Università Politecnica delle Marche – Facoltà di Ingegneria

**INGEGNERIA INFORMATICA E DELL’AUTOMAZIONE**

******

**APPLICAZIONE PER LA GESTIONE DI UN AUTONOLEGGIO**

**RELAZIONE**

ABBADINI LORENZO, COMPAGNONI PAOLO, SCISCI VALERIO

**OOP1617Gruppo08**

Sommario

[Introduzione 3](#_Toc491278802)

[Note 3](#_Toc491278803)

[Analisi del dominio di interesse 4](#_Toc491278804)

[Tipologie di clienti 4](#_Toc491278805)

[Tipologie di noleggio 4](#_Toc491278806)

[Tipologie di utenti dell’applicazione 5](#_Toc491278807)

[Note 5](#_Toc491278808)

[Strutturazione dei requisiti 6](#_Toc491278809)

[Diagramma dei casi d’uso 6](#_Toc491278810)

[Gestione clienti 7](#_Toc491278811)

[Gestione contratti 7](#_Toc491278812)

[Gestione flotta 8](#_Toc491278813)

[Funzionalità extra 8](#_Toc491278814)

[Progettazione del database 9](#_Toc491278815)

[Note 10](#_Toc491278816)

[Struttura del progetto 11](#_Toc491278817)

[Struttura della cartella “src” 11](#_Toc491278818)

[Diagramma delle classi 12](#_Toc491278819)

[Implementazione 18](#_Toc491278820)

[Package “autonoleggio” 18](#_Toc491278821)

[Autonoleggio 18](#_Toc491278822)

[Note 19](#_Toc491278823)

[Login 19](#_Toc491278824)

[Package “db” 21](#_Toc491278825)

[DBConnect 21](#_Toc491278826)

[Package “entita” 22](#_Toc491278827)

[Contratto 22](#_Toc491278828)

[Preventivo 26](#_Toc491278829)

[Package “finestre” 30](#_Toc491278830)

[Finestra 30](#_Toc491278831)

[Package “pannelli” 32](#_Toc491278832)

[PannelloCliente 32](#_Toc491278833)

[Package “moduli” 34](#_Toc491278834)

[ModuloCliente 34](#_Toc491278835)

[Package “moduliOpzionali” 37](#_Toc491278836)

[ModuloElencoClienti 38](#_Toc491278837)

[ModuloCalendario 40](#_Toc491278838)

[Bibliografia 41](#_Toc491278839)

# **Introduzione**

Il progetto è stato sviluppato per informatizzare la gestione dell’autonoleggio RentForYou situato a Giulianova (TE).

L’applicazione offre all’utente le seguenti funzionalità:

* calcolo del profitto mensile/annuale
* creazione di statistiche
* gestione dei clienti
* gestione dei contratti di noleggio
* gestione del parco veicoli
* monitoraggio delle scadenze

Per un corretto utilizzo dell’applicazione si raccomanda di consultare il *Manuale di Prima Esecuzione* ed il *Manuale di Utilizzo*.

## Note

Le figure contenute nella presente relazione sono riportate nella cartella **indicare path cartella** per una migliore leggibilità.

# **Analisi del dominio di interesse**

## Tipologie di clienti

* Associazioni
* Aziende
* Privati

Le **associazioni** di qualsiasi genere (sportive, culturali, ecc.) hanno diritto ad uno sconto del 20% sul costo totale del noleggio.

Le **aziende** hanno diritto ad uno sconto del 10% sul costo totale, nel caso di noleggio a lungo termine.

## Tipologie di noleggio

* Noleggio a breve termine
* Noleggio a lungo termine

Il **noleggio a breve termine**, riguarda tutti quei casi in cui l’auto viene noleggiata temporaneamente, da un giorno fino anche a sei mesi. Il noleggio a breve termine prevede tariffe che variano in base al periodo, e sono più basse quanto più si allunga il periodo di noleggio. Questa formula è una soluzione molto utilizzata da privati ed aziende che necessitano di un’auto per spostamenti brevi (da uno a più giorni), per sostituire veicoli in assistenza, o per specifiche esigenze aziendali (dotare di auto un collaboratore temporaneo, ad esempio).

