Università degli Studi di Salerno
Corso di Ingegneria Del Software

BEerHAPPY

Object Design Document

Docente:
Prof. Andrea De Lucia

Studenti:
Giovanni Forlenza
Gaetana Galdi

Sommario

Introduzione

Dopo la realizzazione dei documenti Requirements Analysis Document (RAD) e System Design Document (SDD) abbiamo descritto in linea di massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti implementativi.
L'Object Design Document (ODD) è un documento utilizzato nel processo di sviluppo del software per descrivere la progettazione dettagliata degli oggetti o delle classi che costituiscono il sistema, l'ODD è un documento tecnico che viene utilizzato per descrivere come il codice sorgente del software deve essere scritto e organizzato per ottenere un prodotto software funzionante.

Trade-off

Prestazioni vs Costi

Non disponendo di sponsorizzazioni esterne, il sistema sarà realizzato sfruttando le tecnologie open-source che garantiranno la sua funzionalità anche in maniera totalmente gratuita. Più nello specifico, sarà utilizzato un database relazionale per gestire i dati del sistema ed un web server per l'interazione con gli utenti.

Interfaccia vs Usabilità

L'interfaccia sarà realizzata tenendo conto de canoni dello "user-friendly", disporrà solo degli elementi strettamente essenziali come bottoni e form, in modo da rendere l'esperienza dell'utente quanto più semplice ed intuitiva possibile.

Sicurezza vs Efficienza

Il sistema disporrà di un sistema di sicurezza in grado di evitare accessi non autorizzati per proteggere informazioni personali e dati sensibili.

Linee guida per la documentazione delle interfacce

Per l'implementazione del sistema saranno utilizzate le seguenti convenzioni.
Naming Convention

I nomi utilizzati per identificare i concetti principali, le funzionalità e le componenti generiche del sistema dovranno rispettare tali condizioni:

- Descrittivi
- Pronunciabili
- Di uso comune
- Di lunghezza medio-corta
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Le variabili devono:

- Rispettare la Camel Notation
- Iniziare con carattere minuscolo
- Essere composti da caratteri compresi tra [0-9, a-z, A-Z]

I metodi devono:

- Cominciare con carattere minuscolo
- Consistere in un verbo che identifica una azione, seguita dal nome di un oggetto
- Rispettare la Camel Notation

Le classi e le interfacce devono:

- Rispettare la Camel Notation
- Iniziare con carattere Maiuscolo
- Concludersi con il tipo di elemento che rappresentano

Definizioni, acronimi e abbreviazioni

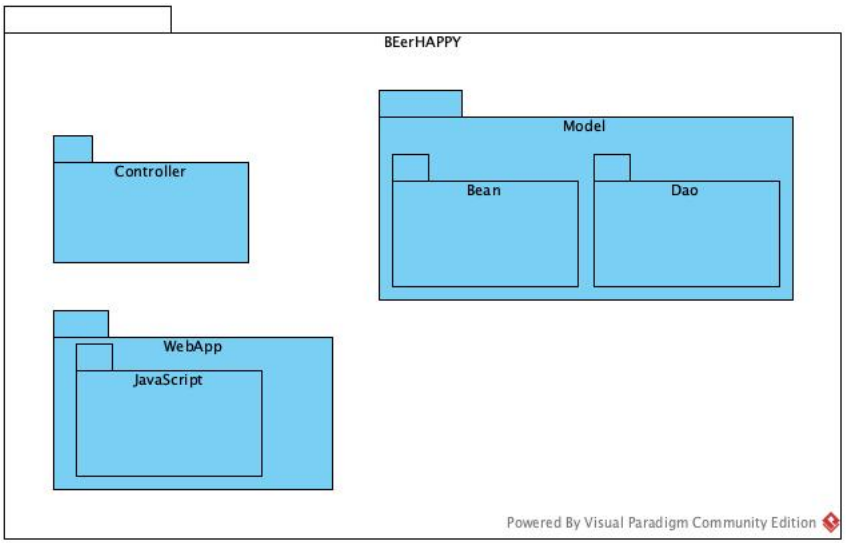
- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document
- DBMS: Database Management System
- API: Application Programming Interface
- JDBS: Java Database Connectivity

Design Pattern

Il Design Pattern Data Access Object (DAO) è un pattern architetturale utilizzato nella progettazione del software per separare la logica di accesso ai dati dalla logica di business.
Il pattern DAO prevede la creazione di un'interfaccia che definisce le operazioni CRUD (Create, Read, Update, Delete) che il sistema deve eseguire sui dati, e di una classe di implementazione che gestisce la connessione al database e l'effettiva esecuzione delle operazioni.
L'uso del pattern DAO consente di separare la logica di accesso ai dati dalla logica di business, rendendo il codice più modulare, riutilizzabile e facile da manutenere. Inoltre, il DAO fornisce una maggiore flessibilità al sistema, poiché consente di cambiare la fonte dati senza dover modificare la logica di business.

