

Università degli Studi di Salerno

Corso di Ingegneria del Software

EdenJewelry
ODD_EdenJewelry
Versione 1.7



Data: 15/01/2025

Progetto: Eden Jewelry	Versione: 1.7
Documento: ODD_EdenJewelry	Data: 15/01/2025

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
Gaetano D'Alessio	0512110836
Miriam Eva De Santis	0512117121
Luigi Montuori	0512117799

Scritto da:	Gaetano D'Alessio
--------------------	-------------------

Revision History

Data	Versione	Descrizione	Autore
05/01/2025	1.0	Prima stesura del file	Gaetano D'Alessio
05/01/2025	1.1	Prima scrittura dell'indice	Miriam Eva De Santis
05/01/2025	1.2	Scrittura dei Off-the-shelf components	Gaetano D'Alessio, Miriam Eva De Santis
10/01/2025	1.3	Aggiunta dei packages	Gaetano D'Alessio
13/01/2025	1.4	Riscrittura delle interfacce	Gaetano D'Alessio
15/01/2025	1.5	Completamento del primo punto del documento	Miriam Eva De Santis
15/01/2025	1.6	Scrittura design pattern	Gaetano D'Alessio
15/01/2025	1.7	Ultimi ritocchi	Gaetano D'Alessio

Indice

1. INTRODUZIONE	4
1.1. Object design trade offs	4
1.2 Off-the-shelf components	4
1.3 Interface documentation guidelines	5
1.4 Definitions, acronyms, and abbreviations	5
1.5 References	5
2. Packages	5
2.1 Gestione account	6
2.2 Gestione prodotti	6
2.3 Gestione ordini	7
3. Specifica interfacce dei sottosistemi	7
4. Design Patterns	26
4.1 DAO	26
4.2 Proxy	26
4.3 Strategy	27

1. INTRODUZIONE

1.1.Object design trade offs

Generalità vs Specificità	La promozione della specificità del sito permette a quest'ultimo di rappresentare pienamente l'ambito di utilizzo del sito (cioè la vendita di gioielli).
Flessibilità vs Semplicità	La nostra scelta è ricaduta sulla promozione della semplicità, in modo da permettere un facile utilizzo da un più vasto pubblico e una fase di implementazione agevolata. Questo comporta una riduzione dell'adattabilità del sito.
Astrazione vs Trasparenza	Abbiamo deciso di favorire la trasparenza, evitando di utilizzare astrazioni che avrebbero gravato sulle prestazioni del sistema.
Leggibilità vs Release Time	Considerando le tempistiche stringenti per la consegna del progetto, non sarà possibile commentare il codice in maniera esaustiva. Nonostante ciò, il codice sarà comunque comprensibile e manutenibile.
Buy vs Build	Trattandosi di un progetto sviluppato da una manciata di studenti senza budget, le soluzioni adottate sono progettate dagli stessi e non si avvalgono di parecchie componenti off-the-shelf.
Sicurezza vs Efficienza	Il sistema utilizzerà dei meccanismi di criptazione delle password. Questi sono reputati indispensabili, anche al costo di pesare su spazio ed efficienza.

1.2.Off-the-shelf components

Per quanto riguarda i componenti cosiddetti "off-the-shelf", è stato deciso di ridurre il loro utilizzo al minimo per assenza di budget. Sono tuttavia risultati essenziali i seguenti tool per agevolare le fasi di testing:

J-unit: L'idea dello unit test in Java è quella di valutare ogni singolo metodo in funzione dei valori attesi. Per automatizzare questo passaggio, sfruttiamo J-Unit. Quest'ultimo mette a disposizione dei tag per: seguire i metodi richiamati da una classe durante il suo ciclo di vita, per creare dei testcase in maniera semplice e veloce utilizzando @Test;

Selenium: Permette di automatizzare il testing di siti web. Questo prodotto è suddiviso nei tools: Selenium WebDriver, Selenium IDE, Selenium Grid;

