Università degli Studi di Salerno
Corso di Ingegneria del Software

Mille Miglia In Viaggio
Object Design Document
Versione 1.0

Data: 08/02/2019

Progetto: Mille Miglia In Viaggio	Versione: 1.0
Documento: Object Design Document	Data: 08/02/2019

Coordinatore del progetto:

Nome	Matricola
De Lucia Andrea	

Partecipanti:

Nome	Matricola
Antonio Scognamiglio	0512104868
Giuliana Muto	0512104598
Matteo Fasolino	0512103038

Scritto da:	Giuliana Muto

Revision History

Data	Versione	Descrizione	Autore
08/02/2019	1.0	Introduzione	Matteo Fasolino
08/02/2019	1.0	Design pattern Singleton	Antonio Scognamiglio
08/02/2019	1.0	Design pattern MVC	Giuliana Muto
08/02/2019	1.0	Package	Antonio Scognamiglio
08/02/2019	1.0	Class interfaces	Giuliana Muto
08/02/2019	1.0	Glossario	Giuliana Muto

Sommario

1. INTRODUZIONE

1.1 Object Design Trade

Il team di progettazione, sulla base della formazione del documento dell'analisi dei requisiti, si lancia verso la fase implementativa. Questa nuova fase mira ad esplicare una panoramica del sistema con annesse le interfacce delle classi; operazioni e tipi di dato supportati; convenzioni e linee guida accennate precedentemente nell'SDD.

Spazi di memoria vs Tempi di risposta

Nonostante il sistema sfrutti un database per la gestione dei dati, persino durante il caricamento di immagini per la messa in vendita di un pacchetto, non soffre di pesantezza o rallentamenti di velocità perché ci sono delle imposizioni sui limiti di grandezza e tipo di file durante la fase di caricamento.

Prestazioni vs Costi

Tutte le fasi salienti in cui vengono inseriti i dati persistenti e sensibili (IBAN, numero di conto), durante la fase di acquisto o di vendita, demandano il controllo di questi ai gestori finanziari terzi; questo permette agli sviluppatori di concentrarsi maggiormente su una piattaforma semplice e leggera.

Interfaccia vs Usabilità

Nonostante il sistema sia previsto di una buona interfaccia grafica, in modo da rendere maggiormente appetibile alla vista il sito, il team si impegna maggiormente a puntare sull'usabilità perché lo scopo è quello di allargare il più possibile il golfo di appartenenza degli utenti. Anche chi non ha dimestichezza in informatica può approcciarsi al sistema.

Sicurezza vs Efficienza

Sono entrambi fattori a cui si punta sempre al 100%, ma quando vengono messi a confronto per fare delle scelte si cerca sempre un compromesso, limitando a rendere ottimali tutte le funzionalità che descrivono il sistema e abolendo preventivamente quelle che possono causare una cattiva gestione. Il sistema gode dell'efficienza e della sicurezza nella sua semplicità.

1.2 Linee guida per la documentazione di interfacce

Il team di sviluppo adopera le seguenti convenzioni:
I nomi sono descrittivi, quindi rispecchiano il concetto essenziale di cui si sta trattando; sono di facile comprensione, quindi pronunciabili, di breve lunghezza e non vi si consente di usare abbreviazioni per non creare confusione; infine vengono utilizzati solo caratteri e termini consentiti.
Le classi avranno un singolo nome, senza lo stile Camel, e rispecchiano l'essenza dell'oggetto che avrà uno stato ed un comportamento; inizierà con la lettera maiuscola e sarò senza alcuna abbreviazione. Ogni classe ha la propria lista di metodi, mantenendo una stretta coesione ed un buono compromesso di accoppiamento quando lo richiede.
Le variabili iniziano con le lettere minuscole e, come anche per i metodi, seguono uno stile a cammello (nel caso si ritenga opportuno di aggiungere un altro termine, il successivo inizierà consecutivamente con la lettera maiuscola senza lasciare caratteri di spazi bianchi e underscore).
Il carattere "_" è destinato esclusivamente alle costanti.
I metodi sono raggruppati in un'unica classe rispecchianti le ristrette ed indispensabili funzionalità, restituendo un valore di ritorno per ogni variabile d'istanza in modo da evitare effetti collaterali e da mantenere sempre il valore dell'istanza aggiornato.
Le pagine JSP devono, quando eseguite, produrre sempre un documento conforme allo standard HTML. Le parti Java delle pagine devono aderire alle convenzioni per la codifica in Java.
Le pagine HTML statiche e dinamiche devono aderire allo standard HTML. 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;
- Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;