Il **noleggio a lungo termine** è una formula di abbonamento mensile che permette di utilizzare un’auto senza acquistarla e senza spendere soldi per la sua manutenzione. Il noleggio a lungo termine prevede un contratto che va dai 12 ai 36 mesi, con rate annuali o mensili fisse che variano in funzione del veicolo e dei chilometri che si prevedono di effettuare. Le rate mensili vanno da un minimo di 200€ ad un massimo di 500€. Per il lungo termine le tariffe vanno calcolate in funzione della quota mensile.

## Tipologie di utenti dell’applicazione

* Amministratore
* Utente standard

L’**utente standard** ha accesso solo alla gestione dei clienti e dei contratti di noleggio.

L'**amministratore** ha accesso completo a tutte le funzionalità del programma.

## Note

Per informazioni più dettagliate circa le tipologie di veicoli, le tariffe e le dinamiche interne dell’attività si consiglia di consultare le specifiche di progetto fornite con l’applicazione.

# **Strutturazione dei requisiti**

In seguito allo studio del dominio di interesse, abbiamo strutturato i requisiti dell’applicazione utilizzando il linguaggio UML.

## Diagramma dei casi d’uso

Seguendo le indicazioni del testo di riferimento sul linguaggio UML (Fowler, 2010), abbiamo cercato di rendere il tutto più chiaro trattando separatamente i diversi casi d’uso.

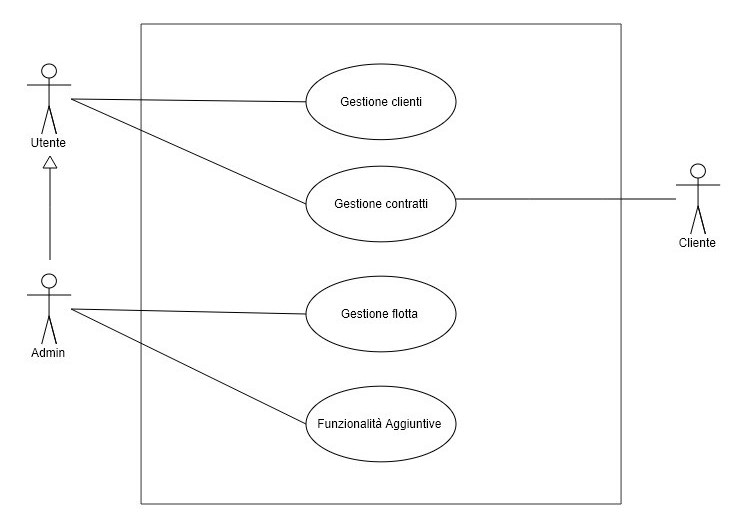
Gli attori del diagramma dei casi d’uso sono:

* Amministratore
* Cliente
* Utente standard

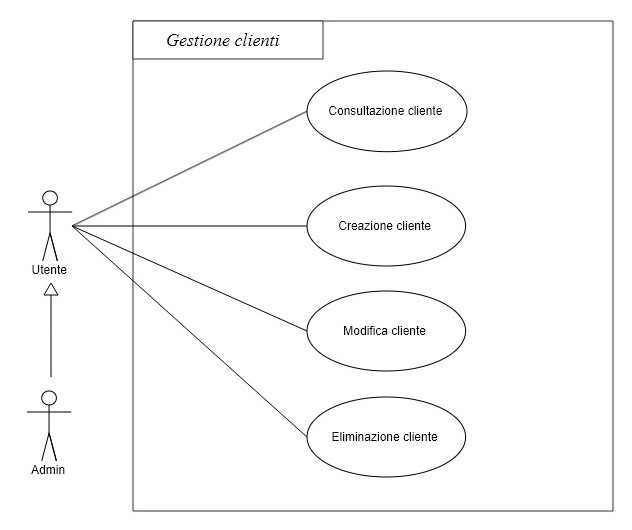
L’amministratore è il titolare dell’attività. I casi d’uso ad esso associati rappresentano le principali funzionalità fornite dall’applicazione.

