Laurea Triennale in informatica - Università di Salerno
Corso di *Ingegneria del Software* - Prof. C. Gravino

Design Pattern

Riferimento	NC24_ODD_ver.0.7
Versione	0.7
Data	17/12/25
Destinatario	Prof. Carmine Gravino
Presentato da	NC24
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
17/12/2025	0.1	Prima stesura	Paolo Visconti Simone Sammartano
18/12/2025	0.2	Definizione del 1° Design Pattern	Paolo Visconti Simone Sammartano
21/12/2025	0.3	Definizione del 2° Design Pattern	Mario Branca Gabriele de Luca
28/12/2025	0.4	Aggiunta del Glossario	Paolo Visconti
03/01/2026	0.5	Correzione di alcuni errori	Gabriele De Luca
29/01/2026	0.6	Revisione sulla scelta dei Design Pattern	Mario Branca Simone Sammartano
06/02/2026	0.7	Sistemazione per la consegna finale	Tutto il team

Team Members

Nome	Ruolo	Acronimo	Informazioni di contatto
Mario Branca	Team Member	M.B.	m.branca2@studenti.unisa.it
Gabriele De Luca	Team Member	G.D.L.	g.deluca65@studenti.unisa.it
Simone Sammartano	Team Member	S.S.	s.sammartano1@studenti.unisa.it
Paolo Visconti	Team Member	P.V.	p.visconti4@studenti.unisa.it

Sommario

1. Cenni su scelte di Object Design

In questo documento vengono presentati i design pattern scelti per l'implementazione nel sistema Readify e le motivazioni correlate.

1.1 Design Patterns

Un design pattern è una soluzione generale e riutilizzabile per un problema ricorrente nel contesto della progettazione del software. Si tratta di un modello che fornisce linee guida su come strutturare il codice in modo efficiente, migliorando leggibilità, manutenzione e riusabilità, di seguito presentiamo 2 design pattern individuati ed implementati nel codice.

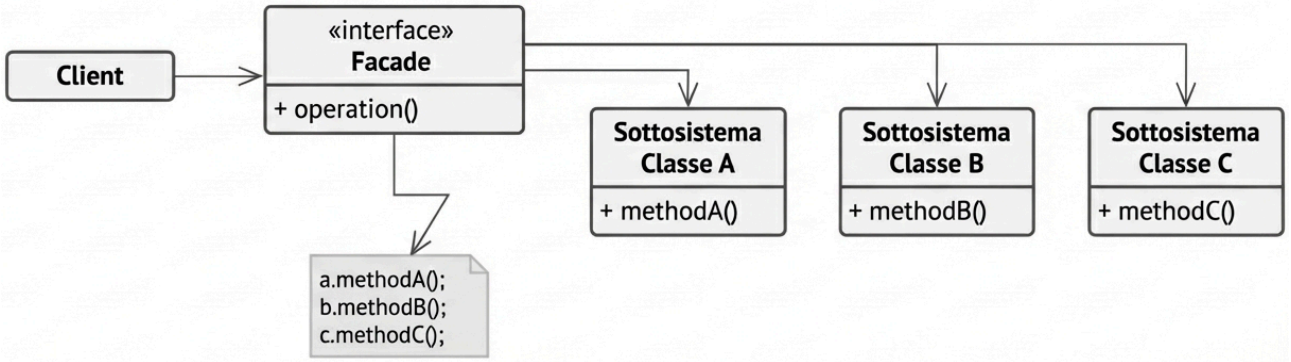
1.1.1 DP1: Facade

Il pattern Facade permette di offrire un'interfaccia unica e più semplice verso un sottosistema che, internamente, risulta articolato. La Facade si occupa di coordinare i vari componenti coinvolti (ad esempio servizi, repository e logiche di validazione), nascondendo al resto dell'applicazione i dettagli implementativi.

In questo modo si riduce l'accoppiamento tra i livelli del sistema: invece di dipendere direttamente da molte classi interne, i client interagiscono con un solo punto di accesso. Ne derivano benefici in termini di leggibilità e manutenibilità, oltre alla possibilità di modificare o estendere l'implementazione interna senza influenzare le componenti che la utilizzano.

Nel sistema Readify, il pattern Facade è stato applicato al sottosistema Carrello, introducendo una CartFacade che espone operazioni ad alto livello come l'aggiunta e la rimozione di libri, l'aggiornamento delle quantità e il calcolo del totale. Le Servlet richiamano quindi direttamente queste operazioni, mentre la Facade gestisce internamente le interazioni con le classi di persistenza e le verifiche necessarie, mantenendo separata la logica applicativa dai dettagli tecnici.

La Facade opera sul carrello persistente associato agli utenti registrati, mentre il carrello di sessione dei visitatori non rientra nell'object design persistente.

