
Laurea Triennale in Informatica - Università degli Studi di Salerno

Corso di *Ingegneria del Software* - Prof. C.Gravino


# Object Design Document

## ReVampAscent

| | |
|---|---|
| Riferimento | ODD_V1.0 |
| Versione | 1.0 |
| Data | 10/12/2025 |
| Destinatario | C.Gravino |


Laurea Triennale in Informatica - Università degli Studi di Salerno

Corso di *Ingegneria del Software* – Prof. C.Gravino

| Presentato da | Antonio Aliberti, Raffaella Di Pasquale, Vincenzo Martucci |
|---|---|

# Team composition

| Ruolo | Nome | Acronimo | Contatti |
|---|---|---|---|
| Team Member | Aliberti Antonio | AA | a.aliberti56@studenti.unisa.it |
| Team Member | Di Pasquale Raffaella | RDP | r.dipasquale5@studenti.unisa.it |
| Team Member | Martucci Vincenzo | VM | v.martucci5@studenti.unisa.it |

## Sommario

# Revision History

| Data | Versione | Descrizione | Autori |
|---|---|---|---|
| 14/12/2025 | 0.1 | Prima stesura del documento | [TEAM MEMBER] |
| 15/12/2025 | 0.6 | Design Pattern | AA |
| 07/01/2026 | 1.0 | Revisione in base ai Suggerimenti del Docente | AA, RDP, VM |
| 08/01/2026 | 1.1 | Revisione in base ai Suggerimenti del Docente | AA, RDP, VM |

# 1. Introduzione

Il sistema ReVamp Ascent è progettato per consentire la compravendita di beni d'arredamento e complementi per la casa attraverso una piattaforma e-commerce moderna, sicura e centralizzata.

Lo scopo principale è facilitare l'incontro tra domanda e offerta mediante un'infrastruttura digitale che automatizza i processi di ricerca, selezione, ordine, pagamento e gestione post-vendita, garantendo un'esperienza d'acquisto fluida e personalizzata per l'utente finale.

# 2. Elementi di Riuso

## 2.1 Design Pattern usati

All'interno del sistema sono stati adottati design pattern (tra quelli trattati a lezione) per affrontare problemi ricorrenti di progettazione, rendere più chiara la separazione delle responsabilità e aumentare la manutenibilità del codice.

### **Adapter Pattern (**Adattamento dell'accesso ai dati JDBC/MySQL al dominio a oggetti)

La persistenza del sistema è realizzata tramite un DBMS relazionale (MySQL) accessibile attraverso JDBC. JDBC espone un'interfaccia di basso livello, orientata a connessioni e risultati tabellari (Connection, PreparedStatement, ResultSet), che non coincide con l'interfaccia attesa dal dominio applicativo, organizzato invece in entità e relazioni modellate tramite oggetti (JavaBeans come Articolo, Cliente, Ordine, Indirizzo, MetodiPagamento, ecc.). Questo disallineamento tra rappresentazione relazionale e modello a oggetti genera complessità nella logica applicativa se non viene confinato in un punto specifico dell'architettura.

Per isolare tale complessità e adattare l'interfaccia JDBC ai bisogni del dominio, è stato adottato l'Adapter Pattern: l'accesso al DB viene mediato da componenti che traducono operazioni applicative "ad alto livello" (centrate su concetti di dominio) in operazioni "a basso livello" (query SQL e mapping su ResultSet), e viceversa.

Istanza del pattern nel progetto e mappatura dei ruoli:

Client: livello applicativo (Servlet) e/o presentazione (JSP) che richiede operazioni di persistenza senza conoscere i dettagli JDBC. Esempi: DettaglioArticoloServlet, CarrelloServlet, ConfermaOrdineServlet, RecensioneServlet.

Target: interfaccia logica attesa dai Client, espressa come insieme di operazioni di persistenza coerenti con il dominio (es. recuperaArticolo(codice), carrelloPerUtente(username), salvaOrdine(ordine), recuperaImmaginePrincipale(codice), ecc.). Nel progetto questa interfaccia è concretizzata dai metodi pubblici dei DAO (API applicativa di persistenza).

