GREEN PROJECT

Green Project è un applicativo desktop che calcola un pedaggio corretto in base all’ autostrada percorsa dall’utente.

L’ applicativo si divide in 2 parti diverse;

Parte operatore;

Il programma deve saper simulare un sistema che racchiude i vari elementi sotto specificati:

1. Il sistema deve saper calcolare la tariffa unitaria con la seguente formula; prodotto dei chilometri (km) relativi ad ogni percorrenza per la tariffa di competenza più la normativa, più eventuali maggiorazioni ed imposte previste dalla normativa vigente e occasionalmente, maggiorazioni per il terreno.
2. Al prodotto sopra specificato deve essere aggiunta un iva e il sistema deve saper moltiplicare per eccesso o per difetto.
3. La tariffa deve dipendere dalla classe del veicolo utilizzato.

Parte admin;

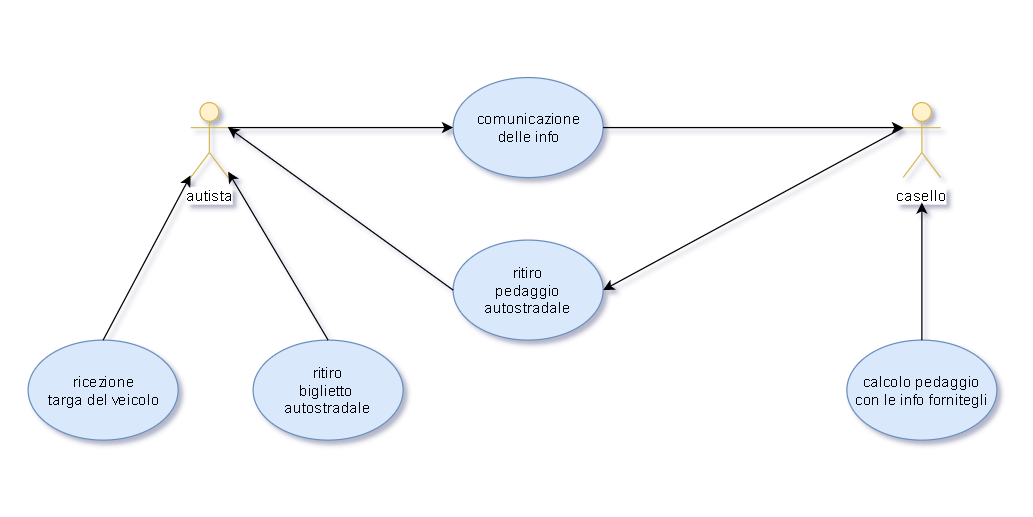
1. Il programma deve avere un pannello amministrativo che permetta di modificare, cancellare o inserire una nuova autostrada.

Come ho implementato la parte per operatore ;

