi che la compongono da "_"
   - Evitare di iniziare con "_"
   - Essere di lunghezza medio-breve

5. I **pacchetti** devono:
   - Contenere solamente caratteri minuscoli
   - Contenere solamente caratteri a-z
   - Di semantica affine con gli elementi di cui è composto

6. I **metodi** devono:
   - Iniziare con lettere minuscole

- Evitare di iniziare con GET o SET se non si trattano di metodi setting o getting della classe corrispondente.
- Rispettare la Camel Notation

7. Le pagine **JSP** devono, quando eseguite, produrre, in ogni circostanza, un documento conforme allo standard HTML versione 5. Le parti Java delle pagine devono aderire alle convenzioni per la codifica in Java, con le seguenti puntualizzazioni:
   - Il tag di apertura (<%) è seguito immediatamente dalla fine della riga
   - Il tag di chiusura (%>) si trova all'inizio di una riga
   - Contengono solamente caratteri minuscoli
   - Sono di lunghezza medio-breve
   - Hanno nomi composte da una singola parola in inglese o in italiano

1.2.2 Basi di dati

Le tabelle della base di dati dovrebbero rispettare la terza forma normale di Codd (3NF). Ove ciò non si verifichi, tale fatto deve essere riportato e motivato nella documentazione della base di dati. Le scelte di gestione nel trattamento dell'integrità referenziale devono essere riportate e motivate nella documentazione della base di dati.

1.2.3 Pagine HTML

Le pagine HTML, statiche e dinamiche, devono essere totalmente aderenti allo standard HTML versione 5. Inoltre, il codice HTML statico deve utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

- Un'indentazione consiste in una tabulazione
- Ogni tag deve avere un'indentazione maggiore del tag che lo contiene
- Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura
- I tag di commento devono seguire le stesse regole che si applicano ai tag normali

```
<html>
        <head>
        </head>
    <body>
                <div>
                        Contenuto DIV
                </div>
     </body>
</html>
```

1.2.4 Gestione codici Servlet

Le Servlet devono seguire il seguente formato:

```java
package src;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *  Descrizione di cosa fa la Servlet
 */

@WebServlet("/ExampleServlet")
public class ExampleServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;


    public ExampleServlet() {
    }

    /**
     * Descrizione di cosa fa il doGet
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        /* Content of GET */
    }

    /**
     * Descrizione di cosa fa il doPost
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        /* Content of POST */
    }

}
```

1.2.5 Gestione JavaScript

Gli script JavaScript devono essere collocati in file dedicati.

Il codice JavaScript deve seguire le stesse convenzioni adottate per il linguaggio Java.

Gli scipt devono attenersi al seguente formato:

```
/**
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti del costruttore (@param)
*
* Metodo nomeMetodo1
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti (@param)
* Specifica dei risultati (@return)
*
* Metodo nomeMetodo2
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti (@param)
* Specifica dei risultati (@return)
*
* ...
*/
function ClasseX(a, b, c) {
```

1.2.6 CSS

Tutti gli stili devono essere collocati in file separati.

Ogni regola CSS deve essere formattata come segue:

1. L'ultimo selettore della regola è seguito da parentesi graffa aperta "{";
2. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
3. La regola termina con la parentesi graffa chiusa "}"

1.3 Riferimenti

Il presente documento, fa riferimento ai documenti precedentemente rilasciati:

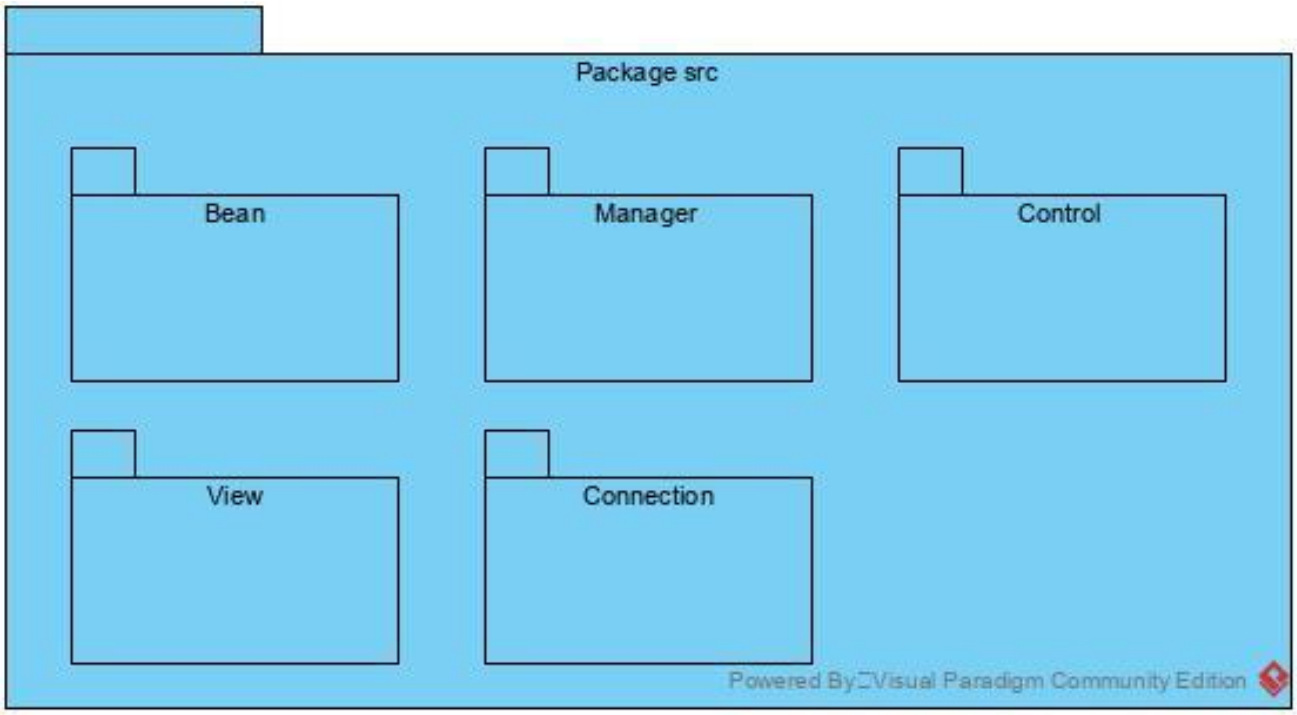
- RAD;
- SDD.

