Object Design

Document

GoldenOnlineStore

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autori** |
| 27/11/2023 | 0.1 | Prima stesura | Antonio Spatuzzi  Vittorio Giordano |
| 30/11/2023 | 0.2 | Aggiunta Interface documentation guideline | Tutti i membri |
| 02/12/2023 | 0.3 | Aggiunti Design Pattern | Tutti i membri |
| 04/12/2023 | 0.4 | Aggiunti i packages | Tutti i membri |
| 12/12/2023 | 0.5 | Aggiunti Class Interface e Class Diagram | Tutti i membri |
| 10/01/2024 | 0.6 | Revisione e correzione errori | Tutti i membri |

[1. Introduction 4](#_Toc157437264)

[1.1. Object Design Trade-Offs 4](#_Toc157437265)

[1.2. Interface documentation guideline 4](#_Toc157437266)

[1.2.1. Classi ed interfacce Java 4](#_Toc157437267)

[1.2.2. Pagine Lato Server (JSP) 5](#_Toc157437268)

[1.2.3. Pagine HTML 5](#_Toc157437269)

[1.2.4. Fogli di stile CSS 5](#_Toc157437270)

[1.2.5. Database SQL 5](#_Toc157437271)

[1.3. Architecture 6](#_Toc157437272)

[1.3.1. Three Tiers Architecture 6](#_Toc157437273)

[1.4 Design Pattern 6](#_Toc157437274)

[1.4.1. DAO (Data Access Object) Pattern 6](#_Toc157437275)

[1.5. Definitions, acronyms and abbreviations 6](#_Toc157437276)

[1.6. References 7](#_Toc157437277)

[2. Packages 7](#_Toc157437278)

[2.5. Bean Packages 7](#_Toc157437279)

[2.6. Presentation Layer 8](#_Toc157437280)

[2.7. Storage Layer 10](#_Toc157437281)

[2.8. Application Layer 12](#_Toc157437282)

[3. Class Interfaces 14](#_Toc157437283)

[4. Class Diagram 18](#_Toc157437284)

# Introduction

L'obiettivo primario di questo documento è fornire una panoramica approfondita della progettazione degli oggetti che costituiranno il cuore del sistema, garantendo una base solida per la fase di implementazione e manutenzione.

## Object Design Trade-Offs

* Funzionalità vs. Usabilità: Il sistema mira ad essere intuitivo da utilizzare, consentendo la navigazione agevole e permettendo l’utilizzo delle funzionalità offerte anche senza consultare la documentazione.
* Costi vs. Robustezza: Il sistema mira ad essere robusto e tollerante ai guasti, in modo da garantire la correttezza delle informazioni e delle funzionalità fornite.
* Efficienza vs. Portabilità: Il sistema mira ad essere portabile, infatti l’utilizzo di GOS avviene mediante un web browser senza interazione con il sistema sottostante, c’è quindi indipendenza dal sistema operativo.
* Sviluppo rapido vs. Funzionalità: Il sistema mira ad offrire all’utente un’esperienza ottimale, consentendo l’utilizzo di funzionalità complete e sviluppate adeguatamente

## Interface documentation guideline

Nell’implementazione del sistema i programmatori dovranno attenersi alle linee guida definite di seguito:

### Classi ed interfacce Java

Nella scrittura del codice per le classi Java, è necessario conformarsi allo standard di programmazione di Google per Java, consultabile al seguente link:

<http://google.github.io/styleguide/javaguide.html#s4.6-whitespace>.

Secondo questo standard, ogni metodo e ogni file possono essere privi di commenti, tuttavia, è possibile aggiungere commenti per spiegare decisioni specifiche o calcoli particolari.

La convenzione utilizzata per i nomi delle variabili è il camelCase, che prevede la scrittura di parole composte in modo che ogni parola, tranne la prima, inizi con una lettera maiuscola.

Quando si codificano classi e interfacce Java, è importante rispettare le seguenti regole di formattazione:

1. Non inserire spazi tra il nome del metodo e la parentesi tonda "(" che apre la lista dei parametri.

2. La parentesi graffa aperta "{" deve trovarsi sulla stessa linea dell'istruzione di dichiarazione.

3. La parentesi graffa chiusa "}" deve trovarsi su una nuova riga vuota, allo stesso livello di indentazione del nome della classe o dell'interfaccia.