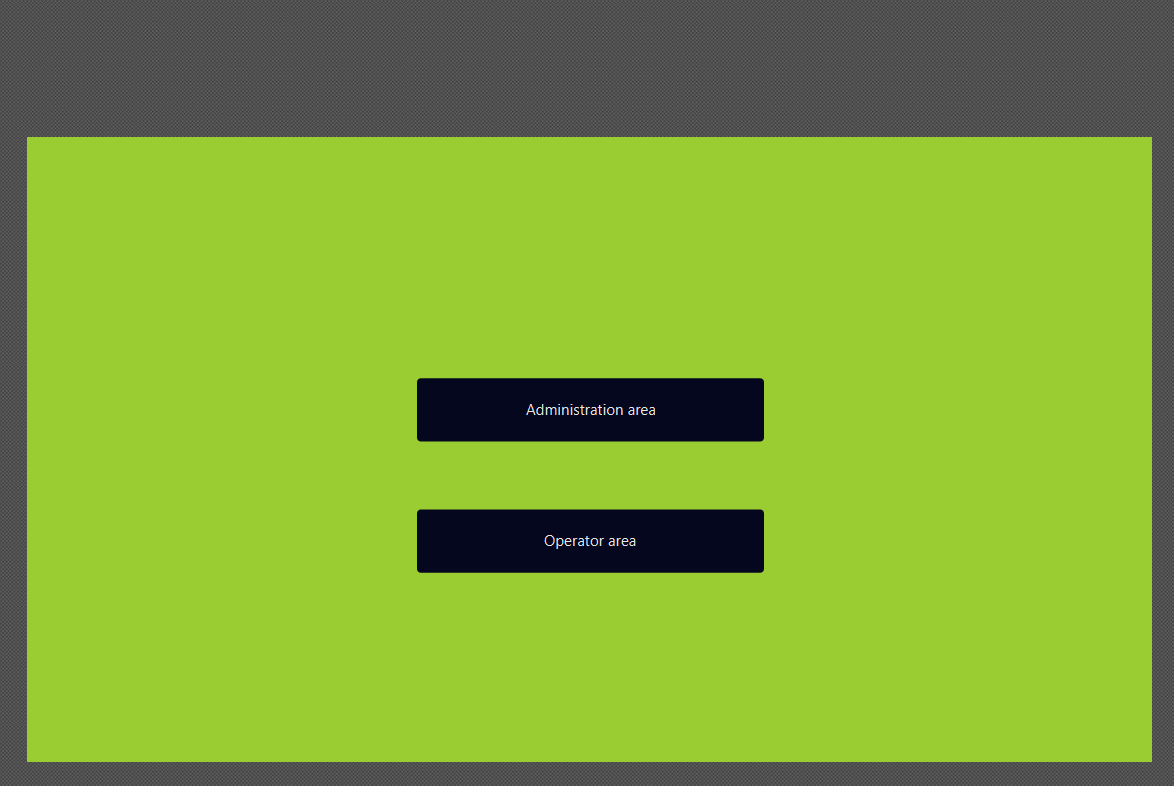
il programma è in grado di calcolare un eventuale pedaggio, grazie ai due elementi fornitegli, ovvero la targa del veicolo e il biglietto d’ ingresso.

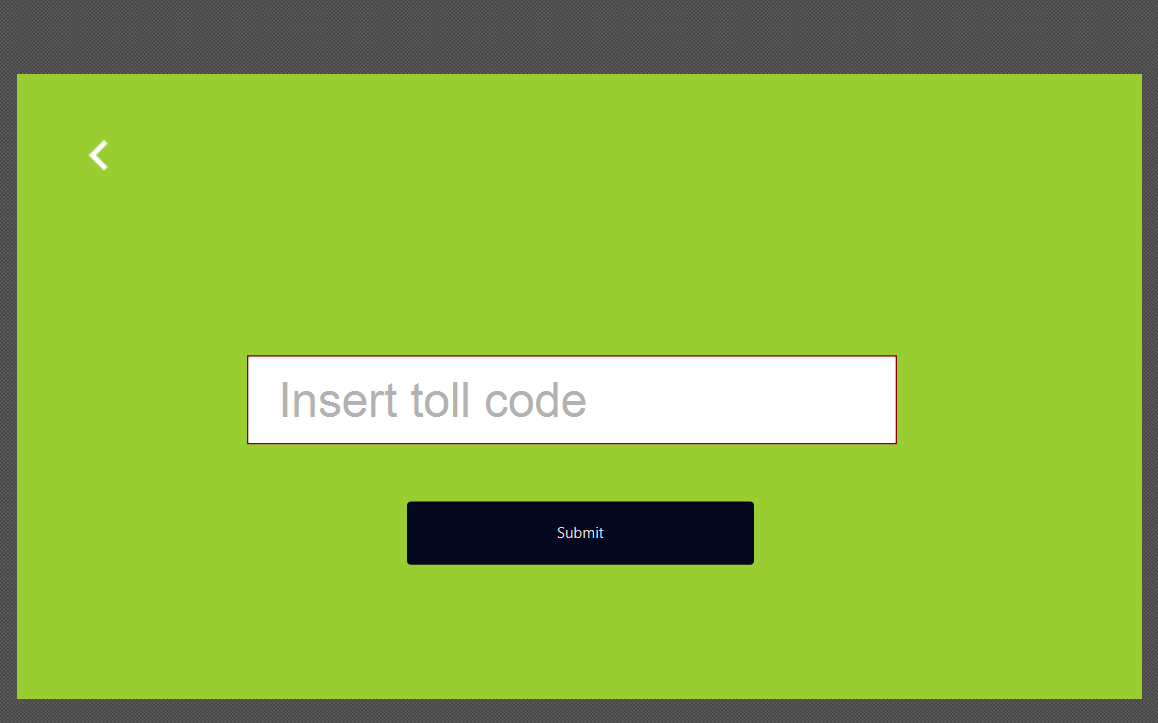
La targa viene ripresa da delle telecamere e in seguito verrà riconosciuta, mentre invece il casello di ingresso viene riconosciuto tramite un codice nel biglietto d’ ingresso, questi due elementi, uniti alle eventuali maggiorazioni attuati dall’ ecotassa ,dal terreno e dall’applicazione dell’ iva.