Packages

La decomposizione in sottosistemi è la seguente:



La suddivisione in packages è motivata dall'utilizzo dell'architettura MVC (Model-View-Controller) è un pattern architetturale utilizzato per separare le componenti dell'applicazione in tre parti distinte: il Model, la View e il Controller. L'obiettivo dell'architettura MVC è di separare la logica di business, l'interfaccia utente e la gestione delle richieste dell'utente in tre componenti indipendenti, in modo da migliorare la manutenzione e la scalabilità del sistema.

Package Model
Il package "model" rappresenta la componente responsabile della gestione dei dati e della logica di business dell'applicazione.
Il package model contiene le classi che rappresentano i dati del dominio dell'applicazione. Queste classi sono generalmente progettate per rappresentare le entità e le relazioni tra di esse nel sistema. Le classi del package model sono responsabili di fornire i metodi e le operazioni necessarie per accedere, manipolare e persistere i dati. Ciò include l'implementazione delle regole di validazione, la gestione delle relazioni tra gli oggetti e l'interazione con il database o altre fonti di dati.

Package WebApp
Il package "webapp" contiene i file e le risorse che compongono l'interfaccia utente della web app, come ad esempio le pagine HTML, i file CSS, le immagini e i file JavaScript. Questi file definiscono il layout, il design e il comportamento della web app nel browser.

Package Controller
Il package "controller" contiene le classi che ricevono le richieste HTTP inviate dal client e coordinano le azioni necessarie per soddisfare tali richieste. Queste classi fungono da intermediari tra la view (interfaccia utente) e il model (dati e logica di business).

Controller package

il package controller nell'architettura MVC rappresenta la componente che gestisce le interazioni tra l'utente e il sistema. Si occupa di ricevere e interpretare le richieste del client, coordinare le azioni del model per soddisfare tali richieste e comunicare con la view per aggiornare l'interfaccia utente di conseguenza.

I nomi delle classi Java iniziano per lettera maiuscola (così come le lettere iniziali delle parole composte), mentre i nomi di variabili e di metodi iniziano con la prima lettera minuscola e con le parole composte con le iniziali maiuscole.

Tutte le componenti Java termineranno con la dicitura "Servlet.java".

Le classi contenute nel package **Controller:**

- AccedereServlet.java
- AggiuntaCartaServlet.java
- AggiuntaIndirizzoServlet.java
- AgiuntaProdottoCarrelloServlet.java
- AggiuntaProdottoServlet.java
- AggiuntaUtenteBOServlet.java
- AnnullaOrdineServlet.java
- CambioPasswordServlet.java
- DettagliProdottoServlet.java

- EliminazioneProdottoServlet.java
- EliminazioneServlet.java
- EliminazioneUtenteBOServlet.java
- GestioneCarrelloServlet.java
- GestioneCatalogoServlet.java
- GestioneOrdiniServlet.java
- GestioneUtentiServlet.java
- LogoutServlet.java
- ModificaDatiServlet.java
- ModificaProdottoServlet.java
- ModificaRuoliUtentiBOServlet.java
- ModificaStatoServlet.java
- OrdineUtenteServlet.java
- RecuperoPasswordServlet.java
- RecuperoProdottiServlet.java
- RegistrazioneServlet.java
- RichiestaBirreServlet.java
- RimozioneCartaServlet.java
- RimozioneIndirizzoServlet.java
- SelezioneCartaServlet.java
- SearchServlet.java
- SelezionaIndirizzoServlet.java

Model package

Il package model nell'architettura MVC rappresenta la componente che gestisce i dati e la logica di business dell'applicazione, fornendo le funzionalità necessarie per l'interazione con i dati e la loro manipolazione.

Le classi contenute nel package **model.dao:**

- AddressModel.java
- CardModel.java
- CartModel.java
- CatalogoModel.java
- DriverManagerConnectionPool.java
- UserModel.java
- OrderModel.java
- ProductModdel.java

Le classi contenute nel package **model.bean:**

- Carrello.java
- Carta.java
- Indirizzo.java
- Ordine.java
- Prodotto.java
- Stato.java
- Utente.java
- UtenteBO.java

Webapp package

Il package webapp nell'architettura MVC rappresenta la componente che si occupa dell'interfaccia utente e della gestione delle richieste e delle risposte HTTP. Include i file e le risorse per la presentazione della web app.

