EcoToll

Progetto OOSD 2018/2019 realizzato tramite l'utilizzo di Eclipse e di MySql da:

Rea Gianluca ~ Progettazione ed implementazione

Fossemò Daniele ~ Progettazione ed implementazione

Ricci Davide ~ Progettazione ed implementazione

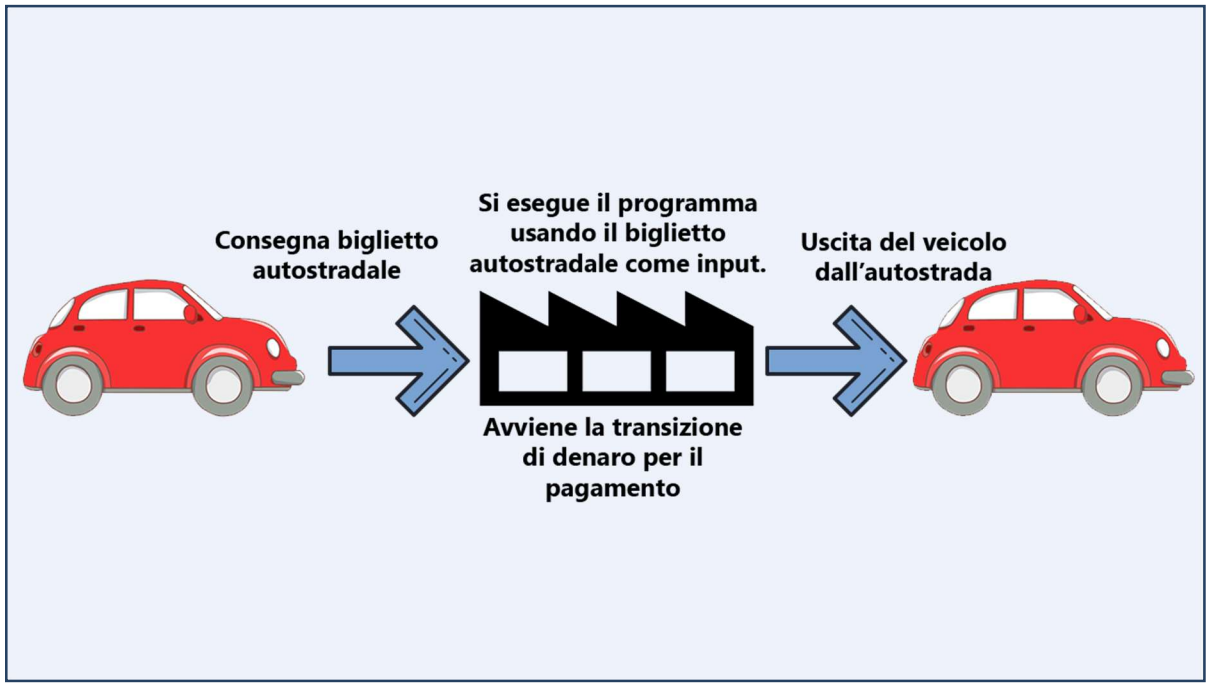
Link repository: https://github.com/GianlucaRea/EcoToll

INDICE

INTRODUZIONE

Il progetto EcoToll ha come obiettivo quello di fornire un'applicazione per il calcolo del pedaggio autostradale in base a differenti variabili tra cui l'iva, la regolamentazione UE sull'emissioni, la tipologia di veicolo ed ovviamente il percorso; oltre che alla gestione dei dati relativi alle infrastrutture autostradali e normativa vigente.
Tale applicazione è stata realizzata tenendo conto delle varie entità (**attori**) che la utilizzeranno, dividendole in due tipologie, l'amministratore, che gestirà i dati relativi all'autostrada, ed il casello, che invece si occuperà del calcolo del pedaggio e del pagamento del biglietto (utente).
Il programma è pensato, come da consegna, per funzionare ricevendo da input un biglietto (dalla quale si otterranno i dati relativi al percorso) che è stato consegnato al guidatore al casello di partenza; il programma dunque restituirà il pedaggio da pagare in conformità con la normativa vigente.
Nel capitolo a seguire verrà spiegata la progettazione concettuale del software, quindi l'**application flow**, l'**elenco dei requisiti** (funzionali e non funzionale) e l'**use case diagram**.