In sostanza funziona così;

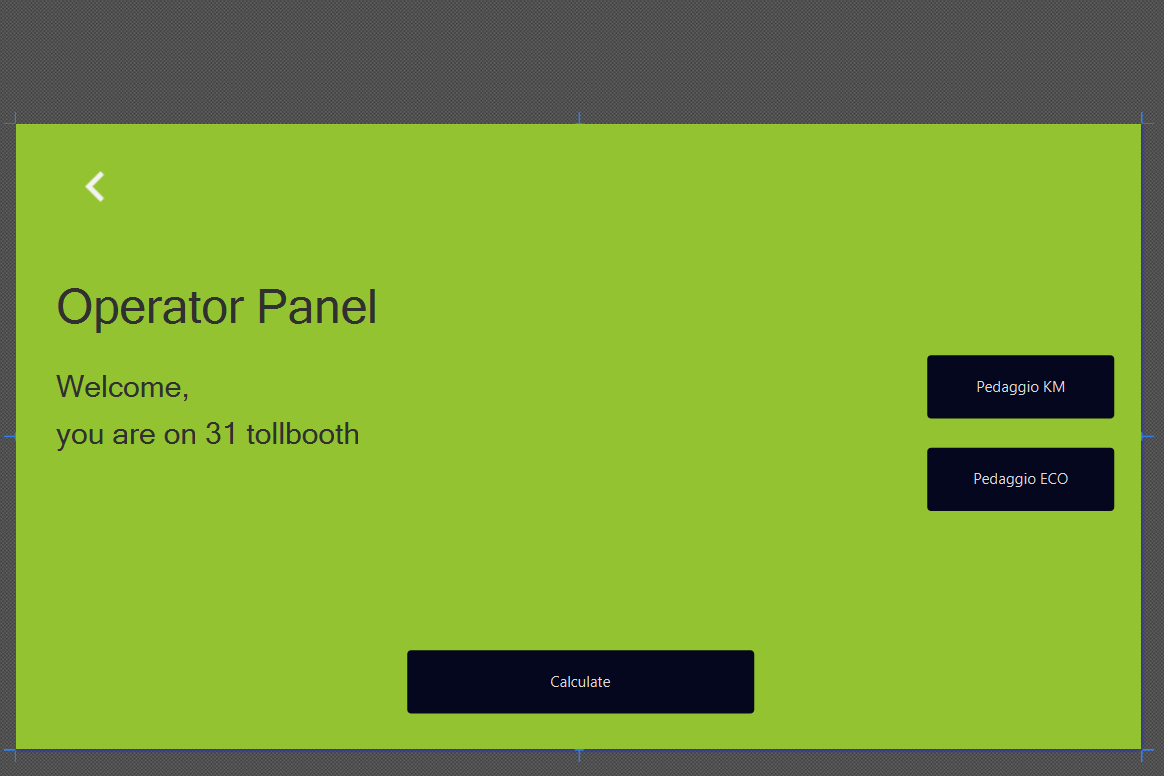


1. Il veicolo entra nel casello di ingresso, il conducente ritira il biglietto.
2. Le targa viene salvata dalle telecamere per far riconoscere il veicolo
3. Il veicolo arriva al casello, il conducente immette il biglietto ed il sistema riconosce il casello di ingresso.
4. Grazie agli elementi acquisiti e alle altre impostazioni il sistema calcola il pedaggio.
5. Una volta calcolato il sistema comunica il pedaggio al conducente.

