



Laurea Triennale in Informatica

Università di Salerno

Corso di Ingegneria del Software

Prof. Andrea De Lucia



Progetto EasyDrive Object Design Document(ODD)

Versione 0.1

Data: 19/12/2025

Coordinatore del progetto:

Nome	Matricola
Mario Mascheri	0512120157
Claudio Brizio	0512119716

Partecipanti:

Nome	Matricola
Brizio Claudio	0512119716
Cannella Vincenzo	0512119065
Coscia Matteo	0512121210
Mascheri Mario	0512120157

Scritto da: Vincenzo Cannella

Cronologia delle Revisioni

Data	Versione	Descrizione	Autore
25/11/2025	0.1	Prima stesura e bozza del ODD	Brizio Claudio
09/12/2025	0.2	Seconda stesura del ODD	Brizio Claudio
30/01/2026	0.3	Terza stesura del ODD	Mascheri Mario
03/02/2026	1.0	Versione finale e ufficiale del ODD	Coscia Matteo

Indice

1 Scopo	4
2 Destinatari	4
3 Introduzione	4
3.1 Object design trade-offs	4
3.2 Linee Guida per la Documentazione delle Interfacce	5
3.3 Definizioni, Acronomi e Abbreviazioni	5
3.4 Riferimenti	6
4 Package Decomposition	6
4.1 Panoramica dei Package	6
4.2 Panoramica dei Package	6
4.3 Architettura Three-Tier (Tre Livelli) dei package	7
4.4 Root Package (Presentation Tier)	7
5 Interfacce delle Classi	8
6 Package it.unisa.easydrive.account - Logic Tier	8
6.1 Classe AccountLogic	8
6.2 Sintassi OCL e Note di Design	9
7 Descrizione del Package it.unisa.easydrive.account	10
8 Specifica della Classe AccountDAO	10
9 Glossario della Sintassi OCL (Standard Bruegge)	11
9.1 Componenti del Presentation Tier (Package account)	12
10 Package it.unisa.easydrive.booking - Data Tier	12
10.1 Classe BookingDAO	12
10.2 Componenti del Presentation Tier (Package booking)	14

11 Package it.unisa.easydrive.catalog - Data Tier	14
11.1 Classe CatalogDAO	14
12 Package it.unisa.easydrive.catalog - Logic Tier	15
12.1 Classe CatalogLogic	15
12.2 Componenti del Presentation Tier (Package catalog)	17
13 Package it.unisa.easydrive.core	17
13.1 Descrizione del Package	17
13.2 Classe Database (Data Tier - Singleton)	17
13.3 Classe CoreDAO (Data Tier)	18
13.4 Classe CoreLogic (Logic Tier)	18
14 Package it.unisa.easydrive.payment	18
14.1 Descrizione del Package	18
14.2 Classe PaymentLogic (Logic Tier)	19
14.3 Componenti del Presentation Tier (Package payment)	19
15 Package it.unisa.easydrive.sales - Data Tier	19
15.1 Classe SalesDAO	19
15.2 Legenda e Formalismo OCL (Standard Bruegge)	21
16 Package it.unisa.easydrive.sales - Logic Tier	21
16.1 Classe SalesLogic	21
16.2 Componenti del Presentation Tier (Package sales)	23

1 Scopo

La progettazione a oggetti del sistema è documentata mediante l'**Object Design Document (ODD)**. Questo documento descrive in dettaglio:

- **I compromessi di progettazione** adottati dagli sviluppatori, spiegando le scelte effettuate in termini di prestazioni, manutenibilità, scalabilità e riusabilità dei componenti.
- **Le linee guida seguite per le interfacce dei sottosistemi**, comprese le convenzioni di denominazione, le regole di accesso ai dati e le modalità di gestione degli errori.
- **La decomposizione dei sottosistemi in package e classi**, illustrando la struttura logica del sistema, le responsabilità di ciascun package e le relazioni tra le classi.
- **Le interfacce pubbliche delle classi**, indicando metodi, attributi accessibili, eccezioni previste e modalità di utilizzo da parte di altri componenti.

L'ODD è uno strumento fondamentale per lo scambio di informazioni sulle interfacce tra i diversi team di sviluppo e serve come riferimento durante le attività di test, garantendo coerenza e comprensibilità del sistema.

2 Destinatari

I destinatari principali dell'ODD comprendono:

- **Architetti di sistema**, ovvero gli sviluppatori che partecipano alla progettazione complessiva del sistema.
- **Sviluppatori**, responsabili dell'implementazione dei singoli sottosistemi, che utilizzano l'ODD come guida per la realizzazione coerente delle classi e delle interfacce.
- **Tester**, che fanno riferimento all'ODD per verificare il corretto funzionamento delle interfacce e dei componenti secondo le specifiche di progetto.

3 Introduzione

3.1 Object design trade-offs

Questa sezione analizza i principali compromessi (*trade-offs*) che gli sviluppatori devono affrontare durante la progettazione a oggetti. Bilanciare questi fattori è cruciale per il successo di un progetto software.

Funzionalità vs. Usabilità (Functionality vs. Usability): L'usabilità è un obiettivo primario del design. Tuttavia, l'integrazione di un numero eccessivo di funzionalità può appesantire l'interfaccia utente, rendendola complessa o difficile da apprendere, riducendo la facilità d'uso del sistema.

Affidabilità vs. Costo di Sviluppo (Reliability vs. Development Cost): La robustezza garantisce che il software operi correttamente anche a fronte di input imprevisti. Raggiungere standard di affidabilità elevatissimi richiede investimenti ingenti in termini di tempo, personale specializzato e strumenti di testing, aumentando sensibilmente i costi complessivi del progetto.

Disponibilità vs. Tolleranza ai guasti (Availability vs. Fault Tolerance): In caso di errore interno, il sistema isola e disabilita temporaneamente solo le componenti colpite. Questo approccio previene la propagazione del guasto (*effetto domino*) all'intero sistema, garantendo che la maggior parte dei servizi rimanga operativa (**disponibilità**) a discapito della sospensione temporanea della funzione difettosa.