Inoltre, si fa riferimento anche al libro di testo. "B.Bruegge, A.H. Dutoit, Object Oriented Software Engineering – Using UML, Patterns and Java, Prentice Hall."

2. Package

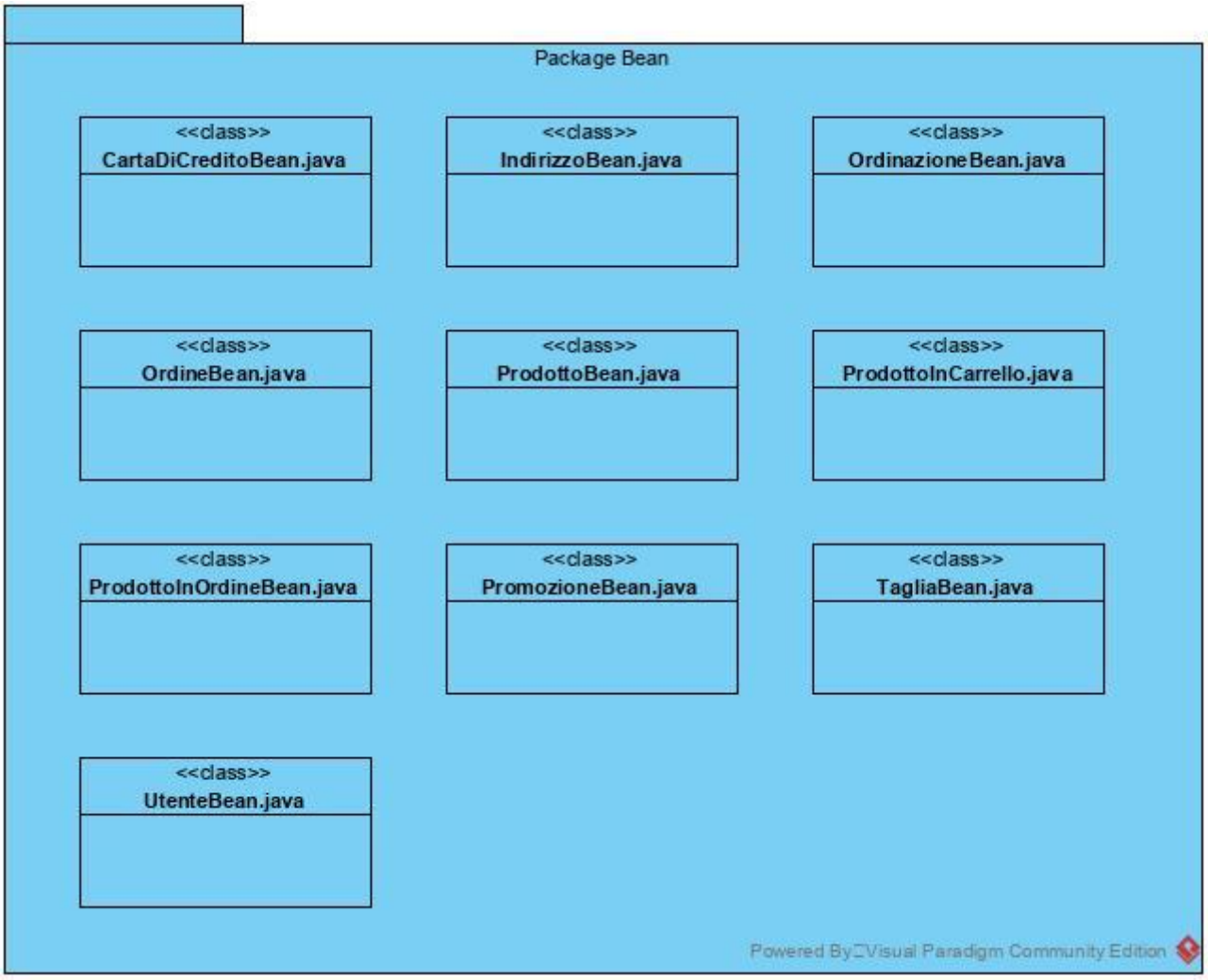
2.1 Package src

Qui di seguito verrà illustrato lo schema dei package presenti all'interno del progetto. Questi package verranno divisi secondo determinate operazioni che eseguiranno all'interno del sistema. Di seguito è riprodotto il package src che raggruppa tutti i sottopackage che verranno utilizzati nella fase di implementazione:



2.1.1 Package Bean

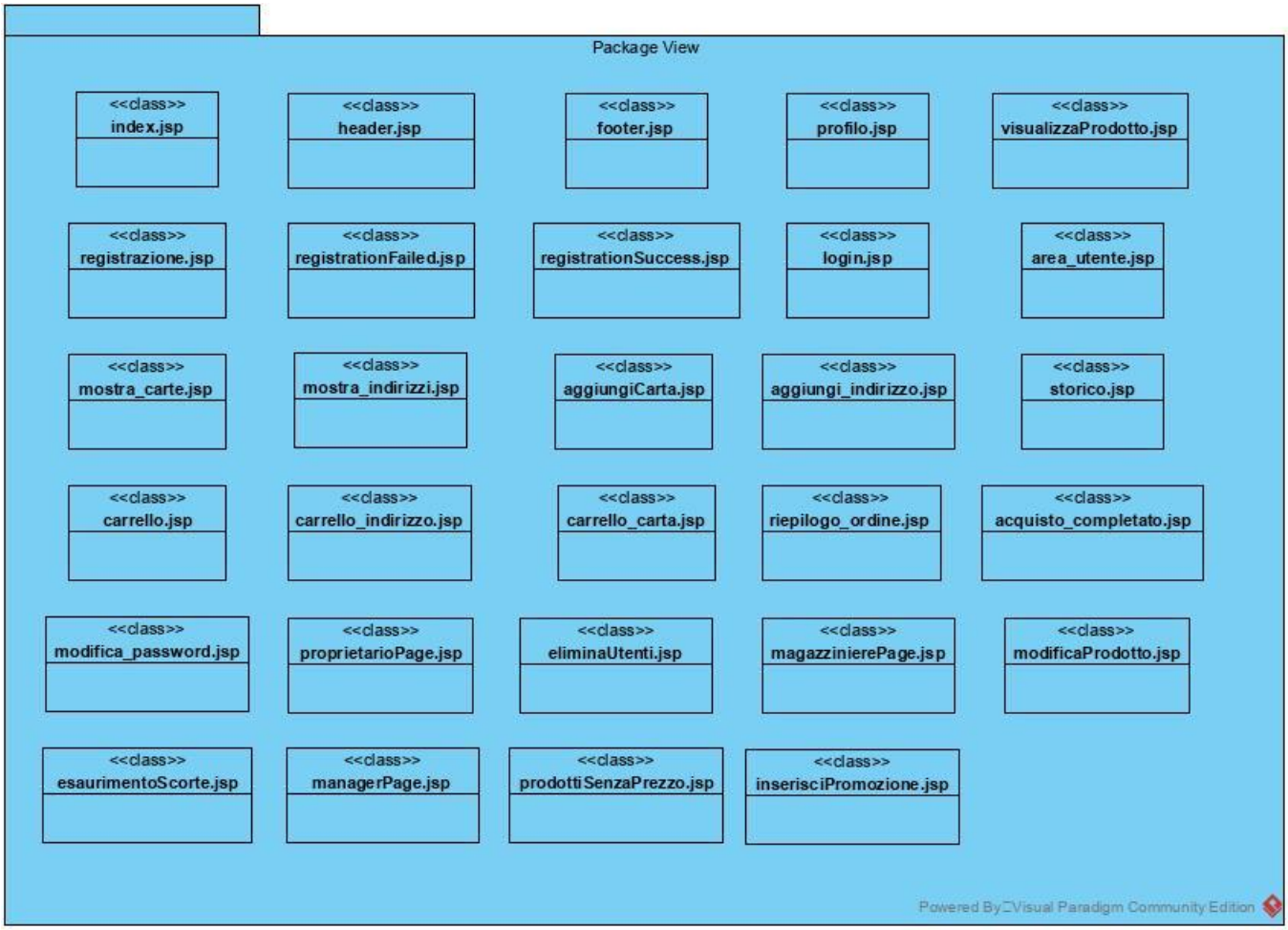
Il Package Bean include tutte le classi JavaBean.



Nome:	Descrizione:
CartaDiCreditoBean.java	Descrive una carta di credito creata da un utente.
IndirizzoBean.java	Descrive un indirizzo di un utente.
OrdinazioneBean.java	Descrive la relazione tra un prodotto in ordine e un ordine effettuato da un utente.
OrdineBean.java	Descrive un ordine effettuato da un utente.
ProdottoBean.java	Descrive un prodotto del sito.
ProdottoInCarrello.java	Descrive un prodotto aggiunto al carrello da un utente.
ProdottoInOrdineBean.java	Descrive un prodotto ordinato da un utente.
PromozioneBean.java	Descrive una promozione applicata ad un prodotto all'interno del sito.
TagliaBean.java	Descriva la taglia di un prodotto.
UtenteBean.java	Descrive un utente iscritto al sito.

2.1.2 Package View

Il Package View contiene tutte le pagine JSP, ovvero le pagine che gestiscono la visualizzazione dei contenuti da parte di un utente del sistema.



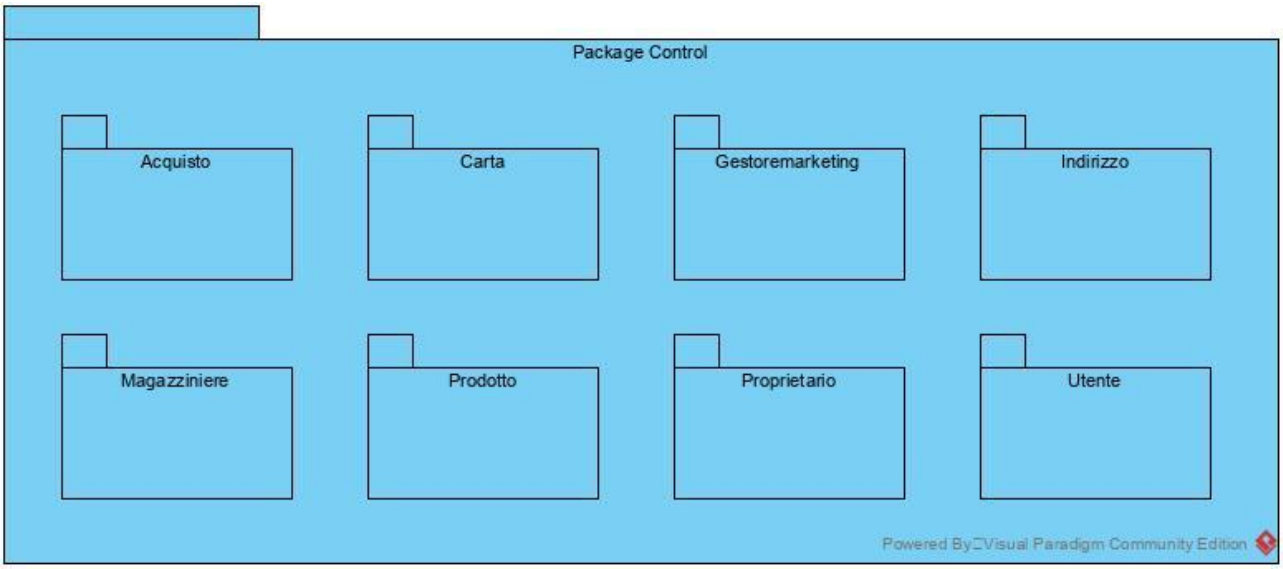
Nome:	Descrizione:
index.jsp	Pagina principale della piattaforma, mostra alcuni prodotti presenti nel sito.
header.jsp	Una sezione che viene mostrata in cima ad ogni pagina, contiene il logo e i tasti: Carrello, Area Utente e login (o logout se l'utente ha già effettuato l'accesso).
footer.jsp	Una sezione che viene mostrata in fondo ad ogni pagina.
profilo.jsp	Pagina che contiene le informazioni di un utente loggato come nome, cognome, email e password.
visualizzaProdotto.jsp	Pagina che contiene le informazioni di un prodotto.
registrazione.jsp	Pagina che consente ad un utente di effettuare una registrazione al sito tramite dei form.
registrazioneFailed.jsp	Pagina che mostra un messaggio di errore se la registrazione ha riscontrato dei problemi.

