Università degli Studi di Salerno
Corso di Ingegneria del Software

MusicParadise
Object Design Documentation
Versione 2.0

LOGO PROGETTO

Data: 19/12/2017

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
Antonio Spera	0512103538
Alessandro De Riso	0512103802
Domenico Pannone	0512103730
Vincenzo Pandolfo	0512103694

Scritto da:	

Revision History

Data	Versione	Descrizione	Autore
22/12/2017		Aggiunta descrizione classe Ordine	Antonio Spera
22/12/2017		Aggiunta descrizione classe Prodotto	Alessandro De Riso
22/12/2017		Aggiunta descrizione classe Indirizzo	Domenico Pannone
22/12/2017		Aggiunta descrizione classe Utente	Vincenzo Pandolfo
23/12/2017		Aggiunta package core	Antonio Spera
27/12/2017		Aggiunta package bean	Antonio Spera
30/12/2017		Aggiunta package control	Antonio Spera
5/01/2018		Aggiunta package control	Alessandro De Riso
10/01/2018		Aggiunta descrizione delle classi bean: utente, cliente gestore ordini	Alessandro De Riso
12/01/2018		Aggiunte descrizione delle classi bean: indirizzo, carta, prodotto.	Antonio Spera
20/01/2018		Revisione delle classi bean e inserimento sequence diagram	Alessandro De Riso
29/01/2018		Revisione del documento	Alessandro De Riso

1 SOMMARIO

2 INTRODUZIONE

2.1 OBJECT DESIGN TRADE-OFF

2.1.1 MODULARITA' VS EFFICIENZA

L'ampia modularità descritta nel documento MusicParadise.com SDD si scontra con un'efficienza necessaria nelle elaborazioni lato server. Ciò facilità la creazione e la manutenzione del programma (principio del divide et impera) ed, inoltre, aumenta la possibilità di riutilizzare lo stesso codice per altre applicazioni. La scelta di preferire la modularità avvale il sistema di una notevole indipendenza nelle differenti gestioni delle caratteristiche, offrendo una buona manutenibilità pur riducendo l'efficienza dei tempi di risposta dei moduli che si occupano di determinati servizi.

2.1.2 SICUREZZA VS EFFICIENZA

Nel nostro sistema ogni richiesta viene validata attraverso l'uso della sessione e un controllo a livello di utenza. Questo ci permette di impedire un accesso non autorizzato. Per fare ciò ogni pagina servlet o jsp deve autenticare l'utente oppure verificare che l'utente sia stato autenticato in precedenza. Questa caratteristica però potrebbe far alzare il tempo di risposta del sistema soprattutto con carichi di lavoro molto alti. Eliminando i controlli per l'esistenza e validità della sessione e introdurre la precondizione di sessione esistente, può portare a rischi di sicurezza molto elevati, in quanto utenti maliziosi potrebbero richiamare servizi non consentiti provocando danni al sistema. Per questo motivo anche se a discapito dell'efficienza optiamo per questi controlli.

2.1.3 PORTABILITA' VS EFFICIENZA

La portabilità del sistema MusicParadise.com è garantita dalla scelta del linguaggio di programmazione Java. Lo svantaggio dato da questa scelta è nella perdita di efficienza introdotta dal meccanismo della macchina virtuale Java. Tale compromesso è accettabile per i numerosi supporti forniti dal linguaggio Java.

2.2 INTERFACCE DOCUMENTATION GUIDELINES

2.2.1 FILE JAVA

Ogni file deve contenere:

- Commenti per comprendere al meglio il codice
- Dichiarazione dei package
- Sezione import
- Dichiarazione di classe o interfaccia
    - Attributi pubblici
    - Attributi privati
    - Attributi protetti
    - Costruttori
    - Metodi getter e setter
    - Altri metodi
- Classi interne

2.2.2 NAMING

Vengono utilizzate convenzioni su nomi per rendere il programma più leggibile e comprensibile.

Classi e interfacce

I nomi delle classi possono essere composti da più parole (ad esempio **ProdottoBean),** ogni parola che compone il nome della classe ha l'iniziale maiuscola. I nomi devono essere semplici e descrittivi in modo da capire il ruolo della classe. Evitare abbreviazioni.

Metodi

I **metodi** devono essere verbi (composti anche da più parole) con iniziale maiuscola. Ad esempio: addProdotto().

Costanti

Rispettando le convenzioni suggerite da Sun, i nomi delle **costanti** vengono indicati da nomi con tutti i caratteri maiuscoli. Le parole vengono separate da **"_".** Ad esempio: **static final MAX_COUNT = 10**

2.2.3 USO DEI COMMENTI

E' possibile utilizzare due diversi tipi di commenti:

- Commenti **Javadoc** (testo compreso tra /** e */)
- Commenti per singole righe (delimitate da //)

L'utilizzo dei commenti **Javadoc** è suggerito prima della dichiarazione di:

- Classi e interfacce
- Costruttori
- Metodi
- Variabili di classe

I commenti compresi tra /** e */ devono essere semplici e chiari, in modo da rendere leggibile la documentazione.

2.2.4 ALTRE REGOLE DI STILE

Ulteriori regole sono le seguenti:

- Nomi di package, classi e metodi devono essere nomi descrittivi
- Si consiglia l'utilizzo di parti standard dei nomi in casi come:
  - Classi astratte (es: AbstractClass)
  - Design pattern (es: ProdottoModel)
  - Eccezioni (es: ProdottoNonTrovatoException)
- è consigliato l'uso di suffissi standard come "get", "set", "is","has"

2.3 DEFINIZIONI, ACRONIMI E ABBREVIAZIONI

RAD	Documento di analisi dei requisiti
SDD	System Design Document
MVC	Model Control View

2.4 RIFERIMENTI

RAD-MusicParadise.com

SDD-MusicParadise.com

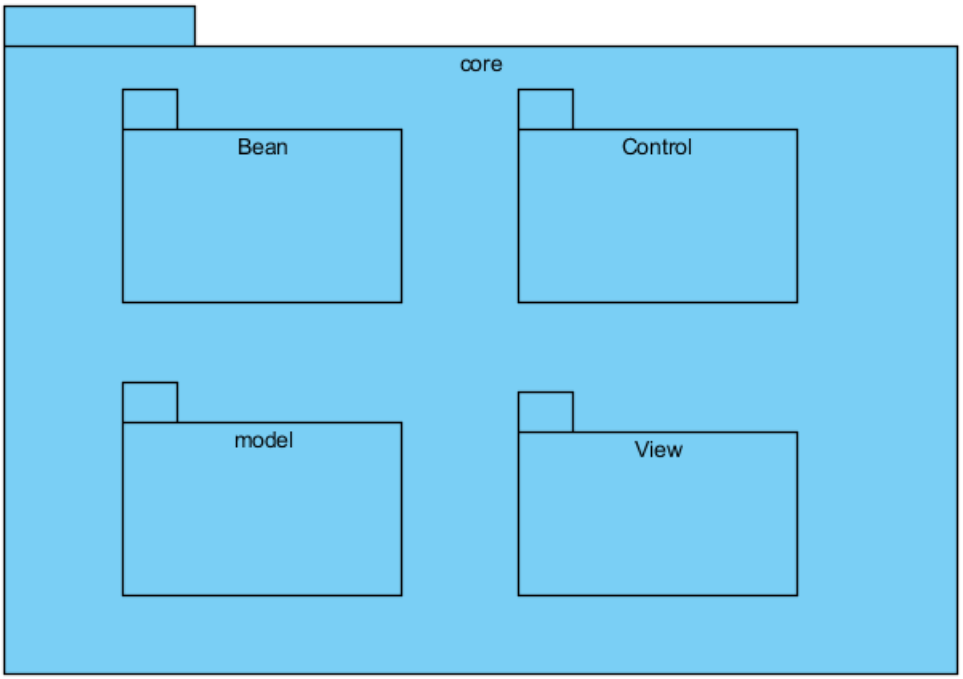
2.5 OVERVIEW

3 PACKAGES

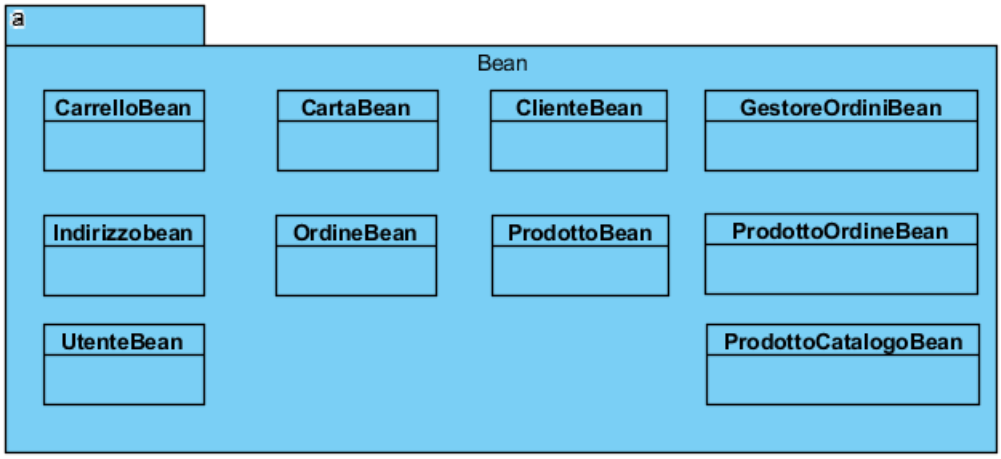
Le componenti base che costituiscono il sistema sono raggruppati in 3 livelli:

- Interface layer
- Application Logic layer
- Storage layer

3.1 PACKAGE CORE

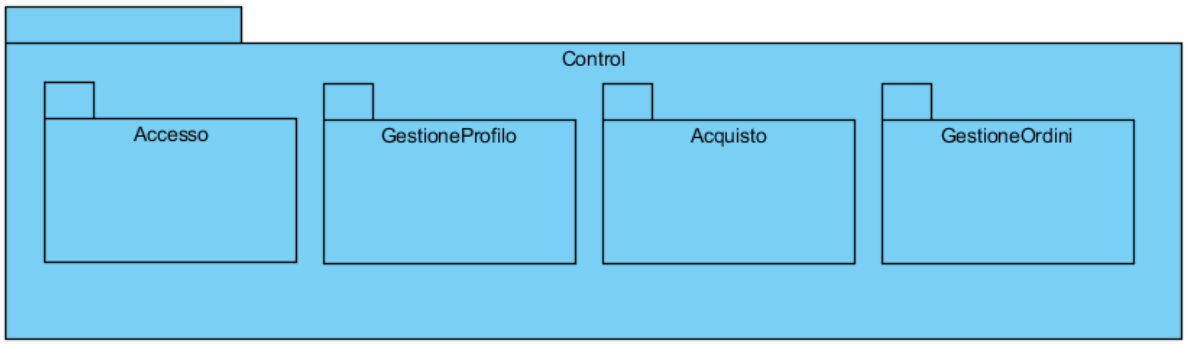


3.2 PACKAGE BEAN

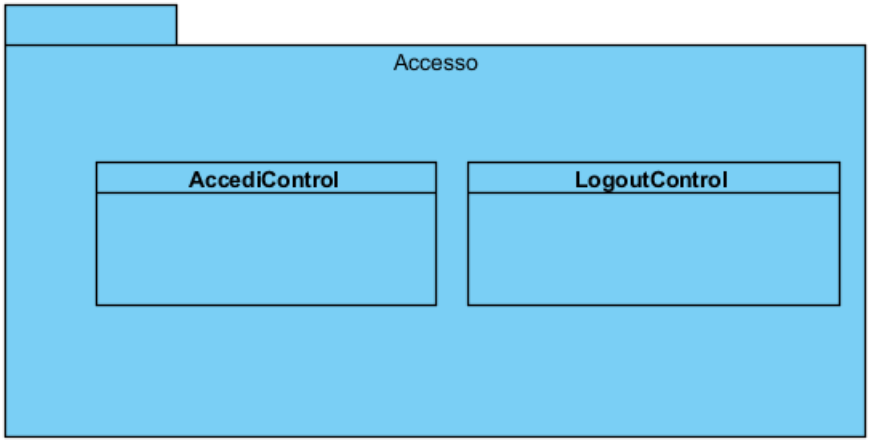


UtenteBean	Reppresenta l'oggetto utente
ProdottoBean	Rappresenta l'oggetto prodotto
OrdineBean	Rappresenta l'oggetto ordine
IndirizzoBean	Rappresenta l'oggetto indirizzo
CartaBean	Rappresenta l'oggetto carta
CarrelloBean	Rappresenta l'oggetto carrello
ProdottoCatalogoBean	Rappresenta l'oggetto prodotto presente nel catalogo
ProdottoOrdineBean	Rappresenta l'oggetto prodotto presente nell'ordine
ClienteBean	Rappresenta l'oggetto cliente
GestoreOrdiniBean	Rappresenta l'oggetto gestore ordini