I nomi delle classi devono essere sostantivi con l'iniziale maiuscola, semplici, descrittivi e inerenti al dominio applicativo. Evitare l'uso di underscore per legare i nomi. I nomi dei metodi devono iniziare con una lettera minuscola, non devono contenere caratteri speciali e devono essere semplici, descrittivi e inerenti al dominio applicativo. Sia i nomi delle classi che quelli dei metodi devono seguire la notazione camelCase descritta precedentemente.

### Pagine Lato Server (JSP)

Le pagine JSP, durante l'esecuzione, devono generare un documento conforme allo standard HTML 5. Il codice Java incorporato nelle JSP deve seguire le convenzioni indicate nella sezione 1.2.1, con le seguenti specifiche:

* Il tag di apertura (<%) è posizionato immediatamente alla fine della riga;
* Il tag di chiusura (%>) si trova all'inizio di una nuova riga;
* È consentito derogare alle due regole precedenti nel caso in cui il corpo del codice Java consista in una singola istruzione (<%= %>).

### Pagine HTML

Le pagine HTML devono aderire allo standard HTML 5, e il codice deve essere strutturato con l'indentazione secondo le seguenti direttive:

* L'indentazione deve consistere in una tabulazione.
* Ogni tag deve essere indentato più profondamente rispetto al tag che lo contiene.
* Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura.
* Le regole di indentazione per i tag normali si applicano anche ai tag di commento.

### Fogli di stile CSS

I fogli di stile (CSS) devono aderire alla seguente convenzione:

* Tutti gli stili non inline devono essere inseriti in fogli di stile separati.
* Ogni regola CSS deve seguire il seguente formato:
* I selettori della regola devono essere posizionati nella stessa riga.
* L'ultimo selettore della regola è seguito da una parentesi graffa aperta "{".
* Le proprietà che costituiscono la regola devono essere elencate una per riga, con un'indentazione rispetto ai selettori.
* La regola è conclusa con una parentesi graffa chiusa "}" posta su una riga separata.

### Database SQL

I criteri per la denominazione delle tabelle devono attenersi alle seguenti linee guida:

* Il nome deve costituire un sostantivo singolare che rifletta il dominio del problema e fornisca una chiara spiegazione del contenuto.

Per quanto riguarda i nomi dei campi delle tabelle, si applicano le seguenti disposizioni:

* Ogni nome deve essere scritto in minuscolo.
* Il nome deve rappresentare un sostantivo singolare pertinente al contesto del problema e comunicare in modo esplicito il contenuto.

## Architecture

### Three Tiers Architecture

L’architettura Three-Tier, suddividendo il sistema in tre livelli, semplifica lo sviluppo di un sistema con numerose interfacce utente, ognuna con la propria logica di interazione e di trasformazione dei dati trattati.

Il primo, il Presentation Layer include l'insieme delle interfacce grafiche destinate a ciascun tipo di utente del sistema.

L’Application Layer è costituito da oggetti con lo scopo di interagire con il database e di notificare le modifiche al Presentation Layer.

Lo Storage Layer si occupa di memorizzare i dati e di rispondere alle interrogazioni.