Adapter: classi DAO che espongono il Target e incapsulano la traduzione verso JDBC. Esempi: ArticoloDAO, CarrelloDAO, ImmagineArticoloDAO, OrdineDAO, ContenimentoDAO, IndirizzoDAO, MetodiPagamentoDAO, RecensioneDAO.

LegacyClass (Adaptee): l'insieme delle API JDBC e del DB relazionale MySQL (connessioni, statement, ResultSet, schema tabelle) che costituiscono il componente esterno con interfaccia non compatibile con quella attesa dai Client.

Funzionamento operativo (mapping e isolamento della dipendenza)

1. Lettura (DB -> Oggetti)

- Il Client invoca un metodo DAO (es. ArticoloDAO.doRetrieveById(codice)).
- Il DAO costruisce ed esegue query SQL tramite JDBC, ottenendo un ResultSet.
- Il DAO effettua il mapping riga→oggetto, traducendo colonne e tipi SQL (INT, VARCHAR, DECIMAL, DATE/TS) nelle proprietà dei JavaBeans (int, String, double, GregorianCalendar, ecc.), restituendo al Client un oggetto di dominio pronto all'uso.
- In questo modo la logica applicativa lavora solo con Articolo, Ordine, Indirizzo, ecc., senza gestire ResultSet, parsing o gestione statement.

2. Scrittura (Oggetti → DB)

- Il Client crea o modifica un oggetto di dominio e invoca un metodo DAO (es. OrdineDAO.doSave(ordine), RecensioneDAO.doSave(recensione)).
- Il DAO "adatta" lo stato dell'oggetto in parametri SQL, popolando PreparedStatement e gestendo conversioni/normalizzazioni necessarie (ad esempio percentuali di sconto salvate come valore decimale).
- Il DAO esegue l'operazione (INSERT/UPDATE/DELETE) e, dove richiesto, gestisce aspetti tecnici come chiavi generate e aggiornamento dell'identificativo nell'oggetto (es. id_ordine dopo l'inserimento).

Esempi di istanza nel sistema:

- CarrelloDAO: espone operazioni di dominio come aggiungiAlCarrello(username, codiceArticolo, quantita), getCarrelloByUtente(username), pulisciCarrello(username). Internamente traduce tali operazioni in query SQL sullo schema del carrello e ritorna List<Carrello>.

- ImmagineArticoloDAO: espone findMainImage(codiceArticolo) per ottenere direttamente l'immagine principale come oggetto ImmagineArticolo, isolando join/filtri SQL (es. is_principale) dalla logica applicativa.

- OrdineDAO + ContenimentoDAO: l'inserimento ordine (OrdineDAO.doSave) e lo "snapshot" dei dettagli (ContenimentoDAO.doSave) permettono al livello applicativo di operare su concetti di dominio ("salva ordine", "salva righe ordine") senza gestire query multiple e mapping manuali.

Benefici progettuali:

- Disaccoppiamento: Servlet/JSP non dipendono direttamente da JDBC, SQL, ResultSet o schema tabelle; dipendono da un'interfaccia di persistenza coerente con il dominio.

- Separazione responsabilità: la logica applicativa si concentra sul flusso (autenticazione, calcolo totale, validazioni), mentre il DAO concentra le responsabilità di accesso ai dati e conversione tipi.

- Manutenibilità ed evolvibilità: modifiche a schema SQL o a dettagli di persistenza (query, join, tipi) restano confinate nei DAO; la modifica non si propaga al resto del sistema.

- Coerenza architetturale: si riduce la duplicazione di codice JDBC e si uniforma il modo con cui tutte le entità (Articolo, Cliente, Ordine, ecc.) vengono caricate e salvate.

- Testabilità: avendo un punto unico di accesso ai dati, è più semplice isolare e verificare i comportamenti di persistenza (e, se necessario, sostituire l'implementazione dei DAO in contesti di test).