Negli Script Javascript i vari Statement dei vari costrutti devono essere indentati tra loro attraverso l'utilizzo di TAB. I nomi dei metodi devono essere in italiano, devono iniziare in lettera minuscola e devono riferirsi a verbo. Nel caso in cui si vuole unire il nome del metodo con un nome sarà necessario unirlo al nome del verbo con la lettera maiuscola. Ad ogni fine istruzione si dovrà andare a capo per poter scrivere l'istruzione successiva. All'inizio di ogni script deve essere fatto un commento in cui si fornisce la spiegazione dello script prodotto. Lo script avrà il seguente formato: /* Spiegazione metodo */
Ogni regola CSS deve essere formattata come segue:

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

- L’ ultimo selettore della regola è seguito da parentesi graffa aperta ({);
- Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
- La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

Le proprietà e le regole poco chiare dovrebbero essere precedute da un commento esplicativo.

1.3 Definizioni acronimi e abbreviazioni

1.3.1 Definizioni

- Design Pattern: descrizione o modello logico da applicare per la risoluzione di un problema incontrato durante le fasi di progettazione e sviluppo del software.
- Package: collezione di classi e interfacce correlate.
- Package model: pacchetto che include i model del progetto, ossia la componente del modello MVC che si occupa dell’interazione tra l’applicazione e il database.
- Package routes: pacchetto che include i controller del progetto, ossia la componente del modello MVC che si occupata di elaborare le richieste di un utente e di comunicare con il model.
- Package docs: pacchetto che include le view del progetto, ossia la componente del modello MVC scritto in linguaggio HTML e visualizzato nella pagina web dall’utente che permette all’utente di interagire col sistema.
- MVC: modello architetturale che si basa sull’utilizzo di 3 componenti fondamentali (model,view e controller) per lo sviluppo di web application.
- Class diagram: Tipo di diagramma che descrive la struttura di un sistema mostrando le sue classi, gli attributi di tali classi, le operazioni (o metodi) e le relazioni tra gli oggetti.

1.3.2 Acronimi e abbreviazioni

- ODD: Object Design Document;
- RAD: Requirement Analisis Document;
- SDD: System Design Document;

1.4 Riferimenti

- RAD;
- SDD.

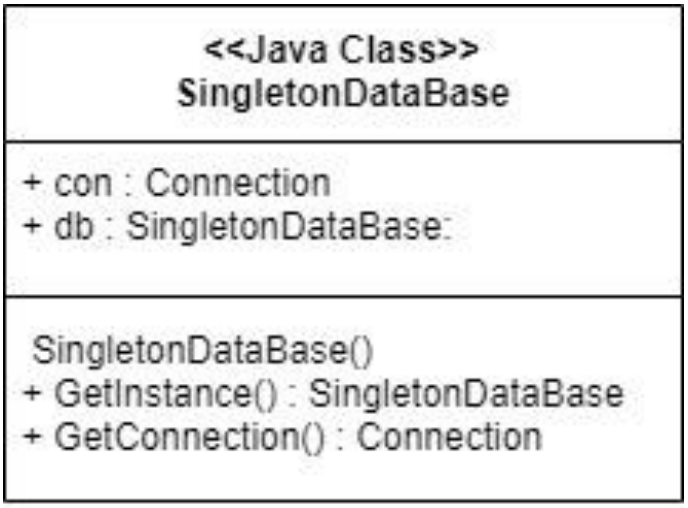
2. DESIGN PATTERN

2.1 Design pattern Singleton

Il singleton è un design pattern che ha lo scopo di:

