ODD Object Design Document

Management Penta Motor

ODD 2.0 – ultima modifica 14/12/2016

Destinatario del documento:
Top Manager – Prof.ssa F. Ferrucci

Presentato da:

Gianmarco Bassano	Ernesto Pesce	Angela Cesarano
Andrea Russo	Carmelo Sottile	Roberta Gesumaria

Approvato da:

Ivan Rizzo

Storia dei cambiamenti

Data	Versione	Cambiamenti	Autori
09/12/2016	0.1	Struttura del documento e inserimento delle sezioni relative all'Introduzione.	Tutti
13/12/2016	0.2	Inserimento delle sezioni relative ai Packages e alle Interfacce delle classi.	Tutti
14/12/2016	0.3	Inserimento della sezione relativa ai Design Pattern.	Tutti
15/12/2016	1.0	Revisione	Carmelo Sottile
Fase manutenzione			
20/02/2020	2.0	Aggiornamento dei package e delle interfacce delle classi in relazione alla nuova funzionalità aggiunta.	Roberta Gesumaria, Francesca Tassatone

Sommario

1. Introduzione

1.1 Trade-offs

- *Comprensibilità vs costi*

  La comprensibilità del codice è un aspetto molto importante, soprattutto per la fase di testing. Ogni classe e metodo deve essere facilmente interpretabile anche dalle persone non coinvolte nel progetto o dalle persone coinvolte che non hanno lavorato a quella parte in particolare. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorare la comprensibilità agevola il processo di modifica. Ovviamente questa caratteristica aggiungerà dei costi allo sviluppo del nostro progetto.

- *Prestazioni vs Costi*

  Non avendo a disposizione alcun budget, utilizziamo materiale open source per la realizzazione del software MPM con lo scopo di rendere il sistema efficiente.

- *Costi vs Mantenimento*

  L'utilizzo di materiale open source rende il sistema facilmente modificabile e manutenibile, per questo i costi di manutenzione non saranno molto alti.

- *Interfaccia vs Easy-use*

  L'interfaccia si presenta semplice ed intuitiva, permettendo una facile gestione del database (Easy-Use).

- *Memoria vs efficienza*

  Si ha una leggera perdita di velocità iniziale per recuperare le informazioni che andrà poi a favore dell'efficienza nell'utilizzo del sistema al fine di rendere subito disponibili i dati. Quindi preferiamo le prestazioni invece della memoria.

- *Sicurezza vs efficienza*

  Nel nostro sistema ogni richiesta del client viene validata controllando la validità di username e password. Questa caratteristica potrebbe far aumentare il tempo di risposta del sistema soprattutto in situazioni di carico elevato, ma è necessario effettuare tali controlli per rispettare i requisiti iniziali del sistema.

- *Tempo di risposta vs Spazio di memoria*

  La scelta di utilizzare un database relazionale è scaturita dai diversi vantaggi che ne derivano:
  - gestione consistente dei dati;
  - tempo di risposta basso rispetto all'utilizzo del file system;
  - accesso veloce e concorrente ai dati;

  Lo svantaggio nell'utilizzo di un database relazionale è che richiede il triplo dello spazio di memoria rispetto ad un file system.

1.2 Componenti off-the-shelf

Per il progetto software che andiamo a realizzare facciamo uso di componenti off-the-shelf, che sono componenti software disponibili sul mercato per facilitare la creazione del progetto. Per il sistema che si vuole realizzare, siamo interessati ad un framework per applicazioni web. Il framework che andremo ad utilizzare è Bootstrap, un framework open source che contiene una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Questo framework contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, bottoni e navigazione, e altri componenti dell'interfaccia, così come alcune estensioni opzionali di JavaScript. Bootstrap è compatibile con le ultime versioni di tutti i principali browser. Dalla versione 2.0 supporta anche il responsive web design. Ciò significa che il layout delle pagine web si regola dinamicamente, tenendo conto delle caratteristiche del dispositivo utilizzato, sia esso desktop, tablet o telefono cellulare.