Mockito: Un framework java che consente di simulare il comportamento di oggetti e dipendenze esterne, rendendo più facile testare il funzionamento isolato di una classe (mock) o un metodo (all'atto pratico serve per scrivere i test stub). Dunque, è possibile utilizzarlo per scrivere test unitari efficaci, garantendo che la logica dell'applicazione sia robusta e che le dipendenze non interferiscono con il processo di testing;

1.3. Interface documentation guidelines

- Le classi hanno nomi significativi e singolari.
- I metodi hanno dei nomi che aiutano a comprendere il loro utilizzo, possibilmente verbi che descrivano il comportamento atteso.
- Per i nomi di classi, metodi e variabili viene utilizzata la notazione cosiddetta a "gobba di cammello", classica del linguaggio Java.
- Nel codice devono essere presenti i commenti JavaDoc, comprensivi di annotazioni, al fine di generare la documentazione.

Ecco un esempio di commento che segue la sintassi JavaDoc:

```
/**  
 *This is a Java doc comment  
 */
```

1.4. Definitions, acronyms, and abbreviations

DAO: sta per Data Access Object, è il design pattern che permette di interagire con il database.

Servlet: classe java residente sul server capace di gestire le richieste generate dai client.

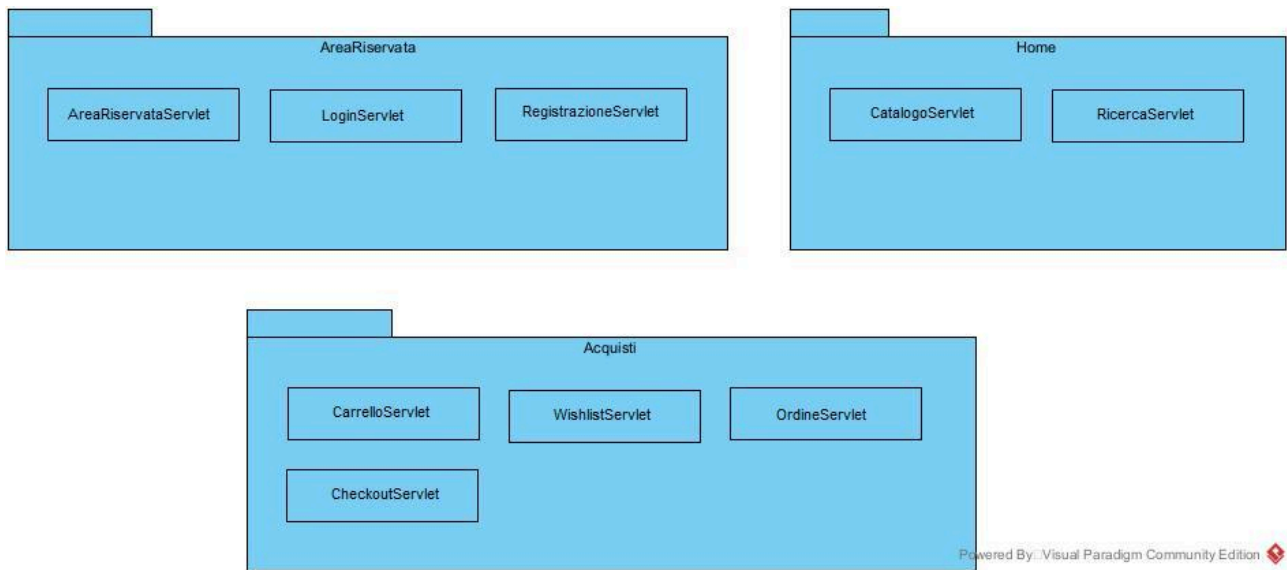
1.5. References

SDD_EdenJewelry.