1.1 APPLICATION FLOW



L'utente finale dell'autostrada, una volta arrivato al casello di arrivo, consegnerà il biglietto che ha ricevuto al casello di entrata all'operatore o alla macchina (in base alla corsia di pagamento in cui entra, se con l'operatore o se con la macchinetta) che scannerizzerà il biglietto fornendo in input i dati al programma e calcolando quindi il pedaggio.
Assumiamo che il sistema sarà sincronizzato con un servizio esterno che permette di reperire i dati del veicolo che paga il pedaggio una volta che questi consegnerà il biglietto. Per esempio, assumiamo che nel casello vi è presente una telecamera che, riconosciuta la targa, manda quest'ultima ad un servizio esterno che si occuperà di restituire al casello tutte le informazioni relative al veicolo associato a quella targa.
Simuleremo questa funzionalità attraverso il database.
Una volta calcolato il pedaggio avverrà la transizione di denaro, alla fine della quale verrà alzata la sbarra del casello e il guidatore potrà uscire dall'autostrada.

1.2 SPECIFICA DEI REQUISITI

I requisiti di base comprendono la facilità di utilizzo con cui utenti e amministratori possono usufruire del software tramite l'interfaccia grafica, tenendo conto sia ad un'implementazione efficiente (tramite l'utilizzo di interfacce e di patterns) che alla realizzazione di un programma semplice ed intuitivo da utilizzare per gli utenti finali (tramite un'interfaccia grafica).

Requisiti Funzionali

- Dal punto di vista dell'Utente, l'applicazione dovrà saper calcolare il pedaggio dello stesso tramite il percorso dal casello di entrata a quello di uscita, il tipo di vettura e la normativa vigente.

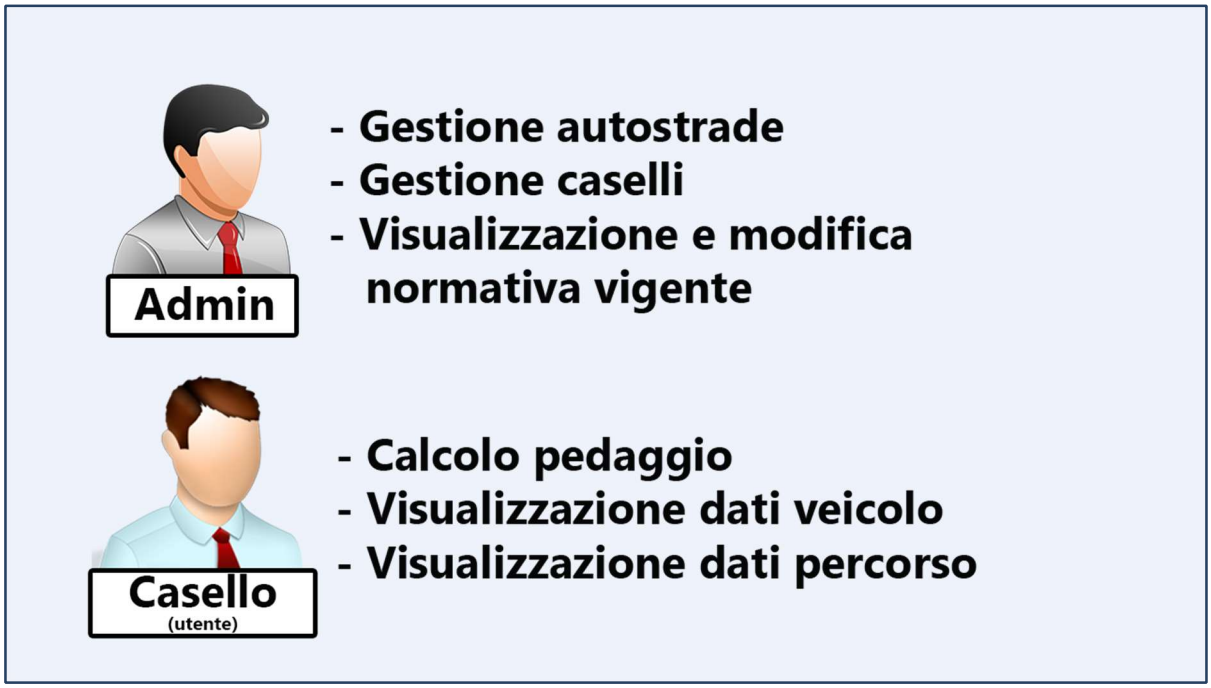
- Dal punto di vista dell'Amministratore del Sistema , l'applicazione dovrà garantire ad esso la possibilità di interfacciarsi con le risorse del software (il database) e di poter gestire gli elementi al suo interno (Autostrade, caselli, ecc.) oltre che la possibilità di aggiornare la normativa.