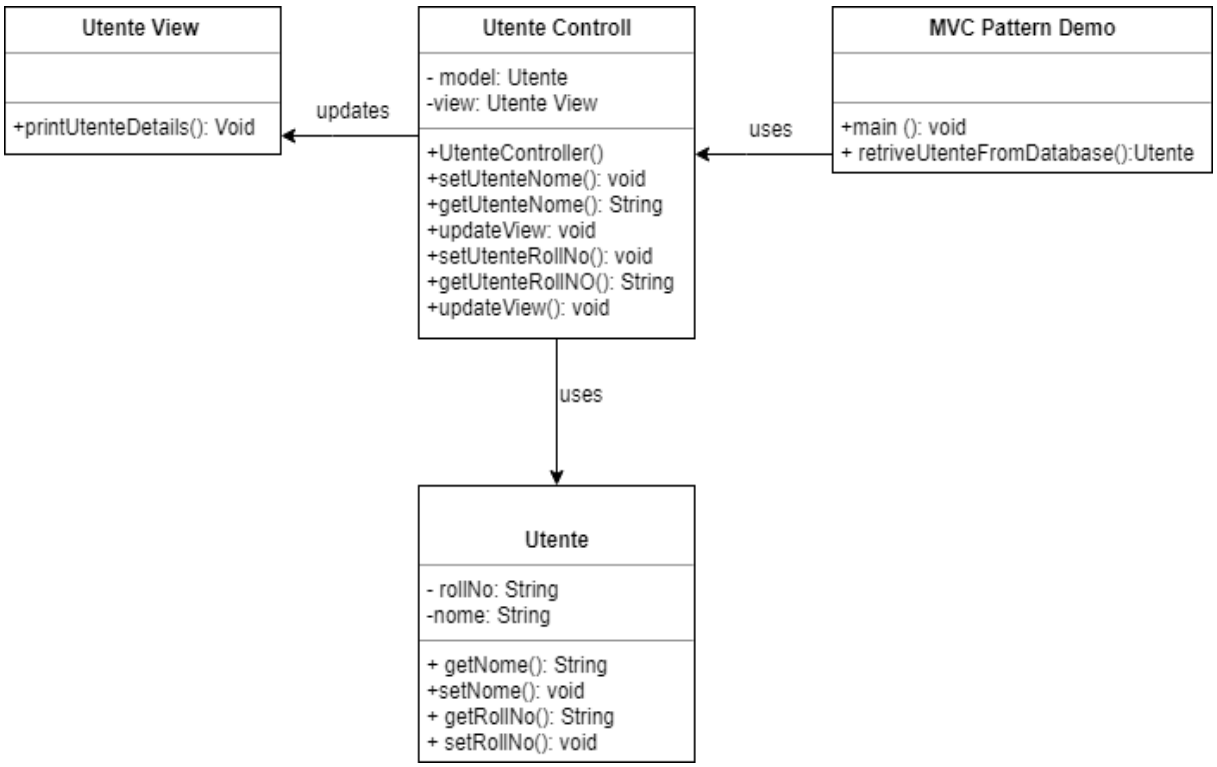
- garantire che di una determinata classe ne venga creata una ed una sola istanza;
- fornire un punto di accesso globale alla classe.

Nel nostro caso è importante che una classe abbia esattamente un'istanza. La classe stessa è responsabile di avere traccia della sua unica istanza e fornire un modo per accedere a tale istanza, in questo modo abbiamo una sola istanza per connettersi al database e possiamo evitare di avere dati non aggiornati o errati.



2.3 Design pattern MVC

L’applicazione deve fornire una interfaccia grafica (GUI) costituita da più schermate, che mostrano vari dati all'utente. Inoltre le informazioni che devono essere visualizzate devono essere sempre quelle aggiornate
E' possibile accedere alla gestione dei dati con diverse tipologie di GUI. I dati dell'applicazione possono essere aggiornati tramite diverse interazioni da parte dei client (richieste HTTP...)
Il supporto di varie GUI ed interazioni non influisce sulle funzionalità di base dell'applicazione.
L'applicazione deve separare i componenti software che implementano il modello delle funzionalità di business, dai componenti che implementano la logica di presentazione e di controllo che utilizzano tali funzionalità. Vengono quindi definiti tre tipologie di componenti che soddisfano tali requisiti:
- il Model, che implementa le funzionalità di business
- la View: che implementa la logica di presentazione
- il Controller: che implementa la logica di controllo



3. PACKAGE

3.1 Package model

Cartelle:

- CSS: Contiene il CSS
- XML: Contiene i file XML
- JSP: Contiene tutte le view
- IMG: Contiene tutte le immagini
- Classi comuni: Contiene le classi per la memorizzazione dei dati.
- Gestione Acquisto: Contiene le classi per la gestione dell'acquisto, della ricerca e del carrello.
- Gestione Vendita: Contiene le classi per la gestione della vendita.
- Gestione Utente: Contiene le classi per la gestione dell'utente
- Servlet Vendita: Contiene le servlet per la gestione Vendita
- Servlet Utente: Contiene le servlet per la gestione Utente
- Servlet Acquisto: Contiene le servlet per la gestione Acquisto, Carrello e ricerca.
- Storage: Contiene il Singleton che consente la connessione al Data Base

