



Laurea Triennale in Informatica

Università di Salerno

Corso di Ingegneria del Software

Prof. Andrea De Lucia



Progetto EasyDrive

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 SDD	Claudio Brizio
09/12/2025	0.2	Seconda stesura del SDD	Claudio Brizio

20 dicembre 2025

Indice

1	Scopo	3
2	Destinatari	3
3	Introduzione	3
3.1	Object design trade-offs	3
3.2	Linee Guida per la Documentazione delle Interfacce	4
3.3	Definizioni, Acronimi e Abbreviazioni	4
3.4	Riferimenti	5
4	Package	5
5	Interfacce delle Classi	5

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 descrive i principali compromessi adottati dagli sviluppatori durante la progettazione a oggetti, ad esempio:

- **Funzionalità vs. Usabilità (Functionality vs. Usability):** L'usabilità è un obiettivo fondamentale del design. L'inserimento di troppe funzionalità può rendere l'interfaccia utente complessa o difficile da apprendere, riducendo la facilità d'uso del sistema.
- **Affidabilità vs. Costo di Sviluppo (Reliability vs. Development Cost):** La robustezza del sistema garantisce che il software funzioni correttamente anche in caso di errori o input imprevisti. Raggiungere un livello di affidabilità molto elevato richiede però investimenti significativi in termini di tempo, personale 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 funzionalità direttamente colpite dal malfunzionamento. Questo approccio serve a prevenire la propagazione del guasto all'intero sistema, garantendo che la maggior parte dei servizi rimanga operativa (disponibilità) a discapito della temporanea sospensione della componente 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, Acronimi 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

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 sei package funzionali principali che rispecchiano i sottosistemi individuati nel SDD.

2.1 Panoramica dei Package

it.unisa.easydrive.catalog: Gestisce la logica di presentazione e ricerca dei veicoli per l'utente finale.

it.unisa.easydrive.user: Include i componenti per l'identificazione, la gestione del profilo.

it.unisa.easydrive.booking: Implementa la logica di business relativa alla ricerca di disponibilità e alla gestione dei noleggi.

it.unisa.easydrive.sales: Gestisce il flusso d'acquisto, gli ordini e lo storico delle transazioni.

it.unisa.easydrive.payment: Interfaccia il sistema con il database per la gestione della validazione e dell'esecuzione dei pagamenti.

it.unisa.easydrive.staff: Fornisce le funzionalità CRUD avanzate per la manutenzione del catalogo da parte dei vari gestori del sistema.

it.unisa.easydrive.core: Package trasversale che contiene le classi di base (es. connessione DB) e le entità condivise (es. Veicolo, Utente).

5 Interfacce delle Classi

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.