Requisiti Non Funzionali

Il sistema deve essere scalabile (cioè permettere di calcolare il pedaggio anche se la normativa vigente cambia), sicuro, affidabile ed implementato tramite Java e MySql (per la realizzazione di un database da collegare con il programma). Deve essere inoltre supportato da un'interfaccia grafica che ne permette un utilizzo facilitato per gli utenti del sistema, nascondendo a questi ultimi l'effettivo modo in cui il programma funziona.

1.3 USE CASE DIAGRAM

In questo paragrafo vedremo il diagramma descrivente le funzioni utilizzate dagli attori (che abbiamo riconosciuto, come precedentemente detto, nell'amministratore ed il casello). Di seguito un'immagine rappresentativa e la descrizione delle operazioni.



ADMIN

Gestione autostrade: Insieme di operazioni che si identificano nell'inserimento (tramite nome e tariffa, i caselli verranno aggiunti in un secondo momento tramite inserisci casello nelle impostazioni casello), nell'eliminazione (tramite nome), nella modifica (del nome e della tariffa) e della visualizzazione delle autostrade.

Gestione caselli: Insieme di operazioni che si identificano nell'inserimento (tramite nome, altezza in km nell'autostrada e autostrada nella quale è contenuto), nell'eliminazione (tramite nome), nella modifica (del nome, della tariffa o dell'autostrada di appartenenza) e della visualizzazione dei caselli.

Visualizzazione e modifica della normativa vigente: Insieme di operazioni che si identificano nella possibilità di visualizzare la normativa vigente e di modificarla.

CASELLO (utente)

Calcolo pedaggio: Operazione che permette, dato in input un biglietto contenente le informazioni del percorso, ed i dati del veicolo (ottenuti tramite un sistema esterno) che ha effettuato il percorso, di calcolare il pedaggio in base ai dati di input e alla normativa vigente.