3.3 PACKAGE CONTROL

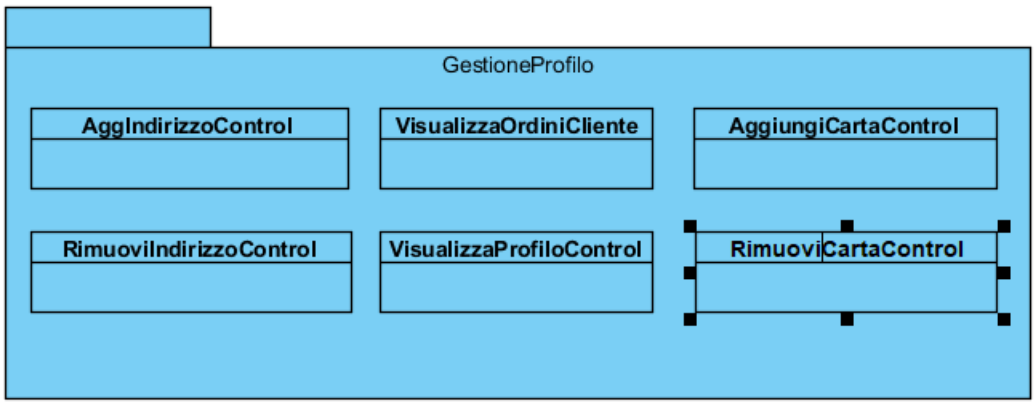


3.3.1 Package Accesso



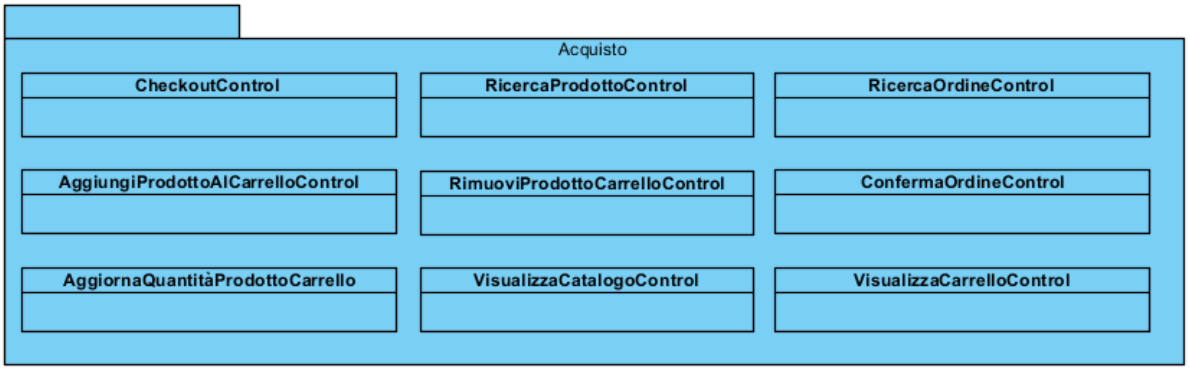
AccediControl	Permette di effettuare il login al sistema
LogoutControl	Permette di effettuare il logout dal sistema

3.3.2 Package GestioneProfilo



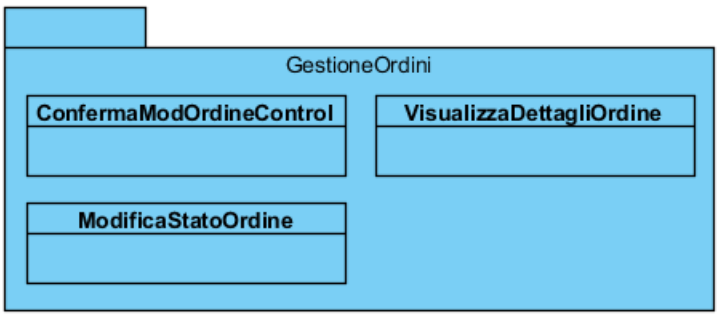
AggIndirizzoControl	Permette di aggiungere un nuovo indirizzo all'account
AggiungiCartaControl	Permette di aggiungere una nuova carta collegata all'account
VisualizzaOrdiniCliente	Permette di ottenere gli ordini eseguiti dal cliente
RimuoviIndirizzoControl	Permette la rimozione di un indirizzo
RimuoviCartaControl	Permette la rimozione di una carta
VisualizzaProfiloControl	Permette di visualizzare il profilo utente

3.3.3 Acquisto



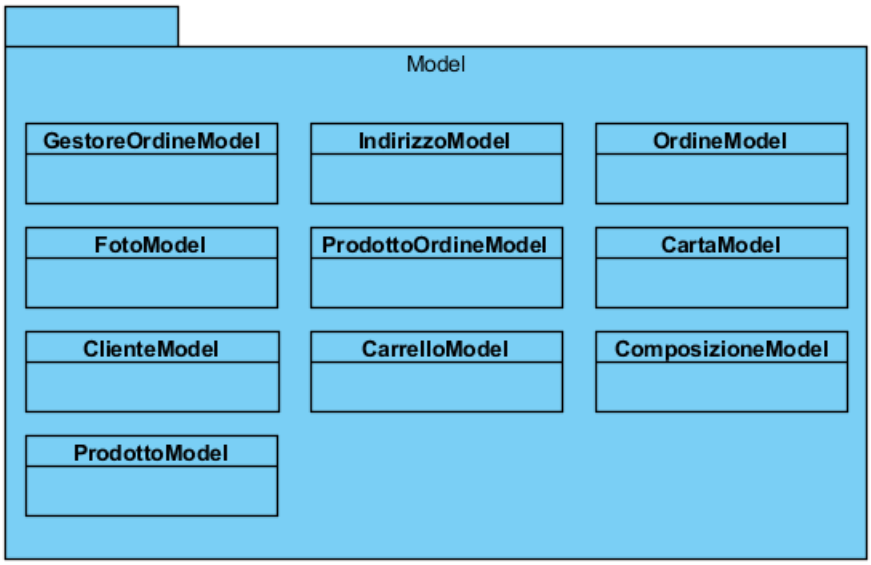
AggiungiProdottoAlCarrelloControl	Permette di aggiungere un prodotto al carrello
RimuovoProdottoCarrelloControl	Permette di rimuovere un prodotto dal carrello
RicercaProdottoControl	Permette di effettuare una ricerca di un prodotto
RicercaOrdineControl	Permette di effettuare la ricerca degli ordini
CheckoutControl	Permette di accedere all'area checkout
ConfermaOrdineControl	Permette di confermare l'ordine
VisualizzaCatalogoControl	Permette la visualizzazione di tutti i prodotti
AggiornaQuantitàProdottoCarrello	Permette di aggiornare la quantità dei prodotti presenti nel carrello
VisualizzaCarrelloControl	Permette la visualizzazione del carrello

3.3.4 GestioneOrdini



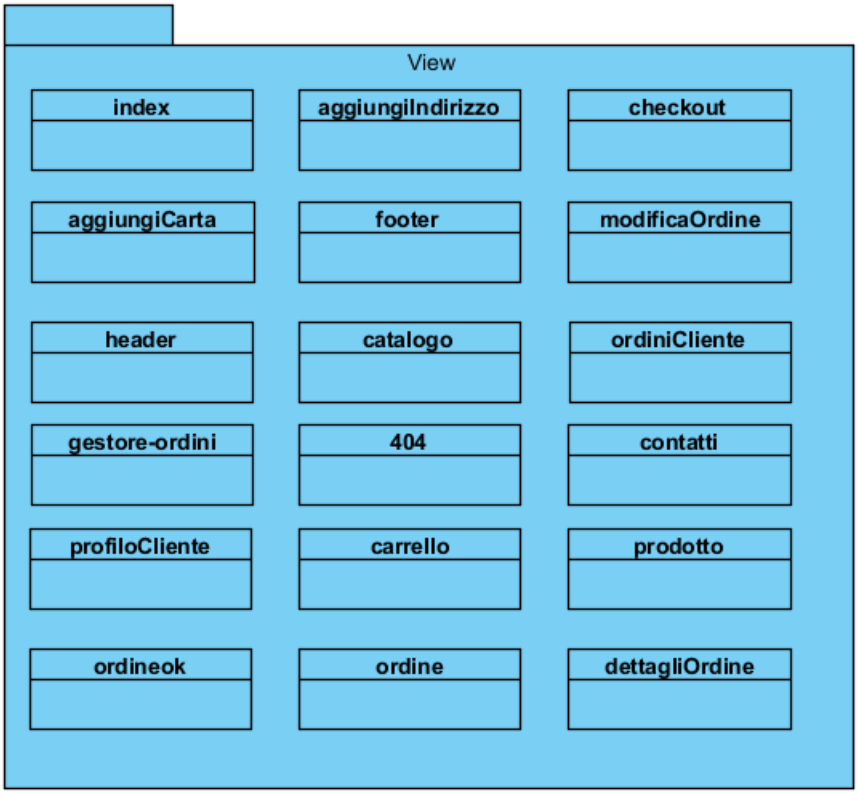
ModificaStatoOrdineControl	Permette la modifica dello stato di un ordine
ConfermaModOrdineControl	Permette la conferma della modifica dello stato di un ordine
VisualizzaDettagliOrdine	Permette la visualizzazione dei dettagli dell'ordine

3.4 PACKAGE MODEL



ClienteModel	Permette di effettuare le query riguardanti al cliente
ProdottoModel	Permette di effettuare le query riguardanti al prodotto
FotoModel	Permette di effettuare le query riguardanti alle foto dei prodotti
OrdineModel	Permette di effettuare le query riguardanti all'ordine
GestoreOrdineModel	Permette di effettuare le query riguardanti il GesotoreOrdine
ProdottoOrdineModel	Permette di effettuare le query riguardanti il ProdottoOrdine
IndirizzoModel	Permette di effettuare le query riguardanti all'indirizzo
CartaModel	Permette di effettuare le query riguardanti alla carta
ComposizioneModel	Permette di effettuare le query riguardanti alla composizione degli ordini
CarrelloModel	Permette di effettuare le query riguardanti al salvataggio persistente del carrello

3.5 PACKAGE VIEW



4 INTERFACCE DELLE CLASSI

4.1.1 AccediControl

Nome della classe	AccediControl
Descrizione	Questa classe è una servelt che si occupa del login dell'utente.
Metodi	+doGet(HttpServletRequest request,HttpServletResponse response): void +doPost(HttpServletRequest request, HttpServletResponse response): void
Metodo	**+doPost(HttpServletRequest request,HttpServletResponse response): void**
Descrizione	Questo metodo si occupa di effettuare il login da parte dell'utente utilizzando la coppia di stringhe user/password inserite dall'utente negli appositi campi di testo
Pre-Condizione	**Context:** AccediControl : doPost(request,response) **Pre:** request.getParameter("nick") !=null && request.getParameter("numPassword") !=null && request.getParameter("action") !=null && request.getParameter("i") !=null nick.length() >= 5 && nick.length() <=15 password.length()>=8 && password.length()<=20
Post-Condizione	If(cliente!=null || gestoreOrdini!=null) Then loginOK

4.1.2 LogoutControl

Nome della classe	LogoutControl
Descrizione	Questa classe è una servelt che si occupa del logout dell'utente.
Metodi	+doGet(HttpServletRequest request,HttpServletResponse response): void +doPost(HttpServletRequest request, HttpServletResponse response): void

Metodo	**+doGet(HttpServletRequest request,HttpServletResponse response): void**
Descrizione	Questo metodo permette il logout da parte dell'utente.