3.1 Package Classi Comuni

Classe	Descrizione
Utente.java	Contiene le informazioni dell'utente
DatiAlloggio.java	Contiene le informazioni delle strutture
DatiVolo.java	Contiene le informazioni dei voli
DatiPacchetti.java	Contiene le informazioni dei Pacchetti

3.2 Package Gestione Acquisto

Classe	Descrizione
Acquisto.java	Contiene le informazioni relative all'acquisto
Carrello.java	Visualizza il carrello
Ricerca.java	Gestisce la ricerca di un pacchetto

3.3 Package Gestione Vendita

Classe	Descrizione
Vendita.java	Contiene le informazioni relative alla vendita
Pacchetti.java	Gestisce le informazioni relative al pacchetto

3.5 Package Gestione Utente

Classe	Descrizione
Accesso.java	Gestisce l'accesso dell'utente
RecuperaPassword.java	Gestisce il recupero della password
Registrazione.java	Gestisce la registrazione di un utente

3.8 Package Servlet Vendita

Classe	Descrizione
AggiungiPacchetto.java	Permette di aggiungere un pacchetto nel sistema
Elimina.java	Elimina un pacchetto
ModificaPacchetto.java	Gestisce la modifica di un pacchetto
VisualizzaPacchettoJSON.java	Recupera i dati dei pacchetti
CreaPaginaPacchetto.java	Gestisce la creazione di un pacchetto nel sistema

3.9 Package Servlet Utente

Classe	Descrizione
Accedo.java	Permette all'utente di accedere al sistema
AggiornaPassword.java	Permette di modificare la password
Logout.java	Permette all'utente di disconnettersi da sistema
Memorizza.java	Permette di memorizzare i dati dell'utente.
RecuperaPassword.java	Permette di recuperare la password
VisualizzaDati.java	Permette di visualizzare tutti i dati dell'utente

3.12 Package Servlet Acquista

Classe	Descrizione
Acquista.java	Permette di acquistare un pacchetto dal sistema
ServletRicerca.java	Permette di effettuare una ricerca in base ai filtri
AggiungiPacchOffline.java	Permette di aggiungere pacchetti nel carrello senza aver effettuato l'accesso
AggiungiCarrello.java	Permette di aggiungere un pacchetto al carrello
Elimina.java	Permette di eliminare un pacchetto dal carrello
CarrelloJSON.java	Permette di visualizzare i pacchetti nel carrello