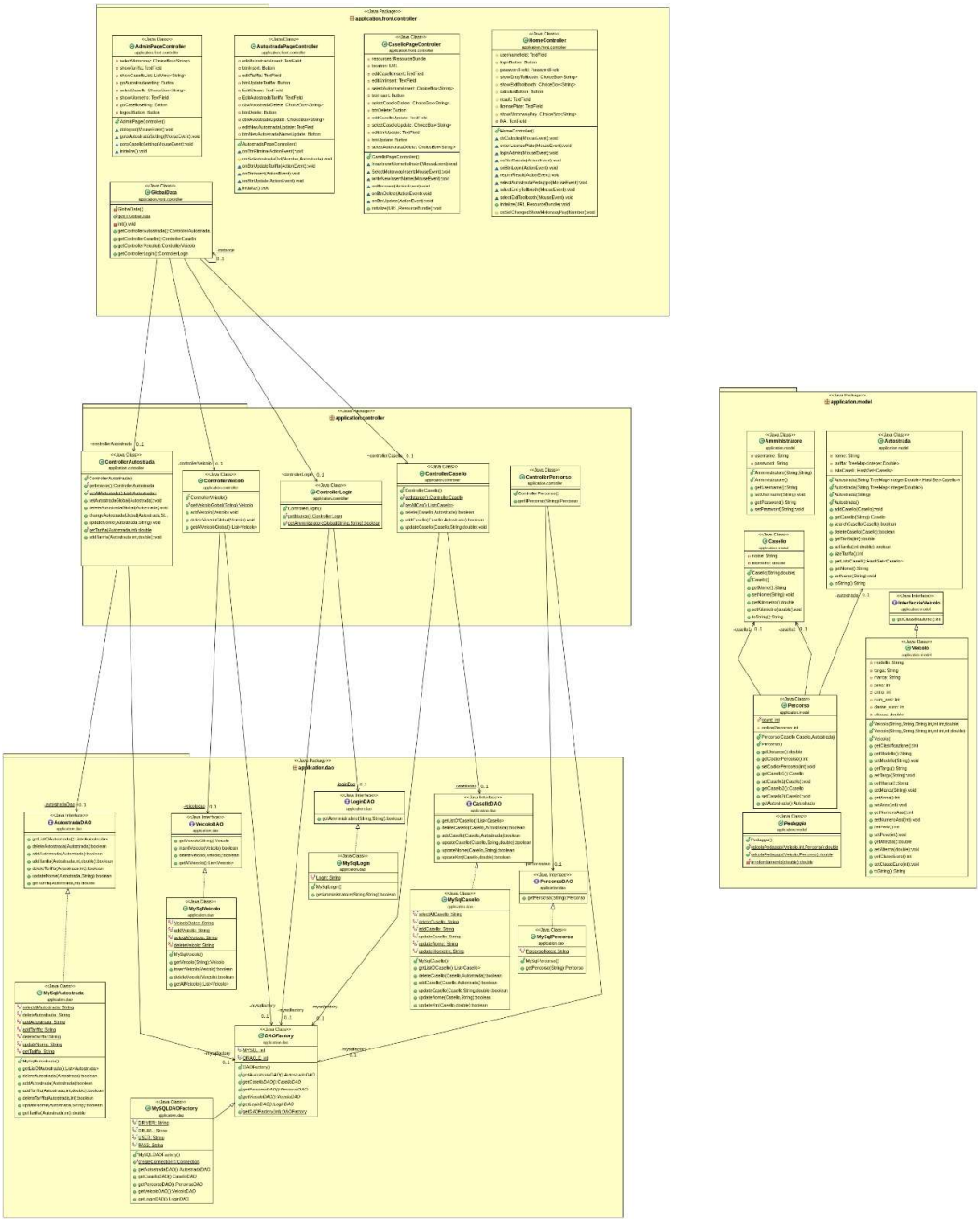
Visualizzazione dati veicolo: Operazione che permette al casello di visionare tutte le informazioni relative al veicolo a cui sta facendo pagare il pedaggio.

Visualizzazione dati Percorso: Operazione che permette al casello di visionare tutte le informazioni relative al percorso di cui sta facendo pagare il pedaggio.

ARCHITETTURA DEL SOFTWARE

Nel seguente capitolo verrà descritta l'architettura del software, quindi le scelte di design intraprese dal gruppo per realizzare questo progetto ed il diagramma dei pacchetti.

2.1 MODELLO DELL'ARCHITETTURA SOFTWARE



Per immagine ad alta risoluzione visionare nella directory l'immagine package diagram

2.2 DESCRIZIONE DELL'ARCHITETTURA SOFTWARE

L'architettura software è stata realizzato utilizzando il design pattern **MVC** (model, view, controller).
Nel **model** sono presenti le classi relative alle entità che abbiamo identificato nel progetto, insieme alle operazioni get e set, oltre che ad alcune aggiuntive (in base alla entità). Le entità presenti nel model verranno astratte dal metodo con il quale verranno connesse al database. Nel model abbiamo inserito le seguenti entità:
1- Autostrada
2- Casello
3- Amministratore
4- Veicolo
5- Percorso
6- Pedaggio