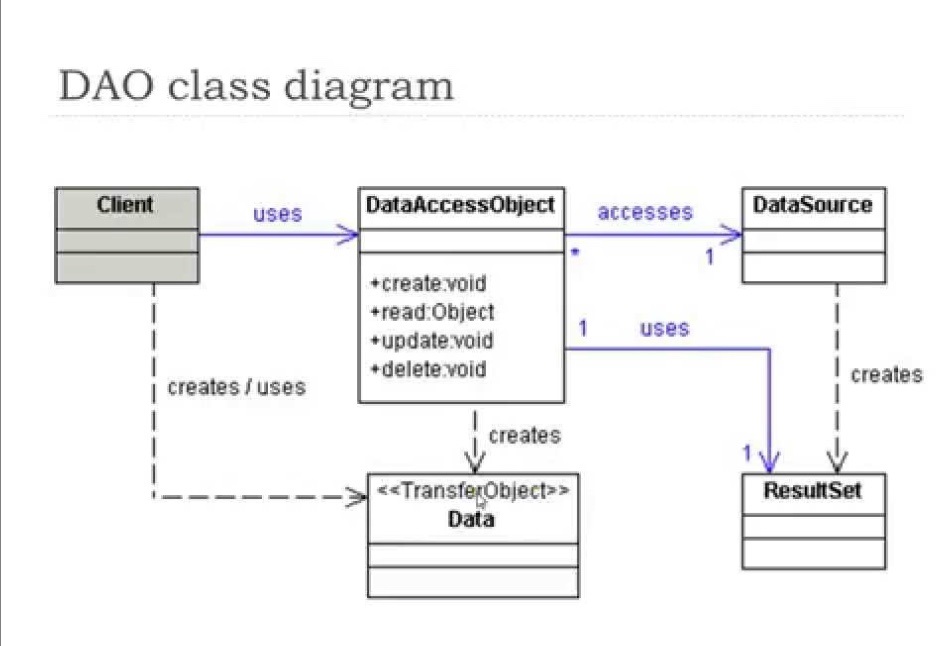
## Design Pattern

### 1.4.1. DAO (Data Access Object) Pattern

Il pattern DAO è adottato per mantenere una chiara separazione tra le componenti dell’architettura del Three-Tier. Questo pattern è costituito dai seguenti elementi:

* Interfaccia del Data Access Object (DAO): Un'interfaccia che definisce le operazioni da eseguire sugli oggetti all'interno del modello Three-Tier. Questa interfaccia fornisce una specifica standard per le operazioni di accesso ai dati.
* Classe concreta del Data Access Object (DAO):Una classe concreta che implementa l'interfaccia del DAO descritta in precedenza. Questa classe contiene la logica effettiva per eseguire le operazioni di accesso ai dati specificate nell'interfaccia.
* Model Object: Plain Old Java Objects (POJO) contenenti metodi get/set per salvare i dati recuperati attraverso le classi DAO. Questi oggetti rappresentano gli elementi del modello e contengono i dati necessari nell'ambito del modello Three-Tier.

Il DAO pattern, attraverso l'utilizzo di un'interfaccia e classi concrete, facilita la gestione della persistenza dei dati all'interno del modello Three-Tier, contribuendo così a preservare la modularità e la chiarezza del codice.



## Definitions, acronyms and abbreviations

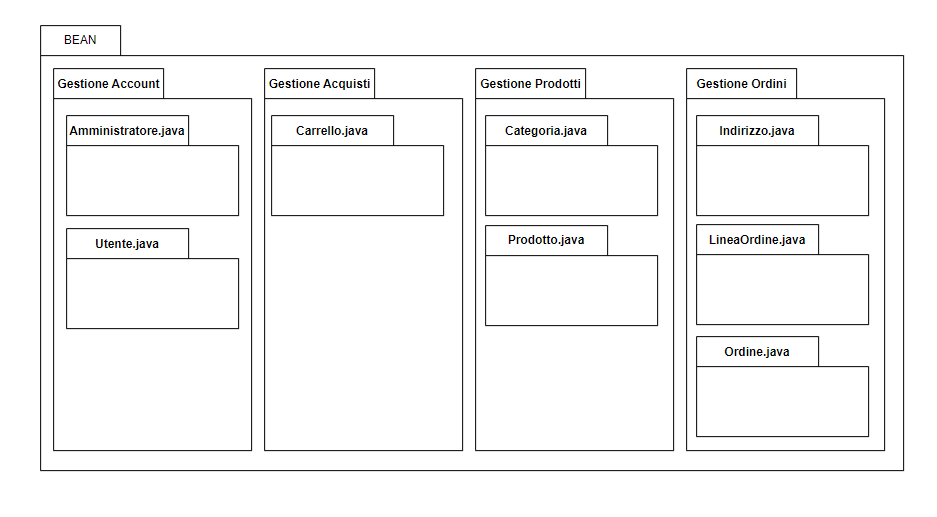
* JSP (JavaServer Pages): Tecnologia di programmazione web in Java utilizzata per lo sviluppo della logica di presentazione, spesso seguendo il pattern architetturale MVC, nelle applicazioni web.
* Bootstrap: Raccolta di strumenti liberi utilizzati per la creazione di siti e applicazioni web.
* HTML (HyperText Markup Language): Linguaggio di markup utilizzato per lo sviluppo di pagine web.
* CSS (Cascading Style Sheets): Linguaggio utilizzato per definire la formattazione delle pagine web.
* AJAX (Asynchronous JavaScript and XML): Tecnica di sviluppo software per realizzare applicazioni web interattive.
* camelCase: Tecnica di denominazione delle variabili adottata dallo standard Google Java, che consiste nel scrivere più parole insieme delimitando l'inizio di ogni nuova parola con una lettera maiuscola.
* Servlet: Oggetti scritti in linguaggio Java che operano all'interno di un server web.
* Tomcat (Apache Tomcat): Web server open source che implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo una piattaforma software per l'esecuzione di applicazioni web sviluppate in linguaggio Java.
* JSTL (JavaServer Pages Standard Tag Library): è una libreria di tag per lo sviluppo di applicazioni Java basate su tecnologia JSP (JavaServer Pages). JSTL fornisce un insieme di tag personalizzati che semplificano la creazione di pagine web dinamiche e la manipolazione dei dati all'interno di esse.
* JavaScript: È un linguaggio di scripting lato client, eseguito dal browser dell'utente, che consente di manipolare il contenuto della pagina, gestire gli eventi utente e comunicare con il server. JavaScript è versatile e supporta paradigmi di programmazione come la programmazione orientata agli oggetti e la programmazione funzionale.