1.3 Linee guida per la documentazione dell'interfaccia

Prima dell'implementazione della logica è opportuno sottometterla al Project Manager, l'algoritmo che si intende seguire per l'implementazione, in modo che eventuali correzioni della logica da seguire possano essere apportate prima di imbattersi nella sintassi degli strumenti scelti.
Ogni metodo e ogni file devono essere preceduti da un commento, o più precisamente da una documentazione che riporti l'obiettivo che si vuole e deve raggiungere con il nome/i dell'autore/i. Inoltre bisogna commentare, giustificare delle decisioni particolari o dei calcoli.
La convenzione che deve essere adottata da tutti i team member per quanto riguarda i nomi delle variabili è inserire un underscore per separare più parole che compongono un unico nome.

Organizzazione dei file

Ogni file deve essere:

- Sviluppato e diviso in base alla categoria di appartenenza, ovvero deve essere correlato ad un'unica funzionalità che persegue. Ogni pagina di MPM (login, visualizza veicoli, statistiche, etc.) deve essere implementata in file separati;
- Diviso in più file, se raggiunge una lunghezza tale da divenire difficile da leggere e comprendere.

Se sono presenti diversi script per le singole funzionalità è opportuno organizzarli in una apposita cartella. La convenzione per quanto riguarda i nomi dei file, delle operazioni e delle variabili è quella di avere nomi evocativi, ma soprattutto in lingua italiana. Le uniche eccezioni sono legate al linguaggio di programmazione utilizzato. Organizzare in una cartella i file delle librerie usate e le altre risorse scaricate necessarie per lo sviluppo del progetto.

Spostamento di linee

Quando un'espressione supera la lunghezza della linea, occorre spezzarla secondo i seguenti principi generali:

- Interrompere la linea dopo una virgola;
- Interrompere la linea prima di un operatore;

- Preferire interruzioni di alto livello rispetto ad interruzioni di basso livello (interrompere laddove non si interrompe un discorso logico, discorso valido soprattutto per le formule es. (3+4) * 2 interrompere prima della moltiplicazione senza spezzare gli operandi in parentesi);
- Allineare la nuova linea con l'inizio dell'espressione nella linea precedente;
- Se le regole precedenti rendono il codice più confuso o il codice è troppo spostato verso il margine destro, utilizzare solo otto spazi di indentazione.

Indentazione

L'indentazione deve essere effettuata con un TAB e qualunque sia il linguaggio usato per la produzione di codice, ogni istruzione deve essere opportunamente indentata.

Es.

```
<html>
<head>
</head>
<body>
</body>
</html>
```

Deve essere sostituita da:

```
<html>
    <head>
    </head>
    <body>
    </body>
</html>
```

Questa pratica deve essere usata soprattutto per le istruzioni FOR, IF.

È buona pratica scendere di livello.

Inizializzazione

Inizializzare le variabili locali nel punto in cui sono state dichiarate a meno che il loro valore iniziale non dipenda da un calcolo che occorre eseguire prima.

Posizione

Mettere le dichiarazioni all'inizio dei blocchi. Non aspettare di dichiarare le variabili al loro primo uso: può confondere il programmatore inesperto e impedire la portabilità del codice dentro lo scope. L'unica eccezione a questa regola sono gli indici dei cicli for che in Java possono essere dichiarati nell' istruzione stessa. Evitare dichiarazioni locali che nascondono dichiarazioni a più alto livello. Ad esempio, non dichiarare una variabile con lo stesso nome in un blocco interno.