Per quanto riguarda la connessione al database abbiamo utilizzato il pattern del **DAO** (Data access Object).
Nel **controller** sono state implementate tutte le funzioni atte a svolgere compiti più complessi, quali funzioni di coordinamento col database (inserimento, eliminazione, get, etc.) oltre che a soddisfare i requisiti della **view**.
La **View** è stata pensata con delle viste che sono state realizzate in funzione all'uso che ne faranno l'utente finale (che dovrà semplicemente calcolare il pedaggio) e l'utente amministratore (che dovrà occuparsi della gestione delle autostrade, caselli, etc.). Di seguito la descrizione delle viste e delle loro composizioni:

Vista Generica

Home : Questa finestra è divisa in due pannelli , a sinistra vi è il pannello di login dell'utente amministratore mentre a destra vi è il pannello per il calcolo del pedaggio.

Viste Admin:

AdminPage: Questa è la finestra iniziale dopo il Login dell'Amministratore. È Divisa in 3 Pannelli e possiede un bottone per il Logout.

1- pannello relativo ai **caselli**, in cui è possibile visualizzare il kilometro di altezza di questi ultimi nell'autostrada di appartenenza, oltre che accedere alle loro impostazioni.
2- pannello relativo alle **autostrade** dove possiamo visualizzare la tariffa sull'Autostrada e la lista di Caselli che ne fanno parte oppure possiamo accedere alle impostazioni delle Autostrade.

AutostradaPage: Questa è la finestra per la gestione delle Autostrade e possiede un bottone per tornare indietro alla schermata iniziale dell'Amministratore. È divisa in 3 pannelli.

1- pannello per l'**inserimento** di una nuova autostrada con Nome
2- pannello per l'**inserimento** di una nuova tariffa ad una data autostrada.
3- pannello che, data un'Autostrada, ne gestisce l'**eliminazione**.
4- pannello che, data un'Autostrada, gestisce la **modifica** del nome.

CaselloPage: Questa è la finestra per la gestione delle Autostrade e possiede un bottone per tornare indietro alla schermata iniziale dell'Amministratore. È divisa in 3 pannelli.

1- pannello per l'**inserimento** di un nuovo casello con Nome, Altezza (km) e Autostrada di appartenenza
2- pannello che, dato un casello, ne gestisce l'**eliminazione**
3- pannello che, dato un casello, gestisce la **modifica** o del nome, dell'altezza (km) e dell'autostrada di appartenenza.

Le viste fanno uso di **file FXML** per la configurazione delle **interfacce** e dei **controller** di **front end** per la gestione degli eventi degli oggetti **JavaFX** della **GUI**.

2.3 DESCRIZIONE SCELTE E STRATEGIE

Il pattern MVC è stato realizzato per dare un'architettura solida ed efficace, capace di dividere i concetti tra di loro, permettendo tuttavia tra di questi una comunicazione (ad esempio tra il model ed il database mediante il DAO e il package controller).

Il DAO è stato scelto per connettere il programma al database in quanto risultava un metodo efficace per astrarre il modello da quest'ultimo, usando un'architettura (tramite la Factory) intuitiva e scalabile in base alle esigenze.

Nel progetto vi è anche l'utilizzo di **eccezioni** lì dove servono (ad esempio, nella classe Percorso, nel costruttore, se i due caselli del percorso non appartengono alla stessa autostrada

viene sollevata un'eccezione in quanto nella realtà dei fatti non sarebbe possibile uno scenario del genere).

Vi è stato anche l'utilizzo di **Collection**, come ad esempio di una HashSet<Casello> in autostrada che associa ad una determinata autostrada una lista di caselli. Abbiamo usato l'HashSet<Casello> per via della sua efficienza e dato che non serviva definire un ordine tra i caselli oltre che Treemap<Integer, Double> per le tariffe dove classe veicolo è la chiave e valore della tariffa è il "value" della treemap.