3.2 Linee Guida per la Documentazione delle Interfacce

Le linee guida per la documentazione delle interfacce e le convenzioni di codifica sono i fattori più importanti per migliorare la comunicazione tra sviluppatori durante la progettazione a oggetti.

Queste linee guida includono regole che gli sviluppatori devono seguire nella progettazione e denominazione delle interfacce.

Gli sviluppatori seguiranno alcune linee guida comuni per la scrittura del codice, al fine di garantire coerenza, leggibilità e facilità di manutenzione del sistema:

- Le classi sono denominate utilizzando sostantivi singolari.
- I metodi sono denominati utilizzando frasi verbali, mentre attributi e parametri utilizzano sostantivi.
- Gli errori devono essere gestiti tramite meccanismi di eccezione, anziché tramite valori di ritorno.
- I nomi delle variabili devono iniziare con una lettera minuscola e utilizzare una notazione in cui ogni parola successiva inizia con una lettera maiuscola.
Le variabili devono essere dichiarate all'inizio del blocco di codice, una sola per riga, mantenendo un allineamento coerente al fine di migliorarne la leggibilità. È consentito corredare le dichiarazioni con commenti esplicativi.
- I nomi dei metodi devono iniziare con una lettera minuscola e seguire la convenzione in cui ogni parola successiva è scritta con l'iniziale maiuscola. Il nome di un metodo dovrebbe generalmente includere un verbo che descrive l'operazione svolta, eventualmente seguito dal riferimento all'oggetto su cui l'azione viene eseguita. I metodi che si riferiscono all'accesso e all'aggiornamento delle variabili devono seguire il modello `getNomeVariabile()` e `setNomeVariabile()`.
- I nomi delle classi e delle pagine devono iniziare con una lettera maiuscola e adottare la convenzione in cui anche le parole successive sono scritte con l'iniziale maiuscola. La denominazione deve essere significativa e descrivere chiaramente il ruolo o la funzionalità della classe o della pagina all'interno del sistema.

3.3 Definizioni, Acronomi e Abbreviazioni

- Interfaccia di Classe: gli attributi e le operazioni pubbliche di una classe.

- Package: raggruppamento di classi, interfacce o file correlati.
- Sottosistema: insieme di servizi legati da una relazione funzionale.
- Interfacce delle Classi: insieme delle signature delle operazioni offerte dalla classe.
- RAD: Requirements Analysis Document – Documento di Analisi dei Requisiti.
- SDD: System Design Document – Documento di Progettazione del Sistema.
- ODD: Object Design Document – Documento di Progettazione a Oggetti.

3.4 Riferimenti

Riferimenti rilevanti includono:

- Documento di Problem Statement relativo a questo progetto. Link alla risorsa: PROBLEM STATEMENT.
- Requirements Analysis Document relativo a questo progetto. Link alla risorsa: RAD.
- System Design Document relativo a questo progetto. Link alla risorsa: SDD.

4 Package Decomposition

Questa sezione descrive la traduzione della scomposizione logica del sistema EasyDrive in una struttura fisica di package, definendo le dipendenze e l'organizzazione del codice sorgente. Il sistema è suddiviso in package funzionali che rispecchiano fedelmente i sottosistemi individuati nel System Design Document (SDD).

4.1 Panoramica dei Package

Il sistema EasyDrive è suddiviso in package funzionali che mappano direttamente la struttura delle cartelle del progetto. Ogni package raggruppa gli script PHP responsabili di una specifica area del sistema.

4.2 Panoramica dei Package

it.unisa.easydrive.core: Cross-Cutting Package. Rappresenta il cuore del *Data Access Tier*. Contiene le utility di connessione al database MySQL e le classi base per la gestione della persistenza condivise da tutti gli altri package.

it.unisa.easydrive.account: Gestisce l'identità degli utenti (login, logout, registrazione).

it.unisa.easydrive.catalog: Gestisce l'esposizione e il dettaglio tecnico dei veicoli.

it.unisa.easydrive.booking: Gestisce la logica dei noleggi e dei preventivi.

it.unisa.easydrive.sales: Package principale per la gestione del carrello e la finalizzazione degli ordini di vendita.

it.unisa.easydrive.payment: Gestisce la validazione dei dati sensibili e l'elaborazione dei pagamenti.

4.3 Architettura Three-Tier (Tre Livelli) dei package

Il sistema è stato progettato e suddiviso seguendo rigorosamente l'architettura *Three-Tier*, al fine di garantire manutenibilità, scalabilità e una chiara separazione delle responsabilità, in conformità a quanto definito nello *Software Design Document (SDD)*.

Tutti i package definiti durante lo sviluppo del codice rispettano coerentemente tale architettura.

I tre livelli individuati nei package per l'implementazione dell'architettura sono i seguenti:

- **Presentation Tier (View):** Comprende i file .php (ad esempio `registrazione.php`, `noleggio.php`) responsabili dell'interazione con l'utente e della presentazione dei dati. Questo livello non contiene logica applicativa né accede direttamente al database.
- **Logic Tier (Business Logic):** Include le classi con suffisso `Logic`, incaricate di implementare le regole di business dell'applicazione. In questo livello vengono gestiti i calcoli (ad esempio il calcolo dei prezzi) e le validazioni dei dati, senza alcun accesso diretto alla persistenza.
- **Data Tier (Persistence):** Comprende le classi con suffisso `DAO` (*Data Access Object*) e la classe `Database`. Questo livello è responsabile esclusivamente dell'accesso ai dati e dell'esecuzione delle query SQL, garantendo l'isolamento della logica di persistenza dal resto dell'applicazione.

4.4 Root Package (Presentation Tier)

Alcuni componenti del sistema non appartengono a un package logico specifico, in quanto risiedono nella cartella radice del progetto e svolgono un ruolo di supporto alla gestione della presentazione.

Tali elementi sono direttamente coinvolti nel controllo del flusso di navigazione e nella composizione delle pagine, senza implementare logica di business né meccanismi di persistenza.

In particolare, rientrano in questa categoria:

- il file `index.php`, che funge da punto di ingresso principale dell'applicazione e gestisce il flusso di controllo della presentazione;
- i fogli di stile CSS, responsabili esclusivamente della definizione dell'aspetto grafico dell'interfaccia;
- i componenti di layout condivisi, quali `header.php` e `footer.php`, utilizzati per garantire coerenza visiva e strutturale tra le diverse pagine.