- aggiuntaCarta.jsp
- aggiuntaIndirizzo.jsp
- aggiuntaProdotto.jsp
- aggiuntaUtente.jsp
- beerCollection.jsp
- cambioPassword.jsp
- carrello.jsp
- catalogo.jsp
- effettuaOrdine.jsp
- footer.jsp
- gestioneCatalogo.jsp
- gestioneOrdini.jsp
- gestioneUtenti.jsp
- homePage.jsp
- homePageStore.jsp
- login.jsp
- modificaProdotto.jsp
- modificaRuoliUtenteBO.jsp
- modificaStatoOrdine.jsp
- navBarBO.jsp
- navBarGuest.jsp
- navBarStore.jsp
- ordinePagina.jsp
- prodottoPage.jsp
- profilo.jsp
- recuperoPassword.jsp
- ricerca.jsp
- selezioneRuolo.jsp
- signUp.jsp

Interfacce delle classi

Interfacce delle classi dao:

AddressModel	
Metodo	Descrizione
public Indirizzo doRetrieveByKey(int key)	Permette di recuperare l'indirizzo di un utente tramite chiave.
public Boolean doDelete(int indirizzoID)	Permette a un utente di eliminare uno dei suoi indirizzi.
public ArrayList<Indirizzo> recuperoIndirizzo(Utente utente)	Permette di recuperare tutti gli indirizzi associati a un utente.
public Indirizzo aggiuntaIndirizzo(Indirizzo indirizzo, Utente utente)	Permette a un utente di aggiungere un nuovo indirizzo alla lista di indirizzi a lui associata.
public boolean controlloFormato(Indirizzo indirizzo)	Permette di effettuare i controlli sul formato dell'indirizzo.

CardModel	
Metodo	Descrizione
public Carta doRetrieveByKey(int key)	Permette di recuperare una carta di un utente tramite chiave.
public ArrayList<Carta> recuperoCarte(Utente utente)	Permette di recuperare tutte le carte associate ad un utente.
public boolean doDelete(int cartaID)	Permette a un utente di eliminare una delle sue carte.
public Carta aggiuntaCarta(Carta carta, Utente utente)	Permette a un utente di aggiungere una nuova carta alla lista di carte a lui associate.
public boolean controlloFormato(Carta carta)	Permette di effettuare i controlli sul formato della carta.

CartModel	
Metodo	Descrizione
public void salvaCarrello(String email , Prodotto prodotto)	Permette di salvare il carrello di un utente, se l'utente effettua il logout e il carrello non è vuoto.
public void svuotaCarrello(Utente utente)	Permette di eliminare il carrello di un utente
public Carrello recuperoCarrello(Utente utente, Carrello cartSession)	Permette di recuperare il carrello di un utente, se il carrello recuperato non è vuoto salva il carrello nella sessione dell'utente.
public boolean doUpdate(Prodotto prodotto)	Permette di aggiornare il carrello.

CatalogoModel	
Metodo	Descrizione
public ArrayList<Prodotto> recuperoProdotti()	Permette di recuperare tutti i prodotti che sono presenti nel database.
public boolean addProduct(Prodotto prodotto)	Permette di aggiungere un nuovo prodotto al catalogo di prodotti.
public boolean updateProduct(Prodotto oldProduct, Prodotto newProduct)	Permette di aggiornare un prodotto presente nel database con nuove informazioni.
public boolean searchProductByKey(String nome, String birrificio)	Permette di cercare un determinato prodotto tramite chiave.
public boolean controlloFormato(Prodotto prodotto)	Permette di controllare che il formato del prodotto sia corretto.

OrderModel	
Metodo	Descrizione
public Ordine aggiuntaOrdine(Ordine ordine, Utente utente)	Permette di aggiungere un nuovo ordine alla lista degli ordini.
public ArrayList<Ordine> recuperoOrdini(Utente utente)	Permette di recuperare gli ordini di un utente.
public void aggiuntaProdottiOrdine(Prodotto prodottoOrdinato, Ordine ordine)	Permette di aggiungere un prodotto a un ordine.
public ArrayList<Ordine> recuperoOrdini()	Permette di recuperare tutti gli ordini presenti nel sistema.
public ArrayList<Prodotto> recuperoProdotti(Ordine ordine)	Permette di recuperare tutti i prodotti di un determinato ordine.
public void updateState(String stato, Ordine ordine)	Permette di aggiornare lo stato di un ordine.

ProductModel	
Metodo	Descrizione
public Prodotto doRetrieveByKey(String nome, String birrificio)	Permette di recuperare un prodotto tramite chiave.
public void doDelete(Prodotto product)	Permette di eliminare un prodotto dal sito.
public ArrayList<Prodotto> doRetrieveAll()	Permette di recuperare tutti i prodotti presenti nel sito.
public ArrayList<Prodotto> doRetrieveProducts(String name)	Permette di recuperare tutti i prodotti tramite nome.