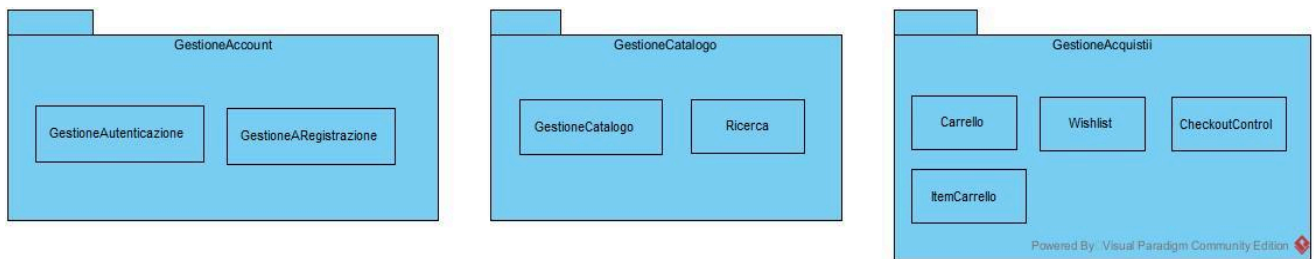
2. PACKAGES

La suddivisione dei sottosistemi in package riprende la struttura delle componenti già vista nel System Design Document. Andiamo adesso a vedere, nello specifico, le classi che compongono i vari package e le dipendenze tra questi.

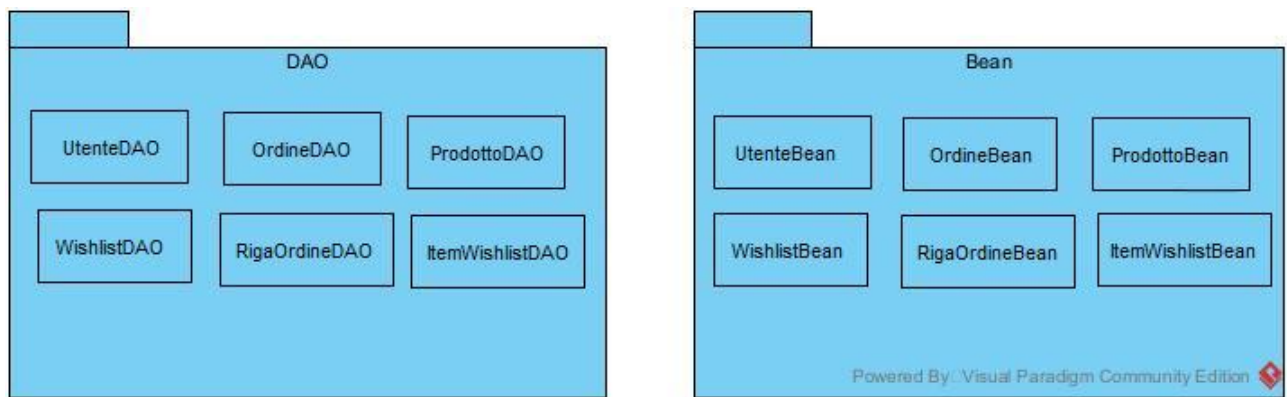
2.1. Presentation Layer



2.2. Application Layer



2.3 Data Management Layer



3. SPECIFICHE INTERFACCE DEI SOTTOSISTEMI

Presentation Layer

Nome classe	GestioneAutenticazione
Descrizione	Classe che si occupa di permettere, all'utente, l'accesso e la disconnessione all'account che ha realizzato al momento della registrazione
Metodi	<ul style="list-style-type: none">• login• logout
Invariante	

Nome metodo	login
Descrizione	Metodo che permette di accedere al sito tramite email e password
Precondizione	Context: LoginServlet:: login(email: String, password: String): UtenteBean pre: email != null AND utenteDAO.doRetrieveByEmail(email) != null AND password != null
Postcondizione	Context: LoginServlet:: login(email: String, password: String): UtenteBean post: utente = utenteDAO.doRetrieveByEmail(email)