Questi elementi sono pertanto inclusi nel *Presentation Tier*, pur non essendo associati a un package logico specifico, in quanto non rappresentano unità funzionali autonome ma componenti trasversali a supporto dell'interfaccia utente.

5 Interfacce delle Classi

6 Package it.unisa.easydrive.account - Logic Tier

6.1 Classe AccountLogic

Descrizione: La classe AccountLogic implementa i servizi di business relativi alla gestione dell'identità. Essa funge da intermediario tra il Presentation Tier (script PHP) e il Data Tier (AccountDAO), validando i dati e orchestrando le operazioni di autenticazione e sincronizzazione.

Metodo autentica(identificativo: String, password: String): Any

- **Descrizione:** Verifica la validità delle credenziali fornite per l'accesso al sistema.
- **Pre-condizione (OCL):** context AccountLogic::autentica(id, pass) pre: id.size() > 0 and pass.size() > 0
- **Significato:** Prima di tentare l'accesso, l'identificativo e la password forniti dall'utente devono essere non vuoti.
- **Post-condizione (OCL):** post: result <> false implies result.oclIsTypeOf(Map)
- **Significato:** Il sistema garantisce che, in caso di successo, venga restituita una struttura dati contenente le informazioni dell'utente autenticato.

Metodo validaEInregistra(dati: Map, password_raw: String): Any

- **Descrizione:** Valida i requisiti di sicurezza e anagrafici prima di persistere un nuovo account nel database.
- **Pre-condizione (OCL):** context AccountLogic::validaEInregistra(dati, pass) pre: dati->includesKey('username', 'email', 'nome', 'cognome', 'cap')
- **Significato:** Il metodo richiede obbligatoriamente un set completo di chiavi informative per procedere con l'elaborazione.
- **Post-condizione (OCL):**

```
post: if (pass.size() >= 8 and dati['cap'].size() = 5 and dati['provincia'].size() = 2)
      then result = true
      else result.oclIsTypeOf(String)
      endif
```
- **Significato:** Se la password è sicura e i dati geografici sono formalmente corretti, l'operazione ha successo; altrimenti, viene garantita la restituzione di una stringa descrittiva dell'errore.

Metodo sincronizzaCarrello(username: String): void

- **Descrizione:** Allinea lo stato del carrello in sessione con gli interessi salvati nel database, rimuovendo veicoli non più disponibili.
- **Pre-condizione (OCL):** context AccountLogic::sincronizzaCarrello(username) pre: username <> null
- **Post-condizione (OCL):** post: session['carrello']->forAll(t | self.dao.checkVeicoloDisponibile(t) = true)
- **Significato:** Il sistema garantisce che, dopo la sincronizzazione, ogni veicolo presente nel carrello dell'utente sia effettivamente disponibile alla vendita o al noleggio.

Metodo ottieniProfilo(username: String): Map

- **Descrizione:** Recupera le informazioni complete dell'utente e formatta le etichette per la visualizzazione.
- **Pre-condizione (OCL):** context AccountLogic::ottieniProfilo(username) pre: username <> null
- **Post-condizione (OCL):** post: result <> null implies result->includesKey('ruolo_{formattato}')

• **Significato:** In caso di successo, il sistema garantisce la presenza di un attributo aggiuntivo relativo al ruolo, pronto per essere mostrato nell'interfaccia utente.

Metodo logout(): void

- **Descrizione:** Esegue la chiusura sicura della sessione, rimuovendo dati temporanei e cookie.
- **Pre-condizione (OCL):** true
- **Post-condizione (OCL):** post: session.isEmpty()
- **Significato:** Il sistema garantisce la distruzione completa dello stato di sessione sul server.

6.2 Sintassi OCL e Note di Design

- **oclIsTypeOf(String):** Utilizzato per mappare il ritorno dinamico di PHP; specifica che se l'operazione non restituisce un valore booleano, deve restituire un messaggio testuale.
- **size():** Sostituisce l'uso di espressioni regolari per il controllo della lunghezza di password, CAP e sigle provinciali.
- **includesKey:** Verifica l'integrità del dizionario (Map) passato dal Presentation Tier.
- **session:** In OCL, questo termine rappresenta l'oggetto di stato globale del sistema durante l'esecuzione della richiesta.

7 Descrizione del Package it.unisa.easydrive.account

Il package gestisce la persistenza e la logica degli accessi. In particolare, la classe **AccountDAO** funge da interfaccia tra il sistema e il database MySQL, incapsulando la creazione delle query e l'estrazione dei dati per le tabelle Account e Wishlist_Interesse.

8 Specifica della Classe AccountDAO

Metodo getAccountByIdentifier(id: String): Map

- **Pre-condizione (OCL):** context AccountDAO::getAccountByIdentifier(id) pre: id <> null
- **Spiegazione:** Prima dell'invocazione, è necessario che l'identificativo fornito (username o email) non sia nullo.
- **Post-condizione (OCL):** post: result <> null implies (result['username'] = id or result['email'] = id)
- **Spiegazione:** Il sistema garantisce che, se viene trovato un account, i suoi attributi identificativi corrispondano a quanto richiesto.

Metodo esisteGia(username: String, email: String): Boolean

- **Pre-condizione (OCL):** context AccountDAO::esisteGia(username, email) pre: username <> null and email <> null
- **Spiegazione:** Per verificare l'unicità, entrambi i parametri di ricerca devono essere validi.
- **Post-condizione (OCL):** post: result = Account.allInstances()->exists(a | a.username = username or a.email = email)
- **Spiegazione:** Il metodo restituisce vero se esiste almeno una riga nel database che viola il vincolo di unicità per lo username o per l'email.