UserModel	
Metodo	Descrizione
public void addUser(Utente utente)	Permette di salvare le credenziali di un utente che si è appena registrato al sito.
public Utente loginUtente(String email, String password)	Permette di verificare se le credenziali inserite per accedere al sito sono corrette.
public boolean controlloEmailRegistrazione(Utente utente)	Permette di verificare se la mail che l'utente sta usando in fase di registrazione e già presente nel sistema.
public void removeUser(Utente utente)	Permette di rimuovere un utente dal sistema.
public boolean cercaUtente(String email)	Permette di verificare se la mail inserita dall'utente in fase di recupero password e presente nel sistema.
public String recuperoPassword(String email)	Permette all'utente di effettuare il recupero della password.
public static String generaPassword()	Permette di generare una password in modo totalmente casuale.
public UtenteBO loginAmministratore(String email, String password)	Permette di effettuare l'accesso all'utente back-office nel caso in cui le credenziali siano corrette.
public boolean cercaUtenteBO(String email)	Permette di controllare se la mail inserita appartiene già ad un utente back-office.
public Utente modificaDati(Utente utente, String nome, String cognome)	Permette ad utente store di modificare il proprio nome o cognome.
public void cambioPassword(Utente utente, String password)	Permette ad un utente store di cambiare la password che usa per accedere al sito.
public ArrayList<UtenteBO> recuperoUtentiBO()	Permette recuperare tutti gli utenti back-office che sono presenti nel sito.
public void updateRoles(UtenteBO utenteBO, int ruolo)	Permette di modificare i ruoli di un utente back-office.
public void deleteUtenteBO(UtenteBO utenteBO)	Permette di rimuovere un utente back-office del sistema.
public void addUserBO(UtenteBO utenteBO)	Permette di aggiungere un nuovo utente back-office.
public boolean controlloFormatoAmministratore (String email, String password)	Alla creazione di un nuovo utente back office permette di effettuare un controllo sul formato dei dati.
public boolean controlloFormatoUtente (String email, String password)	Permette di effettuare un controllo sui dai dell'utente quando prova ad effettuare il login.
public boolean controlloFormato(Utente utente)	Permette di effettuare un controllo sui dati prima di effettuare la registrazione.
public boolean controlloFormatoModifica(String nome, String cognome)	Permette di effettuare un controllo sul formato dei dati quando l'utente prova a modificare il suo profilo.

Interfacce delle classi, non sono indicati i metodi get e set e non saranno presenti le classi che presentano solo tali metodi.

Carrello	
Metodo	Descrizione
public void removeProdotti(Prodotto prodotto)	Permette di rimuovere un prodotto dal carrello.
public void clearProdotti()	Permette di rimuovere tutti i prodotti dal carrello.
public void addProdotti(Prodotto prodotto)	Permette di aggiungere un prodotto al carrello.

Ordine	
Metodo	Descrizione
public void addProdotto(Prodotto prodottoOrdinato)	Permette di aggiungere un prodotto alla lista degli ordini.
public boolean compareKeys(Ordine ordine)	Permette di controllare id dell'ordine.
public static ArrayList<Ordine> remove(ArrayList<Ordine> ordini, Ordine ordine)	Permette di rimuovere un ordine dalla lista degli ordini.

Prodotto	
Metodo	Descrizione
public int aggiornaQuantita(int quantita)	Permette di aggiornare la quantità di un prodotto.
public void aggiungiQuantita(int quantita)	Permette di aumentare la quantità disponibile di un prodotto.
public boolean compareKeys(Prodotto prodotto)	Permette di controllare la chiave di un prodotto.
public static ArrayList<Prodotto> remove(ArrayList<Prodotto> prodotti, Prodotto prodotto)	Permette di rimuovere di un prodotto dalla lista dei prodotti.

Utente	
Metodo	Descrizione
public void addIndirizzo(Indirizzo indirizzo)	Permette di aggiungere un nuovo indirizzo alla lista di indirizzi dell'utente.
public void removeIndirizzo(int indirizzoID)	Permette di rimuovere un indirizzo dalla lista di indirizzi dell'utente.
public void removeCarta(int cartaID)	Permette di rimuovere una carta dalla lista delle carte dell'utente.
public void addCarta(Carta carta)	Permette di aggiungere una nuova carta alla lista di carte dell'utente.
public void annullaOrdine(int idOrdine)	Permette all'utente di annullare un ordine.

UtenteBO	
Metodo	Descrizione
public boolean compareKeys(UtenteBO utenteBO)	Permette di controllare la mail di un utente back-office.
public static ArrayList<UtenteBO> remove(ArrayList<UtenteBO> utenti, UtenteBO utenteBO)	Permette di rimuovere un utente back-office dalla lista di utenti back-office presenti nel sito.