## References

Bernd Bruegge e Allen H. Dutoit - Object-Oriented Software Engineering (using UML, Patterns and JavaTM) – Prentice Hall.

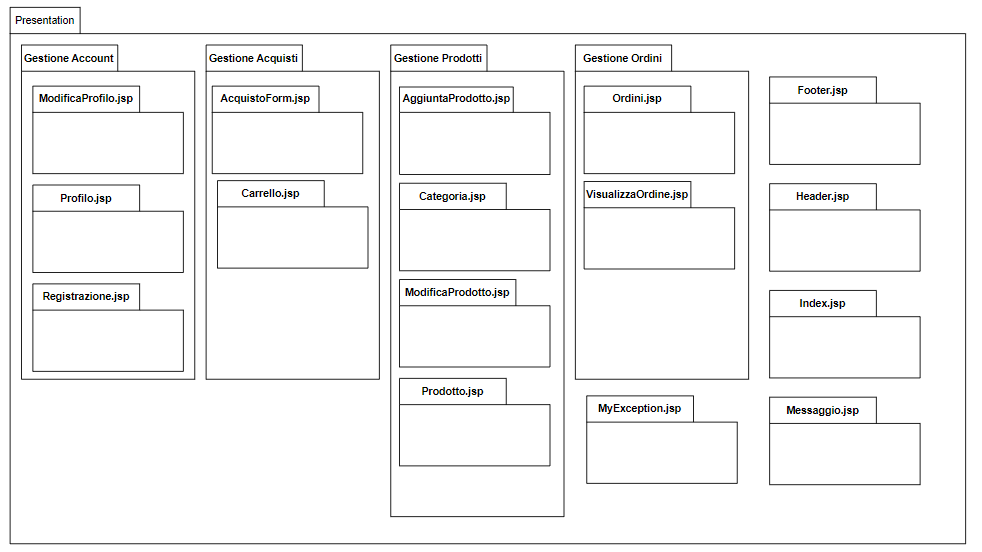
# Packages

## Bean Packages



|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| Utente.java | Classe che descrive un utente registrato |
| Amministratore.java | Classe che descrive un utente amministratore |
| Carrello.java | Classe che descrive il carrello |
| Categoria.java | Classe che descrive una categoria |
| Prodotto.java | Classe che descrive un prodotto |
| Ordine.java | Classe che descrive un ordine |
| LineaOrdine.java | Classe che rappresenta una singola voce nell'elenco degli articoli che un cliente ha scelto di acquistare |
| Indirizzo.java | Classe che descrive un indirizzo |

## Presentation Layer



|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| ModificaProfilo.jsp | Pagina che visualizza il form per modificare le credenziali del profilo |
| Profilo.jsp | Pagina che visualizza i dati di un account |
| Registrazione.jsp | Pagina che visualizza il form per la registrazione |
| AcquistoForm.jsp | Pagina che visualizza il form per l’acquisto di un prodotto |
| Carrello.jsp | Pagina che visualizza il carrello di un utente |
| AggiuntaProdotto.jsp | Pagina che visualizza il form per l’aggiunta di un nuovo prodotto |
| Categoria.jsp | Pagina che visualizza tutti i prodotti di una determinata categoria |
| ModificaProdotto.jsp | Pagina che visualizza il form per la modifica delle caratteristiche di un prodotto |
| Prodotto.jsp | Pagina che visualizza i dettagli di un prodotto |
| Ordini.jsp | Pagina che visualizza la lista degli ordini effettuati da un cliente |
| VisualizzaOrdine.jsp | Pagina che visualizza i dettagli di un singolo ordine |
| MyException.jsp | Pagina per visualizza le eccezioni delle servlet |
| Footer.jsp | Pagina che contiene il footer |
| Header.jsp | Pagina che contiene l’header |
| Index.jsp | Pagina principale del sito |
| Messaggio.jsp | Pagina che mostra un messaggio all’utente e contiene un redirect alla Home |