Parentesi

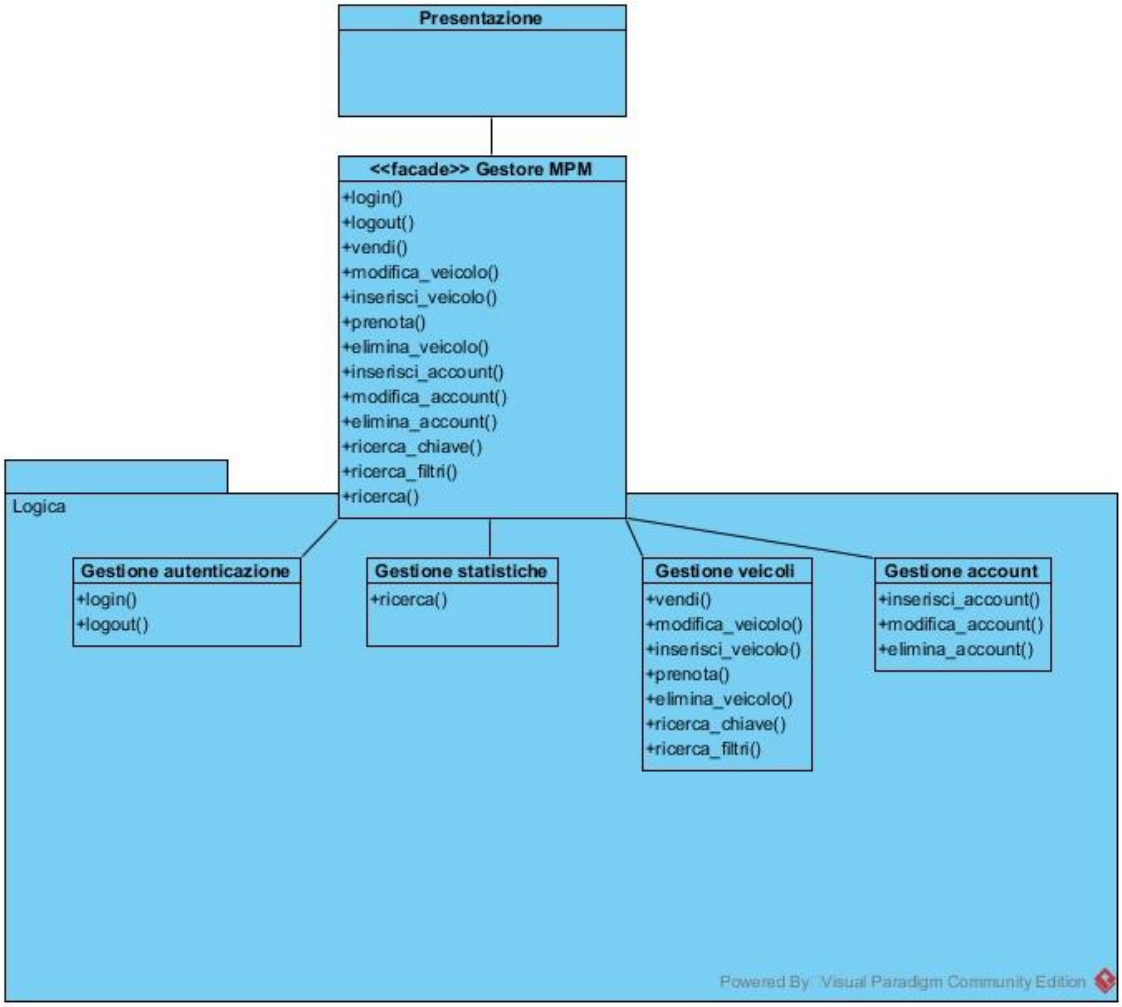
A prescindere dalle istruzioni che seguono un IF, è necessario, laddove ci fosse anche una sola istruzione, riportare il blocco di istruzioni tra parentesi graffe.

Ogni tag di apertura deve essere necessariamente seguito dall'apposito tag di chiusura (eccetto i tag self-closing).

Una convenzione importante, per quanto riguarda l'inserimento di numeri o di valori costanti, è quella di non usare una codifica fissa (hard coding) che è fortemente sconsigliata ma di associare sempre il valore ad una variabile o semplicemente definire una macro che può essere richiamata da eventi ed essere parametrizzata. In questo modo si facilita la modifica, sostituendo solo il valore della variabile o macro, in un unico posto.