Object Constrain Language

L'Object Constraint Language (OCL) è un linguaggio di specifica e di espressione dei vincoli che consente di definire regole, proprietà e condizioni che devono essere rispettate dai modelli e dai sistemi software.

AddressModel.java

public Indirizzo doRetrieveByKey(int key)	
Descrizione	Questo metodo perette di trovare un indirizzo tramite key.
Precondizione	key > 0
Post condizione	Il metodo ritorna un Indirizzo, se presente nel database.
Invariante	Il database deve essere funzionante.

public Boolean doDelete(int indirizzoID)	
Descrizione	Questo metodo perette di rimuovere un indirizzo tramite indirizzoID.
Precondizione	indirizzoID > 0
Post condizione	Il metodo ritorna un true, se presente nel database.
Invariante	Il database deve essere funzionante.

public ArrayList<Indirizzo> recuperoIndirizzo(Utente utente)	
Descrizione	Questo metodo perette di recuperare tutti gli indirizzi di un utente.
Precondizione	utente.getEmail != null
Post condizione	Il metodo ritorna un array list di indirizzi.
Invariante	Il database deve essere funzionante.

public Indirizzo aggiuntaIndirizzo(Indirizzo indirizzo, Utente utente)	
Descrizione	Questo metodo perette di aggiungere un indirizzo ad un utente.
Precondizione	indirizzo != null && utente != null controlloFormato(indirizzo) == true
Post condizione	Il metodo aggiunge il nuovo indirizzo al database, ritorna l'indirizzo aggiunto.
Invariante	Il database deve essere funzionante.

CardModel.java

public Carta doRetrieveByKey(int key)	
Descrizione	Questo metodo perette di trovare una carta tramite key.
Precondizione	key > 0
Post condizione	Il metodo ritorna una carta, se presente nel database.
Invariante	Il database deve essere funzionante.

public ArrayList<Carta> recuperoCarte(Utente utente)	
Descrizione	Questo metodo perette di trovare tutte le carte di un utente.
Precondizione	utente != null
Post condizione	Il metodo ritorna tette le carte collegate all'utente.
Invariante	Il database deve essere funzionante.

public boolean doDelete(int cartaID)	
Descrizione	Questo metodo perette cancellare una carta.
Precondizione	cartaID > 0
Post condizione	Il metodo ritorna true se la carta viene rimossa dal database.
Invariante	Il database deve essere funzionante.

public Carta aggiuntaCarta(Carta carta, Utente utente)	
Descrizione	Questo metodo perette a un utente di aggiungere una carta.
Precondizione	carta != null && utente != null controlloFormato(carta) == true
Post condizione	Il metodo ritorna la carta che è stata aggiunta.
Invariante	Il database deve essere funzionante.

CartModel.java

public boolean doUpdate(Prodotto prodotto)	
Descrizione	Questo metodo perette di aggiornare i prodotti nel carrello.
Precondizione	prodotto != null
Post condizione	Il metodo ritorna true se il carrello viene aggiornato.
Invariante	Il database deve essere funzionante.

CatalogoModel.java

public ArrayList<Prodotto> recuperoProdotti()	
Descrizione	Questo metodo permette di recuperare tutti i prodotti salvati nel database
Precondizione	-
Post condizione	Il metodo ritorna la lista di prodotti presenti nel database.
Invariante	Il database deve essere funzionante.

public boolean addProduct(Prodotto prodotto)	
Descrizione	Questo metodo permette di aggiungere un nuovo prodotto al database
Precondizione	prodotto != null controlloFormato(prodotto) == true
Post condizione	Il metodo ritorna true se il prodotto viene aggiunto al database con successo.
Invariante	Il database deve essere funzionante.

public boolean updateProduct(Prodotto oldProduct, Prodotto newProduct)	
Descrizione	Questo metodo permette di aggiornare un prodotto già presente nel database.
Precondizione	oldProduct != null && newProduct != null controlloFormato(newProdotto) == true
Post condizione	Il metodo ritorna true se il prodotto viene aggiornato con successo.
Invariante	Il database deve essere funzionante.

public boolean searchProductByKey(String nome, String birrificio)	
Descrizione	Questo metodo permette di trovare un prodotto presente nel database.
Precondizione	nome != null && birrificio != null
Post condizione	Il metodo ritorna true se il prodotto viene trovato nel database.
Invariante	il database deve essere funzionante.