Pre-Condizione	**Context:** LogoutControl: doGet(request,response) **Pre:** request.getParameter("action") !=null
Post-Condizione	il cliente ha effettuato correttamente il logout dal sistema

4.1.3 AggiungiCartaControl

Nome della classe	AggiungiCartaControl
Descrizione	Questa classe è una servelt che si occupa di poter aggiungere una carta all'account dell'utente.
Metodi	+doGet(HttpServletRequest request,HttpServletResponse response): void +doPost(HttpServletRequest request, HttpServletResponse response): void

Metodo	+doPost(HttpServletRequest request,HttpServletResponse response): void
Descrizione	Questo metodo permette l'inserimento di una carta di credito utilizzando un form dove vengono specificati: proprietario, numero carte, data di scadenza e codice cv.
Pre-Condizione	**Context:** AggiungiCartaControl : doPost(request,response) **Pre**: request.getParameter("numCarta") !=null && request.getParameter("mese") !=null && request.getParameter("anno") !=null && request.getParameter("nomProrietario") !=null && cliente != null numCarta.matches("[0-9]{16}") && scadenza.matches("[0-9]{1,2}[/]{1}[0-9]{2}") && nomeProprietario.matches("[A-Za-z]{4,10}[ ]{0,1}[A-Za-z]{4,10}")
Post-Condizione	Il cliente ha aggiunto correttamente una nuova carta di credito.

4.1.4 RicercaOrdineControl

Nome della classe	RicercaOrdineControl
Descrizione	Questa classe è una servlet che si occupa di gestire la ricerca degli ordini
Metodi	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa di effettuare la ricerca degli ordini all'interno del database usando come riferimento il codice dell'ordine da ricercare.

Pre-condizione	**Context:**RicercaOrdine::doGet(request,response) **Pre:**request.getParameter("CodiceOrdine")!=null && ordiniUtente != null
Post-condizione	Il cliente visualizza tutti gli ordini da lui effettuati

4.1.5 VisualizzaOrdiniCliente

Nome della classe	**VisualizzaOrdiniControl**
Descrizione	Questa classe è una servelt che si occupa di far visualizzare gli ordini da parte dell'utente.
Metodi	`+doGet(HttpServletRequest request,HttpServletResponse response): void` `+doPost(HttpServletRequest request, HttpServletResponse response): void`

Metodo	**`+doGet(HttpServletRequest request,HttpServletResponse response): void`**
Descrizione	Questo metodo permette la visualizzazione degli ordini effettuati dal cliente.
Pre-Condizione	**Context:** VisualizzaOrdiniControl: doGet(request,response) **Pre:** `request.getParameter("nickname") != null && cliente != null`
Post-Condizione	Il cliente visualizza tutti gli ordini da lui effettuati

4.1.6 RicercaProdottoControl

Nome della classe	**RicercaProdottoControl**
Descrizione	Questa classe è una servlet che si occupa di gestire la ricerca di prodotti dal catalogo.
Metodi	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	**+doGet(HttpServletRequest request, HttpServletResponse response): void**
Descrizione	Questo metodo si occupa di effettuare la ricerca di prodotti all'interno del catalogo utilizzando la stringa inserita dall'utente nell'apposita barra di ricerca.
Pre-condizione	**Context:**RicercaProdottoControl :doGet(request,response) **Pre:** request.getParameter("stringaRicerca")!=null
Post-condizione	il cliente visualizza tutti i prodotti riguardanti la chiave di ricerca inserita

4.1.7 RimuoviProdortoCarrelloControl

Nome della classe	RimuoviProdottoControl
Descrizione	Questa classe è una servlet che permette di rimuovere un prodotto dal database
Metodi	+doGet(HttpServletRequest request, HttpServletResponse response): void +doPost(HttpServletRequest request, HttpServletResponse response): void

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response): void
Descrizione	Questo metodo si occupa della cancellazione del prodotto passando come parametro il codice del prodotto da eliminare.
Pre-condizione	**Context:**RimuoviProdotto::doGet(request,response) **Pre:**request.getParameter("CodiceProdotto")!=null && cart != null & prd !=null
Post-condizione	`il prodotto è stato rimosso dal carrello sizeCarrello() > sizeCarrelloPostRemove()`

4.1.8 AggiungiProdottoAlCarrelloControl

Nome della classe	AggiungiProdottoAlCarrelloControl
Descrizione	Questa classe è una servlet che permette l'aggiunta di un prodotto al carrello.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doGet (HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo si occupa di aggiungere un prodotto nel carrello del cliente.
Pre - condizione	**Context**: AggiungiProdottoAlCarrelloControl : doGet (request, response) **Pre** : cart != null && prodotto != null && `numdisp> 0`
Post - condizione	Il prodotto è stato aggiunto al carrello

4.1.9 CheckoutControl

Nome della classe	CheckoutControl
Descrizione	Questa classe salva le informazioni dell'ordine nel database.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doGet (HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo consente il salvataggio dei dati nel database.
Pre - condizione	**Context**: CheckoutControl :: doGet (request, response) **Pre** : utente != null && cart != null
Post - condizione	Il cliente visualizzerà la form per inserire metodo di pagamento e indirizzo di spedizione.

4.1.10 ConfermaOrdineControl

Nome della classe	ConfermaOrdineControl
Descrizione	Questa classe permette di confermare l'ordine.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doPost (HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo consente la conferma dell'ordine effettuato.
Pre - condizione	**Context**: ConfermaOrdineControl :: doPost (request, response) **Pre** : utente != null && carrello != null && codIndirizzo != null &&codCarta !=null `le carte e gli indirizzi devono essere associate all'utente presente nella sessione`
Post - condizione	Il cliente ha effettuato l'ordine, il quale apparirà nella pagina personale "i miei ordini"

4.1.11 ConfermaModOrdineControl

Nome della classe	ConfermaModOrdineControl
Descrizione	Questa classe permette la conferma della modifica dello stato di un ordine.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doGet (HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo consente la conferma della modifica dello stato dell'ordine effettuato.
Pre - condizione	**Context**: ConfermaModOrdineControl :: doGet(request, response) **Pre** : gestore != null dataConsegna != null && corriere != null && numeroTracking != null && ordine != null (presente nel database) `dataConsegna.matches("[0-9]{2}[/]{1}[0-9]{2}[/]{1}[0-9]{4}") && numeroTracking.matches("[0-9]{10}")&&(corriere.matches("[A-Za-z]{3,20}")`

Post - condizione	Lo stato di un ordine è passato da “In preparazione” a “Spedito”

4.1.12 ModificaStatoOrdineControl

Nome della classe	**ModificaStatoOrdineControl**
Descrizione	Questa classe permette la modifica dello stato di un ordine.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	**+doGet (HttpServletRequest request, HttpServletResponse response) : void**
Descrizione	Questo metodo consente la dello stato di un ordine.
Pre - condizione	**Context**: ModificaStatoOrdineControl :: doGet(request, response) **Pre** : Gestore != null && codiceOrdine != null && ordine != null (presente nel database)
Post - condizione	Il gestore visualizzerà a video la form per modificare lo stato di un ordine.

4.1.13 VisualizzaCatalogoControl

Nome della classe	**VisualizzaCatalogoControl**
Descrizione	Questa classe è una servlet che permette di caricare tutti i prodotti presenti nel catalogo
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	**+doGet (HttpServletRequest request, HttpServletResponse response) : void**
Descrizione	Questo metodo carica tutti i prodotti presenti nel catalogo.
Pre - condizione	**Context**: VisualizzaCatalogoControl :: doGet(request, response) **Pre** :
Post - condizione	Il cliente visualizzarà tutti i prodotti presenti nel catalogo.

4.1.14 AggIndirizzoControl

Nome della classe	AggIndirizzoControl
Descrizione	Questa classe è una servlet che gestisce l'inserimento di un nuovo indirizzo da parte del cliente
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doGet (HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo gestisce l'inserimento di un nuovo indirizzo da parte di un cliente.
Pre - condizione	**Context**: AggIndirizzoControl:: doGet(request, response) **Pre** :cliente != null && nome != null && cognome != null && indirizzo != null && città != null && cap != null && telefono != null `nome.matches("[a-zA-Z]{4,10}[ ]{0,1}[a-zA-Z]{0,10}") && cognome.matches("[a-zA-Z]{4,10}[ ]{0,1}[a-zA-Z]{0,10}") &&città.matches("[a-zA-Z]{4,10}[ ]{0,1}[a-zA-Z]{0,10}")&& indirizzo.matches"[a-zA-Z0-9]{3,6}[ ]{0,1}[a-zA-Z0-9]{0,10}[ ]{0,1}[a-zA-Z0-9]{0,4}")&& cap.matches("[0-9]{5}") && telefono.matches("[0-9]{10}")`
Post - condizione	`Il cliente ha aggiunto correttamente un nuovo indirizzo.`

4.1.15 RimuoviIndirizzoControl

Nome della classe	RimuoviIndirizzoControl
Descrizione	Questa classe è una servlet che gestisce la rimozione di un indirizzo da parte del cliente
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doPost(HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo che gestisce la rimozione di un indirizzo da parte del cliente
Pre-condizione	**Context**: RimuoviIndirizzoControl:: doPost(request, response) **Pre:** cliente!=null && codice!=null
Post-condizione	Il cliente ha rimosso correttamente l'indirizzo

4.1.16 RimuoviCartaControl

Nome della classe	RimuoviCartaControl
Descrizione	Questa classe è una servlet che gestisce la rimozione di una carta da parte del cliente
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doPost(HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo che gestisce la rimozione di una carta da parte del cliente
Pre-Condizione	**Context**: RimuoviCartaControl:: doPost(request, response) **Pre:** cliente!=null && codice!=null
Post-Condizione	Il cliente ha rimosso correttamente la carta

4.1.17 AggiornaQuantitàProdottoCarrello

Nome della classe	AggiornaQuantitàProdottoCarrello
Descrizione	Questa classe è una servlet che gestisce l'aggiornamento della quantità di un prodotto già presente nel carrello.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doPost(HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo permette l'aggiornamento della quantità di un prodotto già presente nel carrello.
Pre-Condizione	**Context** AggiornaQuantitàProdottoCarrello: doPost(request, response) `cart != null && q != null && c != null && quantità >0 && quantità > numdisponibilità`
Post-Condizione	se nel carrello c'è un prodotto con il codice uguale a quello passato allora la quantità del prodotto nel carrello viene incrementata

4.1.18 VisualizzaDettagliOrdineControl

Nome della classe	VisualizzaDettagliOrdineControl
Descrizione	Questa classe è una servlet che gestisce la visualizzazione di un ordine da parte del gestore.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doPost(HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo permette di visualizzare l'ordine con con codice passato come parametro.
Pre-Condizione	**Context** VisualizzaDettagliOrdineControl: doPost(request, response) `codice != null && ordine != null && gestore != null`
Post-Condizione	Ordine != null e il gestore visualizza i dettagli dell'ordine

4.1.19 VisualizzaCarrelloControl

Nome della classe	VisaulizzaCarrelloControl
Descrizione	Questa classe è una servlet che gestisce la visualizzazione del carrello.
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doPost(HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo permette di accedere alla pagina che permette di visualizzare il carrello
Pre-Condizione	**Context** VisualizzaDettagliOrdineControl: doPost(request, response)
Post-Condizione	L'utente visualizza il carrello

4.1.20 VisualizzaProfiloClienteControl

Nome della classe	VisualizzaProfiloControl
Descrizione	Questa classe è una servlet permette di visualizzare il profilo del cliente
Metodi	+doGet (HttpServletRequest request, HttpServletResponse response): void +doPost (HttpServletRequest request, HttpServletResponse response) : void

Nome del metodo	+doPost(HttpServletRequest request, HttpServletResponse response) : void
Descrizione	Questo metodo permette di accedere alla pagina che permette di visualizzare il carrello
Pre-Condizione	**Context** VisualizzaDettagliOrdineControl: doPost(request, response) Cliente != null
Post-Condizione	Il clinete visualizza il profilo

4.1.21 CarrelloModel

Nome della classe	CarrelloModel
Descrizione	Questa classe gestisce l'informazione persistente carello nel database
Metodi	doSave(List<ProdottoCatalogoBean> list, UtenteBean utente) leggi(String nickname) remove(UtenteBean utente)

Nome del metodo	+ doSave(List<ProdottoCatalogoBean> list, UtenteBean utente): void
Descrizione	Questo metodo consente il salvataggio del carrello nel database
Pre - condizione	List != null && utente != null
Post - condizione	Il sistema memorizza il carrello nel database

Nome del metodo	+ leggi(String nickname): ArrayList<ProdottoCatalogoBean>
Descrizione	Questo metodo restituisce i prodotti del carrello del cliente passato come parametro
Pre - condizione	Nickname != null
Post - condizione	Il metodo ritorna l'ArrayList di prodotti.