A questa finestra vi si può accedere prima passando per un menu comune che si presenta cosi;

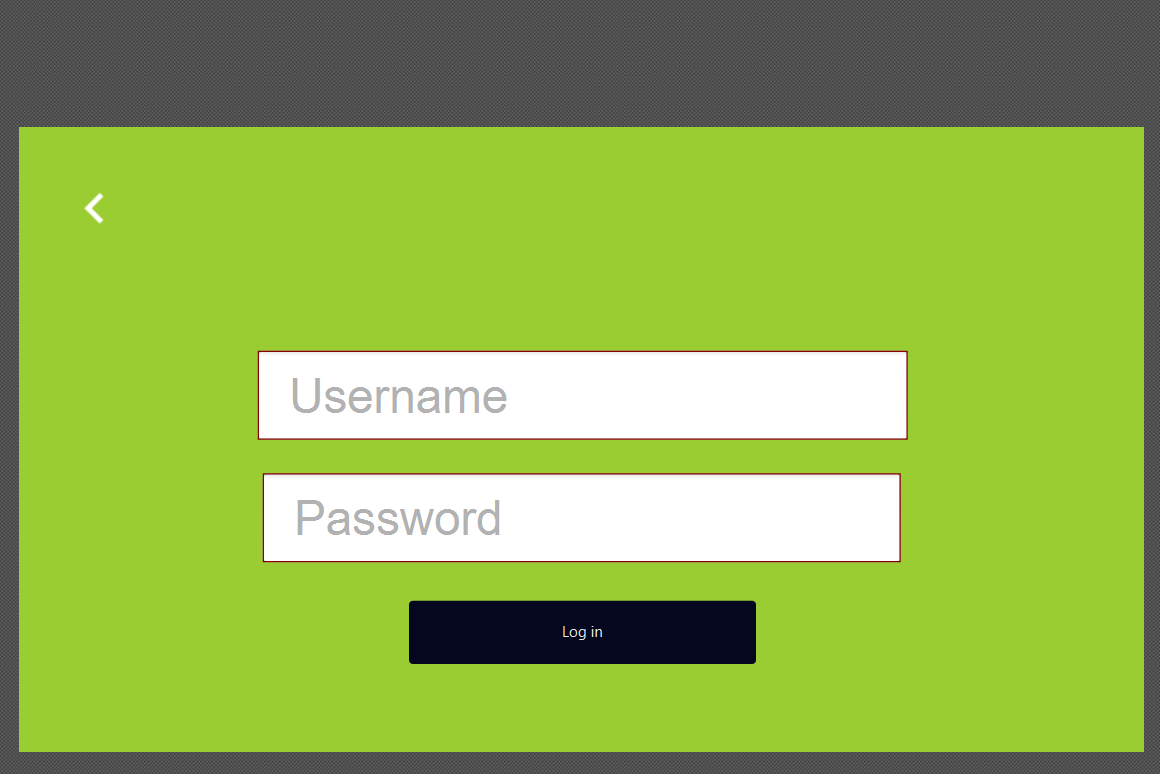
Poi selezionando la scelta operator area, si accede ad un finestra cosi mostrata;

In cui si dovrà inserire il codice del biglietto, una volta fatto, si arriva alla finestra finale in cui ci verrà comunicato il pedaggio;



Come ho implementato la parte per admin;

Il programma dispone di un interfaccia privata a cui è possibile accedere solo tramite password, che permette di aggiungere, modificare o cancellare le autostrade e i caselli.