Metodo inserisciAccount(dati: Map): Boolean

- **Pre-condizione (OCL):** context AccountDAO::inserisciAccount(dati) pre: Account.allInstances()->select(a | a.username = dati['username'])->isEmpty()
- **Spiegazione:** L'inserimento può avvenire solo se lo username scelto non appartiene a nessuna istanza già registrata nel sistema.
- **Post-condizione (OCL):** post: result = true implies Account.allInstances()->size() = Account.allInstances()@pre->size() + 1
- **Spiegazione:** In caso di successo, il sistema garantisce che il numero totale di record nella tabella Account sia aumentato esattamente di un'unità.

Metodo checkVeicoloDisponibile(telaio: String): Boolean

- **Pre-condizione (OCL):** context AccountDAO::checkVeicoloDisponibile(telaio) pre: telaio <> null
- **Spiegazione:** Il controllo della disponibilità richiede un codice telaio valido come parametro di ricerca.
- **Post-condizione (OCL):** post: result = true implies Veicolo.allInstances()->select(v | v.telaio = telaio).stato = 'Disponibile'
- **Spiegazione:** Se il risultato è positivo, viene garantito che il veicolo sia nello stato corretto per la vendita o il noleggio.

Metodo removeWishlistItem(username: String, telaio: String): Boolean

- **Pre-condizione (OCL):** context AccountDAO::removeWishlistItem(user, tel) pre: user <> null and tel <> null
- **Post-condizione (OCL):** post: result = true implies Wishlist_Interesse.allInstances()->select(w | w.username = user and w.telaio = tel)->isEmpty()
- **Spiegazione:** Dopo la rimozione, il sistema garantisce che non esista più alcuna associazione tra l'utente e il veicolo nella tabella degli interessi.

Metodo getFullAccountByUsername(username: String): Map

- **Pre-condizione (OCL):** context AccountDAO::getFullAccountByUsername(username) pre: username <> null
- **Post-condizione (OCL):** post: result <> null implies result['username'] = username
- **Spiegazione:** Il recupero dei dati anagrafici completi garantisce la corrispondenza con lo username dell'utente loggato.

9 Glossario della Sintassi OCL (Standard Bruegge)

- **allInstances()**: Insieme di tutti i record presenti in una tabella (classe persistente).
- **exists**: Operatore logico che verifica l'esistenza di almeno un elemento che soddisfi la condizione.
- **isEmpty()**: Verifica che una collezione risultante sia vuota (usato per l'unicità).
- **@pre**: Indica il valore o lo stato di una collezione prima dell'esecuzione del metodo.
- **result**: Parola chiave che identifica il valore di ritorno dell'operazione.

9.1 Componenti del Presentation Tier (Package account)

I componenti elencati in questa sezione, pur risiedendo logicamente nel package `it.unisa.easydrive.account`, rappresentano il punto di ingresso del sistema (*Presentation Tier*).

A differenza delle classi del Logic e Data Tier, questi file (.php) agiscono come **Controller di interfaccia**: il loro compito è raccogliere le richieste HTTP, gestire la sessione e delegare l'intera logica di business e di persistenza ai metodi delle classi `AccountLogic` e `AccountDAO`.

Assenza di Interfaccia Pubblica Formale Si specifica che per tali componenti **non viene definita un'interfaccia pubblica formalizzata in OCL**, in quanto non espongono metodi richiamabili da altri oggetti del sistema. Essi sono script procedurali che fungono da orchestratori.

- **registrazione.php**: Raccoglie i dati anagrafici e di residenza dal form e invoca il metodo `validaEInregistra()` della classe `AccountLogic`. Gestisce la visualizzazione dei messaggi di errore (Flash Messages) in caso di violazione dei contratti.
- **login.php**: Riceve le credenziali di accesso e interroga `AccountLogic::autentica()`. In caso di successo, inizializza le variabili di sessione e invoca la sincronizzazione del carrello.
- **profilo.php**: Funge da vista per l'utente autenticato. Recupera le informazioni necessarie invocando `AccountLogic::ottieneProfilo()`.
- **Logout.php**: Script dedicato alla terminazione della sessione. Delega la distruzione dei dati persistenti in memoria temporanea al metodo `AccountLogic::logout()`.

10 Package `it.unisa.easydrive.booking` - Data Tier

10.1 Classe BookingDAO

Descrizione: La classe `BookingDAO` è il componente del Data Tier responsabile della persistenza dei dati relativi al modulo di noleggio. Gestisce il recupero delle tariffe, il controllo della disponibilità temporale e l'inserimento atomico delle prenotazioni, assicurando la coerenza tra la tabella delle prenotazioni e lo stato del veicolo.

Metodo `getVehicleRate(telaio: String): Map`

- **Pre-condizione (OCL):** context `BookingDAO::getVehicleRate(telaio)` pre: `telaio <> null`
- **Spiegazione:** È necessario fornire un codice telaio valido per recuperare le informazioni economiche del veicolo.
- **Post-condizione (OCL):** post: `result <> null implies result->includesKey('tariffaNoleggioGiorno')`

- **Spiegazione:** Se il veicolo esiste, il sistema garantisce la restituzione di una mappa contenente la tariffa giornaliera per il calcolo del preventivo.

Metodo checkCollisioniDate(telaio: String, inizio: String, fine: String): Boolean

- **Pre-condizione (OCL):** context BookingDAO::checkCollisioniDate(telaio, inizio, fine) pre: inizio < fine
- **Spiegazione:** Il controllo delle collisioni può essere effettuato solo se l'intervallo temporale richiesto è logicamente valido (l'inizio precede la fine).
- **Post-condizione (OCL):** post: result = true implies
`Prenotazione_Noleggio.allInstances()->exists(p | p.telaio = telaio and
(p.data_inizio <= fine and p.data_fine >= inizio) and p.stato <> 'Annullata')`
- **Spiegazione:** Il metodo restituisce vero se e solo se esiste già una prenotazione attiva nel database che si sovrappone temporalmente al periodo richiesto.