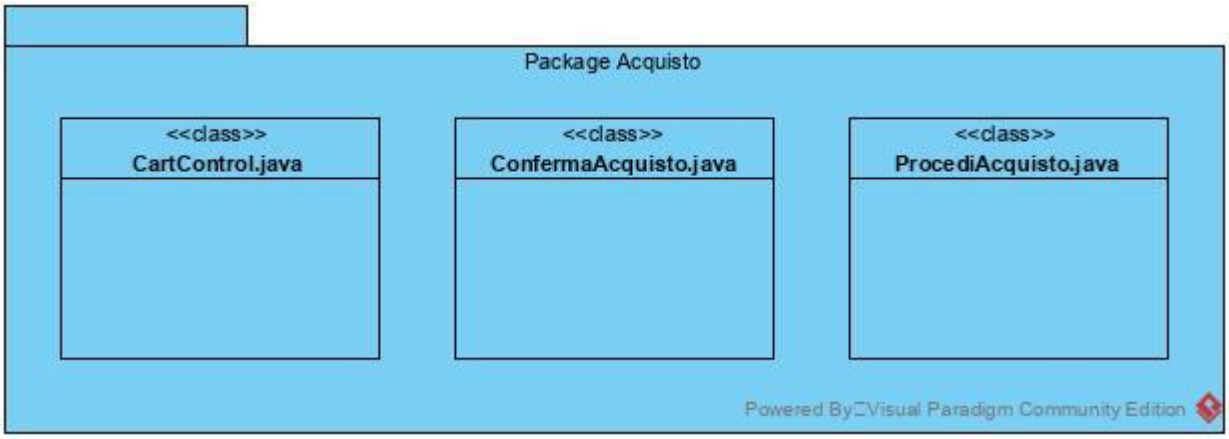
registrazioneSuccess.jsp	Pagina che conferma all'utente l'avvenuta registrazione.
login.jsp	Pagina che consente di accedere al sito tramite dei form ad un utente già registrato.
area_utente.jsp	Pagina che consente ad un utente già loggato di visualizzare alcune informazioni come lo storico acquisti, la lista degli indirizzi e la lista delle carte di credito. Da qui è in oltre possibile eliminare la propria registrazione al sito ed eseguire il logout.
mostra_carte.jsp	Pagina che mostra all'utente le sue carte di credito
mostra_indirizzi.jsp	Pagina che mostra all'utente i suoi indirizzi.
aggiungiCarta.jsp	Pagina che consente all'utente di inserire una nuova carta di credito.
aggiungi_indirizzo.jsp	Pagina che consente all'utente di inserire un nuovo indirizzo.
storico.jsp	Pagina che contiene lo storico acquisti, ovvero tutti gli ordini eseguiti dall'utente.
carrello.jsp	Pagina che contiene i prodotti che l'utente ha aggiunto al proprio carrello.
carrello_indirizzo.jsp	Pagina che viene mostrata quando l'utente decide di effettuare un ordine. Mostra all'utente l'elenco dei suoi indirizzi per consentirgli di sceglierne uno o, eventualmente, di aggiungerne uno nuovo.
carrello_carta.jsp	Pagina che viene mostrata, dopo aver scelto l'indirizzo, quando l'utente decide di effettuare un ordine. Mostra all'utente l'elenco delle sue carte di credito per consentirgli di sceglierne una o, nel caso, di aggiungerne una nuova.
riepilogo_ordine.jsp	Pagina che viene mostrata al termine della scelta di un indirizzo e di una carta di credito quando l'utente decide di effettuare un ordine. Mostra all'utente i prodotti che vuole acquistare, l'indirizzo e la carta di credito che ha scelto.
acquisto_completato.jsp	Pagina che viene mostrata quando l'utente ha ultimato un ordine con successo.
modifica_password.jsp	Pagina che consente all'utente di modificare la propria password attraverso dei form.
proprietarioPage.jsp	Pagina che viene mostrata all'amministratore quando esegue il login. Da questa pagina è possibile eliminare utenti dal sito o di modificare prodotti.
eliminaUtenti.jsp	Pagina che viene mostrata all'amministratore nella quale è possibile eliminare uno degli utenti dal sito.
magazzinierePage.jsp	Pagina che viene mostrata al magazziniere quando esegue il login. Da questa pagina è possibile modificare un prodotto o visualizzare le scorte in esaurimento.
modificaProdotto.jsp	Pagina che viene visualizzata al magazziniere o al proprietario. Consente agli stessi di modificare un prodotto attraverso l'utilizzo di form.

esaurimentoScorte.jsp	Pagina che viene mostrata al magazziniere. Consente allo stesso di visualizzare le scorte che sono in esaurimento.
managerPage.jsp	Pagina che viene mostrata al gestore marketing quando effettua il login. Da questa pagina è possibile modificare il prezzo dei prodotti o inserire una promozione.
prodottiSenzaPrezzo.jsp	Pagina che viene visualizzata dal gestore marketing e che gli consente di aggiungere il prezzo ai prodotti sprovvisti di prezzo.
inserisciPromozione.jsp	Pagina che viene visualizzata dal gestore marketing e che gli consente di aggiungere una promozione ai prodotti già presenti nel sito.