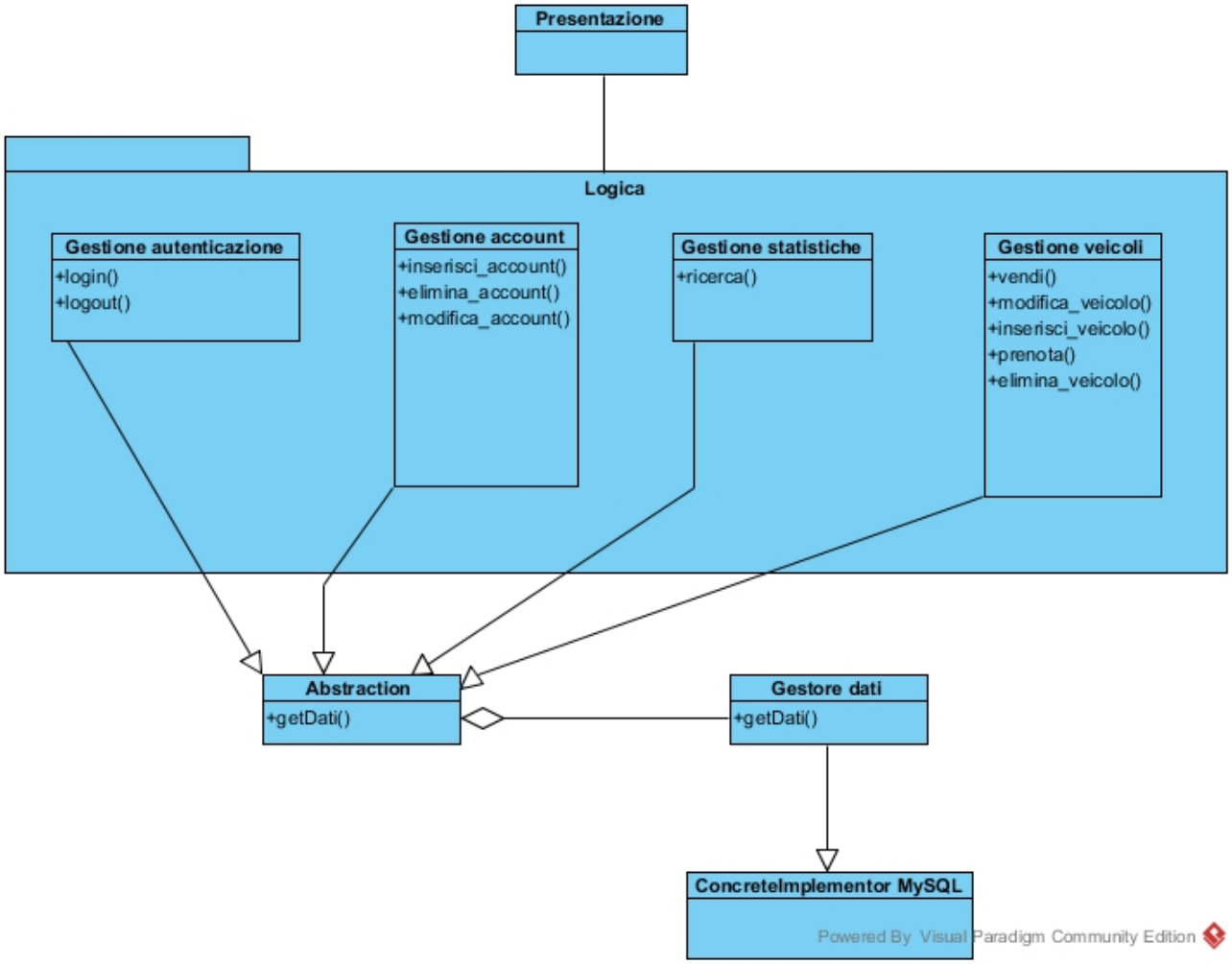
1.4 Design Pattern

Façade Pattern



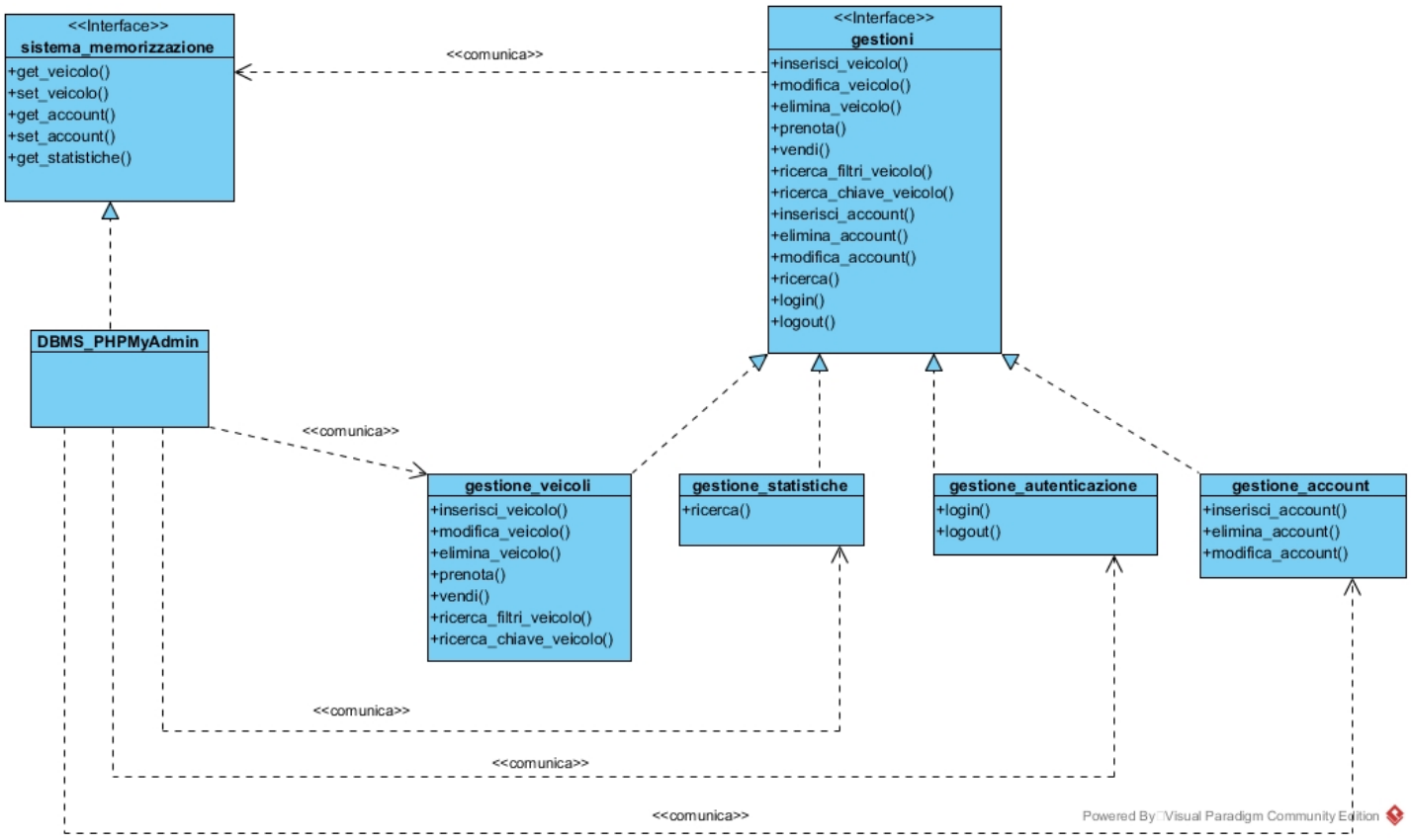
MPM fa uso del Façade Pattern per definire un'unica interfaccia a livello di logica che permette all'utente di interagire, attraverso l'interfaccia, con le funzionalità del sistema vedendole come un unico sistema.

Bridge Pattern



MPM fa uso del Bridge Pattern perché abbiamo bisogno di utilizzare un'unica interfaccia per diversi accessi allo storage: offrendo un'unica interfaccia si garantisce che l'eventuale cambio di implementazione del database usato comporta la modifica solo a una componente e non a svariate componenti del sistema.

Mediator Pattern



MPM fa uso del Mediator Pattern nell'intento di disaccoppiare entità del sistema che devono comunicare fra loro. Il pattern infatti fa in modo che queste entità non si riferiscano reciprocamente ma che comunicano tra loro tramite il "mediatore" che nel nostro caso è il sistema di memorizzazione.

Il beneficio principale è il permettere di modificare agilmente le politiche di interazione, poiché le entità coinvolte devono fare riferimento al loro interno solamente al mediatore, permettendo facilmente un eventuale cambiamento del database.

1.5 Definizioni, acronimi e abbreviazioni