Nome del metodo	+remove(UtenteBean utente): void
Descrizione	Questo metodo rimuove il carrello associato all'utente dal database
Pre - condizione	utente != null
Post - condizione	Se presente rimuove il carrello dal database

4.1.22 ProdottoOrdineModel

Nome della classe	ProdottoOrdineModel
Descrizione	Questa classe gestisce l'informazione persistente del prodotto dell'ordine nel database
Metodi	`doSave(ArrayList<ProdottoCatalogoBean> prodotti)` `prodottiOrdine(int codiceOrdine)`
Nome del metodo	**`+doSave(ArrayList<ProdottoCatalogoBean> prodotti): void`**
Descrizione	Questo metodo memorizza i prodotti dell'ordine nel database
Pre - condizione	prodotti != null
Post - condizione	Memorizza i prodotti nel database

Nome del metodo	**`+ prodottiOrdine(int codiceOrdine):ArrayList<ProdottoOrdineBean>`**
Descrizione	Questo metodo restituisce i prodotti dell'ordine con codice passato come parametro
Pre - condizione	codiceOrdine != null
Post - condizione	Restiuisce l'ArrayList di prodotti dell'ordine

4.1.23 OrdineModel

Nome della classe	OrdineModel
Descrizione	Questa classe gestisce l'informazione persistente ordine nel database
Metodi	doRetrieveByKey(**int** code) doRetrieveByStato(String stato) doSave(IndirizzoBean indirizzo, CartaBean carta, String stato,String corriere, String tracking, **double** totale, String cliente) ordiniUtente(String nickname) listaOrdini() aggiorna(**int** cod, String num_tracking, Date dataConsegna, String corriere) aggiornaConsegnato(int codice)

Nome del metodo	+ doRetrieveByKey(int code):OrdineBean
Descrizione	Questo metodo ritorna l'ordine in base al codice passato come parametro
Pre - condizione	code != null
Post - condizione	ordine != null se trova un ordine corrispondente al codice passato || ordine == null se non trova l'ordine

Nome del metodo	+ doRetrieveByStato(String stato):ArrayList<OrdineBean>
Descrizione	Questo metodo ritorna gli ordini in base allo stato passato come parametro
Pre - condizione	stato != null
Post - condizione	ordini.size() > 0 se trova ordini con lo stato passato come parametro || ordini.size() == 0 se non trova gli ordini

Nome del metodo	+ doSave(IndirizzoBean indirizzo, CartaBean carta, String stato,String corriere, String tracking, double totale, String cliente) : void
Descrizione	Questo metodo salva l'ordine nel database
Pre - condizione	Indirizzo != null && carta != null && stato != null && corriere != null && tracking != null && totale != null && cliente != null
Post - condizione	Memorizza l'ordine nel database

Nome del metodo	`+ ordiniUtente(String nickname): ArrayList<OrdineBean>`
Descrizione	Questo metodo ritorna gli ordini dell'utente passato come parametro
Pre - condizione	nickname != null
Post - condizione	Ritorna ordini.size() > 0 se trova ordini relativi all'utente passato come parametro || Ritorna ordini.size() == 0 se non trova gli ordini

Nome del metodo	`+ listaOrdini(): ArrayList<OrdineBean>`
Descrizione	Questo metodo ritorna gli ordini memorizzati nel database
Pre - condizione	//
Post - condizione	ordini.size() > 0 se trova ordini memorizzati nel database || ordini.size() == 0 se non trova gli ordini

Nome del metodo	`+ aggiorna(int cod, String num_tracking, Date dataConsegna, String corriere): void`
Descrizione	Questo metodo aggiorna le informazioni relative all'ordine da in preparazione a spedito
Pre - condizione	cod != null && num_tracking != null && dataConsegna != null && corriere != null
Post - condizione	Modifica le informazioni dell'ordine da in preparazione a spedito

Nome del metodo	`+ aggiornaConsegnato(int cod): void`
Descrizione	Questo metodo aggiorna le informazioni relative all'ordine da spedito a consegnato
Pre - condizione	cod != null
Post - condizione	Modifica le informazioni dell'ordine da spedito a consegnato

4.1.24 IndirizzoModel

Nome della classe	IndirizzoModel
Descrizione	Questa classe gestisce l'informazione persistente indirizzo nel database
Metodi	`doSave(IndirizzoBean indirizzo, String nickname)` `leggi(String nickname)`

Nome del metodo	`+ doSave(IndirizzoBean indirizzo, String nickname): void`
Descrizione	Questo metodo salva l'indirizzo nel database
Pre - condizione	Indirizzo!=null && nickname!=null
Post - condizione	Memorizza l'indirizzo nel database

Nome del metodo	`+ leggi(String nickname) : ArrayList<IndirizzoBean>`
Descrizione	Restituisce gli indirizzi dell'utente passato come parametro.
Pre - condizione	Nickname != null
Post - condizione	Ritorna indirizzi.size() > 0 se il cliente ha degli indirizzi memorizzati || Ritorna indirizzi.size() == 0 se non ci sono indirizzi memorizzati per il cliente

4.1.25 ProdottoCatalogoModel

Nome della classe	ProdottoCatalogoModel
Descrizione	Questa classe gestisce l'informazione persistente prodotto nel database

Metodi	`doSave( int codice, int numDisp, String nome, String colore, String marca, String descrizione, int peso, double Prezzo, java.sql.Date data, String strumento )`
	`generaCodice()`
	`doDelete(int codice)`
	`doRetriveAll()`
	`doRetriveByInstruments(String strumento)`
	`doRetriveByMarca(String marca)`
	`doRetriveByKey(int codice)`
	`doRetriveByName(String nome)`
	`aggiorna(ArrayList<ProdottoCatalogoBean>)`

Nome del metodo	`+ doSave(int codice, int numDisp, String nome, String colore, String marca, String descrizione, int peso, double Prezzo, java.sql.Date data, String strumento) : void`
Descrizione	Salva i prodotti nel database
Pre - condizione	codice!=null && numDisp>0 && nome!=null && colore!=null && marca!=null && descrizione!=null && peso>0, prezzo>0, data!=null && strumento!=null
Post - condizione	Memorizza i prodotti nel database

Nome del metodo	`+ generaCodice(): int`
Descrizione	Questo metodo genera l'id per l'oggetto da memorizzare nel database
Pre - condizione	//
Post - condizione	Restituisce l'intero che rappresenta l'id

Nome del metodo	`+ aggiorna(ArrayList<ProdottoCatalogo prodotti) : void`
Descrizione	Metodo che aggiorna i prodotti nel database
Pre - condizione	prodotti.size()>0
Post - condizione	`I prodotti sono stati aggiornati.`

Nome del metodo	+ doDelete(int codice) : void
Descrizione	Elimina un prodotto dal database
Pre - condizione	Codice != null
Post - condizione	Il prodotto è stato eliminato

Nome del metodo	+ doRestriveAll() : ArrayList<ProdottoCatalogoBean>
Descrizione	Metodo che restituisce tutti i prodotti presenti nel database nella tabella prodottoCatalogo
Pre - condizione	//
Post - condizione	Prodotti.size() > 0 se ci sono presenti dei prodotti memorizzati nel database || Prodotti.size() == 0 se non ci sono dei prodotti memorizzati nel database

Nome del metodo	+ doRestriveByInstruments(String strumento) : ArrayList<ProdottoCatalogoBean>
Descrizione	Metodo che restituisce i prodotti in base al tipo di strumento
Pre - condizione	strumento!=null
Post - condizione	Prodotti.size() > 0 se ci sono presenti dei prodotti memorizzati nel database del tipo passato come parametro || Prodotti.size() == 0 se non ci sono dei prodotti memorizzati nel database del tipo passato come parametro

Nome del metodo	+ doRestriveByKey(int codice) : ProdottoCatalogoBean
Descrizione	Metodo che restituisce il prodotto in base al codice passato come parametro
Pre - condizione	Codice != null
Post - condizione	Prodotto != null se è prodotto.getCodice() == codice || Prodotto == null se nel database non ci sono prodotti con quel codice

Nome del metodo	+ doRestriveByMarca(String marca) : ArrayList<ProdottoCatalogoBean>
Descrizione	Metodo che restituisce i prodotti in base alla marca passato come parametro
Pre - condizione	marca!=null
Post - condizione	Prodotti.size() > 0 se ci sono presenti dei prodotti memorizzati nel database della marca passato come parametro || Prodotti.size() == 0 se non ci sono dei prodotti memorizzati nel database della marca passato come parametro

Nome del metodo	+ doRestriveByName(String nome) : ArrayList<ProdottoCatalogoBean>
Descrizione	Metodo che restituisce i prodotti in base al nome passato come parametro
Pre - condizione	nome!=null
Post - condizione	Prodotti.size() > 0 se ci sono presenti dei prodotti memorizzati nel database con il nome passato come parametro || Prodotti.size() == 0 se non ci sono dei prodotti memorizzati nel database con il nome passato come parametro

4.1.26 CartaModel

Nome della classe	CartaModel
Descrizione	Questa classe gestisce l'informazione persistente carta di credito nel database
Metodi	`doSave(CartaBean carta, ClienteBean cliente)` `leggi(String nickname)`

Nome del metodo	`+ doSave(CartaBean carte, ClienteBean cliente) : void`
Descrizione	Questo metodo salva la carta di credito nel database
Pre - condizione	carta!=null && cliente!=null
Post - condizione	Memorizza la carta di credito nel database

Nome del metodo	`+ leggi(String nickname) : ArrayList<CartaBean>`
Descrizione	Metodo che legge le carte di credito di un determinato utente nel database.
Pre - condizione	nickname!=null
Post - condizione	`carte.size() > 0` se il cliente ha associato delle carte || `carte.size() == 0` se non ci sono associate delle carte al cliente

4.1.27 COMPOSIZIONE MODEL

Nome della classe	ComposizioneModel
Descrizione	Questa classe permette il salvataggio della coppia di chiavi ordine, prodottoOrdine all'interno del database.
Metodi	`doSave(ArrayList <ProdottoCatalogoBean> prodotti, int codOrdine)`

Nome del metodo	+ doSave(ArrayList <ProdottoCatalogoBean> prodotti): int codOrdine
Descrizione	Questo metodo permette il salvataggio della coppia di chiavi ordine, prodottoOrdine all'interno del database.
Pre - condizione	prodotti!=null && codOrdine!=null
Post - condizione	Il sistema memorizza la coppia ordine e prodottoOrdine nel database.

4.1.22 FOTO MODEL

Nome della classe	FotoModel
Descrizione	Questa classe permette il salvataggio di un'immagine nel database.
Metodi	doSave(String url, int cod) generaCodice()

Nome del metodo	+ doSave(String url, int cod)
Descrizione	Questo metodo permette il salvataggio di un immagine nel database.
Pre - condizione	url!=null && cod!=null
Post - condizione	Il sistema memorizza il path dell'immagine nel database.

Nome del metodo	+ generaCodice()
Descrizione	Questo metodo genera l'id dell'immagine da memorizzare nel database.
Pre - condizione	//
Post - condizione	Restituisce l'ID.

4.1.28 ClienteModel

Nome della classe	ClienteModel
Descrizione	Questa classe gestisce l'informazione persistente cliente nel database.
Metodi	`doSave(ClienteBean cliente)` `leggi(String nickname,String pwd)`

Nome del metodo	+ doSave(ClienteBean cliente) : void
Descrizione	Questo metodo salva il cliente nel database.
Pre - condizione	Cliente != null
Post - condizione	Memorizza il cliente nel database

Nome del metodo	+ leggi(String nickname, String pwd) : cliente
Descrizione	Metodo che restituisce il cliente se presente nel database
Pre - condizione	nickname!=null && pwd!=null
Post - condizione	cliente != null se è presente nel database || cliente == null se non è presente nel database

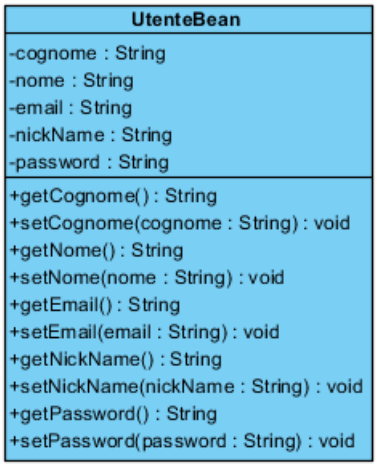
4.1.29 GestoreOrdineModel

Nome della classe	gestoreOrdineModel
Descrizione	Questa classe gestisce l'informazione persistente gestore dell'ordine nel database.
Metodi	`leggi(String nickname,String pwd)`

Nome del metodo	+ leggi(String nickname,String pwd) : gestoreOrdine
Descrizione	Metodo che restituisce il gestore se presente nel database
Pre - condizione	Nickname != null && pwd !=null
Post - condizione	gestore != null se è presente nel database || gestore == null se non è presente nel database

4.2 DESCRIZIONE DELLE CLASSI

4.2.1 UtenteBean



Nickname : nickname dell'utente per effettuare l'accesso.