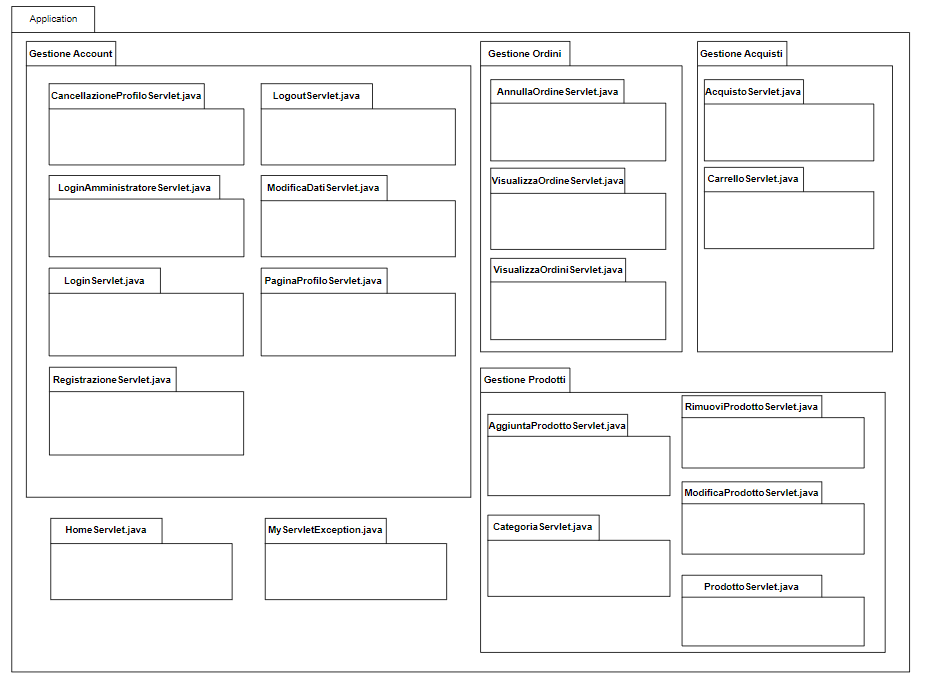
## Storage Layer

Immagine che contiene testo, schermata, diagramma, Rettangolo

Descrizione generata automaticamente

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| AmministratoreDAO.java | Classe contenente metodi utili a manipolare le informazioni del database relative all’amministratore. |
| UtenteDAO.java | Classe contenente metodi utili a manipolare le informazioni del database relative all’utente. |
| ProdottoDAO.java | Classe contenente metodi utili a manipolare le informazioni del database relative al prodotto. |
| CategoriaDAO.java | Classe contenente metodi utili a manipolare le informazioni del database relative alla categoria. |
| IndirizzoDAO.java | Classe contenente metodi utili a manipolare le informazioni del database relative all’indirizzo. |
| LineaOrdineDAO.java | Classe contenente metodi utili a manipolare le informazioni del database relative alla linea d’ordine. |
| OrdineDAO.java | Classe contenente metodi utili a manipolare le informazioni del database relative all’ordine. |
| ConnectionPool.java | Permette la connessione al database. |

## Application Layer



|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| CancellazioneProfiloServlet.java | Classe che permette di cancellare il proprio profilo |
| LoginAmministratoreServlet.java | Classe che permette all’amministratore di effettuare l’accesso |
| LoginServlet.java | Classe che permette al cliente di effettuare l’accesso |
| RegistrazioneServlet.java | Classe che permette ad un utente di registrarsi al sito |
| LogoutServlet.java | Classe che permette al cliente di disconnettersi dal sito |
| ModificaDatiServlet.java | Classe che permette ad un cliente di modificare i propri dati |
| PaginaProfiloServlet.java | Classe che permette ad un cliente di visualizzare il proprio profilo |
| AnnullaOrdineServlet.java | Classe che permette ad un cliente di annullare un ordine |
| VisualizzaOrdineServlet.java | Classe che permette di visualizzare i dettagli di un ordine effettuato da un cliente |
| VisualizzaOrdiniServlet.java | Classe che permette di visualizzare gli ordini effettuati da un cliente |
| AcquistoServlet.java | Classe che permette ad un cliente di effettuare un acquisto |
| CarrelloServlet.java | Classe che permette la gestione del carrello |
| AggiuntaProdottoServlet.java | Classe che permette all’amministratore di aggiungere un nuovo prodotto |
| CategoriaServlet.java | Classe che permette di visualizzare tutti i prodotti di una determinata categoria |
| RimuoviProdottoServlet.java | Classe che permette all’amministratore di rimuovere un prodotto |
| ModificaProdottoServlet.java | Classe che permette all’amministratore di modificare i dettagli di un prodotto |
| ProdottoServlet.java | Classe che permette di visualizzare i dettagli di un prodotto |
| HomeServlet.java | Classe che permette la visualizzazione della pagina principale |
| MyServletException.java | Classe che permette la gestione delle eccezioni delle servlet |