Ed è possibile accedervi, tramite il menu comune, che ci porterà in un interfaccia di login,

Che è protetta da una password.

Una volta effettuato l’ accesso, si accede ad un interfaccia che si presenta cosi:



Che permette di creare nuove autostrade o modificarle, i bottoni sotto la tua immagine profilo, si accede ad altre 2 interfacce simili che permettono la modifica o l eliminazione dei caselli attualmente presenti in memoria;

Architettura di sistema

Leggenda del diagramma:

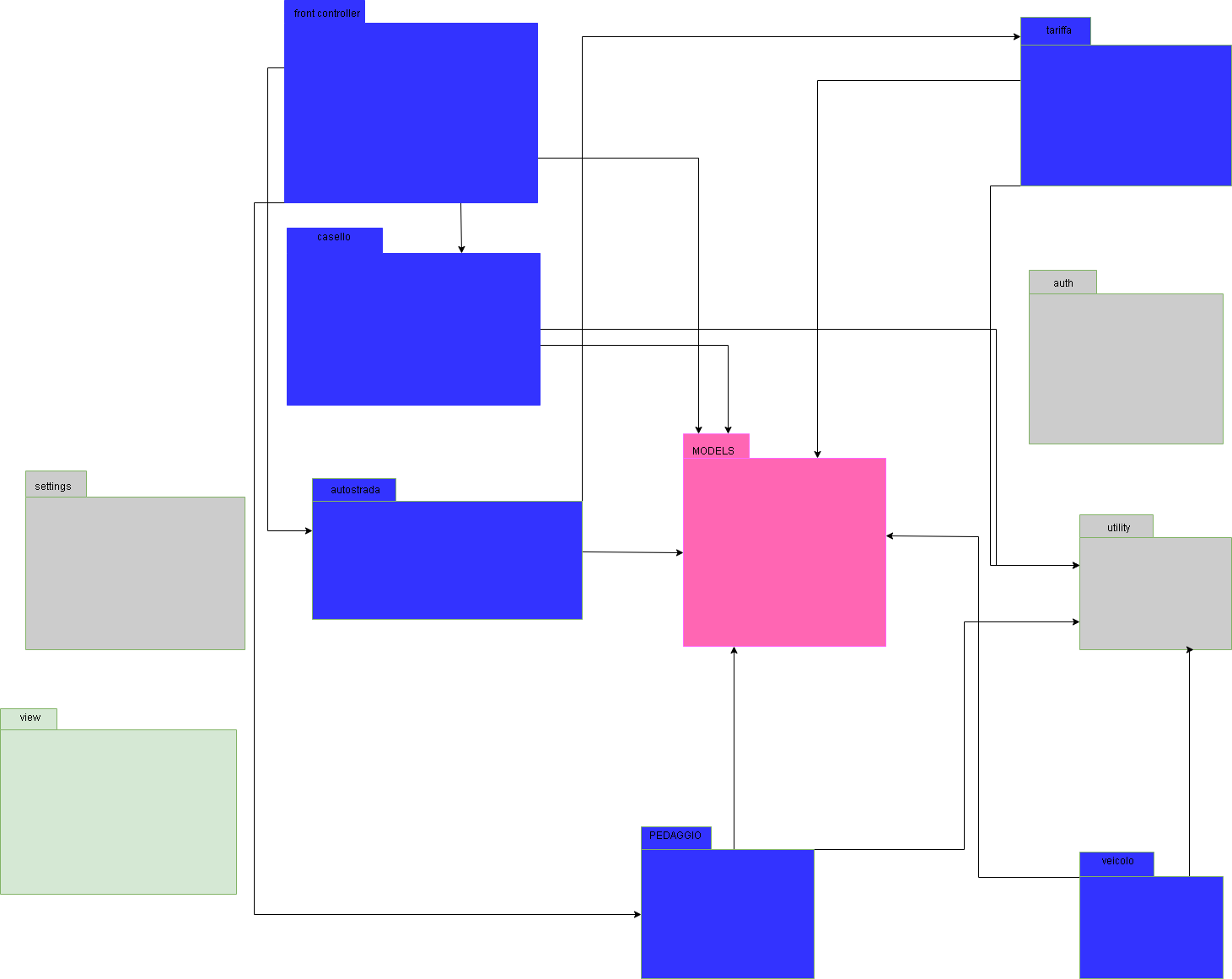
Blu=controller

Rosa=model

Grigio=impostazioni

Verde=view

Le stesse analogie sono state riportate anche nella versione estesa per dare una chiara distinzione all’ osservatore