Definizioni

Parametri veicolo: rappresentano le informazioni relative ad ogni veicolo:

1. **Marca**;
2. **Modello**;
3. **Categoria** (berlina, cabrio, city car, coupé, furgone, monovolume, station wagon, SUV);
4. **Numero telaio**;
5. **Colore**;
6. **Cilindrata**;
7. **Potenza** (Cv);
8. **Alimentazione**;
9. **Anno immatricolazione**;
10. **Numero porte**;

11. **Allestimento** (base, economy, full optional);
12. **Km**;
13. **Cambio** (automatico, manuale, semi-automatico);
14. **Classe emissione** (Euro 1-6);
15. **Usata** (si/no);
16. **Prezzo**;
17. **Stato:** rappresenta un'informazione associata ad un veicolo relativa alla sua situazione di vendita. Può essere:
    - 17.1 **Disponibile**: nel caso in cui ci sia la possibilità dell'immediata fruizione del veicolo.
    - 17.2 **In arrivo**: nel caso in cui il veicolo sia già stato ordinato dal gestore e siamo solo in attesa del suo arrivo nel parco auto per poter essere venduta.
    - 17.3 **Venduto**: nel caso in cui il veicolo sia già stato acquistato da un cliente.
    - 17.4 **In lavorazione**: nel caso in cui il veicolo sia stato prenotato da un cliente e siamo solo in attesa del pagamento.
18. **Dipendente**: indica il dipendente che ha venduto il veicolo;
19. **Data vendita**.