I concetti di **polimorfismo**, **overloading** ed **overriding** son presenti nel progetto:
Nella classe Pedaggio vi è un tipico esempio di **overloading**, infatti vi è la presenza di due metodi "calcolaPedaggio" nella quale però, in uno dei due viene passato come parametro anche l'Iva, nell'altro no (quindi il casello può decidere se usare l'iva di default del programma, cioè del 22%, o applicarne una diversa in concordanza con la normativa vigente).
Tra la classe Veicolo ed interfacciaVeicolo vi è di fatto un esempio di **overriding**, infatti la classe veicolo, implementando interfacciaVeicolo, fa overriding sul metodo "getClassificazione".

Per quanto riguarda il **dinamic binding** anch'esso è presente, infatti a tempo di run time, ad esempio, si sceglie il tipo di veicolo da costruire.

Infine, per quanto riguarda i **modificatori di classe**, essi sono stati utilizzati nella maniera che ritenevamo opportuna:
- public per tutte le classi/metodi che possono essere visibili anche dalle altre classi dello stesso pacchetto (o da quelle che importano il pacchetto in cui sono contenuti);
- private per le variabili delle classi, che saranno

visibili/modificabili solo grazie ai metodi getters/setters;
- abstract per pedaggio (oltre che essere usata nel DAO) che più che rappresenta un'entità del progetto, rappresenta una classe "per fare i calcoli" con la normativa vigente.
- static per le constants (come in DAO per le query).
- final per le constants (come in DAO per le query)

Sono stati inoltre utilizzati altri design patterns. Abbiamo infatti utilizzato anche il **Singleton** che garantisce la creazione di una sola istanza di una determinata classe, esso è stato implementato per restituire un'istanza del driver **JDBS** nella classe GlobalData Nella cartella Front>Controller.

PROGETTAZIONE CLASSI

In questo capitolo vedremo la progettazione delle classi, quindi la loro struttura (da un punto di vista più accurato rispetto a quello usato fino ad ora) e le scelte di design che ci hanno portato a strutturarle in questo modo.

3.1 DESCRZIONE CLASSI, INTERACCE, MEMBRI

La progettazione delle classi e delle interfacce ha richiesto come prima fase la creazione di un modello teorico su cui lavorare concentrandosi sulle principali entità, cioè Autostrada, Casello e Veicolo.

L'Autostrada è stata caratterizzata da un nome, un array di 5 possibili tariffe per essa, ed un HashSet di Caselli.

Il Casello è stato caratterizzato dal nome e dal kilometro di altezza in cui era situato sul tratto autostradale.

Il Veicolo è stato caratterizzato da vari attributi quali il modello, la marca, il peso, l'anno di immatricolazione, il numero degli assi, la classe euro, la sua altezza ed infine la targa che rende unico ogni veicolo.

Tutte le classi possiedono un costruttore ed i metodi get e set ma in particolare la classe Veicolo implementa anche un Interfaccia che definisce la firma del metodo per la classificazione del veicolo che viene implementata nella classe Veicolo in base al numero di assi ed altezza.

Nella seconda fase della costruzione delle classi si è passato alla creazione della classe Percorso e Pedaggio.

La classe Percorso è descritta da un codice, un casello di entrata ed uno di uscita ed infine l'autostrada su cui sono situati i caselli stessi.

La classe Pedaggio è la classe che fornisce solo metodi per il calcolo del pedaggio stesso perciò è implementata come una classe abstract. La classe definisce un metodo per il calcolo del pedaggio attuale, uno per il calcolo del pedaggio in caso di cambio della Normativa ed infine un semplice metodo per l'arrotondamento a 10 centesi.

Nell'ultima fase progettuale è stata introdotta la figura dell'Amministratore, cioè colui che è abilitato ad effettuare le modifiche (inserimento, cancellazione ecc.) delle prime 3 classi principali (Autostrada, Casello, Veicolo). Inoltre, è abilitato anche al cambiamento del sistema del calcolo del Pedaggio in caso del cambiamento della Normativa.

L'Amministratore del sistema è caratterizzato dal proprio username e dalla propria password.