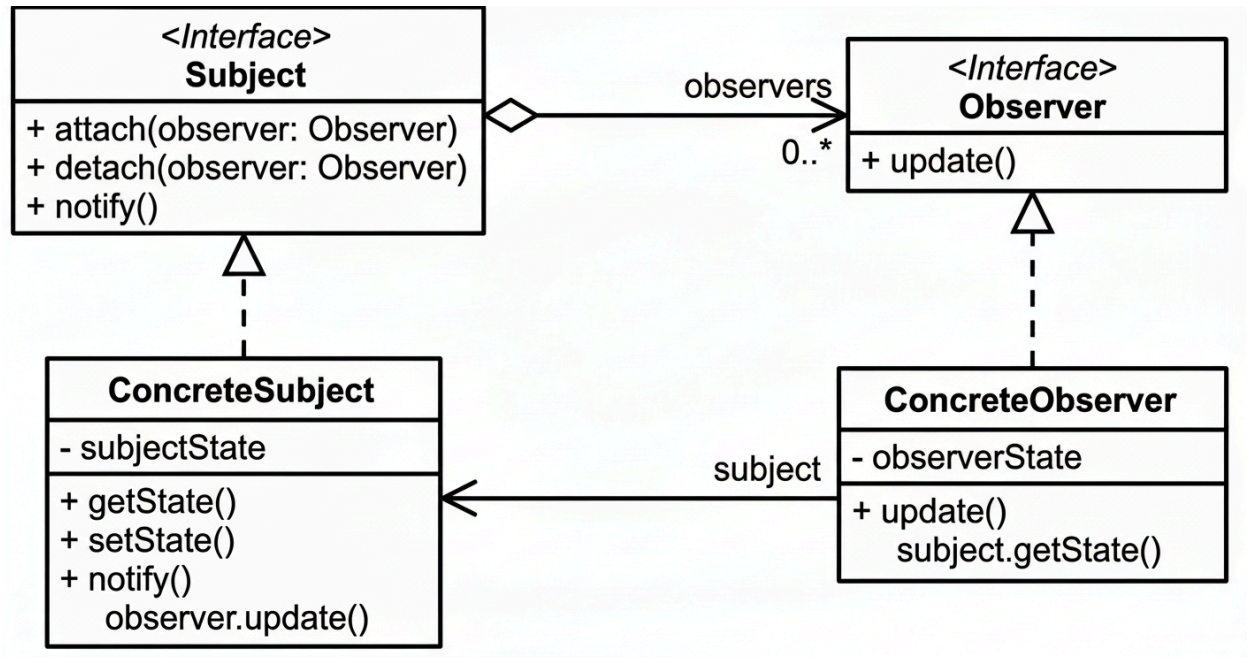


1.1.2 DP2: Observer

L'Observer è un design pattern comportamentale che definisce una dipendenza uno-a-molti tra oggetti, in modo tale che quando un oggetto (detto *subject*) cambia stato, tutti gli oggetti che dipendono da esso (gli *observer*) vengano notificati automaticamente. Lo scopo principale di questo pattern è mantenere sincronizzati più componenti senza creare un forte accoppiamento tra di essi. In questo modo, è possibile aggiungere o rimuovere osservatori dinamicamente, migliorando la flessibilità del sistema e facilitando l'estensione e la manutenzione del codice.

Nella piattaforma, abbiamo utilizzato il pattern Observer per gestire in modo coerente gli effetti che derivano dalle operazioni di gestione del catalogo riservate all'amministratore. Quando l'amministratore aggiunge, modifica o rimuove un libro, oppure aggiorna disponibilità e prezzo, il componente incaricato della gestione del catalogo assume il ruolo di subject e genera un evento che descrive la variazione avvenuta. Tale evento viene quindi propagato agli observer registrati, che, da un lato aggiornano le informazioni mostrate nel catalogo (pagina di dettaglio e risultati di ricerca), dall'altro mantengono coerente il carrello nel caso in cui un libro già inserito subisca variazioni di prezzo o disponibilità, ad esempio segnalando la modifica e adeguando le quantità acquistabili.

Questa scelta progettuale ci consente di separare la logica di aggiornamento del catalogo dalle azioni secondarie che ne conseguono, evitando che un singolo componente debba conoscere e gestire tutte le dipendenze. Di conseguenza, l'applicazione risulta più robusta e facilmente manutenibile rispetto a future estensioni, poiché possono essere aggiunte nuove reazioni agli eventi del catalogo introducendo nuovi observer senza modificare la logica centrale del sottosistema.



2. Glossario

Termine	Definizione
Accoppiamento	Grado di dipendenza tra componenti software; un basso accoppiamento favorisce manutenibilità ed estendibilità del sistema.
Subject	Nel pattern Observer, oggetto che mantiene lo stato osservato e notifica le variazioni agli observer registrati.
Observer	Oggetto che si registra presso un subject per ricevere notifiche in caso di variazione dello stato osservato.
Evento	Oggetto che rappresenta un cambiamento di stato avvenuto all'interno di un subject e che viene propagato agli observer.
Business Logic	Insieme delle regole applicative che definiscono il comportamento funzionale del sistema, indipendentemente dalla presentazione e dalla persistenza dei dati.

Persistenza dei Dati	Capacità del sistema di conservare informazioni in modo stabile su un supporto di memorizzazione, rendendole disponibili anche dopo la terminazione dell'esecuzione dell'applicazione.
DAO (Data Access Object)	Pattern architetturale che incapsula l'accesso ai dati persistenti, separando la logica applicativa dai meccanismi di persistenza
Servlet	Componente server-side che gestisce le richieste HTTP e coordina la logica applicativa.