UserModel.java

public void addUser(Utente utente)	
Descrizione	Questo metodo permette di aggiungere un nuovo utente al database.
Precondizione	utente != null
Post condizione	Il metodo aggiunge il nuovo utente al database.
Invariante	Il database deve essere funzionante.

public Utente loginUtente(String email, String password)	
Descrizione	Questo metodo permette di cercare un utente nel database.
Precondizione	email != null && password != null controlloFormatoUtente(email, password) == true
Post condizione	Il metodo ritorna l'utente se presente nel database, null altrimenti.
Invariante	Il database deve essere funzionante.

public boolean controlloEmailRegistrazione(Utente utente)	
Descrizione	Questo metodo permette di controllare se in fase di registrazione è già presente una stessa mail nel sistema.
Precondizione	utente != null
Post condizione	Il metodo ritorna false se la mail è presente nel sistema true se la mail inserita in fase di registrazione non è presente nel sistema.
Invariante	Il database deve essere funzionante.

public void removeUser(Utente utente)	
Descrizione	Questo metodo permette di rimuovere un utente dal database.
Precondizione	utente != null
Post condizione	Il metodo rimuove l'utente dal sistema quando preme il bottone elimina account.
Invariante	Il database deve essere funzionante.

public boolean cercaUtente(String email)	
Descrizione	Questo metodo permette cercare un utente nel database.
Precondizione	email != null
Post condizione	Il metodo ritorna true se trova l'utente nel sistema, false altrimenti.
Invariante	Il database deve essere funzionante.

public String recuperoPassword(String email)	
Descrizione	Questo metodo permette recuperare la password di un nuovo utente.
Precondizione	cercaUtente(email)
Post condizione	Il metodo controlla che la mail inserita dall'utente sia presente nel sistema, genera una nuova password, aggiorna la password nel sistema e invia una mail alla posta elettronica dell'utente con la password generata.
Invariante	Il database deve essere funzionante.

public static String generaPassword()	
Descrizione	Questo metodo genera una password temporanea per l'utente che non ricorda più la sua vecchia password.
Precondizione	-
Post condizione	Il metodo genera una password in maniera casuale.
Invariante	Il database deve essere funzionante.

public UtenteBO loginAmministratore(String email, String password)	
Descrizione	Questo metodo permette controllare se l'utente che sta effettuando l'accesso è un amministratore.
Precondizione	email != null && password != null
Post condizione	Il metodo effettua un controllo nel database, ritorna l'utente se il controllo ha successo, null altrimenti.
Invariante	Il database deve essere funzionante.

public boolean cercaUtenteBO(String email)	
Descrizione	Questo metodo permette di cercare un utenteBO nel database.
Precondizione	email != null
Post condizione	Il metodo ritorna true se l'utente è presente nel sistema, false altrimenti.
Invariante	Il database deve essere funzionante.

public String recuperoPasswordUtenteBO(String email)	
Descrizione	Questo metodo permette di recuperare la password di un utenteBO.
Precondizione	cercaUtenteBO(email)
Post condizione	Il metodo aggiorna la password nel database.
Invariante	Il database deve essere funzionante.

public Utente modificaDati(Utente utente, String nome, String cognome)	
Descrizione	Questo metodo permette di modificare il nome o il cognome di un utente.
Precondizione	utente != null && nome != null && cognome != null controlloFormatoModifica(nome, cognome) == true
Post condizione	Il metodo aggiorna il nome o il cognome dell'utente nel sistema.
Invariante	Il database deve essere funzionante.

public void cambioPassword(Utente utente, String password)	
Descrizione	Questo metodo permette di effettuare il cambio password di un utente.
Precondizione	utente != null && password != null password.length() >= 8 && password.length() <= 30
Post condizione	Il metodo aggiorna la password dell'utente con una nuova aggiornando il database.
Invariante	Il database deve essere funzionante.

public ArrayList<UtenteBO> recuperoUtentiBO()	
Descrizione	Questo metodo permette di recuperare una lista di amministratori.
Precondizione	
Post condizione	Il metodo restituisce un ArrayList di utentiBO.
Invariante	Il database deve essere funzionante.

public void updateRoles(UtenteBO utenteBO, int ruolo)	
Descrizione	Questo metodo permette di aggiornare il ruolo di un amministratore.
Precondizione	utenteBO != null && ruolo > 0 && ruolo < 5
Post condizione	Il metodo aggiorna nel sistema il ruolo dell'utenteBO.
Invariante	Il database deve essere funzionante.

public void deleteUtenteBO(UtenteBO utenteBO)	
Descrizione	Questo metodo permette di eliminare amministratore dal sistema.
Precondizione	utenteBO != null
Post condizione	Il metodo elimina l'utenteBO dal database.
Invariante	Il database deve essere funzionante.

public void addUserBO(UtenteBO utenteBO)	
Descrizione	Questo metodo permette di aggiungere un nuovo amministratore nel sistema.
Precondizione	utenteBO != null controlloFormatoAmministratore(utenteBO.getEmail(), utenteBO.getPassword()) == true
Post condizione	Il metodo aggiunge il nuovo utenteBO al database.
Invariante	il database deve essere funzionante.