3.2 DESCRIZIONE DEI DETTAGLI DI DESIGN SCELTI

Come già spiegato in precedenza, il progetto è stato realizzato utilizzando diversi pattern quale l'MVC per curare l'architettura del software. Nella fattispecie le classi descritte nel precedente paragrafo sono quelle relativa al modello. Infatti, quest'ultima, riporta la rappresentazione tramite oggetti della realtà che dovremo andare a trattare. Altro pattern utilizzato già nominato è il DAO, esso è ritornato molto utile perché permette al modello di colloquiare col database senza che abbia bisogno di sapere come esso stia colloquiando, né tantomeno dovendo implementare dei metodi per farlo all'interno di esso. Ciò ha

permesso di separare al meglio i concetti in concordanza con l'MVC durante la progettazione. Infatti, una volta realizzato il DAO, determinate modifiche del modello non hanno creato alcun problema col resto del progetto.

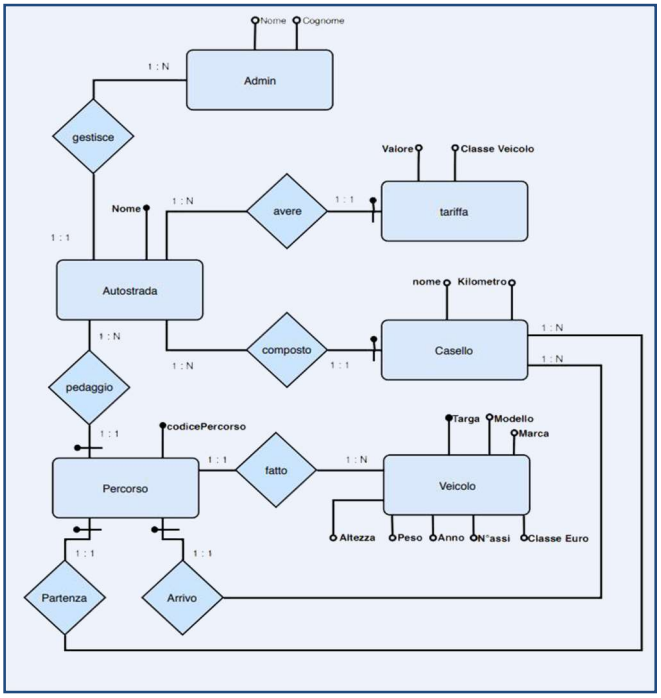
IMPLEMENTAZIONE DATABASE

Il database è stato realizzato tramite MySql utilizzando anche le tecniche apprese durante il corso di "laboratorio di base di dati" ed è stato connesso tramite il driver Java Data Base Connectivity (JDBC) nella versione 5.1.23, per il DBMS MySql. Le classi e i metodi messi a disposizione dal JDBC sono stati utilizzati nei vari model e controller per effettuare le operazioni necessarie da effettuare sul database per il funzionamento del programma.

4.1 SCHEMA E-R DEL DATABASE

Per quanto riguarda la realizzazione del database è stato realizzato in 3 fasi:
1- progettazione concettuale tramite schema E-R
2- progettazione logica tramite schema relazionale
3- implementazione tramite MySql.

Più volte siamo passati per le prime due fasi, ristrutturando lo schema E-R e riprogettando il database, infine il risultato finale è lo schema E-R a lato di questa scritta:



4.2 SCHEMA RELAZIONALE DEL DATABASE

E: Veicolo (Targa, modello, marca, altezza, peso, anno, N°assi, classeEuro);

E: Autostrada (Nome);

E: Amministratore (nome, cognome);

E: Tariffa (*nomeAutostrada*, valore, classeVeicolo);
vincolo di integrità referenziale tra nomeAutostrada in **Tariffa** e Nome in **Autostrada**.

E: Casello (*nomeAutostrada*, nome, kilometro);
vincolo di integrità referenziale tra nomeAutostrada in **Casello** e Nome in **Autostrada**.

E: Percorso *(*codicePercorso, *nomeAutostrada, caselloPartenza*, *caselloArrivo*);
vincolo di integrità referenziale tra nomeAutostrada in **Percorso** e Nome in **Autostrada**.
vincolo di integrità referenziale tra caselloPartenza in **Percorso** e codice in **Casello**.
vincolo di integrità referenziale tra caselloArrivo in **Percorso** e codice in **Casello**.