## Facade Pattern

Il sistema è strutturato secondo architettura three-tier. Nel presentation tier (JSP/JS) sono presenti componenti di UI che invocano servizi applicativi tramite richieste HTTP/Ajax;

nel data tier l'accesso ai dati è incapsulato nei DAO che traducono operazioni in query SQL via JDBC.

Per mantenere l'architettura "chiusa" e ridurre l'accoppiamento tra presentazione e componenti interni, nel tier intermedio (application tier) viene adottato il Facade Pattern: le Servlet/Controller forniscono un punto di accesso unico ai servizi applicativi di alto livello e coordinano internamente le collaborazioni tra più classi del sottosistema (DAO e JavaBeans). In questo modo la presentazione non deve conoscere l'ordine delle chiamate, la presenza di più DAO, né le dipendenze tra componenti: invoca un'unica operazione applicativa e riceve l'esito (forward/redirect o risposta JSON).

Mappatura dei ruoli nel progetto:

Client (presentation tier)

- JSP e JavaScript/Ajax che richiedono un servizio applicativo (es. conferma ordine, aggiunta al carrello, caricamento metodi pagamento, dettaglio articolo).

Facade (application tier)

- Servlet che espongono servizi di alto livello e orchestrano più componenti del model:
  - ConfermaOrdineServlet: facciata del sottosistema checkout/ordine (validazioni, calcolo totale, salvataggio ordine e righe, pulizia carrello).
  - DettaglioArticoloServlet: facciata per recupero aggregato di articolo + immagini + recensioni.
  - CarrelloServlet / RimuoviDalCarrelloServlet: facciata per operazioni di carrello (utente loggato o anonimo).
  - VisualizzaIndirizziServlet / AggiungiIndirizzoServlet / ImpostaPreferitoServlet: facciata gestione indirizzi.
  - InserisciMetodoPagamentoAjax / ListaMetodiPagamentoAjax / RimuoviMetodoPagamentoAjax: facciata gestione metodi di pagamento.
  - AggiungiArticoloServlet / SalvaModificaArticoloServlet / FormAggiungiArticoloServlet: facciata backoffice gestione catalogo.

Subsystem (data tier / componenti interni schermati)

- DAO e Bean coinvolti nelle operazioni: ArticoloDAO, CarrelloDAO, OrdineDAO, ContenimentoDAO, IndirizzoDAO, MetodiPagamentoDAO, ImmagineArticoloDAO, RecensioneDAO, ecc.

Esempio: ConfermaOrdineServlet

La servlet implementa un servizio applicativo "conferma ordine" invocato dalla presentazione. Internamente:

- verifica utente in sessione e input selezione carta
- recupera metodo pagamento e indirizzo preferito
- legge il carrello, recupera i prodotti e calcola totale e quantità
- salva l'ordine e le righe (snapshot) su contenimento
- pulisce il carrello e produce l'esito (redirect)

La presentazione effettua una singola invocazione senza conoscere né orchestrare i componenti interni.

Benefici nel contesto three-tier:

- separazione delle responsabilità: presentazione = UI; application = coordinamento e regole di flusso; data = persistenza
- riduzione accoppiamento e maggiore "chiusura" architetturale: la presentazione non usa direttamente i DAO
- manutenibilità: cambiamenti al flusso (checkout, gestione preferito, ecc.) restano confinati nel tier application e nel model
- facilità di integrazione/test: i servizi applicativi sono invocabili come singoli punti d'ingresso

Nel complesso, l'adozione combinata di Adapter e Facade rende coerente l'architettura three-tier: l'Adapter (realizzato nello strato DAO) isola l'interfaccia a basso livello di JDBC e il mapping relazionale→oggetti, mentre il Facade (realizzato dai controller/servlet) fornisce punti di accesso unici ai servizi applicativi e coordina i sottosistemi interni. Questo riduce l'accoppiamento tra presentation e data tier, mantiene l'architettura "chiusa", migliora manutenibilità ed evolvibilità (cambiamenti a query, schema o flussi restano confinati nei livelli corretti) e semplifica integrazione e test dei servizi tramite endpoint applicativi ben definiti.