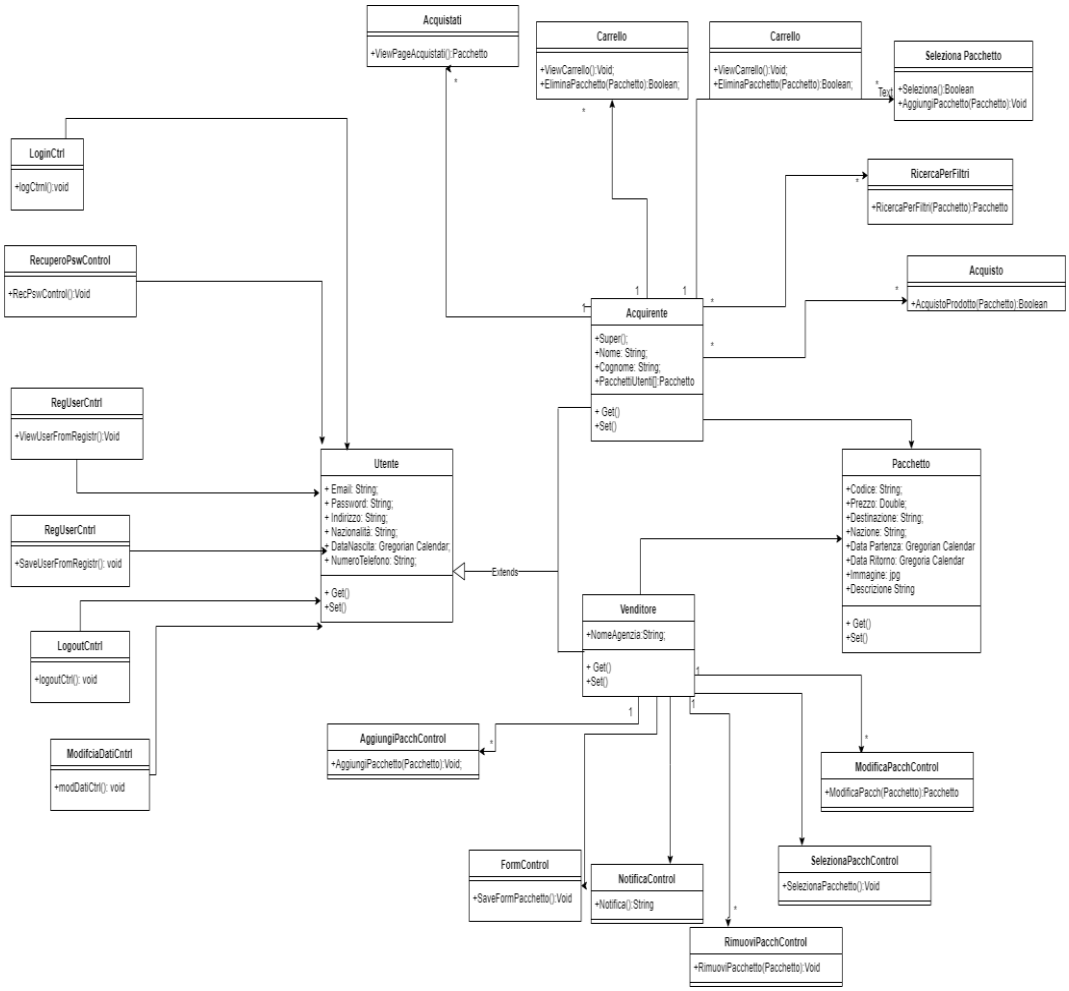
3.14 Package Storage

Classe	Descrizione
SingletonDataBase.java	Classe Java che Consente a tutte le classi la connessione al DB

3.15 Package JSP

Classe	Descrizione
Home.jsp	Pagina Principale
Pacchetti.jsp	Pagina di visualizzazione pacchetto
Registrazione.jsp	Pagina di registrazione
RisultatoRicerca.jsp	Pagina del risultato della ricerca
Ricerca.jsp	Pagina di ricerca
Acquisti.jsp	Pagina di storico acquisto
RecuperaPassword.jsp	Pagina di recupero password
Modifica_bar.jsp	Mostra la barra di navigazione
Aggiungi_Pacchetto.jsp	Pagina di inserimento pacchetto
Modifica_Pacchetto.jsp	Pagina di modifica pacchetto
Modifica_Profilo.jsp	Pagina di modifica profilo utente
Carrello.jsp	Pagina di visualizzazione carrello

4. CLASS INTERFACES



Nome file	Utente
Descrizione	Questa classe rappresenta le informazioni relative ad un utente nel sistema
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome file	Acquirente
Descrizione	Questa classe rappresenta le informazioni relative ad un acquirente nel sistema
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome file	Venditore
Descrizione	Questa classe rappresenta le informazioni relative ad un venditore nel sistema
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome file	Pacchetto
Descrizione	Questa classe rappresenta le informazioni relative ad un pacchetto nel sistema
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome file	RegUserCntrl
Descrizione	Questa classe gestisce la registrazione dell'utente come acquirente o come partner.
Pre-Condizione	RegUserCntrl::SaveUserFormRegistr() Pre:NULL RegUserCntrl:: ViewUserFormRegistr() Pre: NULL
Post-Condizione	
Invarianti	

Nome file	LoginCntrl
Descrizione	Questa classe gestisce se sono corretti i record inseriti nei campi del form dall'utente per l'accesso.
Pre-Condizione	LoginCntrl:: LoginControl(email,password) email != NULL && password != NULL
Post-Condizione	Session.setAttribute("email",email)
Invarianti	

Nome file	LogoutCntrl
Descrizione	Questa classe gestisce la chiusura della sessione dell'utente.
Pre-Condizione	Logout::Logout() Session.getAttribute("email") != NULL
Post-Condizione	Session.invalidate()
Invarianti	

Nome file	RecuperoPSWCntrl
Descrizione	Questa classe gestisce il recupero della password
Pre-Condizione	RecuperoPSWCntrl:: RecuperoPSWCntrl(email)
Post-Condizione	
Invarianti	