Dipendente: rappresenta un qualsiasi addetto alle vendite, ognuno avente un proprio account in cui saranno specificate le seguenti informazioni:

1. **Codice fiscale**;
2. **Nome**;
3. **Cognome**;
4. **Data di nascita**;
5. **Indirizzo**;
6. **Email**;
7. **Numero di telefono**;
8. **Username** (generato automaticamente dal sistema con la seguente notazione: nome_cognome_dipx) con x=1,2,3…;
9. **Password**;
10. **Ruolo**: indica il ruolo del dipendente per distinguerlo dal gestore.

Gestore: rappresenta una specializzazione del concetto di dipendente con le funzionalità aggiuntive di poter visualizzare le statistiche periodiche, le statistiche dei vari dipendenti, inserire, modificare e eliminare un veicolo. Ha inoltre funzioni di gestione degli account.

Cliente: rappresenta un generico acquirente di un veicolo. Al momento dell'acquisto dovranno essere specificati i seguenti parametri:

1. **Codice fiscale**;
2. **Nome**;
3. **Cognome**;
4. **Data di nascita**;
5. **Comune di nascita**;
6. **Indirizzo**;
7. **Email**;
8. **Numero di telefono**.

Acronimi

MPM = Management Penta Motor.

RAD = Requirement Analysis Document.

SDD = System Design Document.

ODD = Object Design Document.

MVC = Model View Controller.

HTML = HyperText Markup Language.

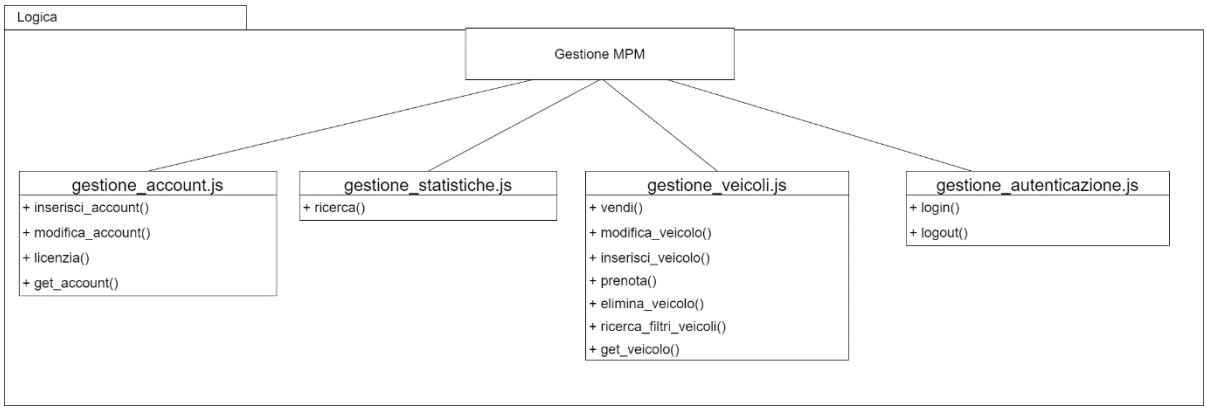
CSS = Cascading Style Sheets.

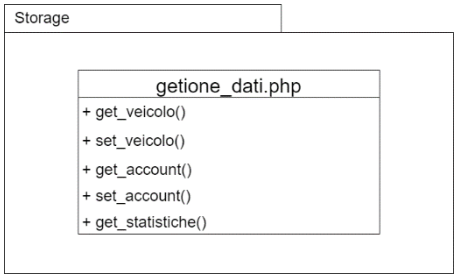
1.6 Riferimenti

- http://www.pentamotor.it
- Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, (2nd edition), Prentice-Hall, 2004.
- Ian Sommerville, Ingegneria del software, (8ª edizione), Pearson, 2007.
- MPM_RAD_Vers3.0.
- MPM_SDD_Vers2.0.