2.1.3 Package Control

Il Package Control contiene tutte le classi Servlet che rappresentano la logica applicativa della piattaforma web. Il sistema divide le varie Servlet in ulteriori sottopackage.

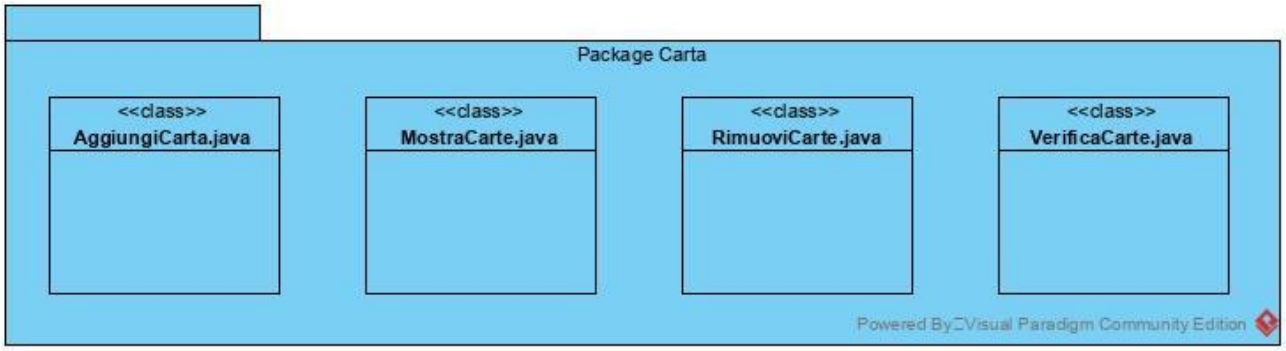




2.1.3.1 Package Acquisto

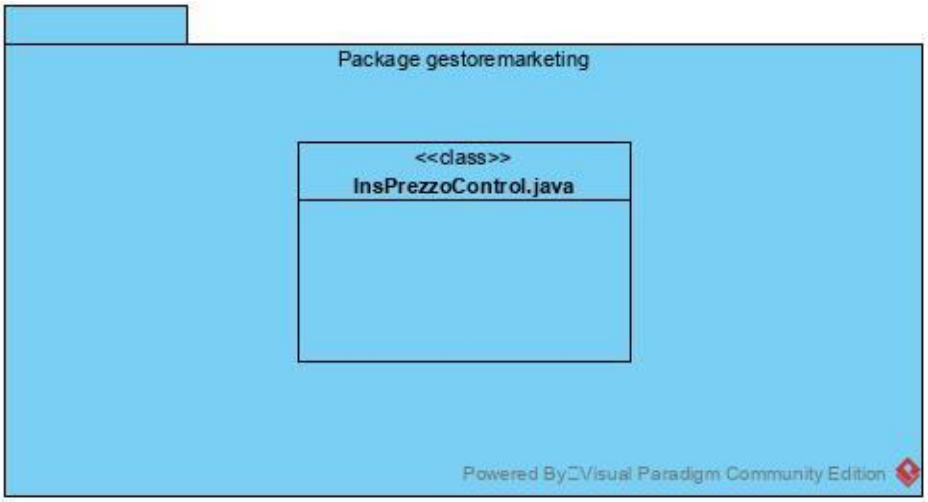
Nome:	Descrizione:
CartControl.java	Controller che consente ad un utente di aggiungere e rimuovere prodotti dal carrello.
ConfermaAcquisto.java	Controller che registra l'effettivo ordine e conferma l'avvenuto acquisto all'utente.
ProcediAcquisto.java	Controller che consente di iniziare la procedura di acquisto da parte di un utente.

2.1.3.2 Package Carta



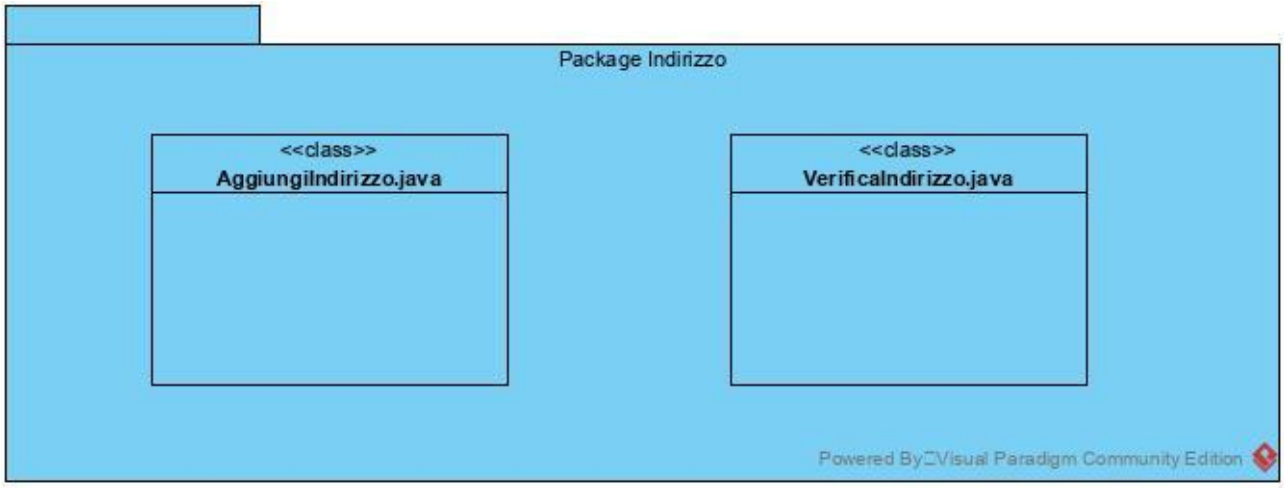
Nome:	Descrizione:
AggiungiCarta.java	Controller che consente di aggiungere una nuova carta di credito.
MostraCarte.java	Controller che consente di mostrare le carte di credito di un utente.
RimuoviCarte.java	Controller che consente di rimuovere una carta.
VerificaCarte.java	Controller che verifica i dati di una carta di credito.