Il diagramma descrive brevemente(per ulteriori dettagli consultare il class diagram in cartella) l’architettura di sistema, realizzata secondo i parametri del design pattern MVC(model,view,controlloer).

Dove i model, astraggono le entità nei DB, fornendo metodi volti alla lettura, scrittura e modifica di esse.

I controller hanno scopo di interpretare i segnali che l’operatore invia al programma attraverso le finestre fornite dalla view.

La view fa uso di file fxml per realizzare un interfaccia grafica, compressibile ad un operatore, rendendo così più semplice svolgere l sua funzione.

Le impostazioni invece sono parametri prefissati o parte dell’ architettura che non fa parte del paradigma MVC; come il database e le costanti che abilitano la connessione.

I controller che sono stati realizzati sono; veicolo, login, autostrada, pedaggio, casello.

Di view invece ne sono state realizzate 3;

1. una in comune che rappresenta quella iniziale che permette di scegliere a quale interfaccia si voglia accedere se a quella privata o a quella da operatore.
2. Una da operatore che fornisce il pedaggio da pagare e i caselli attraversati.
3. E una privata da admin che permette di modificare, inserire e cancellare le autostrade e i caselli

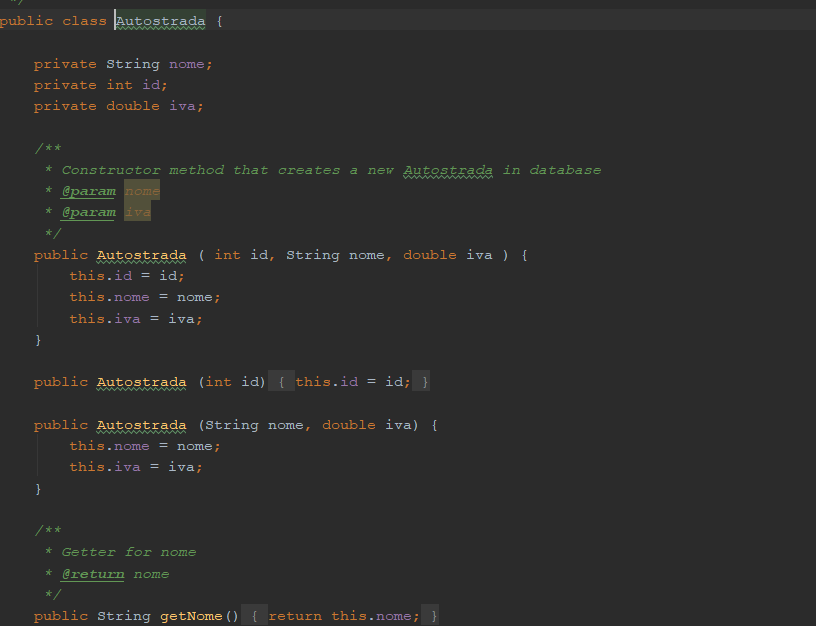
Classi e strategie;

nei package mostrati nel diagramma di sopra sono annidate diverse classi(è illustrato in maniera dettagliata nel class diagram allegato).

Per queste classi sono stati utilizzati alcuni modificatori a proposito del loro utilizzo.