2. Packages

Ogni elemento di un package (Presentazione, Logica, Storage) è collegato con il package successivo, per una maggiore chiarezza dello schema non sono stati riportati i collegamenti.





3. Interfacce delle classi

Nome classe	Gestione_account
Descrizione	Questa classe rappresenta la gestione delle funzionalità relative agli account (dipendenti) quale inserimento, modifica/assunzione e licenziamento.
Pre-condizione	**context** GestioneAccount:: inserisci_account(Codice fiscale, Nome, Cognome, Data nascita, Indirizzo, email, Numero telefono, Username, Password, Ruolo, data_inizio_contratto, data_fine_contratto, tipo_contratto, retribuzione, allegati); **pre** Codice fiscale!=null && Nome!=null && Cognome!=null && Username!=null && Password!=null && data_fine_contratto !=null && tipo_contratto!=null && retribuzione!=null && allegati.length != 0 && Ruolo==gestore **context** GestioneAccount:: modifica_account(Vecchio codice fiscale, Nuovo codice fiscale, Nome, Cognome, Data nascita, Indirizzo, email, Numero telefono, Username, Password, Ruolo); **pre** Vecchio codice fiscale!=null && Nuovo codice fiscale!=null && Nome!=null && Cognome!=null && Username!=null && Password!=null && data_fine_contratto !=null && tipo_contratto!=null && retribuzione!=null && Ruolo==gestore **context** GestioneAccount:: licenzia(id_contratto, motivazione, allegati); **pre** id_contratto!=null && motivazione!=null && allegati.length!=null **context** Gestione_dati:: Dati get_account(); **pre** –
Post-codizione	**context** GestioneAccount:: Dati get_account(); **post** Dati!=null
Invarianti	

Nome classe	Gestione_veicoli
Descrizione	Questa classe rappresenta la gestione dei veicoli e consente le seguenti funzionalità: vendita, modifica, inserimento, prenotazione, eliminazione e ricerca di un veicolo.
Pre-condizione	**context** Gestione_veicoli:: vendi(Numero telaio, Username, Codice fiscale, Nome, Cognome, Data nascita, Comune nascita, Indirizzo, Email, Numero telefono); **pre** Numero telaio!=null && Username!=null && Codice fiscale!=null && Nome!=null && Cognome!=null && Numero telefono!=null **context** Gestione_veicoli:: inserisci_veicolo(Numero telaio, Marca, Modello, Categoria, Colore, Cilindrata, Potenza, Alimentazione, Anno immatricolazione, Numero porte, Allestimento, Km, Cambio, Classe emissione, Usata, Prezzo, Stato, Data vendita); **pre** Numero telaio!=null && Marca!=null && Modello!=null && Categoria!=null && Colore!=null && Cilindrata!=null && Potenza!=null && Alimentazione!=null && Anno immatricolazione!=null && Numero porte!=null && Allestimento!=null && Km!=null && Cambio!=null && Classe emissione!=null && Usata!=null && Prezzo!=null && Stato!=null **context** Gestione_veicoli:: modifica_veicolo(Vecchio numero telaio, Nuovo numero telaio, Marca, Modello, Categoria, Colore, Cilindrata, Potenza, Alimentazione, Anno immatricolazione, Numero porte, Allestimento, Km, Cambio, Classe emissione, Usata, Prezzo, Stato, Data vendita); **pre** Vecchio numero telaio!=null && Nuovo numero telaio!=null && Marca!=null && Modello!=null && Categoria!=null && Colore!=null && Cilindrata!=null && Potenza!=null && Alimentazione!=null && Anno immatricolazione!=null && Numero porte!=null && Allestimento!=null && Km!=null && Cambio!=null && Classe emissione!=null && Usata!=null && Prezzo!=null && Stato!=null && Data vendita!=null **context** Gestione_veicoli:: prenota(Numero telaio, Username, Codice fiscale, Nome, Cognome, Data nascita, Comune nascita, Indirizzo, Email, Numero telefono); **pre** Numero telaio=!null && Username!=null && Codice fiscale=!null && Nome=!null && Cognome=!null && Numero telefono=!null **context** Gestione_veicoli:: elimina_veicolo(numeroTelaio); **pre** numeroTelaio!=null **context** Ricerca:: Dati ricerca_filtri(Numero telaio, Marca, Modello, Categoria, Colore, Cilindrata, Potenza, Alimentazione, Anno immatricolazione, Numero porte, Allestimento, Km,