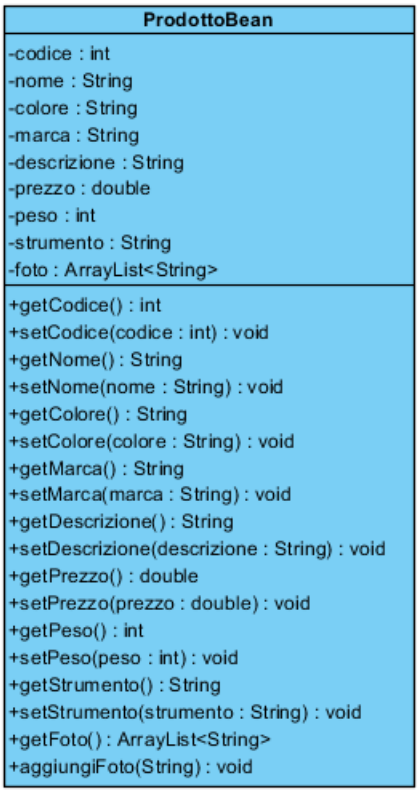
Password : la password che utilizza l'utente per l'accesso.

Nome : nome dell'utente.

Cognome : cognome dell'utente.

Email : e-mail dell'utente.

4.2.2 ProdottoBean



codice : codice che identifica univocamente il prodotto nel sistema.

nome: Nome del prodotto.

descrizione: Descrizione dettagliata del prodotto.

colore: Colorazione in cui è disponibile il prodotto.

peso: Peso del prodotto.

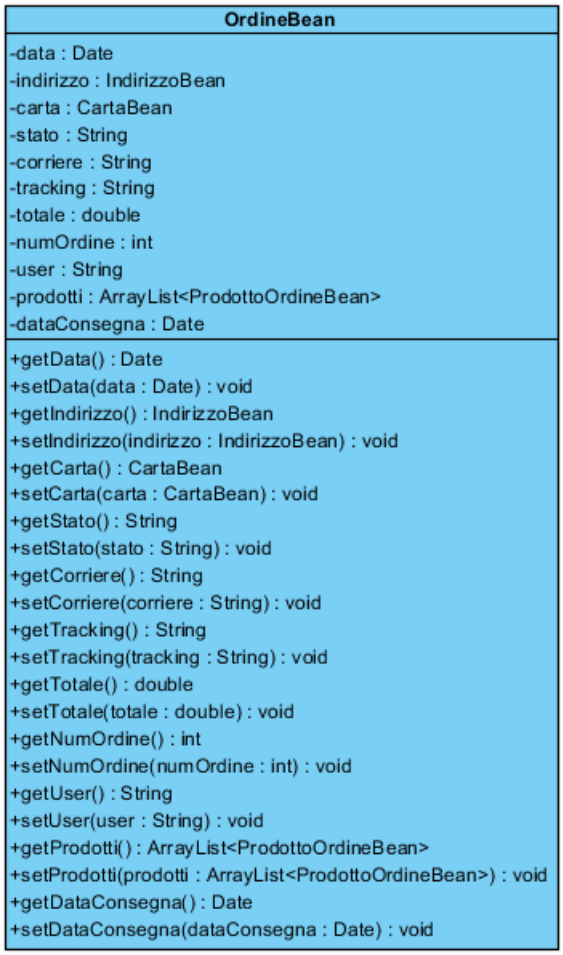
marca: Marca del prodotto.

strumento: Famiglia degli strumenti di cui fa parte il prodotto.

prezzo: prezzo del prodotto.

foto: path delle foto del prodotto.

4.2.3 OrdineBean



numOrdine: identifica univocamente l'ordine.

stato: rappresenta lo stato dell'ordine che può trovarsi "in preparazione" o "spedito".

corriere: rappresenta l'informazione relativa al corriere per la spedizione.

traking: numero che permette di rintracciare l'ordine durante la spedizione.

totale: rappresenta il totale dell'ordine.

indirizzo: rappresenta l'indirizzo dove l'ordine viene spedito.

carta: rappresenta la carta con cui è stato pagato l'ordine.

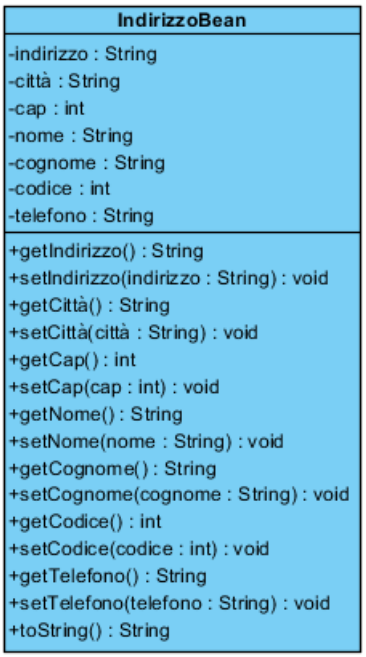
prodotti: rappresentano i prodotti di cui è composto l'ordine.

data: rappresenta la data in cui è stato effettuato l'ordine.

user: rappresenta il nickname dell'utente che ha effettuato l'ordine.

dataConsegna: rappresenta la data in cui è prevista la consegna.

4.2.4 IndirizzoBean



codice: Id dell'indirizzo identifica univocamente un indirizzo nel database.

indirizzo: Indirizzo per la spedizione.

città: città per la spedizione

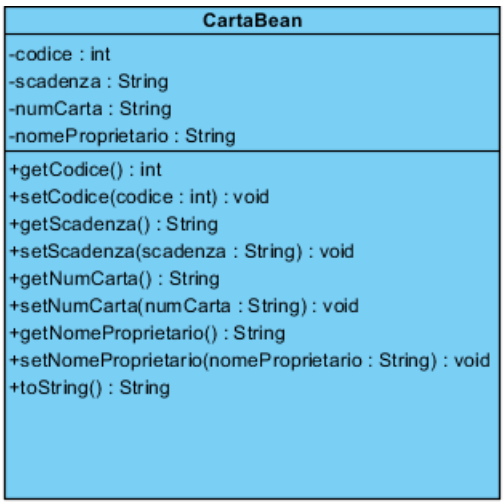
cap: Codice di avviamento postale della città dell'indirizzo fornito.

telefono: numero di telefono

nome: nome del destinatario.

cognome: cognome del destinatario.

4.2.5 CartaBean



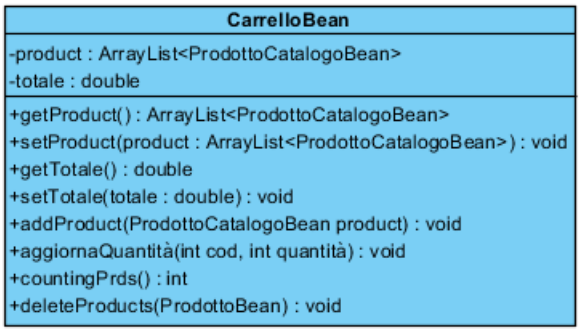
Codice: codice che identifica univocamente la carta.

Scadenza: scadenza della carta.

NumeroCarta: numero della carta prepagata.

NomeProrpeitario: nome e cognome dell'intestatario della carta.

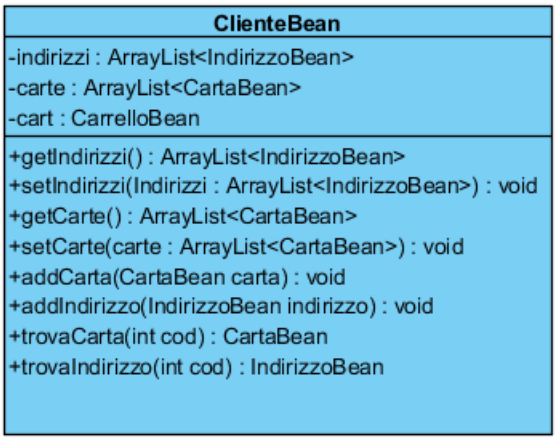
4.2.6 CarrelloBean



Prodotti: prodotti presenti nel carrello.

Totale: prezzo totale del carrello.

4.2.7 ClienteBean

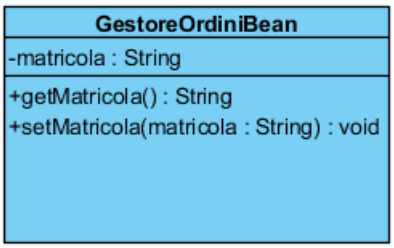


indirizzi: rappresenta tutti gli indirizzi di spedizione che un cliente ha sul suo profilo.

carte: rappresenta tutte le carte di credito che un cliente ha sul suo profilo.

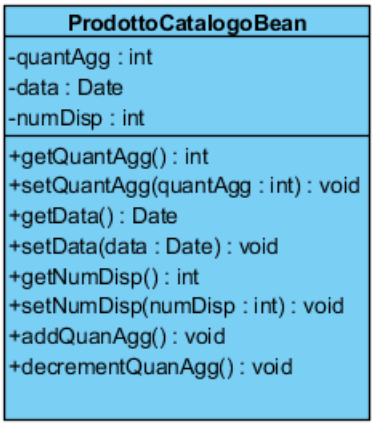
cart: rappresenta il carrello di un cliente.

4.2.8 GestoreOrdiniBean



matricola: rappresenta la matricola di un gestore.

4.2.9 ProdottoCatalogo

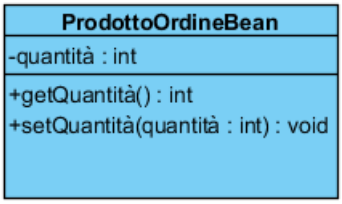


quantAgg: rappresenta la quantità di un prodotto nel carrello.

data: rappresenta la data di inserimento del prodotto nel catalogo.

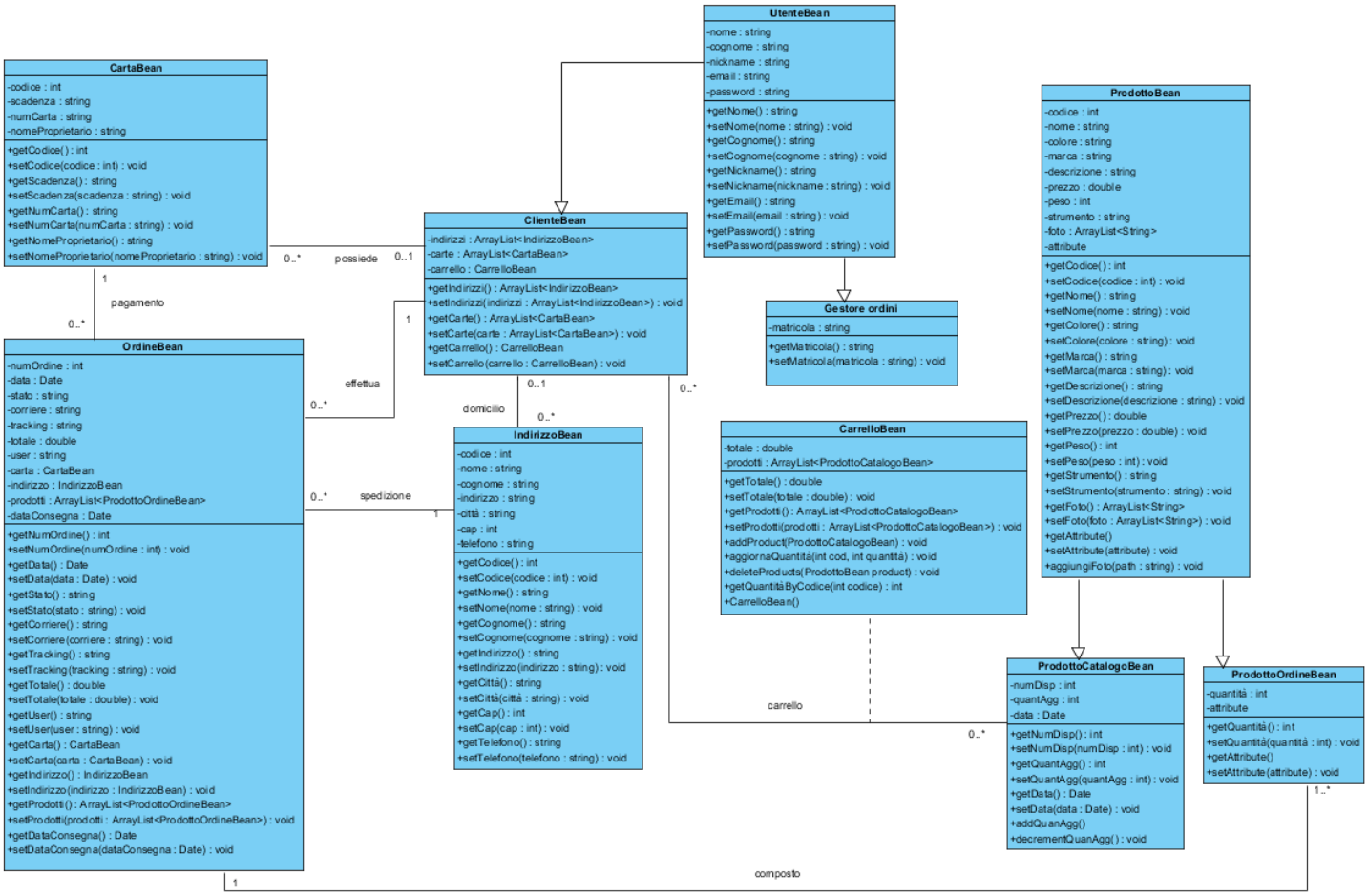
numDisp: rappresenta la quantità di un prodotto disponibile.