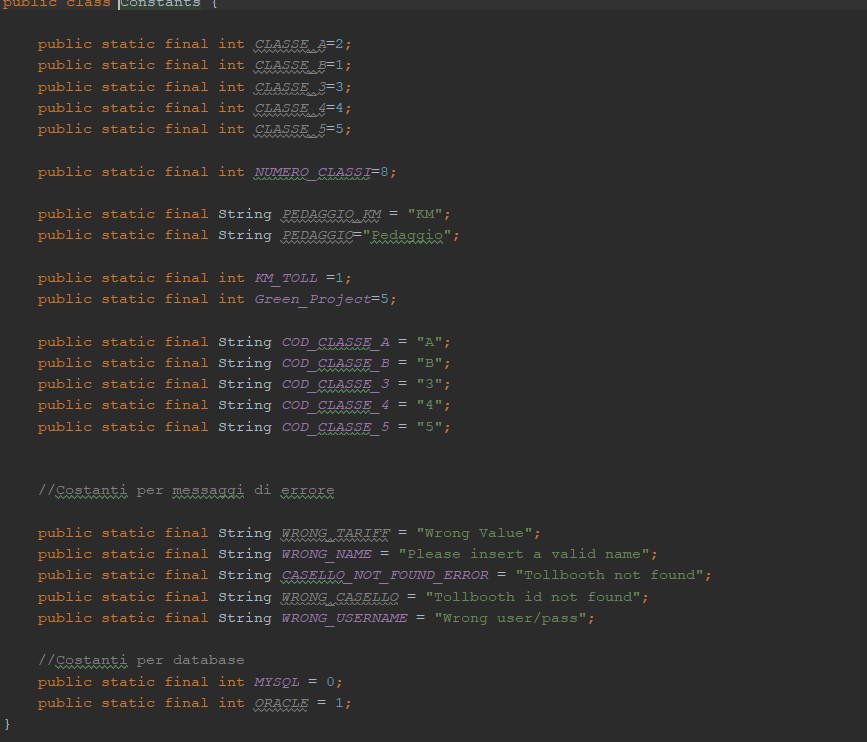
Public; per tutti i metodi e le classi che potrebbero essere utilizzati anche all’esterno della classe d’ origine.

Exp

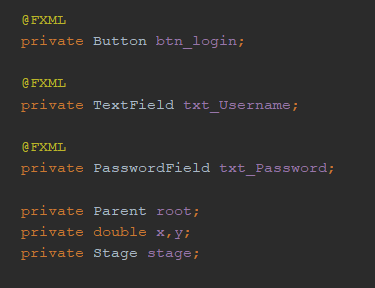


Final; per le costanti e le impostazioni.

Exp:



Private; in tutti i metodi e le variabili che non vengono mai utilizzati al di fuori della classe d’ origine.



Altri 2 modificatori che sono stati utilizzati anche se un po meno sono stati static e abstract; static è stato utilizzato soprattutto nella classe getveicolo e nelle costants assieme a final, abstract è stata sfrutta per creare la classe veicolo e seguire cosi meglio il design pattern della factory.

Parlando di quest’ ultima classe, nella classe veicolo si è fatto un grand’ uso dell’ ereditarietà per poter così passare più facilmente da una categoria all’ altra(A,B,3,4,5).

Il polimorfismo, l’ over ride e l’overlay invece sono stati utilizzati in gran parte del sistema.

Per rendere più agevole, l’associazione categoria veicolo e tariffa fatta dal sistema nel database è stata utilizzata una Map<string,float>.

Varie eccezioni state programmate in diverse parti del sistema per prevedere alcuni possibili errori.

Interfacce usate;

nel programma sono state utilizzate 3 interfacce;