# Class Interfaces

* Amministratore:
  + Metodo “getUsername(): String”: Restituisce l’username dell’amministratore;
  + Metodo “setUsername(username: String)”: Modifica l’username dell’amministratore;
  + Metodo “getEmail(): String”: Restituisce l’email dell’amministratore;
  + Metodo “setEmail(email: String)”: Modifica l’email dell’amministratore;
  + Metodo “getPassword(): String”: Restituisce la password dell’amministratore;
  + Metodo “setPassword(password: String)”: Modifica la password dell’amministratore;
  + Metodo “getNome(): String”: Restituisce il nome dell’amministratore;
  + Metodo “setNome(nome: String)”: Modifica il nome dell’amministratore;
  + Metodo “getCognome(): String”: Restituisce il cognome dell’amministratore;
  + Metodo “setCognome(cognome: String)”: Modifica il cognome dell’amministratore;
  + Metodo “getCodice(): int”: Restituisce l’identificativo dell’amministratore;
  + Metodo “setCodice(codice: int)”: Modifica l’identificativo dell’amministratore.
* AmministratoreDAO:
  + Metodo:”doRetrieveByUsername(username: String): Amministratore”: Recupera un oggetto amministratore dal database utilizzando l’username come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveByCodice(codice: int): Amministratore”: Recupera un oggetto amministratore dal database utilizzando il codice di accesso come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveAll(offset: int, limit: int): List<Amministratore>”: Recupera una lista di amministratori dal database con un indice di partenza e un numero massimo di amministatori da recuperare.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveByUsernamePasswordCode(username: String, password: String, codice: int): Amministratore”: Recupera un oggetto amministratore dal database utilizzando l’username, la password e il codice di accesso come criteri di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
* IstanzaProdotto:
  + Metodo:”getProdotto(): Prodotto”: Restituisce il prodotto di una linea del carrello;
  + Metodo:”setProdotto(prodotto: Prodotto)”: Modifica il prodotto di una linea del carrello;
  + Metodo:”getQuantità(): int”: Restituisce la quantità di un prodotto di una linea del carrello;
  + Metodo:”setQuantità(quantità: int)”: Modifica la quantità di un prodotto di una linea del carrello;
  + Metodo:”getPrezzoTot(): float”: Restituisce il prezzo totale dei prodotti di una linea del carrello;
* Carrello:
  + Metodo:”getProdottiArray(): List<IstanzaProdotto>”: Restituisce tutti i prodotti all’interno del carrello;
  + Metodo:”get(prodId: int): IstanzaProdotto”: Restituisce un prodotto ricercato tramite id;
  + Metodo:”put(prodotto: Prodotto, quantità: int)” Aggiunge un prodotto e la corrispettiva quantità;
  + Metodo:”remove(prodId: int): IstanzaProdotto”: Rimuove un prodotto ricercato tramite id;
  + Metodo:”getPrezzoTot(): float”: Restituisce il prezzo totale di tutti i prodotti del carrello.
  + Metodo:”svuotaCarrello()”: Elimina tutti i prodotti dal carrello;
* Categoria:
  + Metodo:”getId(): int”: Restituisce l’id della categoria;
  + Metodo:”setId(id: int):”Modifica l’id della categoria;
  + Metodo:”getNome(): String”: Restituisce il nome della categoria;
  + Metodo:”setNome(nome: String)”: Modifica il nome della categoria;
  + Metodo:”getDescrizione(): String”: Restituisce la descrizione della categoria;
  + Metodo:”setDescrizione(descrizione: String)”: Modifica la descrizione della categoria.
* CategoriaDAO:
  + Metodo:”doRetrieveByMacro(macro: String): List<Categoria>”: Recupera una lista di categorie dal database utilizzando la macrocategoria come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveAll(): List<Categoria>”: Recupera tutte le categorie dal database;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveById(id: int): Categoria”: Recupera un oggetto categoria dal database utilizzando l’id come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doSave(categoria: Categoria)”: Salva una nuova categoria nel database;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doUpdate(categoria: Categoria)”: Aggiorna una categoria esistente nel database;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doDelete(id: int)”: Elimina una categoria dal database utilizzando l’id come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
* ConnectionPool:
  + Metodo:”getConnection(): Connection”: Ottiene una connessione dal database.