Metodo insertPrenotazione(dati: Map): Boolean

- **Pre-condizione (OCL):** context BookingDAO::insertPrenotazione(dati) pre: self.checkCollisioniDate(dati['telaio'], dati['data_inizio'], dati['data_fine']) = false
- **Spiegazione:** L'inserimento può procedere solo se è stato preventivamente verificato che il veicolo sia libero per l'intero periodo selezionato.
- **Post-condizione (OCL):**
`post: result = true implies (`
`Prenotazione_Noleggio.allInstances()->size() =`
`Prenotazione_Noleggio.allInstances()@pre->size() + 1`
`and Veicolo.allInstances()->select(v | v.telaio = dati['telaio']).stato =`
`'InPrenotazione')`
- **Spiegazione:** In caso di successo, il sistema garantisce l'incremento di una unità dei record di prenotazione e l'aggiornamento simultaneo dello stato del veicolo a 'InPrenotazione'.

Metodo updateStatoVeicolo(telaio: String, stato: String): Boolean

- **Pre-condizione (OCL):** context BookingDAO::updateStatoVeicolo(telaio, stato) pre: telaio <> null and stato <> null
- **Post-condizione (OCL):** post: result = true implies Veicolo.allInstances()->select(v | v.telaio = telaio).stato = stato
- **Spiegazione:** Il sistema garantisce che, dopo l'aggiornamento, lo stato persistente del veicolo corrisponda esattamente al valore passato come parametro.

10.2 Componenti del Presentation Tier (Package booking)

I componenti elencati in questa sezione gestiscono il flusso di interazione per la prenotazione dei noleggi. Pur risiedendo nel package `it.unisa.easydrive.booking`, essi operano come **Controller di interfaccia** all'interno del Presentation Tier.

Assenza di Interfaccia Pubblica Formale Analogamente ai componenti di gestione account, questi script non definiscono classi o metodi richiamabili esternamente. La loro natura è procedurale e la loro correttezza dipende interamente dal rispetto dei contratti OCL definiti nel Logic Tier (`BookingLogic`). Pertanto, non viene fornita una specifica OCL per questi file.

- **Prenota_noleggio.php:** Rappresenta l'interfaccia visuale (form) per l'utente. Il suo compito è verificare lo stato della sessione, recuperare i dati del veicolo selezionato tramite `BookingLogic::getDatiVeicoloPerNoleggio()` e permettere l'inserimento dei parametri temporali (date di inizio e fine) e logistici (luoghi di ritiro e consegna).
- **Noleggio_process.php:** Agisce come orchestratore per la sottomissione dei dati. Riceve i parametri via POST, invoca l'elaborazione del business tramite `BookingLogic::processaPrenotazione()` e, in base all'esito della transazione atomica, reindirizza l'utente verso lo storico delle operazioni (in caso di successo) o restituisce un messaggio di errore nel dettaglio del veicolo.

11 Package `it.unisa.easydrive.catalog` - Data Tier

11.1 Classe CatalogDAO

Descrizione: La classe `CatalogDAO` è responsabile della gestione della persistenza e del recupero delle informazioni relative al parco auto. Gestisce query complesse che includono *join* con la tabella delle immagini, filtri dinamici per la ricerca avanzata e l'estrazione dei dettagli tecnici di singole unità.

Metodo `getFeaturedVehicles(limit: Integer): Collection`

- **Descrizione:** Recupera un set limitato di veicoli contrassegnati come disponibili, solitamente utilizzato per la sezione "In evidenza" della Home Page.
- **Pre-condizione (OCL):** context `CatalogDAO::getFeaturedVehicles(limit)` pre: `limit > 0`
- **Spiegazione:** Il parametro di limitazione dei risultati deve essere un numero intero positivo.
- **Post-condizione (OCL):** post: `result->size() <= limit and result->forAll(v | v.stato = 'Disponibile')`
- **Spiegazione:** Il sistema garantisce che il numero di veicoli restituiti non superi il limite impostato e che ogni veicolo nella collezione sia effettivamente disponibile.

Metodo getFilteredVehicles(filters: Map): Collection

- **Descrizione:** Esegue una ricerca parametrica basata su criteri variabili quali nome (marca/modello), range di prezzo, disponibilità e opzioni di noleggio.
- **Pre-condizione (OCL):** context CatalogDAO::getFilteredVehicles(filters) pre: filters <> null
- **Spiegazione:** È necessario fornire una mappa di parametri (anche vuota) per inizializzare i criteri di filtraggio.
- **Post-condizione (OCL):**

```
post: if (filters['disponibile'] = true)
      then result->forAll(v | v.stato = 'Disponibile')
      else result->notEmpty()
      endif
```

- **Spiegazione:** Se l'utente ha richiesto solo veicoli disponibili, il sistema garantisce che la collezione risultante rispetti rigorosamente tale vincolo.

Metodo getVeicoloByTelaio(telaio: String): Map

- **Descrizione:** Recupera i dati tecnici completi e l'immagine principale di un veicolo specifico identificato dal telaio.
- **Pre-condizione (OCL):** context CatalogDAO::getVeicoloByTelaio(telaio) pre: telaio <> null
- **Spiegazione:** La ricerca richiede obbligatoriamente il codice identificativo univoco del veicolo.
- **Post-condizione (OCL):** post: result <> null implies (result['telaio'] = telaio)
- **Spiegazione:** In caso di riscontro positivo, il sistema garantisce che i dati restituiti appartengano esattamente al veicolo richiesto.

12 Package it.unisa.easydrive.catalog - Logic Tier

12.1 Classe CatalogLogic

Descrizione: La classe CatalogLogic implementa la logica di business relativa alla consultazione del catalogo veicoli. Agisce come mediatore tra il Presentation Tier e il Data Tier, occupandosi di normalizzare i parametri di ricerca, gestire i contenuti in evidenza per la Home Page e arricchire i dati grezzi dei veicoli con attributi calcolati (formattazione URL e flag di disponibilità).

Metodo getHomePageHighlights(): Collection

- **Descrizione:** Fornisce una selezione ridotta di veicoli per la visualizzazione rapida nella pagina principale.
- **Pre-condizione (OCL):** context CatalogLogic::getHomePageHighlights() pre: true
- **Spiegazione:** Il metodo è sempre invocabile senza vincoli di input.
- **Post-condizione (OCL):** post: result->size() <= 4
- **Spiegazione:** Il sistema garantisce che non vengano mai restituiti più di 4 elementi, rispettando il layout della Home Page.
- **Sintassi OCL:** size() è l'operatore standard per contare gli elementi di una collezione.