Nome file	ModificaDatiCntrl
Descrizione	Questa classe permette di modificare i dati dell'utente
Pre-Condizione	ModificaDatiCntrl:: ModificaDatiCntrl() Session.getAttribute(email) Email != NULL
Post-Condizione	
Invarianti	

Nome file	Acquisto
Descrizione	Questa classe gestisce l'acquisto di un pacchetto
Pre-Condizione	Acquisto::Acquisto(Pacchetti) Pacchetti != NULL Session.getAttribute(email) Email != NULL
Post-Condizione	
Invarianti	

Nome file	Acquistati
Descrizione	Questa classe mostra lo storico degli acquisti
Pre-Condizione	Aquistati::Acquistati(Pacchetti) Pacchetti != NULL Session.getAttribute(email) Email != NULL
Post-Condizione	
Invarianti	

Nome file	RicercaPerFiltri
Descrizione	Questa classe gestisce la ricerca di un pacchetto
Pre-Condizione	RicercaPerFiltri::RicercaPerFiltri(pachetto) Session.getAttribute(email) Email != NULL pacchetto != NULL
Post-Condizione	
Invarianti	

Nome file	ViewCarrello
Descrizione	Questa classe mostra il carrello
Pre-Condizione	ViewCarrello::ViewCarello() Session.getAttribute(email) Email != NULL
Post-Condizione	
Invarianti	

Nome file	EliminaPacchetto
Descrizione	Qusta classe gestisce l'eliminazione di un pacchetto nel carrello
Pre-Condizione	EliminaPacchetto::EliminaPacchetto(pacchetto) Session.getAttribute(email) Email != NULL Session.getAttribute(pacchetto) pacchetto != NULL
Post-Condizione	Carrello.removePacchetto(pacchetto)
Invarianti	

Nome file	SelezionaPacchetto
Descrizione	Questa classe permette di selezionare un pacchetto
Pre-Condizione	SelezionePacchetto::SelezionaPacchetto(pacchetto) Session.getAttribute(email) Email != NULL Session.getAttribute(pacchetto) pacchetto != NULL
Post-Condizione	
Invarianti	

Nome file	AggiungiPacchetto
Descrizione	Questa classe permette di aggiungere un pacchetto nel carrello
Pre-Condizione	AggiungiPacchetto::AggiungiPaccheto(pacchetto) Session.getAttribute(email) Email != NULL Session.getAttribute(pacchetto) Pacchetto != NULL
Post-Condizione	Carrello.addPacchetto(Pacchetto)
Invarianti	

Nome file	ModificaPacchettoCntrl
Descrizione	Questa classe permette di modificare un pacchetto
Pre-Condizione	ModificaPacchettoCntrl::ModificaPacchetoCntrl (pacchetto) Session.getAttribute(email) Email != NULL Session.getAttribute(pacchetto) Pacchetto != NULL
Post-Condizione	PacchettoModificato(pacchetto)
Invarianti	

Nome file	SelezionaPacchettoCntrl
Descrizione	Questa classe permette di selezionare un pacchetto
Pre-Condizione	SelezionePacchettoCntrl::SelezionaPacchettoCntrl (pacchetto) Session.getAttribute(email) Email != NULL Session.getAttribute(pacchetto) pacchetto != NULL
Post-Condizione	
Invarianti	

Nome file	EliminaPacchettoCntrl
Descrizione	Qusta classe gestisce l'eliminazione di un pacchetto del venditore
Pre-Condizione	EliminaPacchettoCntrl::EliminaPacchettoCntrl (pacchetto) Session.getAttribute(email) Email != NULL Session.getAttribute(pacchetto) pacchetto != NULL
Post-Condizione	Venditore.removePacchetto(pacchetto)
Invarianti	

Nome file	AggiungiPacchettoCntrl
Descrizione	Questa classe permette di aggiungere un pacchetto del venditore
Pre-Condizione	AggiungiPacchettoCntrl::AggiungiPacchetoCntrl (pacchetto) Session.getAttribute(email) Email != NULL Session.getAttribute(pacchetto) Pacchetto != NULL
Post-Condizione	Venditore.addPacchetto(Pacchetto)
Invarianti	

5. GLOSSARIO

Utente: Rappresenta un utente generico nel sistema;
Pacchetto: Oggetto di interazione nel sistema;
Acquirente: Rappresenta un utente loggato che può acquistare i pacchetti;
Venditore: Rappresenta un utente loggato di tipo Partner che può vendere i pacchetti**.**