OrderModel.java

public Ordine aggiuntaOrdine(Ordine ordine, Utente utente)	
Descrizione	Questo metodo permette di aggiungere un ordine nel sistema.
Precondizione	ordine != null && utente != null
Post condizione	Il metodo aggiunge un nuovo ordine nel database.
Invariante	Il database deve essere funzionante.

public ArrayList<Ordine> recuperoOrdini(Utente utente)	
Descrizione	Questo metodo permette di recupere tutti gli ordini di un utente.
Precondizione	utente != null
Post condizione	Il metodo cerca gli ordini nel database e restituisce un ArrayList di ordini.
Invariante	il database deve essere funzionante.

public void aggiuntaProdottiOrdine(Prodotto prodottoOrdinato, Ordine ordine)	
Descrizione	Questo metodo permette di aggiungere un prodotto ad un ordine.
Precondizione	prodottoOrdinato != null && ordine != null
Post condizione	Il metodo aggiorna l'ordine con l'aggiunta di un prodotto nel database.
Invariante	Il database deve essere funzionante.

public ArrayList<Ordine> recuperoOrdini()	
Descrizione	Questo metodo permette di recuperare tutti gli ordini presenti nel sistema.
Precondizione	-
Post condizione	Il metodo effettua il recupero di tutti gli ordini presenti nel database.
Invariante	Il database deve essere funzionante.

public ArrayList<Prodotto> recuperoProdotti(Ordine ordine)	
Descrizione	Questo metodo permette di recuperare tutti i prodotti di un determinato ordine.
Precondizione	ordine != null
Post condizione	Il metodo effettua il recupero di tutti i prodotti di un ordine presenti nel database.
Invariante	Il database deve essere funzionante.

public void updateState(String stato, Ordine ordine)	
Descrizione	Questo metodo permette di modificare lo stato di un ordine.
Precondizione	stato != null && ordine != null
Post condizione	Il metodo effettua la modifica dello stato di un ordine nel database.
Invariante	Il database deve essere funzionante.

ProductModdel.java

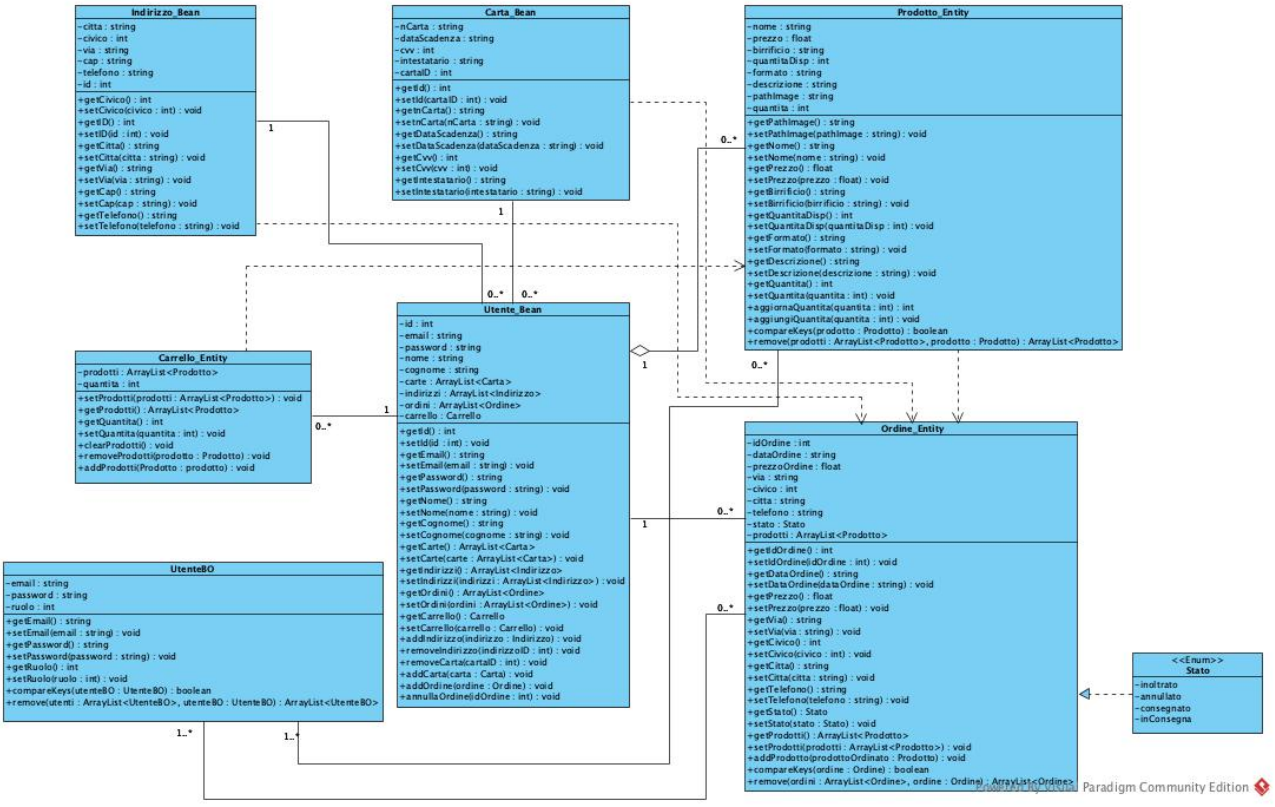
public Prodotto doRetrieveByKey(String nome, String birrificio, String formato)	
Descrizione	Questo metodo permette di ritrovare un prodotto.
Precondizione	nome != null && birrificio != null && formato != null
Post condizione	Il metodo effettua la ricerca di un prodotto nel database.
Invariante	Il database deve essere funzionante.