Nome metodo	logout
Descrizione	Metodo che permette di disconnettersi dal sito
Precondizione	Context: LogoutServlet:: logout(email: String, password: String): boolean

	pre: utenteDAO.doRetrieveAll()->exists(u u.email = email)
Postcondizione	

Nome classe	GestioneRegistrazione
Descrizione	Mette a disposizione un metodo per permettere la registrazione dell'utente al sito e, quindi, la realizzazione di un account
Metodi	<ul style="list-style-type: none"> • register • checkNome • checkEmail
Invariante	

Nome metodo	register
Descrizione	Metodo che permette di registrarsi al sito, utilizzando nome, cognome, email e password
Precondizione	Context: RegistrazioneServlet:: register(nome: String, cognome:String, email: String, password: String, tipo: String): boolean pre: nome != null AND cognome != null AND email != null AND password != null
Postcondizione	Context: RegistrazioneServlet:: register(nome: String, cognome:String, email: String, password: String, tipo: String): boolean post: not UtenteDAO.@pre.doRetrieveAll()->exists(u u.email = email)

Nome metodo	checkNomeCognome
-------------	------------------

Descrizione	Verifica che la stringa inserita non contenga numeri o caratteri speciali. Quindi, che l'input si presenti in un modo da poter essere considerato come un nome o un cognome
Precondizione	Context: GestioneRegistrazione:: checkNomeCognome(s: String): boolean pre: s != null AND s != ""
Postcondizione	Context: GestioneRegistrazione:: checknomeCognome(s: String): boolean post: result = s->forAll(c c.isLetter)

Nome metodo	checkEmail
Descrizione	Verifica che l'email non sia già in uso, e che l'email sia in formato corretto
Precondizione	Context: GestioneRegistrazione:: checkEmail(email: String): boolean pre: email != null AND email != ""
Postcondizione	Context: GestioneRegistrazione:: checkEmail(email: String): boolean post: result = UtenteDAO.doRetrieveAll()->forAll(u u.getEmail() != email)

Nome classe	UtenteDAO
Descrizione	Classe che consente un'astrazione dell'interfacciamento all'entità utente del database
Metodi	+s(utente: UtenteBean): boolean +doDelete(email: String): boolean +doRetrieveAll(): List <UtenteBean> +doRetrieveByEmail(email: String): UtenteBean
Invariante	

Nome metodo	doSave
Descrizione	Metodo che salva un nuovo utente nel database
Precondizione	Context: UtenteDAO:: doSave(utente: UtenteBean): boolean pre: utente.email != null AND utente.nome != null AND utente.cognome != null AND utente.password != null AND utente.tipo != null
Postcondizione	Context: UtenteDAO:: doSave(utente: UtenteBean): boolean post: self.doRetrieveAll()->include(utente)

Nome metodo	doDelete
Descrizione	Metodo che elimina un utente dal database
Precondizione	Context: UtenteDAO:: doDelete(email: String): boolean pre: email != null AND email != "" AND self.doRetrieveByEmail(email) != null
Postcondizione	Context: UtenteDAO:: doDelete(email: String): boolean post: self.doRetrieveByEmail(email) = null

Nome metodo	doRetrieveAll
Descrizione	Metodo che restituisce una lista contenente tutti gli utenti registrati al sito
Precondizione	Context: UtenteDAO:: doRetrieveAll(): List <UtenteBean> pre:
Postcondizione	Context: UtenteDAO:: doRetrieveAll(): List<UtenteBean> post:

Nome metodo	doRetrieveByEmail
Descrizione	Metodo che restituisce l'oggetto utente a cui corrisponde l'email inserita
Precondizione	Context: UtenteDAO:: doRetrieveByEmail(email: String): UtenteBean pre: email != null AND email != ""
Postcondizione	Context: UtenteDAO:: doRetrieveByEmail(email: String): UtenteBean post: result = self.doRetrieveAll()->forAll(u u.email = email)

Abbiamo deciso di omettere i contratti della classe *UtenteBean*, in quanto si tratta di banali metodi getter e setter.

