Università degli Studi di Salerno
Corso di Ingegneria del Software

ODD

Object Design Document

Car – Zone

Versione 0.2

12/02/2025

Coordinatore del progetto:

Nome	**Matricola**
Francesco Pio Cataudo	0512116773

Partecipanti:

Nome	**Matricola**
Francesco Pio Cataudo	0512116773
Francesco Santoro	0512117079
Francesco Pio Bottaro	0512118180
Errico Aquino	0512117730

Revision History

Data	Versione	Descrizione	Autori
13/12/2024	0.1	Strutturazione documento ed informazioni iniziali.	Team
12/02/2025	0.2	Revisione Finale	Team

Sommario

1.Introduzione

1.1 Object design trade-offs

Il progetto CarZone segue un design orientato agli oggetti per strutturare il backend utilizzando Java Servlets, Data Access Objects (DAO) e classi entità. I principali compromessi considerati sono:

- **Acquisto vs. Sviluppo:** Servlets e DAO sviluppati internamente invece di utilizzare un framework come Spring per mantenere alte prestazioni e maggiore controllo sulla logica di business.
- **Memoria vs. Prestazioni:** Recupero dei dati ottimizzato attraverso pattern DAO per bilanciare l'uso della memoria e il tempo di risposta, garantendo un'efficiente gestione delle risorse.
- **Manutenibilità vs. Complessità:** La modularità del codice assicura facilità di manutenzione evitando un'eccessiva astrazione e promuovendo la riusabilità dei componenti.
- **Sicurezza vs. Efficienza:** Il progetto implementa la gestione sicura degli accessi con crittografia delle password e utilizzo di sessioni, bilanciando la sicurezza con la rapidità di accesso ai dati.

1.2 Linee guida per la documentazione dell'interfaccia

Per garantire coerenza e facilità di lettura del codice, vengono adottate le seguenti convenzioni:

- **Convenzioni di Naming:**
  - **Classi:** Nomi con sostantivi singolari (es. Prodotto, Cliente).
  - **Metodi:** Nomi con frasi verbali (es. getDettagliProdotto(), aggiungiAlCarrello()).
  - **Attributi:** Nomi con frasi nominali (es. nomeProdotto, idCliente).
- **Gestione degli Errori:** Gli errori vengono gestiti tramite eccezioni invece di valori di ritorno per garantire maggiore controllo sui flussi di esecuzione.
- **Collezioni:** Gli attributi basati su collezioni devono avere metodi iteratori come elementi(), garantendo coerenza nelle implementazioni e un'interazione sicura con i dati.

- **Commenti e Documentazione:** Ogni classe e metodo deve essere accompagnato da una descrizione Javadoc per facilitare la comprensione del codice e la collaborazione tra sviluppatori.

1.3 Definizioni, acronimi, e abbreviazioni

- **DAO** - Data Access Object, un pattern di progettazione per la manipolazione dei dati.
- **JSP** - Java Server Pages, utilizzato per il rendering frontend.
- **Servlet** - Classe Java che gestisce richieste HTTP nell'applicazione web.
- **MVC** - Model-View-Controller, il pattern di design seguito dal progetto.
- **ORM** - Object Relational Mapping, tecnica utilizzata per gestire la persistenza degli oggetti nel database.

1.4 Riferimenti

- Deliverables github relativa a questo proggetto
- Documentazione Java EE

2. Packages

Il progetto è strutturato in più pacchetti per separare le responsabilità:

2.1 carzone.com.conn

- **Scopo:** Gestione della connessione al database e pooling delle connessioni per ottimizzare l'uso delle risorse.
- **Classi:**
    - DBConnect.java: Gestisce le istanze di connessione al database tramite JDBC e implementa un sistema di connessioni persistenti per ridurre i tempi di latenza.

2.2 carzone.com.dao

- **Scopo:** Strato di accesso ai dati (DAO) per l'interazione con il database.
- **Classi:**
    - DAO.java: Metodi generici DAO.
    - ProductDAO.java, CustomerDAO.java: Implementazioni DAO specifiche per le diverse entità per garantire operazioni CRUD sicure ed efficienti.

2.3 carzone.com.entity

- **Scopo:** Classi entità che rappresentano le tabelle del database.
- **Classi:**
    - Product.java: Rappresenta i prodotti del sistema, con attributi per ID, nome, prezzo e quantità.
    - Customer.java: Contiene i dettagli del cliente, inclusi dati anagrafici e credenziali crittografate.
    - Orders.java: Gestisce i dati relativi agli ordini e implementa relazioni tra clienti e prodotti acquistati.

2.4 carzone.com.servlet

- **Scopo:** Layer controller che gestisce le richieste HTTP e il flusso logico del sistema.
- **Classi:**
    - addproduct.java: Gestisce l'aggiunta di nuovi prodotti al catalogo, con validazione dati.
    - addtocart.java: Gestisce le operazioni del carrello, come l'aggiunta, la rimozione e il calcolo del totale.
    - checkadmin.java: Servlet per l'autenticazione degli amministratori con gestione delle sessioni.

2.5 carzone.com.utility

- **Scopo:** Classi di utilità e helper per funzioni comuni.
- **Classi:**
    - MyUtilities.java: Contiene funzioni per la formattazione delle stringhe, conversioni di dati e altre operazioni di supporto.

3. Interfaccia delle Classi

3. Interfacce delle Classi

Questa sezione descrive le classi e le relative interfacce pubbliche. Ogni descrizione include una panoramica della classe, le sue dipendenze con altre classi e pacchetti, i suoi attributi pubblici, le operazioni disponibili e le eccezioni che possono essere sollevate.