Metodo ricercaVeicoli(params: Map): Collection

- **Descrizione:** Riceve i parametri grezzi dalla richiesta HTTP, li organizza in una struttura di filtri pulita e delega la ricerca al Data Tier.
- **Pre-condizione (OCL):** context CatalogLogic::ricercaVeicoli(params) pre: params <> null
- **Spiegazione:** È necessario che l'oggetto contenente i parametri di ricerca sia stato inizializzato.
- **Post-condizione (OCL):** post: result <> null
- **Spiegazione:** Il sistema garantisce la restituzione di una collezione di risultati, che può essere vuota ma mai nulla.

Metodo fornisceDettagliVeicolo(telaio: String): Map

- **Descrizione:** Recupera le informazioni di un singolo veicolo e applica trasformazioni di business sugli URL delle immagini e sullo stato di vendita.
- **Pre-condizione (OCL):** context CatalogLogic::fornisceDettagliVeicolo(telaio) pre: telaio <> null
- **Spiegazione:** Il metodo richiede un codice telaio valido per poter identificare il veicolo.
- **Post-condizione (OCL):**

```
post: result <> null implies (
    result->includesKey('url_immagine_formattata') and
    result['is_disponibile'] = (result['stato'].toLowerCase() = 'disponibile')
)
```
- **Spiegazione:** Se il veicolo viene trovato, il sistema garantisce l'aggiunta di due attributi calcolati: l'URL dell'immagine pronto per il rendering e un flag booleano che indica se il veicolo è acquistabile.
- **Sintassi OCL:** includesKey verifica l'arricchimento della mappa dati; toLowerCase() normalizza la stringa per un confronto robusto dello stato.

12.2 Componenti del Presentation Tier (Package catalog)

I componenti descritti in questa sezione gestiscono l'esplorazione del parco auto da parte dell'utente. Essendo parte del **Presentation Tier**, questi script fungono da interfaccia tra le richieste dell'utente e la logica applicativa del package `it.unisa.easydrive.catalog`.

Assenza di Interfaccia Pubblica Formale In linea con l'architettura Three-Tier adottata, questi file non implementano classi o metodi richiamabili da altri oggetti, ma operano come script procedurali (**Controller di interfaccia**). La correttezza del loro comportamento è garantita dai contratti OCL definiti in `CatalogLogic`.

- **Catalogo.php:** Gestisce la visualizzazione collettiva dei veicoli. Riceve i parametri di filtraggio (marca, modello, range di prezzo) tramite il metodo GET e invoca `CatalogLogic::ricercaVeicoli()`. Si occupa inoltre di mantenere lo stato dei filtri nella barra laterale per permettere ricerche incrementalî e gestisce il rendering dinamico delle card dei veicoli, differenziando tra opzioni di acquisto e di noleggio.
- **detttaglio_veicolo.php:** Rappresenta la scheda tecnica approfondita di una singola unità. Riceve l'identificativo del veicolo (*telaio*) dall'URL e richiede i dati arricchiti a `CatalogLogic::fornisceDettagliVeicolo()`. Oltre a mostrare le specifiche tecniche (chilometraggio, tipo cambio, ecc.), gestisce condizionalmente la visualizzazione dei pulsanti di azione: permette l'aggiunta al carrello solo se il veicolo è marcato come *disponibile* dalla logica di business.

13 Package `it.unisa.easydrive.core`

13.1 Descrizione del Package

Il package `core` rappresenta il livello infrastrutturale del sistema EasyDrive. Esso implementa il pattern **Singleton** per la gestione della connessione al database e fornisce servizi di utility trasversali, come il recupero e la formattazione delle icone di sistema utilizzate nel Presentation Tier.

13.2 Classe Database (Data Tier - Singleton)

Invariante: `context Database inv: self.instance <> null`

Spiegazione: Una volta inizializzato il sistema, l'istanza della connessione deve essere permanentemente disponibile in memoria.

Metodo `getConnection(): mysqli`

- **Pre-condizione (OCL):** `context Database::getConnection() pre: true`
- **Spiegazione:** Il metodo può essere invocato in qualunque momento per ottenere l'accesso alla risorsa database.
- **Post-condizione (OCL):** `post: result <> null and result.connect_error = null`

- **Spiegazione:** Il sistema garantisce che venga restituito un oggetto di connessione attivo e che non vi siano stati errori durante l'handshake con il DBMS MySQL.
- **Sintassi OCL:** `result` è la parola chiave per il valore di ritorno; `connect_error` è la proprietà dell'oggetto `mysqli` che deve risultare nulla.

13.3 Classe CoreDAO (Data Tier)

Metodo getAllIcons(): Collection

- **Pre-condizione (OCL):** `context CoreDAO::getAllIcons() pre: true`
- **Post-condizione (OCL):** `post: resultoclIsTypeOf(Collection)`
- **Spiegazione:** Il sistema garantisce la restituzione di un set di risultati contenente le coppie nome/url delle icone presenti nella tabella `icone`.
- **Sintassi OCL:** `oclIsTypeOf` verifica che la struttura dati restituita sia una collezione di record.

13.4 Classe CoreLogic (Logic Tier)

Invariante: `context CoreLogic inv: self.dao <> null`

Metodo getFormattedIcons(): Map

- **Pre-condizione (OCL):** `context CoreLogic::getFormattedIcons() pre: true`
- **Post-condizione (OCL):** `post: result->forAll(url | url.size() > 0)`
- **Spiegazione:** Il sistema garantisce la restituzione di una mappa di icone dove ogni URL è stato processato e non risulta vuoto.
- **Sintassi OCL:** `forAll` è il quantificatore universale che verifica la condizione su ogni elemento della mappa restituita.

14 Package it.unisa.easydrive.payment

14.1 Descrizione del Package

Il package `payment` è responsabile della validazione formale dei dati di pagamento forniti dall'utente. Agisce come un filtro di sicurezza finale, assicurando che i dati della carta di credito rispettino gli standard richiesti prima che il sistema proceda alla finalizzazione dell'acquisto nel Data Tier.