2.1.3.3 Package Gestoremarketing



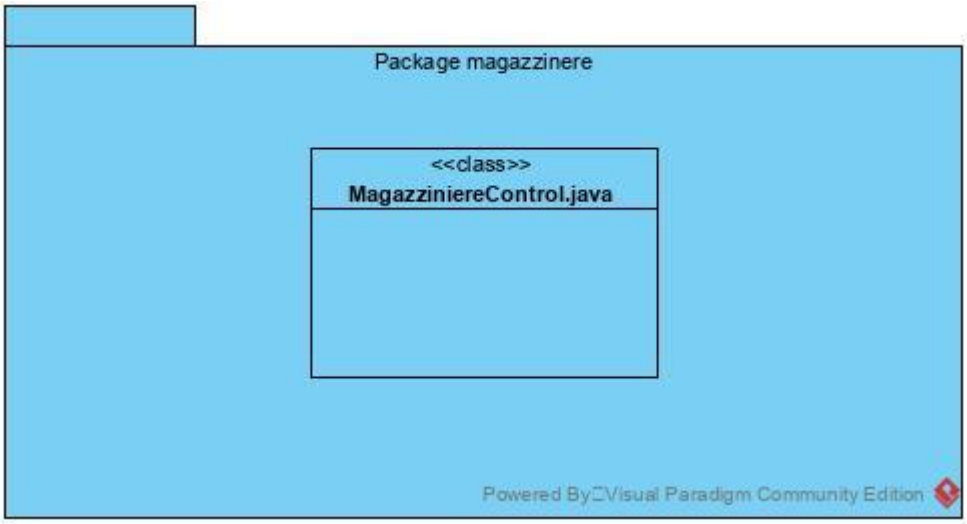
Nome:	Descrizione:
InsPrezzoControl.java	Controller che consente di gestire le operazioni eseguibili dal gestore marketing.

2.1.3.4 Package Indirizzo



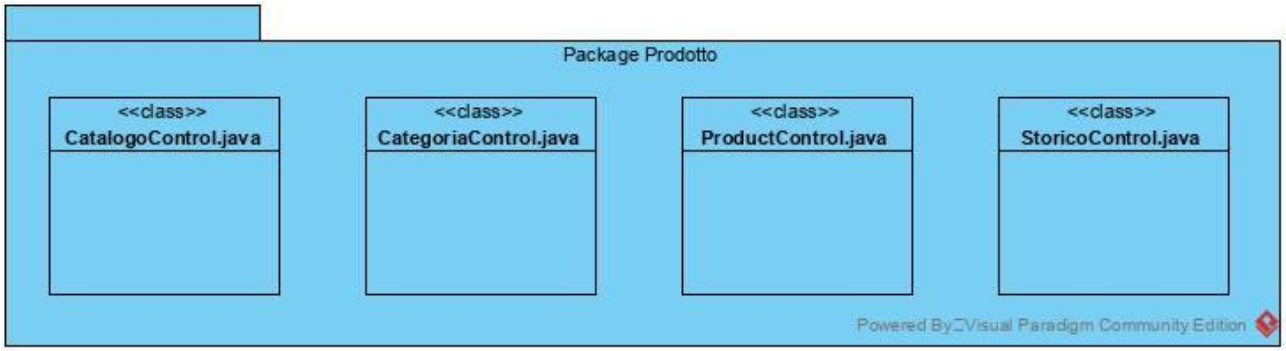
Nome:	Descrizione:
AggiungiIndirizzo.java	Controller che consente di aggiungere un nuovo indirizzo.
VerificaIndirizzo.java	Controller che verifica i dati di un indirizzo.

2.1.3.5 Package Magazziniere



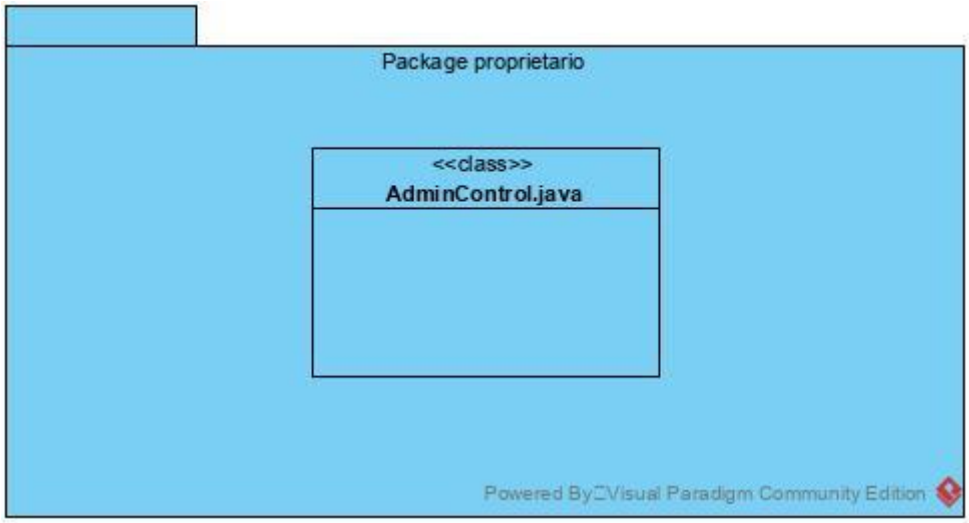
Nome:	Descrizione:
MagazziniereControl.java	Controller che consente di gestire le operazioni eseguibili dal magazziniere.