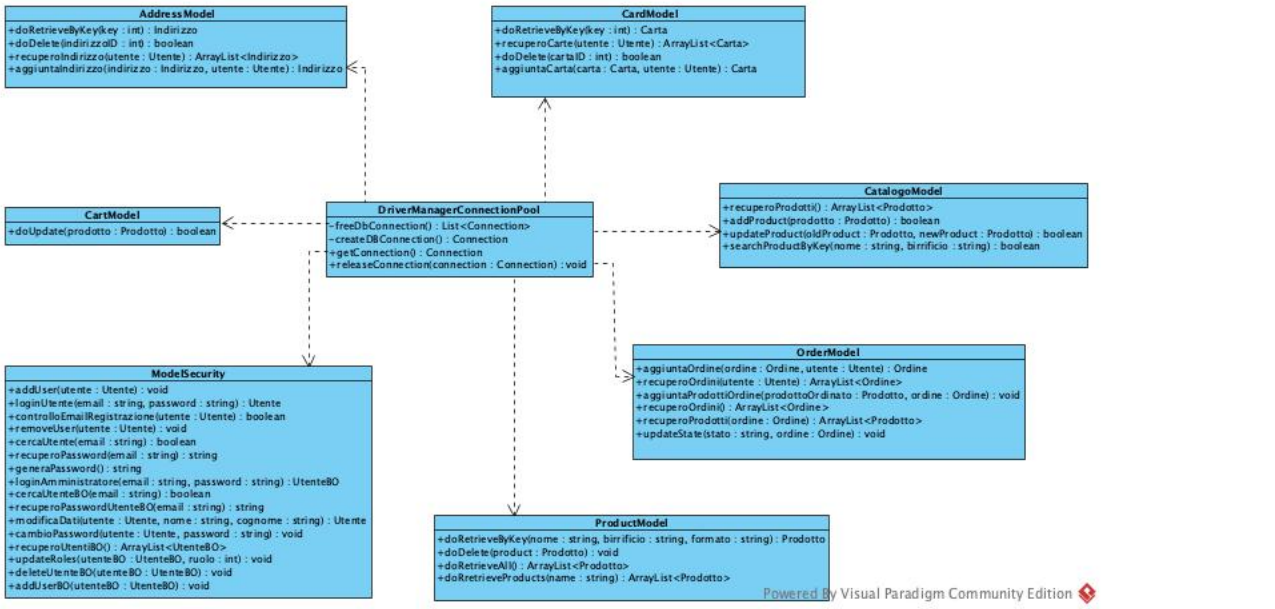
public void doDelete(Prodotto product)	
Descrizione	Questo metodo permette di rimuovere un prodotto
Precondizione	product != null
Post condizione	Il metodo effettua la rimozione di un prodotto nel database.
Invariante	Il database deve essere funzionante.

public ArrayList<Prodotto> doRetrieveAll()	
Descrizione	Questo metodo permette di ritrovare tutti i prodotti presenti nel sistema.
Precondizione	
Post condizione	Il metodo effettua la ricerca di tutti i prodotti nel database.
Invariante	Il database deve essere funzionante.

public ArrayList<Prodotto> doRetrieveProducts(String name)	
Descrizione	Questo metodo permette di ritrovare tutti i prodotti presenti nel sistema tramite nome.
Precondizione	name != null
Post condizione	Il metodo effettua la ricerca di tutti i prodotti con un determinato nome nel database.
Invariante	il database deve essere funzionante.

Class Diagram





Deployment diagram

Il deployment diagram, nell'ambito dell'Unified Modeling Language (UML), serve a visualizzare l'architettura di deployment di un sistema software, indicando come i suoi componenti sono distribuiti su nodi hardware o software. Questo diagramma fornisce una rappresentazione grafica delle risorse su cui verranno eseguiti i componenti del sistema e delle connessioni tra di essi.