14.2 Classe PaymentLogic (Logic Tier)

Descrizione: Classe di business dedicata alla validazione dei parametri finanziari. Non interagisce direttamente con il database, limitandosi a verificare la correttezza formale degli input secondo le regole di business stabilite.

Metodo validaDatiCarta(dati: Map): Boolean

- **Pre-condizione (OCL):** context PaymentLogic::validaDatiCarta(dati) pre: dati->includesKey('numero_carta') and dati->includesKey('cvv')
- **Spiegazione:** Il metodo richiede obbligatoriamente il numero della carta e il codice di sicurezza (CVV) per poter avviare la procedura di validazione.
- **Post-condizione (OCL):** post: result = true implies (dati['numero_carta'].size() = 16 and dati['cvv'].size() = 3)
- **Spiegazione:** Il sistema garantisce che la validazione abbia esito positivo solo se la carta presenta esattamente 16 cifre e il CVV esattamente 3 cifre.
- **Sintassi OCL:** implies (\Rightarrow) stabilisce il vincolo di dipendenza tra l'esito positivo e la correttezza delle lunghezze; size() calcola il numero di caratteri.

14.3 Componenti del Presentation Tier (Package payment)

Assenza di Interfaccia Pubblica Formale Lo script seguente opera come orchestratore di interfaccia e non espone metodi per altri oggetti, pertanto la sua correttezza è derivata dal rispetto dei contratti definiti in PaymentLogic e SalesLogic.

- **Payment.php:** Gestisce l'interfaccia di checkout. Recupera il totale dell'ordine tramite SalesDAO::getCartTotal() per la visualizzazione all'utente e raccoglie i dati del form di pagamento. Prima di consentire la sottomissione finale verso la conferma dell'ordine, delega la validazione formale dei campi a PaymentLogic::validaDatiCarta(). In caso di dati non validi, impedisce il proseguimento del flusso, proteggendo l'integrità delle operazioni di vendita.

15 Package it.unisa.easydrive.sales - Data Tier

15.1 Classe SalesDAO

Descrizione: La classe SalesDAO gestisce l'interfaccia di persistenza per le operazioni di vendita e la gestione del carrello persistente (*Wishlist*). Si occupa del recupero dei prezzi, del calcolo dei totali e dell'aggiornamento atomico degli stati dei veicoli in seguito a un acquisto, garantendo l'integrità dei dati nelle tabelle Ordine_Vendita, Wishlist_Interesse e Veicolo.

Metodo checkDisponibilita(telaio: String): Boolean

- **Descrizione:** Verifica se un veicolo è attualmente disponibile per la vendita o per l'aggiunta al carrello.
- **Pre-condizione (OCL):** context SalesDAO::checkDisponibilita(telaio) pre: telaio <> null
- **Post-condizione (OCL):** post: result = true implies Veicolo.allInstances()->select(v | v.telaio = telaio).stato = 'Disponibile'
- **Spiegazione:** Il sistema garantisce che un esito positivo della funzione corrisponda univocamente allo stato di disponibilità del veicolo nel database.

Metodo getCartTotal(telaiArray: Collection): Real

- **Descrizione:** Calcola la somma algebrica dei prezzi di vendita per un insieme di veicoli identificati dai rispettivi telai.
- **Pre-condizione (OCL):** context SalesDAO::getCartTotal(telaiArray) pre: telaiArray->notEmpty()
- **Post-condizione (OCL):** post: result = telaiArray->collect(t | Veicolo.allInstances()->select(v | v.telaio = t).prezzoVendita)->sum()
- **Spiegazione:** Il sistema garantisce che il totale restituito sia l'esatta sommatoria dei prezzi di vendita di tutti i veicoli passati in input.
- **Sintassi OCL:** collect estrae il prezzo per ogni telaio; sum() esegue la sommatoria sulla collezione risultante.

Metodo insertInteresse(username: String, telaio: String): Boolean

- **Descrizione:** Salva in modo persistente l'associazione tra un utente e un veicolo nel carrello/wishlist del database.
- **Pre-condizione (OCL):** context SalesDAO::insertInteresse(user, tel) pre: self.existsInteresse(user, tel) = false
- **Spiegazione:** L'inserimento può avvenire solo se il veicolo non è già presente tra gli interessi dell'utente (evita duplicati).
- **Post-condizione (OCL):** post: result = true implies Wishlist_Interesse.allInstances()->exists(w | w.username = user and w.telaio = tel)

Metodo createOrderRecord(codice: String, prezzo: Real, metodo: String, user: String, tel: String): Boolean

- **Descrizione:** Crea un nuovo record di vendita definitiva associando l'utente al veicolo acquistato.
- **Pre-condizione (OCL):** context SalesDAO::createOrderRecord(c, p, m, u, t) pre: p > 0 and c.size() > 0
- **Post-condizione (OCL):** post: result = true implies Ordine_Vendita.allInstances()->exists(o | o.codice_univoco = c)
- **Spiegazione:** In caso di successo, il sistema garantisce la persistenza del nuovo ordine con il codice univoco generato dalla logica di business.

Metodo updateVehicleStatus(telaio: String, nuovoStato: String): Boolean

- **Descrizione:** Modifica lo stato operativo di un veicolo (es. da 'Disponibile' a 'Venduto').
- **Pre-condizione (OCL):** context SalesDAO::updateVehicleStatus(telaio, stato) pre: telaio <> null and stato <> null
- **Post-condizione (OCL):** post: result = true implies Veicolo.allInstances()->select(v | v.telaio = telaio).stato = nuovoStato

15.2 Legenda e Formalismo OCL (Standard Bruegge)

- **collect:** Operatore di iterazione che trasforma una collezione di oggetti in una collezione di valori (nel nostro caso, telai in prezzi).
- **sum()**: Operazione aritmetica standard sulle collezioni numeriche.
- **implies:** Definisce il contratto di garanzia: se il metodo restituisce **true**, allora lo stato del sistema deve essere coerente con la clausola successiva.
- **allInstances()**: Rappresenta l'astrazione OCL dell'intera tabella del database.