2.1.3.6 Package Prodotto



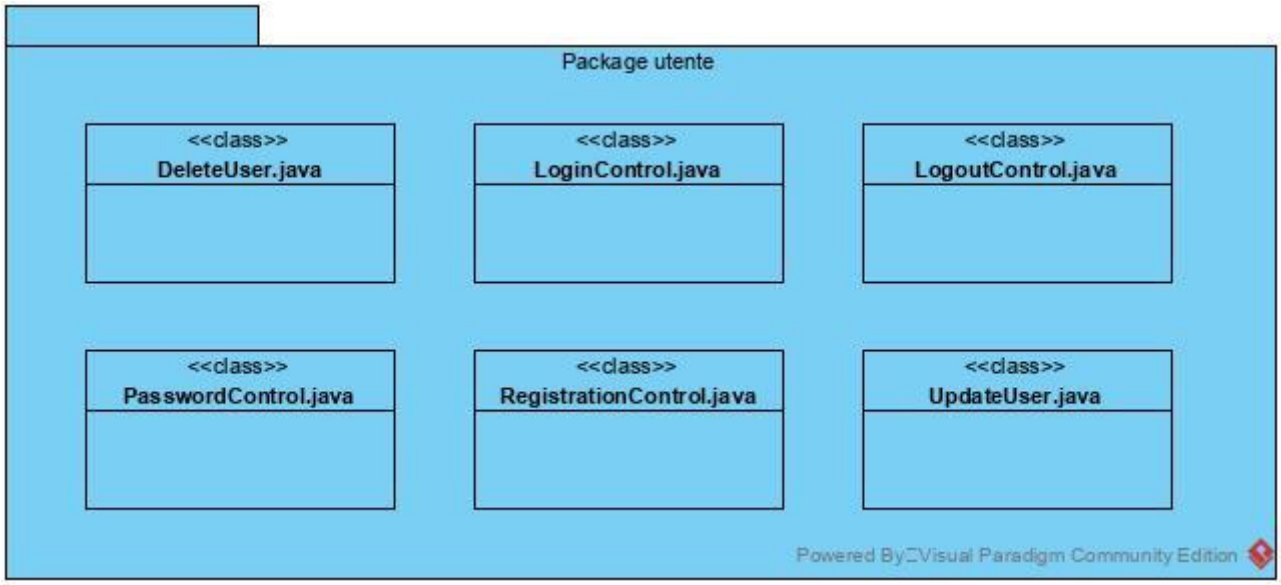
Nome:	Descrizione:
CatalogoControl.java	Controller che consente di visualizzare il catalogo di prodotti.
CategoriaControl.java	Controller che consente di visualizzare le varie categorie dei prodotti.
ProductControl.java	Controller che consente di visualizzare un prodotto.
StoricoControl.java	Controller che consente di visualizzare lo storico acquisti di un utente.

2.1.3.7 Package Proprietario



Nome:	Descrizione:
AdminControl.java	Controller che consente di gestire le operazioni eseguibili dal proprietario.

2.1.3.8 Package Utente



Nome:	Descrizione:
DeleteUser.java	Controller che consente di rimuovere un utente.
LoginControl.java	Controller che verifica i dati durante la procedura di login di un utente.
LogoutControl.java	Controller che consente di effettuare il logout.
PasswordControl.java	Controller che consente di modificare una password di un utente già registrato al sito.
RegistrationControl.java	Controller che consente di aggiungere un nuovo utente al sito.
UpdateUser.java	Controller che consente di modificare i dati di un utente già registrato al sito.

2.1.4 Package Manager

Il Package Manager contiene tutte le classi Java che accedono ai dati presenti nel sistema.

Nome:	Descrizione:
CartaDiCreditoModelDM.java	Classe di gestione dati di una carta di credito.
IndirizzoModelDM.java	Classe di gestione dati di un indirizzo.
OrdinazioneModelDM.java	Classe di gestione dati di un ordinazione.
OrdineModelDM.java	Classe di gestione dati di un ordine.
ProdottoInOrdineModelDM.java	Classe di gestione dati di un prodotto in ordine.
ProdottoModelDM.java	Classe di gestione dati di un prodotto.
PromozioneModelDM.java	Classe di gestione dati di una promozione.
TagliaModelDM.java	Classe di gestione di una taglia.
UtenteModelDM.java	Classe di gestione di un utente.

3.Class Interfaces

Classi Manager progettate per compiere funzionalità atomiche legate ai servizi dei sottosistemi specificati nel documento di System Design.

Nome classe	UtenteManager
Descrizione	Classe che gestisce alcune funzionalità legate ai servizi del sottosistema di gestione Autenticazione e Registrazione
	context UtenteManager::doRetrieveByKey(id_utente) **pre:** id_utente!= null **return** utente **context** UtenteManager::doRetrieveAll() **return** listaUtenti **context** UtenteManager::doRetrieveByEmail(email) **pre:** email != null **return** utente **context** UtenteManager::doSave(utente) **context** UtenteManager::doUpdate(utente) **context** UtenteManager::doUpdatePassword(password, id_utente) **context** UtenteManager::doDelete(id_utente) **post**: result != 0 **return** result **context** UtenteManager::Validate(email, password) **pre**: status = false **post:** status = true **return:** status **context** UtenteManager::CheckUser(email) **pre:** flag = false **return:** flag **context** UtenteManager::getTipo(email, password) **pre:** flag = 0 **post:** flag = 1 || flag = 2 || flag = 3 || flag = 4 **return** flag **context** UtenteManager::getIdUtente() **pre:** id_utente = 0 **post:** id_utente != 0 **return** id_utente **context** UtenteManager::getBean(ResultSet rs) **pre:** rs != null **return** utente **context** UtenteManager::CalculateHash(password) **pre:** password != null **post:** convert password into HashCode

	return hashtext
Invarianti	

Nome classe	TagliaManager
Descrizione	Classe che gestisce una caratteristica di un prodotto e alcune funzionalità legate ai servizi del sottosistema di gestione del Carrello
	context TagliaManager::doRetrieveByProdotto(id_prodotto) **return** listaTaglie **context** TagliaManager::doSave(taglia) **pre**: id_prodotto = 0 **post:** id_prodotto != 0 **context** TagliaManager::doUpdate(taglia) **context** TagliaManager::doDelete(taglia) **pre:** result = 0 **post:** result != 0 **return** result **context** TagliaManager:: getBean(ResultSet rs) **pre:** rs != null **return** taglia
Invarianti	