4.2.10 ProdottoOrdineBean



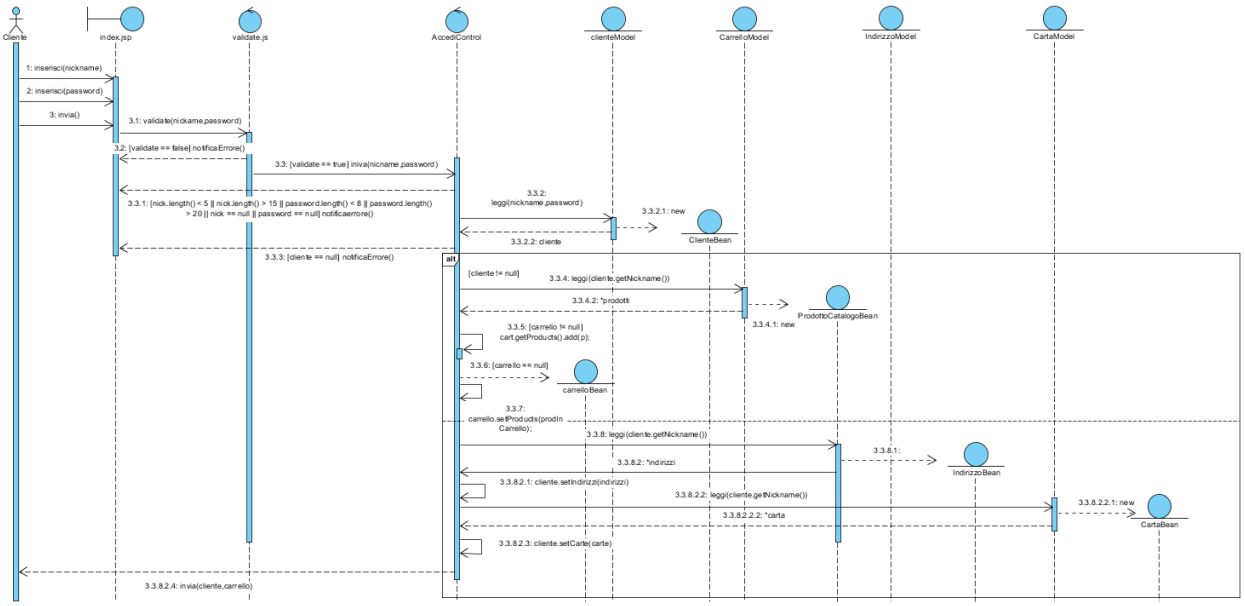
quantità: rappresenta la quantità del prodotto in un ordine.