16 Package it.unisa.easydrive.sales - Logic Tier

16.1 Classe SalesLogic

Descrizione: La classe **SalesLogic** implementa la logica di business per la gestione del carrello e del workflow di vendita. Essa coordina la sincronizzazione tra la sessione utente e la *wishlist* su database, calcola i totali economici e garantisce l'atomicità della transazione di acquisto, gestendo il passaggio dei veicoli dallo stato di disponibilità a quello di vendita.

Metodo aggiungiAlCarrello(telaio: String, username: String): Map

- **Pre-condizione (OCL):** context SalesLogic::aggiungiAlCarrello(t, u) pre:
self.dao.checkDisponibilita(t) = true
- **Significato:** Un veicolo può essere aggiunto al carrello solo se il Data Tier ne conferma lo stato attuale come 'Disponibile'.
- **Post-condizione (OCL):** post: session['carrello']->includes(t) and (u <> null implies WishlistInteresse.allInstances()->exists(w|w.telaio = t and w.username = u))
- **Significato:** Il sistema garantisce l'inserimento del telaio nella sessione e, se l'utente è autenticato, assicura la persistenza dell'interesse nel database.
- **Sintassi OCL:** includes verifica la presenza nell'insieme di sessione; exists verifica la persistenza sul database.

Metodo sincronizzaEPulisciCarrello(username: String): void

- **Pre-condizione (OCL):** context SalesLogic::sincronizzaEPulisciCarrello(u) pre:
session['carrello']->notEmpty()
- **Post-condizione (OCL):** post: session['carrello']->forAll(t |
self.dao.getVehicleStatus(t) = 'Disponibile')
- **Significato:** Il sistema garantisce che, dopo l'esecuzione, il carrello contenga esclusivamente veicoli il cui stato sul database è ancora 'Disponibile', rimuovendo automaticamente quelli venduti o prenotati da altri.
- **Sintassi OCL:** forAll quantifica universalmente l'integrità di ogni elemento della collezione rispetto allo stato del DB.

Metodo finalizzaAcquisto(carrello: Collection, username: String): Any

- **Descrizione:** Gestisce la transazione di vendita atomica, trasformando gli interessi in ordini definitivi.
- **Pre-condizione (OCL):** context SalesLogic::finalizzaAcquisto(c, u) pre:
c->notEmpty() and u <> null
- **Significato:** L'acquisto può essere finalizzato solo se il carrello non è vuoto e l'utente è correttamente identificato.
- **Post-condizione (OCL):**

```
post: result = true implies (
    c->forAll(t | self.dao.getVehicleStatus(t) = 'Venduto') and
    Ordine_Vendita.allInstances()->size() =
    Ordine_Vendita.allInstances()@pre->size() + c->size()
)
```

- **Significato:** In caso di successo della transazione, il sistema garantisce che tutti i veicoli acquistati siano passati allo stato 'Venduto' e che il numero di ordini nel database sia aumentato esattamente del numero di articoli presenti nel carrello.
- **Sintassi OCL:** @pre confronta lo stato del database prima e dopo la transazione atomica; size() conta gli elementi processati.

Metodo rimuoviVeicolo(telaio: String, username: String): Boolean

- **Pre-condizione (OCL):** context SalesLogic::rimuoviVeicolo(t, u) pre: t <> null
- **Post-condizione (OCL):** post: session['carrello']->excludes(t) and (u <> null implies Wishlist_Interesse.allInstances()->select(w | w.telaio = t and w.username = u)->isEmpty())
- **Significato:** Il sistema garantisce la rimozione del veicolo sia dalla memoria temporanea che dalla persistenza su database per l'utente specifico.

16.2 Componenti del Presentation Tier (Package sales)

I componenti descritti in questa sezione costituiscono l'interfaccia utente per la gestione del ciclo di vendita e del carrello. Pur essendo situati nel package `it.unisa.easydrive.sales`, essi operano esclusivamente come **Controller di interfaccia** all'interno del Presentation Tier.

Assenza di Interfaccia Pubblica Formale Come stabilito per l'architettura Three-Tier del sistema, questi script non implementano classi o metodi richiamabili esternamente. La loro natura è puramente procedurale e delegativa: la correttezza delle operazioni dipende dal rigido rispetto dei contratti OCL definiti nel Logic Tier (`SalesLogic`). Di conseguenza, per questi file non viene fornita una specifica formale OCL.

- **aggiungi_al_carrello.php:** Agisce come ricevitore di eventi dal catalogo. Recupera il *telaio* tramite metodo GET e invoca `SalesLogic::aggiungiAlCarrello()`. Gestisce la logica di feedback all'utente, impostando i messaggi di successo o errore in sessione prima di reindirizzare alla visualizzazione del carrello.
- **carrello.php:** Rappresenta la vista principale del carrello utente. Prima del rendering, invoca `SalesLogic::sincronizzaEPulisciCarrello()` per garantire che i veicoli mostrati siano ancora disponibili. Successivamente recupera i dati arricchiti tramite `SalesLogic::getDettagliCompletiCarrello()` per mostrare foto, prezzi e totale calcolato.
- **rimuovi_dal_carrello.php:** Orchestratore per l'eliminazione dei prodotti. Delega la rimozione del veicolo (sia dalla sessione che dal database, se l'utente è loggato) al metodo `SalesLogic::rimuoviVeicolo()`, gestendo poi il riposizionamento dell'utente sulla vista aggiornata.
- **conferma_ordine.php:** Script critico che chiude il workflow di vendita. Viene invocato al termine della fase di pagamento e delega l'intera transazione atomica (creazione ordine e cambio stato veicolo) a `SalesLogic::finalizzaAcquisto()`. In base all'esito, determina il reindirizzamento verso la Home Page o verso la gestione degli errori.

- **storico_operazioni.php**: Gestisce la visualizzazione post-vendita. Recupera l'elenco delle transazioni concluse per l'utente autenticato interpellando SalesLogic::ottieniStoricoUtente(), permettendo la consultazione dei codici ordine e dei dettagli di acquisto o noleggio.

Glossario

Interfaccia di Classe Gli attributi e le operazioni pubbliche di una classe.

Package Raggruppamento logico di classi correlate.

Sottosistema Insieme di package che forniscono un set coerente di servizi.