Nome classe	PromozioneManager
Descrizione	Classe che gestisce una caratteristica di un ordine e alcune funzionalità legate ai servizi del sottosistema di gestione del Carrello
	context PromozioneManager::doRetrieveByKey(id_promozione) **return** promozione **context** PromozioneManager::doRetrieveAll() **return** listaPromozioni **context** PromozioneManager::doSave(promozione) **context** PromozioneManager::doUpdate(promozione) **context** PromozioneManager::doDelete(promozione) **pre:** result = 0 **post:** result != 0 **return** result **context** PromozioneManager::getBean(ResultSet rs) **pre:** rs != null **return** promozione
Invarianti	

Nome classe	ProdottoInOrdineManager
Descrizione	Classe che gestisce lo stato di un prodotto e alcune funzionalità legate ai servizi del sottosistema di gestione del Carrello
	context ProdottoInOrdineManager::doRetrieveByKey(id_prodotto_ordine) **return** prodottoInOrdine **context** ProdottoInOrdineManager::doRetrieveAll() **return** listaProdottiInOrdine **context** ProdottoInOrdineManager::doSave(prodotto) **return** id_prodotto_ordine **context** ProdottoInOrdineManager::doUpdate(prodotto) **context** ProdottoInOrdineManager::doDelete(id_prodotto_ordine) **pre:** result = 0 **post:** result != 0 **return** result **context** ProdottoInOrdineManager::doRetrieveByOrder(id_ordine) **pre:** id_ordine != null **return** listaProdottiInOrdine **context** ProdottoInOrdineManager::getBean(ResultSet rs) **pre:** rs != null **return** prodottoInOrdine
Invarianti	

Nome classe	ProdottoManager
Descrizione	Classe che gestisce un prodotto e alcune funzionalità legate ai servizi del sottosistema di gestione del Carrello
	context ProdottoManager::doRetrieveByKey(id_prodotto) **return** prodotto **context** ProdottoManager::doRetrieveAll() **return** listaProdotti **context** ProdottoManager:: doRetrieveAllIn_vendita() **return** listaProdottiIn_vendita **context** ProdottoManager::doRetrieveAllNot_vendita() **return** listaProdottiNot_vendita **context** ProdottoManager::doRetrieveAllPrezzoZero() **return** listaProdotti **context** ProdottoManager::doSave(prodotto) **return** id **context** ProdottoManager::getUpdate(prodotto) **context** ProdottoManager::getDelete(prodotto) **pre:** result = 0 **post:** result != 0

	return result **context** ProdottoManager::doRetrieveByCategory(categoria) **return** listaProdotti **context** ProdottoManager::doUpdatePrezzo(id, prezzo) **context** ProdottoManager::getBean(ResultSet rs) **pre:** rs != null **return** prodotto
Invarianti	

Nome classe	OrdineManager
Descrizione	Classe che gestisce un ordine e alcune funzionalità legate ai servizi del sottosistema di gestione del Carrello
	context OrdineManager::doRetrieveByKey(id_ordine) **return** ordine **context** OrdineManager::doRetrieveAll() **return** listaOrdini **context** OrdineManager::doSave(ordine) **return** id **context** OrdineManager::doUpdate(ordine) **context** OrdineManager::doDelete(id_ordine) **pre:** result = 0 **post:** result != 0 **return** result **context** OrdineManager::doRetrieveByUtente(utente) **return** listaOrdini **context** OrdineManager::getBean(ResultSet rs) **pre:** rs != null **return** ordine
Invarianti	

Nome classe	OrdinazioneManager
Descrizione	Classe che gestisce funzionalità legate a un ordine e alcune funzionalità legate ai servizi del sottosistema di gestione del Carrello
	context OrdinazioneManager::doSave(ordinazione) **context** OrdinazioneManager::doDelete(ordinazione) **pre:** result = 0 **post:** result != 0 **return** result
Invarianti	

Nome classe	IndirizzoManager
Descrizione	Classe che gestisce gli indirizzi un utente e alcune funzionalità legate ai servizi dei sottosistemi di gestione della Registrazione e dei Clienti
	context IndirizzoManager::doRetrieveByKey(id_indirizzo) **return** indirizzo **context** IndirizzoManager::doRetrieveAll() **return** listaIndirizzo **context** IndirizzoManager::doSave(indirizzo) **context** IndirizzoManager::doUpdate(indirizzo) **context** IndirizzoManager::doDelete(id_indirizzo) **pre:** result = 0 **post:** result != 0 **return** result **context** IndirizzoManager::doRetrieveByUtente(utente) **return** listaIndirizzi **context** IndirizzoManager::checkIndirizzo(via, città) **pre:** flag = false **return** flag **context** IndirizzoManager::getBean(ResultSet rs) **pre:** rs != null **return** indirizzo
Invarianti	

Nome classe	CartaDiCreditoManager
Descrizione	Classe che gestisce le carte di credito un utente e alcune funzionalità legate ai servizi del sottosistema di gestione dei Clienti
	context CartaDiCreditoManager::doRetrieveByKey(numero_carta) **return** cartaDiCredito **context** CartaDiCreditoManager::doRetrieveAll() **return** listaCarte **context** CartaDiCreditoManager::doSave(cartaDiCredito) **context** CartaDiCreditoManager::doUpdate(cartaDiCredito) **context** CartaDiCreditoManager::doDelete(numero_carta) **pre:** result = 0 **post:** result != 0 **return** result **context** CartaDiCreditoManager::doRetrieveByUtente(utente) **return** cartaDiCredito **context** CartaDiCreditoManager::getBean(ResultSet rs) **return** cartaDiCredito **context** CartaDiCreditoManager::checkCarta(numero_carta)

	pre: flag = false **return** flag
Invarianti	