5 CLASS DIAGRAM

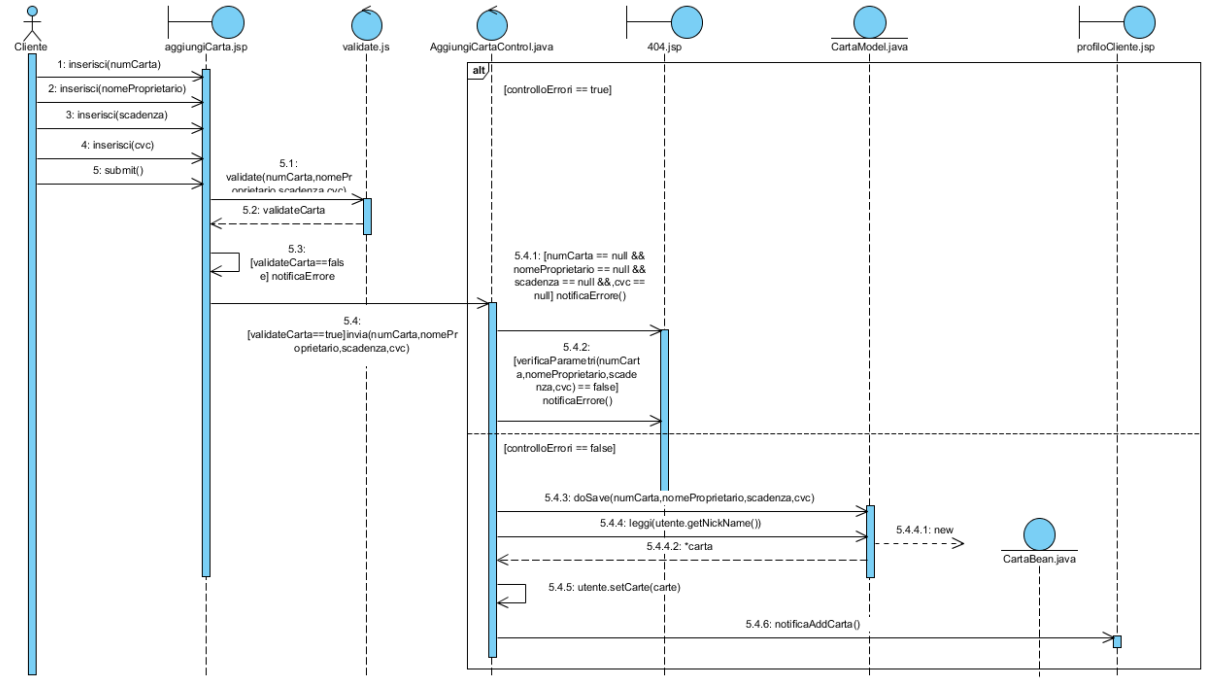


6 SEQUENCE DIAGRAM REVISIONATI

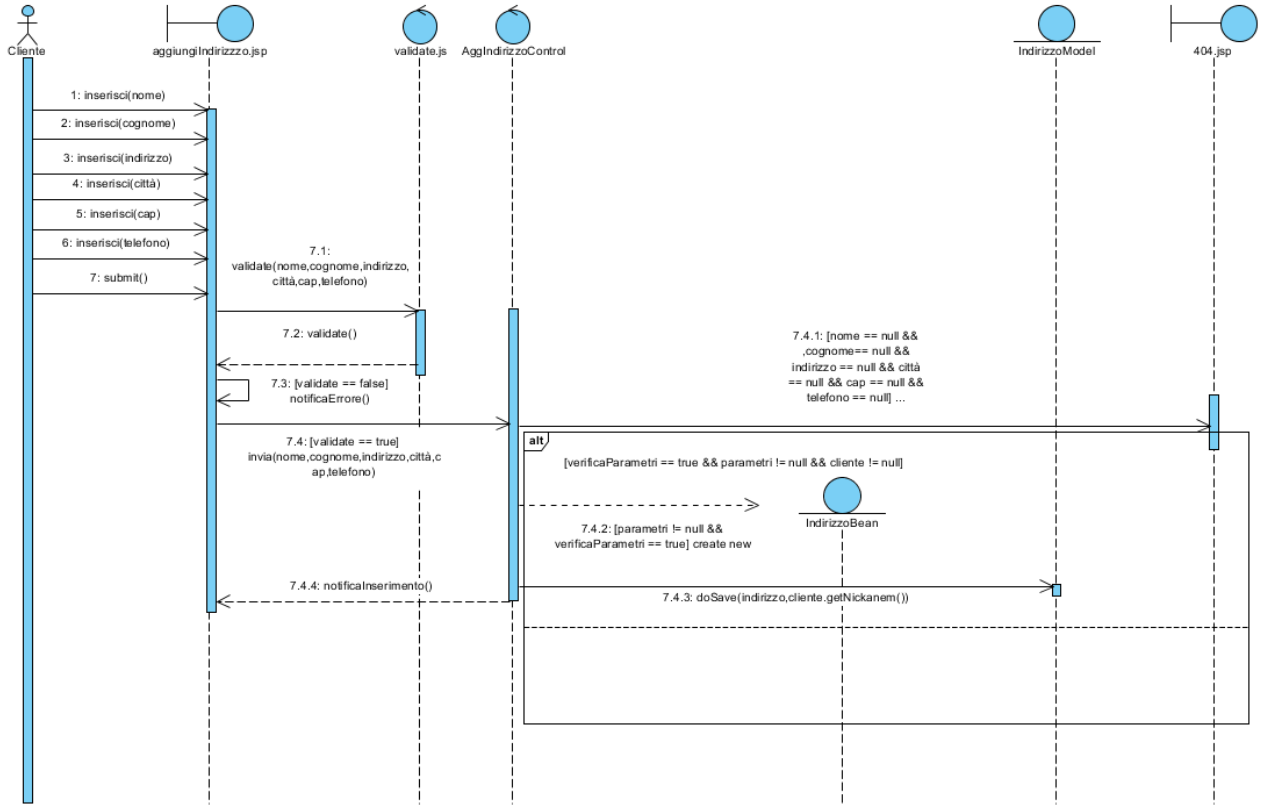
6.1 LOGIN



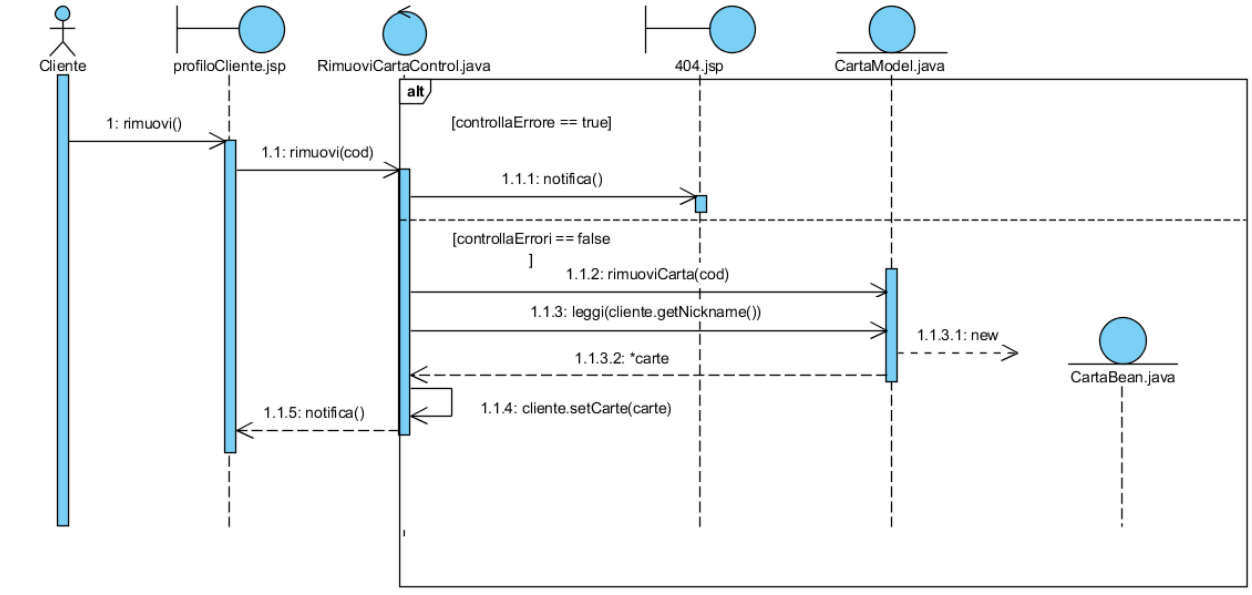
6.2 AGGIUNGI CARTA



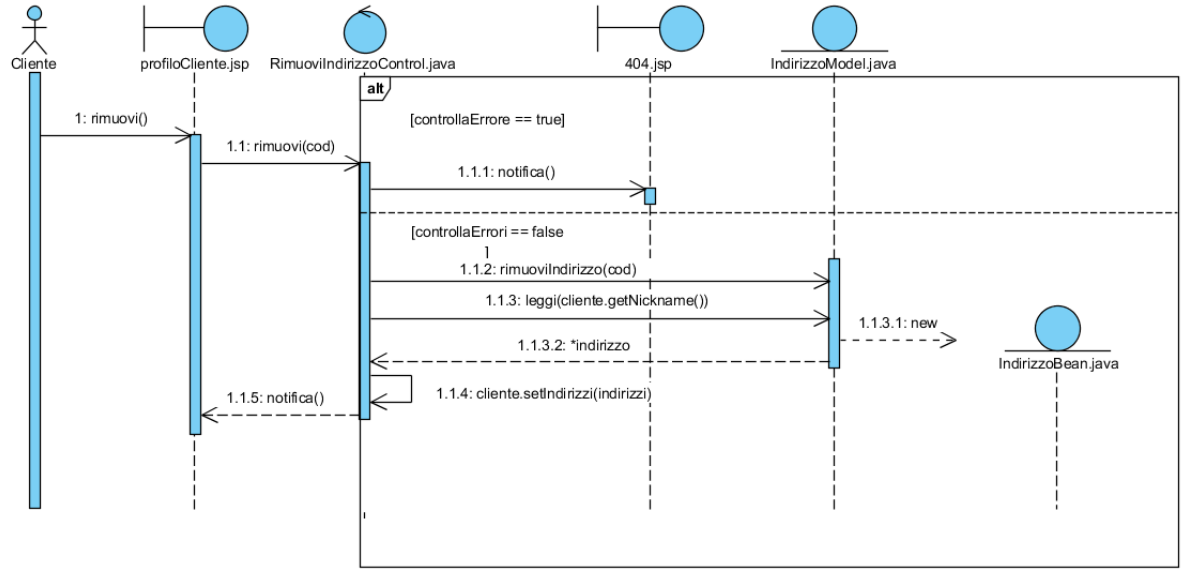
6.3 AGGIUNGI INDIRIZZO



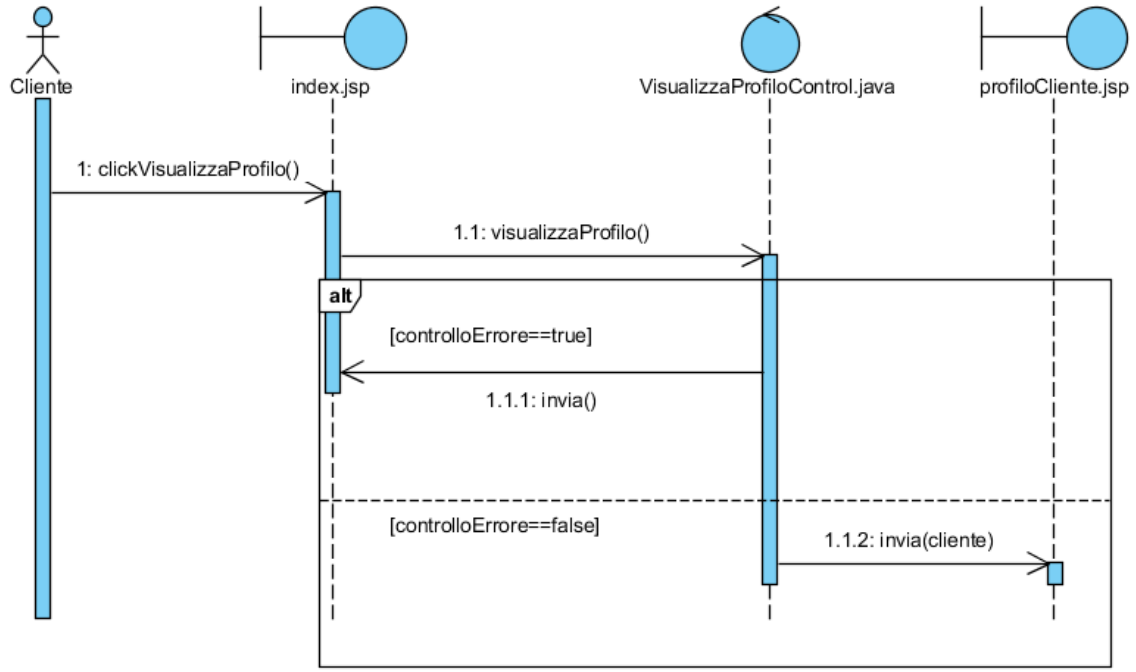
6.4 RIMUOVI CARTA



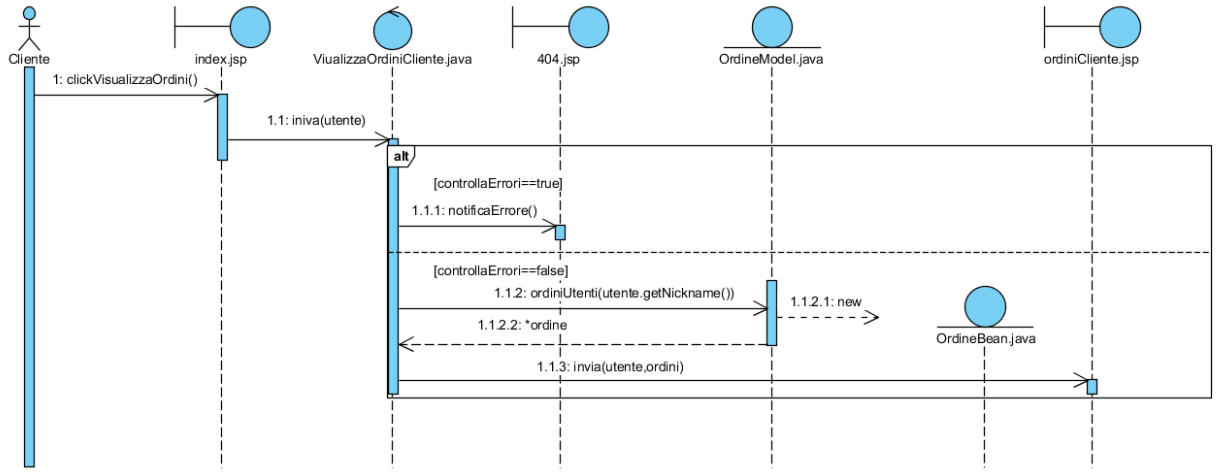
6.5 RIMUOVI INDIRIZZO



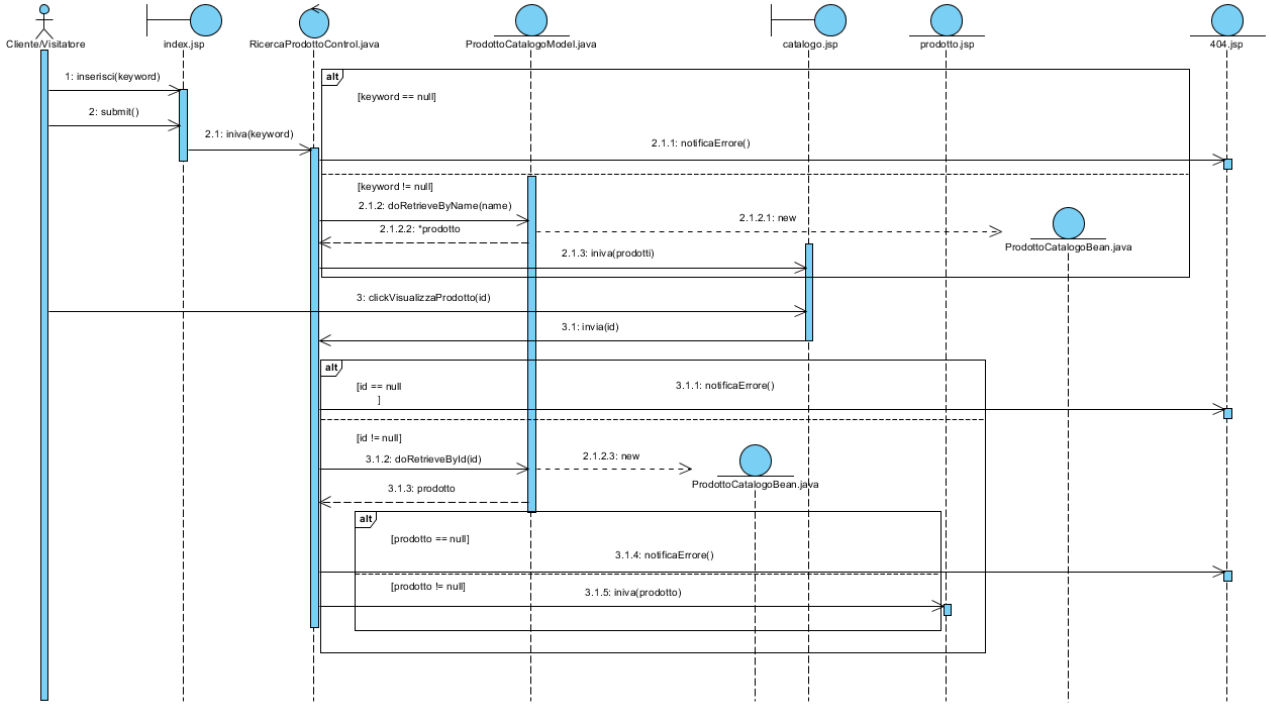
6.6 VISUALIZZA PROFILO



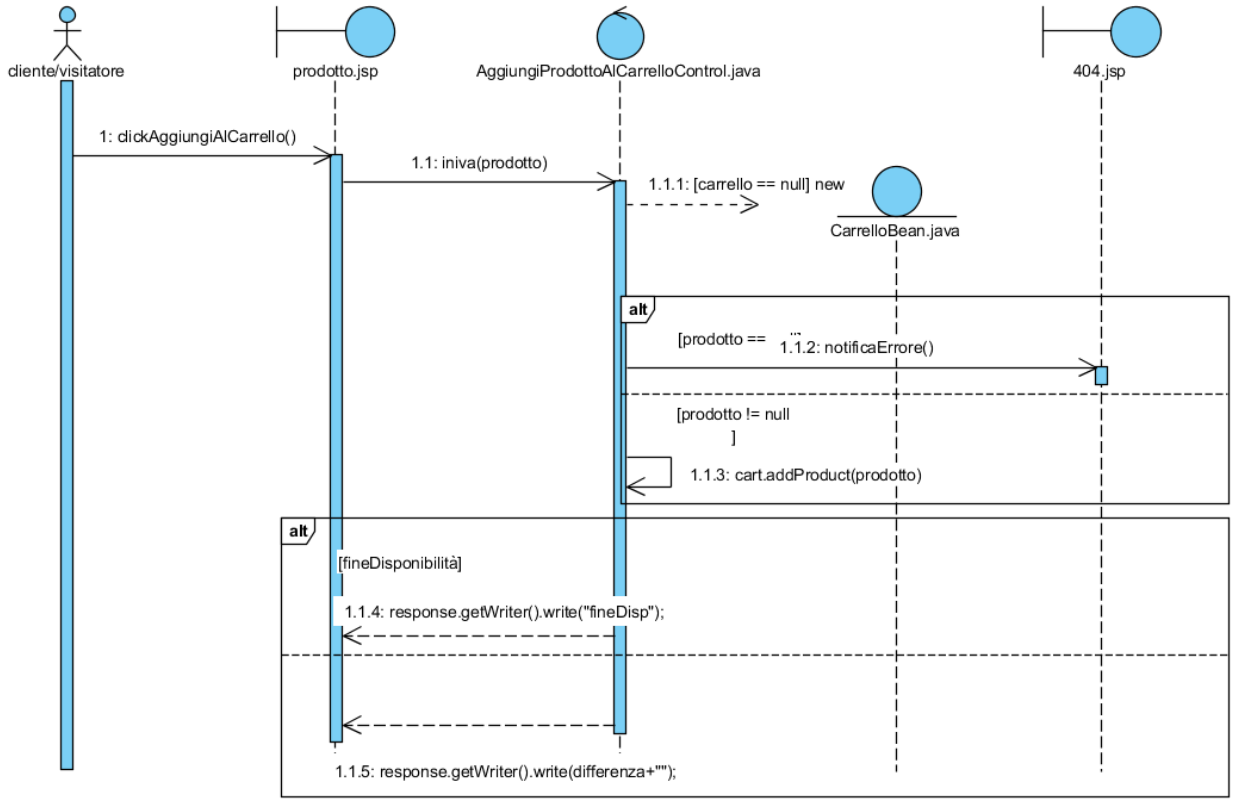