    - Lancia l’eccezione “ClassNotFoundException” quando la classe specificata non viene trovata nel classpath dell'applicazione.
* Indirizzo:
  + Metodo:”getId(): int”: Restituisce l’id dell’indirizzo;
  + Metodo:”setId(id: int)”: Modifica l’id dell’indirizzo;
  + Metodo:”getVia(): String”: Restituisce la via dell’indirizzo;
  + Metodo:”setVia(via: String)”: Modifica la via dell’indirizzo;
  + Metodo:”getCap(): String”: Restituisce il cap dell’indirizzo;
  + Metodo:”setCap(cap: String)”: Modifica il cap dell’indirizzo;
  + Metodo:”getCitta(): String”: Restituisce la città dell’indirizzo;
  + Metodo:”setCitta(citta: String)”: Modifica la città dell’indirizzo.
* IndirizzoDAO:
  + Metodo:”doSave(indirizzo: Indirizzo)”: Salva un nuovo indirizzo nel database.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
* LineaOrdine:
  + Metodo:”getIdOrdine(): int”: Restituisce l’id di un ordine;
  + Metodo:”setIdOrdine(id: int)”: Modifica l’id di un ordine;
  + Metodo:”getIdProdotto(): int”: Restituisce l’id di un prodotto;
  + Metodo:”setIdProdotto(id: int)”: Modifica l’id di un prodotto;
  + Metodo:”getQuantità(): int”: Restituisce la quantità di un prodotto;
  + Metodo:”setQuantità(quantità: int)”: Modifica la quantità di un prodotto;
  + Metodo:”getPrezzoUnitario(): float”: Restituisce il prezzo totale di una linea;
  + Metodo:”setPrezzoUnitario(prezzoUnitario: float)”: Modifica il prezzo totale di una linea.
* LineaOrdineDAO:
  + Metodo:”doRetrieveByOrdine(ordine: int): List<LineaOrdine>”: Recupera le linee d’ordine associate ad un ordine specifico del database;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doSave(lineaOrdine: LineaOrdine)”: Salva una nuova linea d’ordine nel database.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
* Ordine:
  + Metodo:”getCliente(): String”: Restituisce l’username del cliente che ha effettuato l’ordine;
  + Metodo:”setCliente(cliente: String)”: Modifica l’username del cliente che ha effettuato l’ordine;
  + Metodo:”getDataOrdine(): Date”: Restituisce la data in cui è stato effettuato l’ordine;
  + Metodo:”setDataOrdine(data: Date)”: Modifica la data in cui è stato effettato l’ordine;
  + Metodo:”getDataSpedizione(): Date”: Restituisce la data in cui è stato spedito l’ordine;
  + Metodo:”setDataSpedizione(data: Date)”: Modifica la data in cui è stato spedito l’ordine;
  + Metodo:”getId(): int”: Restituisce l’id dell’ordine;
  + Metodo:”setId(id: int)”: Modifica l’id dell’ordine;
  + Metodo:”getPrezzoTotale(): float”: Restituisce il prezzo totale dell’ordine;
  + Metodo:”setPrezzoTotale(prezzoTotale: float)”: Modifica il prezzo totale dell’ordine;
* OrdineDAO:
  + Metodo:”doRetrieveByUsername(username: String): List<Ordine>”: Recupera una lista di ordini dal database utilizzando l’username del cliente come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doSave(ordine: Ordine): int”: Salva un nuovo ordine nel database e restituisce l’id dell’ordine appena creato.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doDelete(idOrdine: int)”: Rimuove l’ordine dal database utilizzando l’id come criterio di ricerca;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveById(idOrdine: int): Ordine”: Recupera un ordine dal database utilizzando l’id come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
* Prodotto:
  + Metodo:”getId(): int”: Restituisce l’id del prodotto;
  + Metodo:”setId(id: int)”: Modifica l’id del prodotto;
  + Metodo:”getNome(): String”: Restituisce il nome del prodotto;
  + Metodo:”setNome(nome: String)”: Modifica il nome del prodotto;
  + Metodo:”getDescrizione(): String”: Restituisce la descrizione del prodotto;
  + Metodo:”setDescrizione(descrizione: String)”: Modifica la descrizione del prodotto;
  + Metodo:”getPrezzo(): float”: Restituisce il prezzo del prodotto;
  + Metodo:”setPrezzo(prezzo: float)”: Modifica il prezzo del prodotto;
  + Metodo:”getMarca(): String”: Restituisce la marca del prodotto;
  + Metodo:”setMarca(marca: String)”: Modifica la marca del prodotto;
  + Metodo:”getCategoria(): String”: Restituisce la categoria del prodotto;
  + Metodo:”setCategoria(categoria: String)”: Modifica la categoria del prodotto;
* ProdottoDAO:
  + Metodo:”doRetrieveAll(offset: int, limit: int): List<Prodotto>”: Recupera una lista di prodotti dal database con un indice di partenza e un numero massimo di prodotti da recuperare.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveById(id: int): Prodotto”: Recupera un singolo prodotto dal database utilizzando l’id come criterio di ricerca;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveByNome(nome: String, offset: int, limit: int): List<Prodotto>”: Recupera una lista di prodotti dal database con un indice di partenza e un numero massimo di prodotti da recuperare utilizzando il nome come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”countByCategoria(categoria: int): int”: Conta il numero totale dei prodotti associati ad una categoria specifica nel database;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveByDescrizione(descrizione: String, offset: int, limit: int): List<Prodotto>”: Recupera una lista di prodotti dal database con un indice di partenza e un numero massimo di prodotti da recuperare utilizzando la descrizione come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveByCategoria(categoria: String, orderBy: OrderBy, offset: int, limit: int): List<Prodotto>”: Recupera una lista di prodotti dal database che appartengono ad una categoria specifica, ordinati secondo un criterio specificato, con un indice di partenza e un numero massimo di prodotti da recuperare.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doSave(prodotto: Prodotto, idCategoria: int):int”: Salva un nuovo prodotto con le informazioni specificate e restituisce il sui identificatore;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doUpdate(prodotto: Prodotto, idCategoria: int)”: Aggiorna le informazioni di un prodotto esistente con i valori specificati;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doDelete(id: String)”: Elimina un prodotto dal database utilizzando l’id specificato.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
* Utente:
  + Metodo:”getId():int”: Restituisce l’id dell’account;
  + Metodo:”setId(id: int):” Modifica l’id dell’account;
  + Metodo:”getUsername(): String”: Restituisce l’username dell’account;
  + Metodo:”setUsername(username: String)”: Modifica l’username dell’account;
  + Metodo:”getEmail(): String”: Restituisce l’email dell’account;
  + Metodo:”setEmail(email: String)”: Modifica l’email dell’account;
  + Metodo:”getPassword(): String”: Restituisce la password dell’account;
  + Metodo:”setPassword(password: String)”: Modifica la password dell’account;
  + Metodo:”getNome(): String”: Restituisce il nome del proprietario dell’account;
  + Metodo:”setNome(nome: String)”: Modifica il nome del proprietario dell’account.
  + Metodo:”getCognome(): String”: Restituisce il cognome del proprietario dell’account;
  + Metodo:”setCognome(cognome: String)”: Modifica il cognome del proprietario dell’account.
* UtenteDAO:
  + Metodo:”doRetrieveByUsername(username: String): Utente”: Recupera un oggetto utente dal database utilizzando l’username come criterio di ricerca;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveByEmail(email: String): Utente” Recupera un oggetto utente dal database utilizzando l’email come criterio di ricerca;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doUpdateCredenziali(utente: Utente, username: String):” Modifica le credenziali di un utente utilizzando l’username come criterio di ricerca;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doSave(utente: Utente): int”: Salva un nuovo utente nel database e restituisce il suo id;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveAll(offset: int, limit: int): List<Utente>”: Recupera una lista di utenti dal database con un indice di partenza e un numero massimo di utenti da recuperare;
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveByUsernamePassword(username: String, password: String): Utente”: Recupera un oggetto utente dal database utilizzando l’username e la password come criteri di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doDelete(username: String):” Elimina un account di un utente dal database utilizzando l’username specificato.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.
  + Metodo:”doRetrieveById(idUtente: int): Utente”: Recupera un utente dal database utilizzando l’id come criterio di ricerca.
    - Lancia l’eccezione “SQLException” durante l’accesso al database nel caso in cui ci siano problemi nella connessione al database.

# Class Diagram