	Cambio, Classe emissione, Usata, Prezzo, Stato, Data vendita, ClienteCodice_fiscale, DipendenteCodice_fiscale); **pre** Numero telaio!=null || Marca!=null || Modello!=null || Categoria!=null || Colore!=null || Cilindrata!=null || Potenza!=null || Alimentazione!=null || Anno immatricolazione!=null || Numero porte!=null || Allestimento!=null || Km!=null || Cambio!=null || Classe emissione!=null || Usata!=null || Prezzo!=null || Stato!=null || Data vendita!=null || ClienteCodice_fiscale!=null || DipendenteCodice_fiscale!=null; **context** Ricerca:: Dati ricerca_chiave(key); **pre** key!=null **context** Gestione_dati:: Dati get_veicolo(stato); **pre** stato==in_arrivo || stato==disponibile || stato==in_lavorazione || stato==venduto
Post-codizione	**context** Ricerca:: Dati ricerca_filtri(marca, modello, categoria, numeroTelaio, colore, cilindrata, potenza, alimentazione, annoImmatricolazione, numeroPorte, allestimento, km, cambio, classeEmissione, usata, prezzo); **post** dati!=null || dati==null **context** Ricerca:: Dati ricerca_chiave(key); **post** Dati!=null || Dati==null **context** Gestione_dati:: Dati get_veicolo(stato); **post** Dati!=null || Dati==null
Invarianti	

Nome classe	Statistiche
Descrizione	Questa classe rappresenta la gestione delle statistiche.
Pre-condizione	**context** Statistiche:: Dati ricerca(data1, data2); **pre** data1!=null && data2!=null
Post-codizione	**context** Statistiche:: Dati ricerca(data1, data2); **post** Dati!=null || Dati==null
Invarianti	

Nome classe	Autenticazione
Descrizione	Questa classe rappresenta la gestione dell'autenticazione.
Pre-condizione	**context** Autenticazione:: Dati login(username, password); **pre** username!=null && password!=null **context** Autenticazione:: logout(); **pre** -
Post-codizione	**context** Autenticazione:: Dati login(username, password); **post** Dati!=null || Dati==null
Invarianti	

Nome classe	Gestione_dati
Descrizione	Questa classe gestisce l'interfaccia tra database e lo strato di logica; essa si occupa in particolare di trasformare delle richieste dell'utente in query da inviare al DBMS.
Pre-condizione	**context** Gestione_dati:: Dati get_account(); **pre** - **context** Gestione_dati:: set_account(codice fiscale, nome, cognome, data nascita, comune nascita, indirizzo, email, numero telefono, username, password, ruolo, data_inizio_contratto, data_fine_contratto, tipo_contratto, retribuzione, allegati); **pre** codice fiscale!=null && nome!=null && cognome!=null && data nascita!=null && comune nascita!=null && indirizzo!=null && email!=null && numero telefono!=null && username!=null && password!=null && data_fine_contratto !=null && tipo_contratto!=null && retribuzione!=null && allegati.length != 0 && Ruolo==gestore **context** Gestione_dati:: Dati get_statistiche(data1, data2); **pre** data1!=null && data2!=null **context** Gestione_dati:: Dati get_veicolo(stato); **pre** stato==in_arrivo || stato==disponibile || stato==in_lavorazione || stato==venduto **context** Gestione_dati:: set_veicolo(Numero telaio, Marca, Modello, Categoria, Colore, Cilindrata, Potenza, Alimentazione, Anno immatricolazione, Numero porte, Allestimento, Km, Cambio, Classe emissione, Usata, Prezzo, Stato, Data vendita);

	pre Numero telaio!=null && Marca!=null && Modello!=null && Categoria!=null && Colore!=null && Cilindrata!=null && Potenza!=null && Alimentazione!=null && Anno immatricolazione!=null && Numero porte!=null && Allestimento!=null && Km!=null && Cambio!=null && Classe emissione!=null && Usata!=null && Prezzo!=null && Stato!=null && Data vendita!=null
Post-codizione	**context** Gestione_dati:: Dati get_account(); **post** Dati!=null || Dati==null **context** Gestione_dati:: Dati get_statistiche(data1, data2); **post** Dati!=null || Dati==null **context** Gestione_dati:: Dati get_veicolo(stato); **post** Dati!=null || Dati==null
Invarianti	