Application Layer

Nome classe	Ricerca
Descrizione	Classe che gestisce la ricerca di prodotti nel catalogo, fornendo una parola chiave. Quest'ultima, viene confrontata con i nomi dei prodotti. Vengono restituiti tutti i prodotti del catalogo che presentano una corrispondenza (cioè, la stringa in input corrisponde ad una porzione di nome di uno o più prodotti)
Metodi	+search(nomeProdotto: String): List <ProdottoBean>
Invariante	

Nome metodo	search
Descrizione	Metodo che restituisce una lista contenente i prodotti contenenti la stringa inserita nel nome

Precondizione	Context: RicercaProdotto:: search(nomeProdotto: String): List <ProdottoBean> pre: nomeProdotto != null AND not ProdottoDAO.doRetrieveAll().isEmpty()
Postcondizione	Context: RicercaProdotto:: search(nomeProdotto: String): List <ProdottoBean> post: result = search(nomeProdotto)

Nome classe	Catalogo
Descrizione	Classe contenente i metodi per la gestione dei prodotti
Metodi	+addProduct(prodotto: ProdottoBean): boolean +deleteProduct(prodotto: ProdottoBean): boolean checkProduct(prodotto: ProdottoBean): boolean
Invariante	

Nome metodo	addProduct
Descrizione	Metodo wrapper che permette al venditore di aggiungere nuovi prodotti al catalogo
Precondizione	Context: Catalogo:: addProduct(prodotto: ProdottoBean): boolean pre: prodotto != null self.checkProduct(prodotto)
Postcondizione	Context: Catalogo:: addProduct(prodotto: ProdottoBean): boolean post: ProdottoDAO.doRetrieveAll()->includes(prodoto)

Nome metodo	deleteProduct
Descrizione	Metodo wrapper che permette al venditore di rimuovere i prodotti dal catalogo
Precondizione	Context: Catalogo:: deleteProduct(prodotto: ProdottoBean): boolean pre: prodotto != null
Postcondizione	Context: Catalogo:: deleteProduct(prodotto: ProdottoBean): boolean post: not ProdottoDAO.doRetrieveAll()->includes(prodotto)

Nome metodo	checkProduct
Descrizione	Metodo che verifica la validità dei parametri del prodotto e che il prodotto non sia già presente nel catalogo
Precondizione	Context: Catalogo:: checkProduct(prodotto: ProdottoBean): boolean pre: prodotto != null
Postcondizione	Context: Catalogo:: addProduct(prodotto: ProdottoBean): boolean post: result =ProdottoDAO.doRetrieveAll()->includes(prodotto)

Nome classe	CarrelloServlet
Descrizione	Servlet che si occupa di mostrare e modificare il contenuto del carrello
Metodi	+showCarrello(carrello: Carrello): void
Invariante	

Nome metodo	showCarrello
-------------	--------------

Descrizione	Metodo che stampa il contenuto del carrello
Precondizione	Context: CarrelloServlet:: showCarrello(carrello: Carrello): void pre: carrello != null
Postcondizione	Context: CarrelloServlet:: showCarrello(carrello: Carrello): void post:

Nome classe	Carrello
Descrizione	Classe che rappresenta il carrello dell'utente, permette di aggiungere e rimuovere prodotti
Metodi	<ul style="list-style-type: none"> • addToCart(prodotto: Prodotto) void • deleteFromCart(prodotto: Prodotto): void • modificaQuantità(prodotto: Prodotto, quantità: int): void • getMapProdotti(): Map <Prodotto, Integer> • getEmail(): String • isEmpty(): boolean
Invariante	

Nome metodo	addToCart
Descrizione	Metodo che permette di aggiungere un prodotto al carrello
Precondizione	Context: Carrello:: addToCart(prodotto: Prodotto): void pre: prodotto != null AND ProdottoDAO.doRetrieveAll()->includes(prodotto)
Postcondizione	Context: Carrello:: addToCart(prodotto: Prodotto): void post: prodottiCarrello->includes(prodotto)