3.1 Package conn

DBConnect.java

- **Descrizione:** Classe per la gestione della connessione al database.
- **Dipendenze:** java.sql.Connection, java.sql.SQLException
- **Attributi Pubblici:** Nessuno
- **Metodi Pubblici:**

  public static Connection getConnection() throws SQLException;
- **Eccezioni Sollevate:** SQLException

3.2 Package dao

DAO.java

- **Descrizione:** Classe base per l'accesso ai dati.
- **Dipendenze:** java.sql.Connection, java.sql.SQLException
- **Attributi Pubblici:** Nessuno
- **Metodi Pubblici:**

public ResultSet executeQuery(String query) throws SQLException;

public int executeUpdate(String query) throws SQLException;

- **Eccezioni Sollevate:** SQLException

DAO2.java, DAO3.java, DAO4.java, DAO5.java

- **Descrizione:** Classi che implementano logiche specifiche per diverse entità del database.
- **Simili a DAO.java, forniscono metodi specifici per la manipolazione dei dati.**

3.3 Package entity

Product.java

- **Descrizione:** Rappresenta un prodotto all'interno del sistema.
- **Dipendenze:** Nessuna
- **Attributi Pubblici:**

private int productId;

private String productName;

private String description;

private double price;

- **Metodi Pubblici:**

public int getProductId();

public void setProductId(int productId);

public String getProductName();

public void setProductName(String productName);

public double getPrice();

public void setPrice(double price);

Customer.java, Orders.java, Cart.java, Category.java, OrderDetails.java, Brand.java

- **Descrizione:** Classi che rappresentano le entità del database (clienti, ordini, carrello, categorie, dettagli ordini, marchi).
- **Dipendenze:** Nessuna
- **Metodi Pubblici:** Definiscono getter e setter per gli attributi corrispondenti.

Berlina.java, Suv.java, Elettrica.java, Dilusso.java

- **Descrizione:** Sottoclassi di Product.java che rappresentano categorie specifiche di veicoli.

UserMaster.java

- **Descrizione:** Gestisce le informazioni sugli utenti amministratori.

3.4 Package servlet

AddProduct.java

- **Descrizione:** Servlet per l'aggiunta di prodotti nel sistema.
- **Dipendenze:** javax.servlet.http.HttpServlet, javax.servlet.ServletException, java.io.IOException
- **Metodi Pubblici:**

  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException;

- **Eccezioni Sollevate:** ServletException, IOException

AddCustomer.java, AddToCart.java, RemoveCart.java, DeleteProduct.java, DeleteCustomer.java

- **Descrizione:** Servlet che gestiscono rispettivamente clienti, carrello e rimozione di prodotti.
- **Metodi Pubblici:** Analoghi a AddProduct.java.

PayProcess.java, CheckAdmin.java, CheckCustomer.java, RemoveOrders.java, RemoveContactUs.java

- **Descrizione:** Servlet per la gestione di pagamenti, autenticazione e gestione ordini.
- **Metodi Pubblici:** Analoghi a AddProduct.java.

UpdateProductName.java, UpdateProductPrice.java, UpdateProductQuantity.java, UpdateProductCategoryId.java, UpdateProductBrandId.java, UpdateProductDescription.java, UpdateProductImage.java

- **Descrizione:** Servlet che aggiornano diversi attributi dei prodotti nel database.
- **Metodi Pubblici:** Analoghi a AddProduct.java.

3.5 Package utility

MyUtilities.java

- **Descrizione:** Classe con metodi di utilità generali.
- **Dipendenze:** Nessuna
- **Metodi Pubblici:**

  public static String formatCurrency(double amount);

  public static boolean isValidEmail(String email);

Ora la sezione rispecchia fedelmente la descrizione richiesta, includendo panoramica, dipendenze, attributi pubblici, operazioni e eccezioni sollevate. Se hai bisogno di altre modifiche, fammelo sapere!

4. Design Pattern

I principali design pattern applicati nel progetto:

- **Data Access Object (DAO):** Utilizzato per separare la logica di accesso ai dati dalla logica di business, migliorando la modularità e la manutenibilità.
- **Model-View-Controller (MVC):** Implementato nelle servlet per separare il modello (gestione dati), la vista (JSP) e il controller (servlet).
- **Object-Oriented Modeling:** Applicato alle entità del database per rappresentare oggetti reali con attributi e metodi.
- **Utility Class:** Pattern usato per definire classi contenenti metodi di supporto riutilizzabili in diverse parti dell'applicazione.

5. Glossario

- **Attributo pubblico**: Variabile dichiarata con visibilità pubblica, accessibile da altre classi.
- **Classe**: Struttura fondamentale della programmazione orientata agli oggetti che rappresenta un'entità con attributi e metodi.
- **DAO (Data Access Object)**: Pattern utilizzato per gestire la persistenza dei dati e l'interazione con il database.
- **Dipendenze**: Classi o pacchetti di cui una determinata classe necessita per funzionare correttamente.
- **Eccezioni**: Errori che possono verificarsi durante l'esecuzione del programma e che devono essere gestiti.
- **Getter e Setter**: Metodi pubblici per accedere e modificare gli attributi privati di una classe.
- **HTTP Servlet**: Classe Java che gestisce richieste e risposte HTTP in un'applicazione web.
- **Interfaccia pubblica**: Insieme di metodi pubblici di una classe accessibili da altre classi.
- **Metodi pubblici**: Funzioni accessibili da altre classi per eseguire operazioni su un oggetto.
- **Pacchetto**: Raggruppamento logico di classi correlate in un progetto.
- **SQL Exception**: Eccezione sollevata durante operazioni di database utilizzando SQL.
- **Utility Class**: Classe contenente metodi di supporto riutilizzabili in diverse parti del codice.