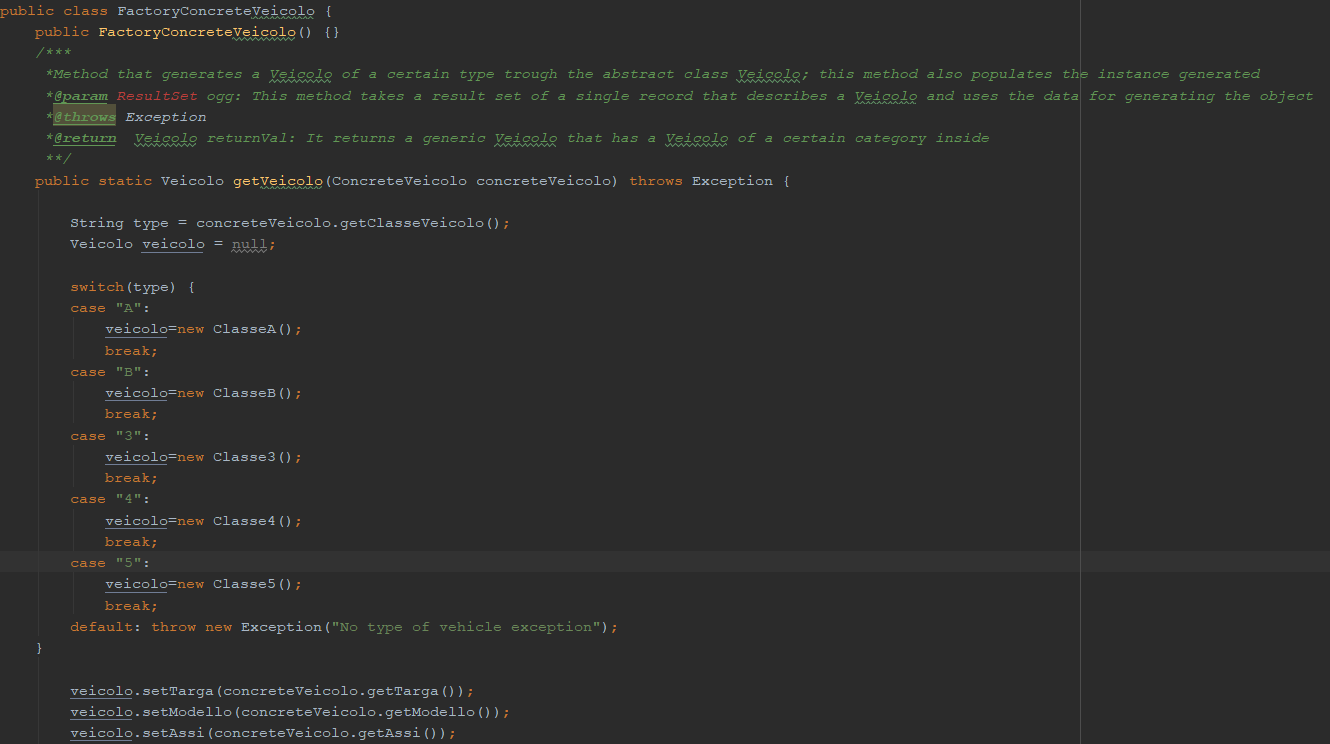
Pedaggio; con la quale è stato definito il comportamento di tutte le classi pedaggio.

Model; definisce i prototipi per i metodi comuni a tutti i model presenti nel sistema.

Controller; definisce i metodi necessari a uniformare la grafica e il comportamento dell’ interfaccia.

Nel programma utilizzo i seguenti design pattern;

Design Pattern usati;

factory; è stato usato nella realizzazione della classe veicolo, che è implementata come abstract, e delle classi categoria che ereditano da essa

Singleton; è stato realizzato per stabilire un’ instanza del driver jdbc nel db.

Decorator; viene utilizzato per aumentare la funzionalità della view per cosi aggiungere più controllo ai controller degli oggetti presenti nella view.

MVC; il design pattern del model view controller è stato utilizzato per strutturare tutta la logica del programma, esso infatti è composto da una view con cui l utente interagisce, il controller che interpreta i segnali che l utente invia al programma grazie alla view e un model che gestisce le entità inserite nei database.

Tecnologie utilizzate

È stata utilizzata la versione di java 10.0.2.

Il gestore dipendenze di Maven 4.0.0.

Il jdbc nella versione 5.1.36,per il DBMS Mysql.

La versione 10 di javaf, fornita dal java development kit(jdk 10.0.2).