Nome metodo	deleteFromCart
Descrizione	Metodo che permette di rimuovere un prodotto dal carrello
Precondizione	Context: Carrello:: deleteFromCart(prodotto: Prodotto): void pre: prodotto != null prodottiCarrello->includes(prodotto)
Postcondizione	Context: Carrello:: deleteFromCart(prodotto: Prodotto): void post: not prodottiCarrello->includes(prodotto)

Nome metodo	modificaQuantità
Descrizione	Metodo che permette di modificare la quantità di un prodotto presente nel carrello
Precondizione	Context: Carrello:: modificaQuantità(prodotto: Prodotto, quantità: int): void pre: prodotto != null AND quantità != 0 AND prodottiCarrello->includes(prodotto)
Postcondizione	Context: Carrello:: modificaQuantità(prodotto, quantità: int): void post: prodotto.quantità = quantità

Nome metodo	getMapProdotti
Descrizione	Metodo che restituisce una map contenente i prodotti del carrello, con relative quantità
Precondizione	Context: Carrello:: getMapProdotti(): Map <Prodotto, Integer> pre:
Postcondizione	Context: Carrello:: getMapProdotti(): Map <Prodotto, Integer> post: result = self.getMapProdotti()

—bisogna aggiungere isEmpty() e getEmail

Nome classe	WishlistDAO
Descrizione	Classe che permette un interfacciamento all'entità wishlist nel database
Metodi	+doSave(wishlist: WishlistBean): boolean +doDelete(email: String): boolean +doRetrieveAll(): List <WishlistBean> +doRetrieveByEmail(email: String): WishlistBean
Invariante	

Nome metodo	doSave
Descrizione	Metodo che permette di salvare la wishlist nel database
Precondizione	Context: WishlistDAO:: doSave(wishlist: WishlistBean): boolean pre: wishlist.email != null AND wishlist.listaProdotti != null
Postcondizione	Context: WishlistDAO:: doSave(wishlist: WishlistBean): boolean post: self.doRetrieveAll()->includes(wishlist)

Nome metodo	doDelete
Descrizione	Metodo che permette di salvare una wishlist nel database
Precondizione	Context: WishlistDAO:: doDelete(email: String): boolean pre: email != null AND email != ""
Postcondizione	Context: WishlistDAO:: doDelete(email: String): boolean post: self.doRetrieveAll()->forall(w w.email != email)

Nome metodo	doRetrieveAll
-------------	---------------

Descrizione	Metodo che permette di salvare una wishlist nel database
Precondizione	Context: WishlistDAO:: doRetrieveAll(wishlist: WishlistBean): List <WishlistBean> pre:
Postcondizione	Context: WishlistDAO:: doRetrieveAll(wishlist: WishlistBean): List <WishlistBean> post:

Nome metodo	doRetrieveByEmail
Descrizione	Metodo che permette di salvare una wishlist nel database
Precondizione	Context: WishlistDAO:: doRetrieveByEmail(email: String): WishlistBean pre: email != null AND email != "" AND UtenteDAO.doRetrievByEmail(email) != null
Postcondizione	Context: WishlistDAO:: doRetrieveAll(wishlist: WishlistBean): WishlistBean post: result = self.doRetrieveAll()->forAll(w w.email = email)

Nome classe	WishlistBean
Descrizione	Classe che rappresenta l'oggetto wishlist, individuale per ogni utente
Metodi	+saveProduct(prodotto: Prodotto): boolean +removeProduct(prodotto: Prodotto): boolean +setEmail(email: String): void +getEmail(): String +getList(): List<Prodotti>
Invariante	