6.7 VISUALIZZA STORICO ORDINI



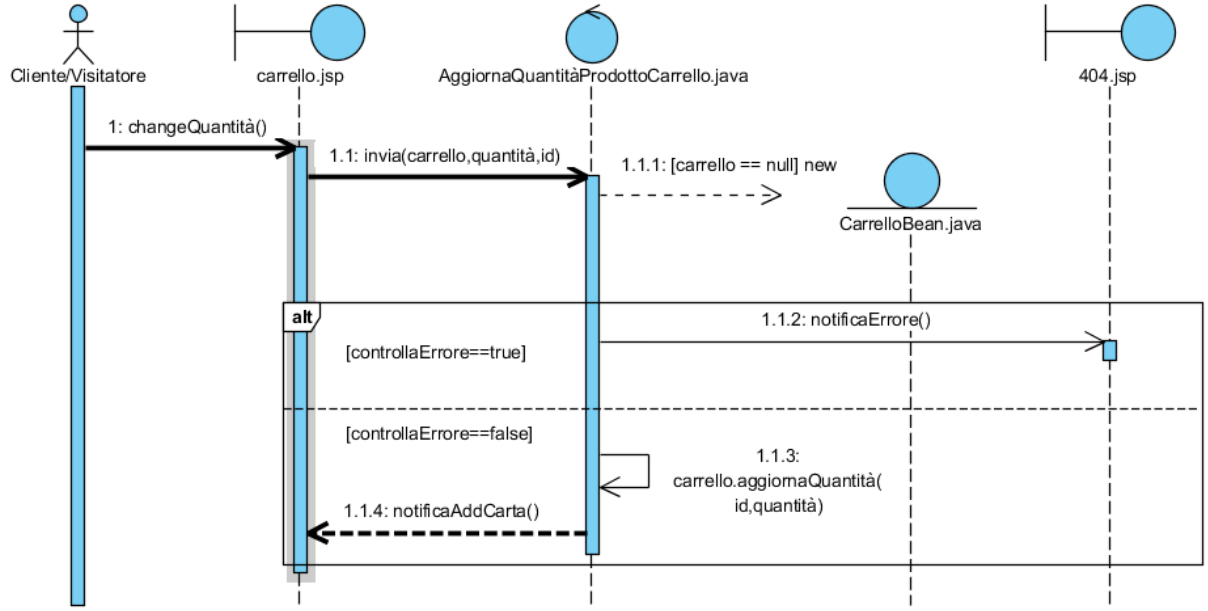
6.8 RICERCA PRODOTTO



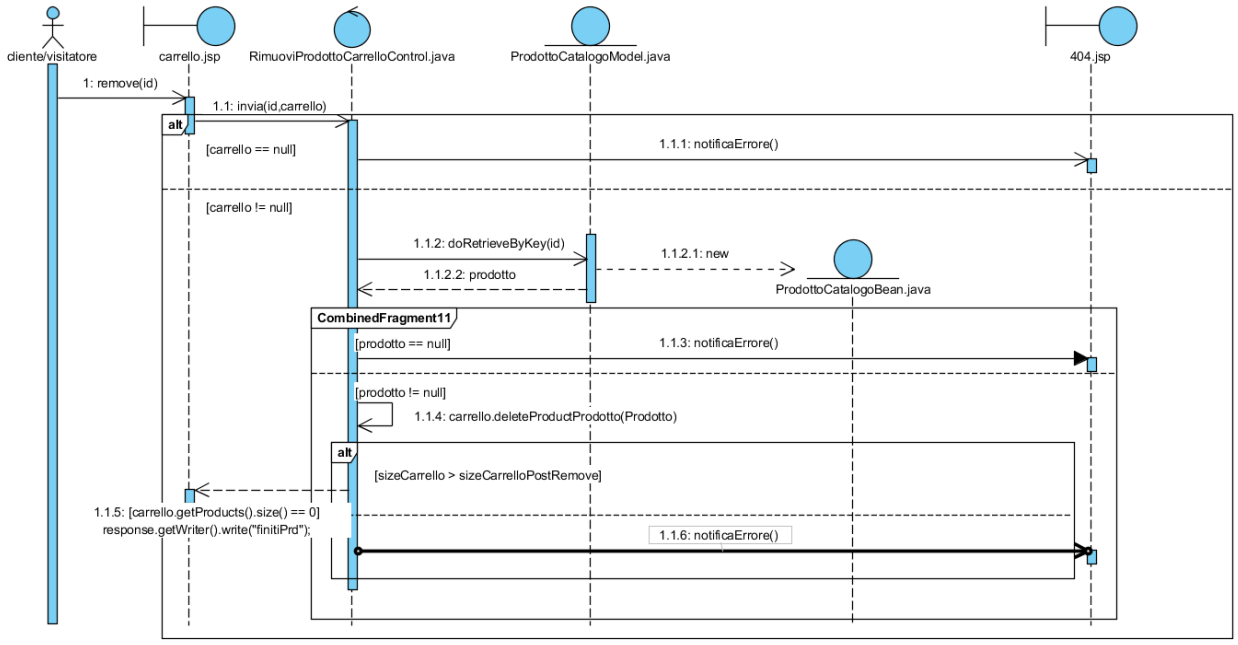
6.9 AGGIUNGI PRODOTTO AL CARRELLO



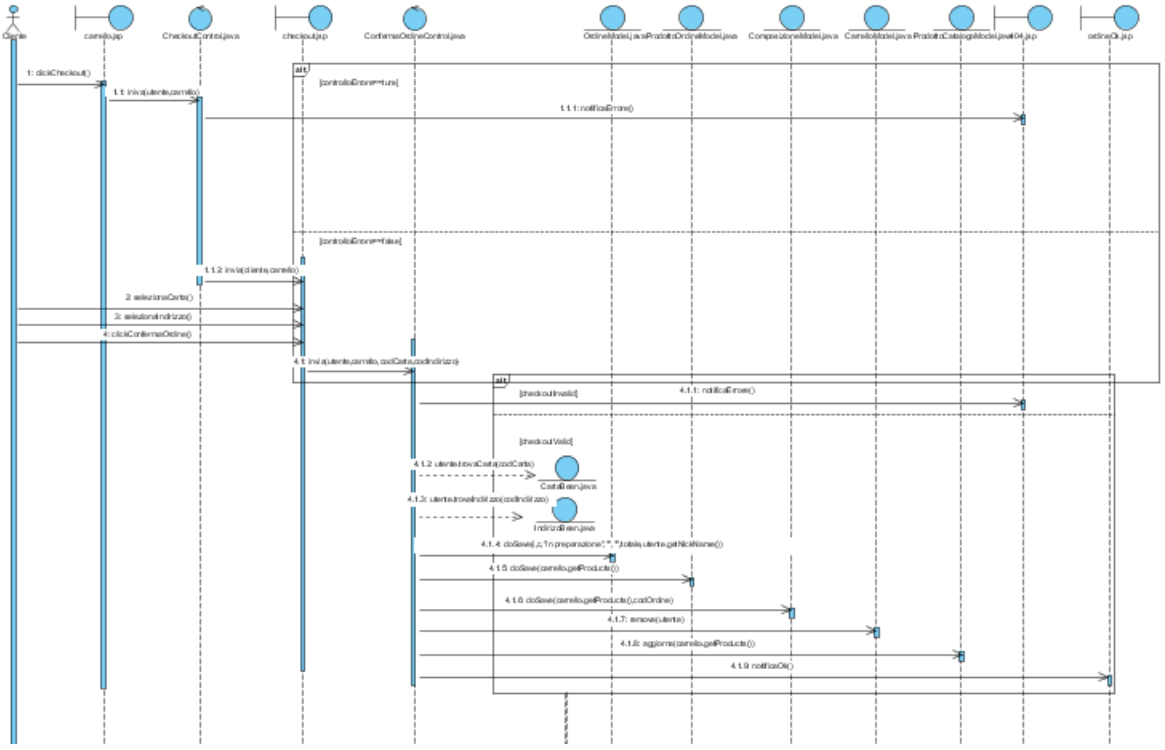
6.10 AGGIORNA QUANTITA' PRODOTTO CARRELLO



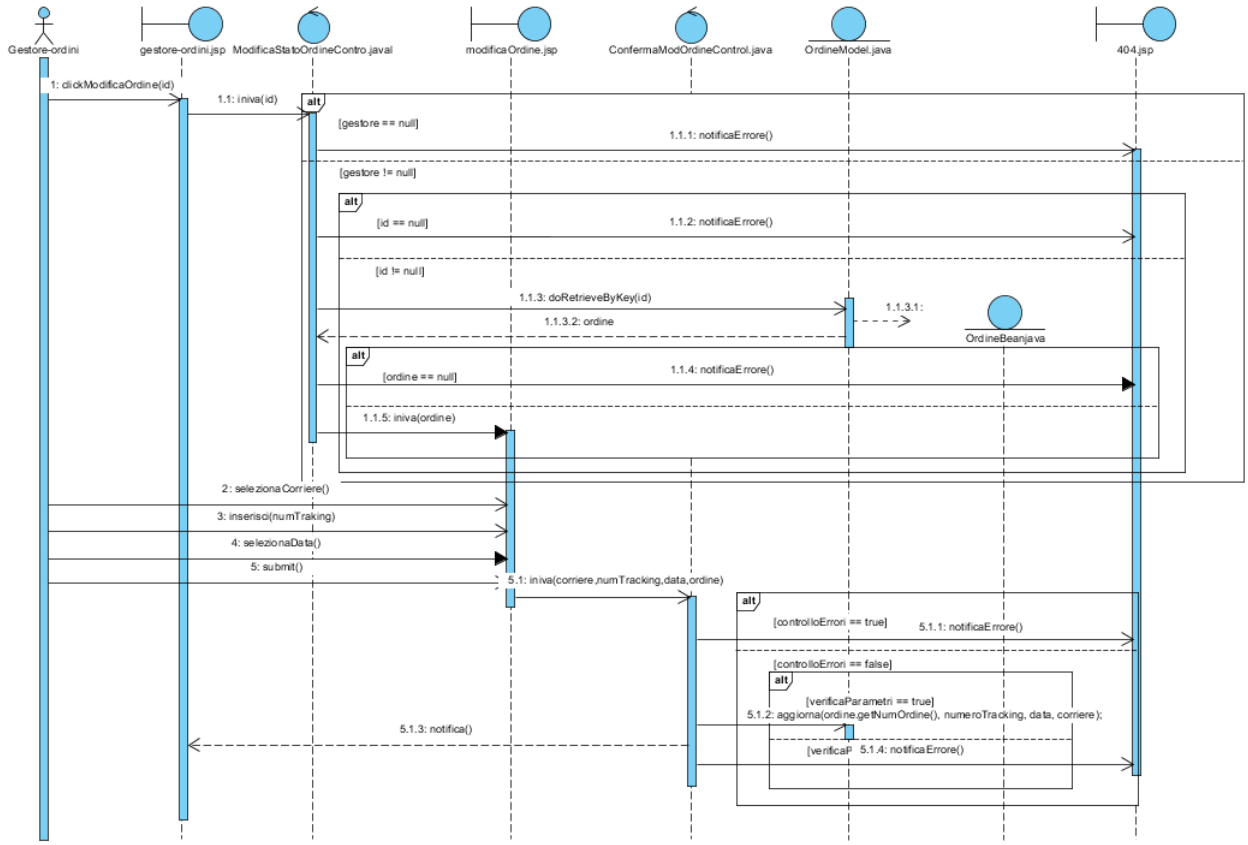
6.11 RIMUOVI PRODOTTO CARRELLO



6.12 CHECKOUT



6.13 MODIFICA STATO ORDINE (DA "IN PREPARAZIONE" A "SPEDITO")



6.14 MODIFICA STATO ORDINE (DA "SPEDITO" A "CONSEGNATO")