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**

**A close up of a flag

Description automatically generated**

**INDICE**

**Introduzione**

* 1. Elenco Dei Requisiti

**Architettura Del Software**

2.1 Modello dell’Architettura Software

2.2 Descrizione Architettura

2.3 Descrizione Scelte e Strategie

**Progettazione Classi e Interfacce**

3.1 Descrizione Classi , Interfacce e membri

3.2 Descrizione dei dettagli di design scelti

**Implementazione Database**

4.1 Schema E-R Del Database

4.2 Schema Relazionale Del Database

**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 e la tipologia di veicolo ed ovviamente il percorso.  
Tale applicazione è stata realizzata tenendo conto delle varie entità che la utilizzeranno, dividendole in due tipologie, l’amministratore ed il casello (utente).  
Il programma è pensato, come da consegna, ricevendo da input un biglietto (dalla quale si otterranno i dati relativi al percorso), il programma restituirà il pedaggio da pagare in conformità con la normativa vigente.

* 1. **Elenco 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 (le, affidabile ed implementato tramite Java e MySql (per la realizzazione di un database da collegare con il programma).

**Architettura del software**

Nel seguente paragrafo verrà descritta l’architettura del software, quindi le scelte di design intraprese dal gruppo per realizzare questo progetto.

**2.1 Modello dell’Architettura Software**

(qui mettere immagine dell’organizzazione in package quando il progetto è pronto)

**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**. (magari il controller essendo di Gianluca ci pensa lui ad ampliarlo… no so che altro dirci)  
La **View** (non essendo ancora completa non so cosa scrivere)

**2.2 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.

Inserire qualcosa relativo a polimorfismo  
Inserire qualcosa relativo a override  
Inserire qualcosa relativo a overload  
inserire qualcosa relativo a dynamic binding

Inserire qualcosa relativo a ereditarietà

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 (tipo davide ha usato il singleton, ma non so che metterci a riguardo)

**Progettazione classi e interfacce**

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 e un HashSet di Caselli collegati tra che vanno a formare la stessa.

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.

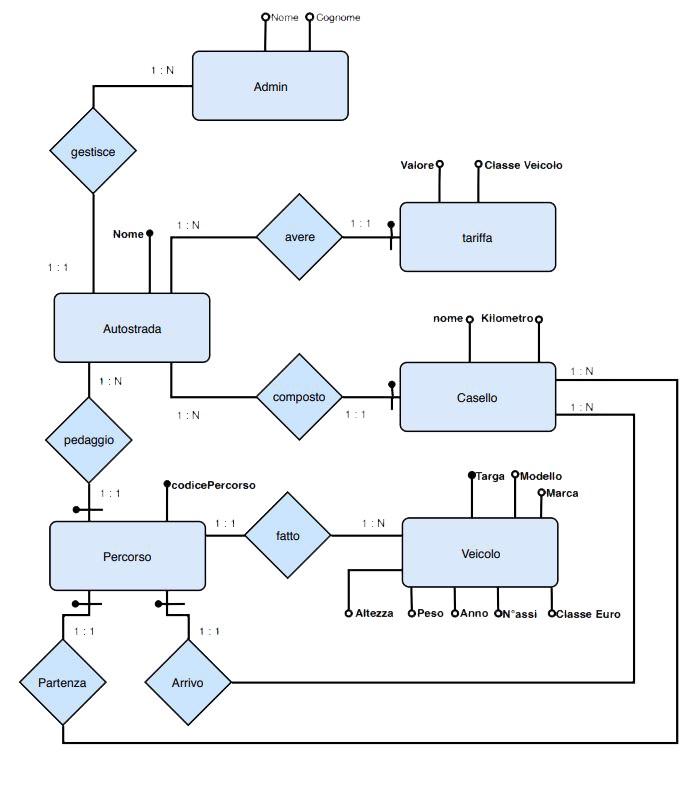
Nell’ultima fase progettuale è stata introdotta la figura dell’Amministratore, cioè colui che è abilitato ad effettuare le modifiche (inserimento, cancellazione ecc.) delle prime 3 classi principali (Autostrada, Casello, Veicolo). Inoltre, è abilitato anche al cambiamento del sistema del calcolo del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

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

**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 WindowBuilder (Davide certifichi?) installando il driver JDBC (Java DataBase Connectivity).

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

Più volte siamo passati per le prime due fasi, ristrutturando lo schema E-R e riprogettando il database, infine il risultato finale è il seguente Schema E-R:

Ed il seguente Schema relazionale:

**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**.