Nome metodo	saveProduct
-------------	-------------

Descrizione	Metodo per aggiungere un prodotto alla wishlist
Precondizione	Context: WishlistBean:: saveProduct(prodotto: Prodotto): boolean pre: prodotto != null AND ProdottoDAO.doRetrieveAll()->includes(prodotto)
Postcondizione	Context: WishlistBean:: saveProduct(prodotto: Prodotto): boolean post: WishlistDAO.doRetrieveByEmail(Utente.email)->includes(prodotto)

Nome metodo	removeProduct
Descrizione	Metodo che rimuove un prodotto dalla wishlist
Precondizione	Context: WishlistBean:: removeProduct(prodotto: Prodotto): boolean pre: prodotto != null AND WishlistDAO.doRetrieveByEmail(Utente.email)->includes(prodotto)
Postcondizione	Context: WishlistBean:: removeProduct(prodotto: Prodotto): boolean post: not self.getList()->includes(prodotto)

Getter e setter sono banali e sono stati perciò omessi.

Nome classe	Wishlist
Descrizione	Classe che mostra i prodotti salvati nella wishlist, permettendo di rimuoverli dalla lista ed aggiungerli al carrello
Metodi	<ul style="list-style-type: none"> aggiungiWishlist (nome: string , email: string): boolean removeWishlist (nome: string, email: string): boolean

Invariante	
------------	--

Nome classe	ProdottoDAO
Descrizione	Classe che si occupa dell'interfacciamento con l'entità prodotto del database
Metodi	+doSave(prodotto: ProdottoBean): boolean +doDelete(nome: String): boolean +doRetrieveAll(): List <ProdottoBean> +doRetrieveByName(nome: String): ProdottoBean
Invariante	

Nome metodo	doSave
Descrizione	Permette di salvare un nuovo prodotto all'interno del database
Precondizione	Context: ProdottoDAO:: doSave(prodotto: ProdottoBean): boolean pre: prodotto.nome != null AND prodotto.prezzo AND prodotto.quantità != null AND prodotto.categoria != null AND prodotto != self.doRetrieveByName(nome);
Postcondizione	Context: ProdottoDAO:: doSave(prodotto: ProdottoBean): boolean post: self.doRetrieveAll()->includes(prodotto)

Nome metodo	doDelete
Descrizione	Permette di rimuovere un prodotto dal database
Precondizione	Context: ProdottoDAO:: doDelete(nome: String): boolean pre: nome != null AND nome != "" AND self.doRetrieveByName(nome) != null
Postcondizione	Context: ProdottoDAO:: doSave(nome: String):

	boolean post: self.doRetrieveByName(nome) = null
--	--

Nome metodo	doRetrieveAll
Descrizione	Recupera la lista completa dei prodotti salvati nel database
Precondizione	Context: ProdottoDAO:: doRetrieveAll(): List <ProdottoBean> pre:
Postcondizione	Context: ProdottoDAO:: doRetrieveAll(): List <ProdottoBean> post:

Nome metodo	doRetrieveByName
Descrizione	Restituisce il prodotto con nome corrispondente
Precondizione	Context: ProdottoDAO:: doRetrieveByName(): ProdottoBean pre: nome != null AND nome != ""
Postcondizione	Context: ProdottoDAO:: doRetrieveByName(): ProdottoBean post: result = self.doRetrieveAll()->forAll(p p.nome = nome)

Abbiamo deciso di omettere i contratti della classe *ProdottoBean*, in quanto si tratta di banali metodi getter e setter.

Data Management Layer

Nome classe	OrdineDAO
Descrizione	Classe che si occupa dell'interfacciamento con l'entità ordine del database

Metodi	+doSave(ordine: OrdineBean): boolean +doDelete(numeroOrdine: int): boolean +doRetrieveAll(): List <OrdineBean> +doRetrieveByNumeroOrdine(numeroOrdine: int): OrdineBean
Invariante	

Nome metodo	doSave
Descrizione	Salva l'ordine effettuato nel database
Precondizione	Context: OrdineDAO:: doSave(ordine: OrdineBean): boolean pre: ordine.numeroOrdine != null AND ordine.totale != null AND ordine.metodoPagamento != null AND ordine.indirizzo != null AND ordine.utente != null AND UtenteDAO.retrieveByName(ordine.utente) != null
Postcondizione	Context: OrdineDAO:: doSave(ordine: OrdineBean): boolean post: self.doRetrieveAll()->includes(ordine)

Nome metodo	doDelete
Descrizione	Elimina l'ordine selezionato dal database
Precondizione	Context: OrdineDAO:: doDelete(numeroOrdine: int): boolean pre: numeroOrdine != 0 AND self.retrieveByNumeroOrdine(numeroOrdine) != null
Postcondizione	Context: OrdineDAO:: doDelete(numeroOrdine: int): boolean post: self.retrieveByNumeroOrdine(numeroOrdine) = null

Nome metodo	doRetrieveAll
Descrizione	Recupera la lista completa degli ordini dal database
Precondizione	Context: OrdineDAO:: doRetrieveAll(): List <OrdineBean> pre:
Postcondizione	Context: OrdineDAO:: doRetrieveAll(): List <OrdineBean> post:

Nome metodo	doRetrieveByNumeroOrdine
Descrizione	Restituisce l'oggetto OrdineBean con il nome corrispondente a quello inserito come parametro
Precondizione	Context: OrdineDAO:: doRetrieveByNumeroOrdine(numeroOrdine: int): OrdineBean pre: numeroOrdine != 0
Postcondizione	Context: OrdineDAO:: doRetrieveByNumeroOrdine(numeroOrdine: int): OrdineBean post: result = self.doRetrieveAll()->forAll(o o.numeroOrdine = numeroOrdine)

La classe *OrdineBean* è stata omessa, in quanto contenente solo banali metodi getter e setter.

Nome classe	CheckoutControl
Descrizione	Classe che si occupa di validare e gestire la fase finale d'acquisto
Metodi	checkIndirizzo(indirizzo: String) checkMetodoPagamento(totale: float) checkout(carrello: Carrello, metodoPagamento: String, totale: float,

	indirizzo: String)
Invariante	

Nome metodo	checkIndirizzo
Descrizione	Controlla che l'indirizzo inserito sia valido
Precondizione	Context: CheckoutControl:: checkIndirizzo(indirizzo: String) pre: indirizzo != null AND indirizzo != ""
Postcondizione	Context: CheckoutControl:: checkIndirizzo(indirizzo: String) post: result = indirizzo.isValid()

Nome metodo	checkout
Descrizione	Metodo che si occupa di finalizzare la procedura di acquisto, come dice il nome
Precondizione	Context: CheckoutControl:: checkout(carrello: Carrello, metodoPagamento: String, totale: float, indirizzo: String) pre: metodoPagamento != null AND metodoPagamento != "" AND totale != 0.0f AND indirizzo != null AND indirizzo != "" AND carrello != null
Postcondizione	Context: CheckoutControl:: checkout(metodoPagamento: String, totale: float, indirizzo: String) post: carrello.prodottiCarrello.isEmpty() AND ProdottoDAO.doRetrieveAll().size() = ProdottoDAO.doRetrieveAll().@pre.size() + 1

4. DESIGN PATTERNS

4.1 DAO

Si è scelto di utilizzare il DAO design pattern (**Data Access Object**) per semplificare il prelievo e l'inserimento dei dati nel database. Questo pattern permette di astrarre ed incapsulare la gestione dei dati nel database.

Le classi DAO racchiudono e raggruppano il codice relativo al database, rendendo il resto del codice piu' leggibile e permettendo alle altre classi un accesso semplificato ai dati. Per fare cio', vengono messi a disposizione una serie di metodi generici che vanno poi implementati per ogni classe (*doSave*, *doDelete*, *doRetrieveAll*, *doRetrieveByKey*).