L’utente standard è l’impiegato dell’attività. I casi d’uso ad esso associati sono un sottoinsieme di quelli associati all’amministratore.

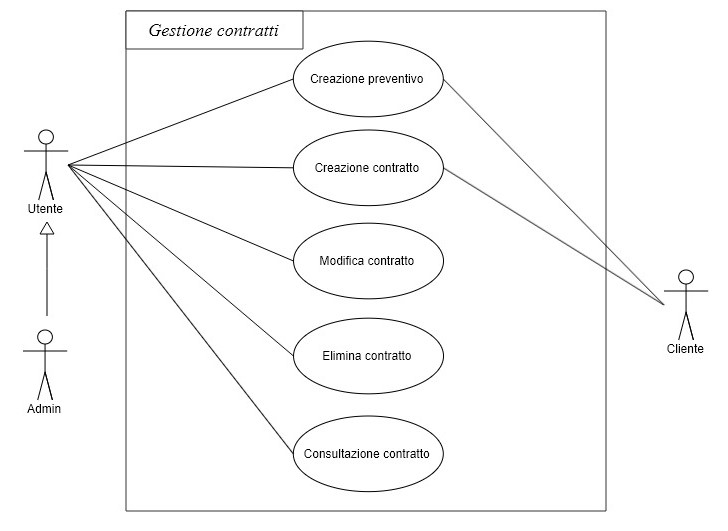
Al cliente è associato un solo caso d’uso. Come evidenziato nella sezione Gestione contratti, esso può soltanto richiedere un preventivo o stipulare un contratto di noleggio.



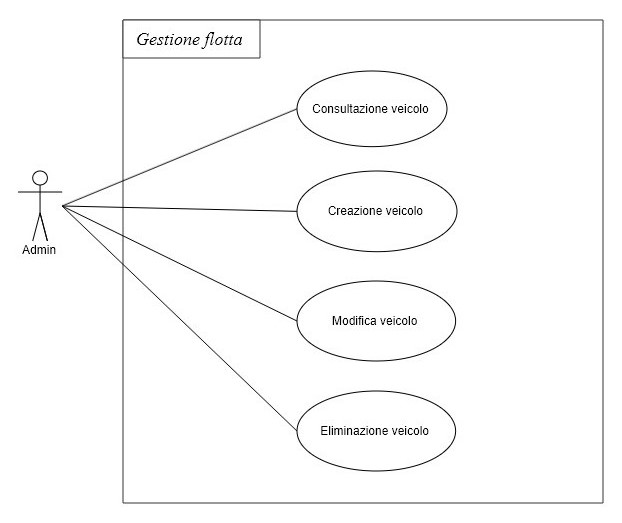
### Gestione clienti



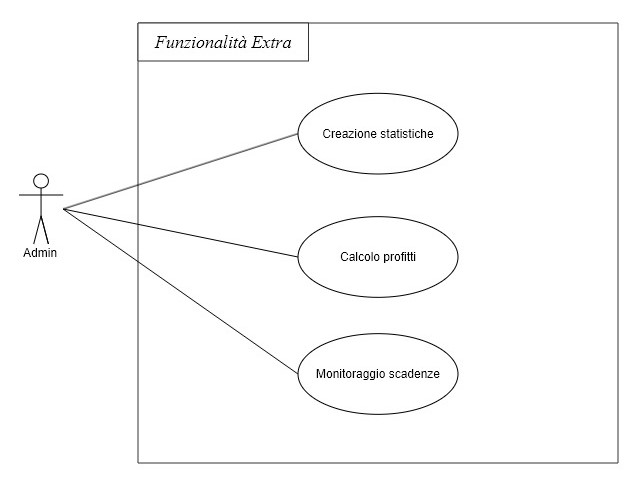
### Gestione contratti



### Gestione flotta



### Funzionalità extra



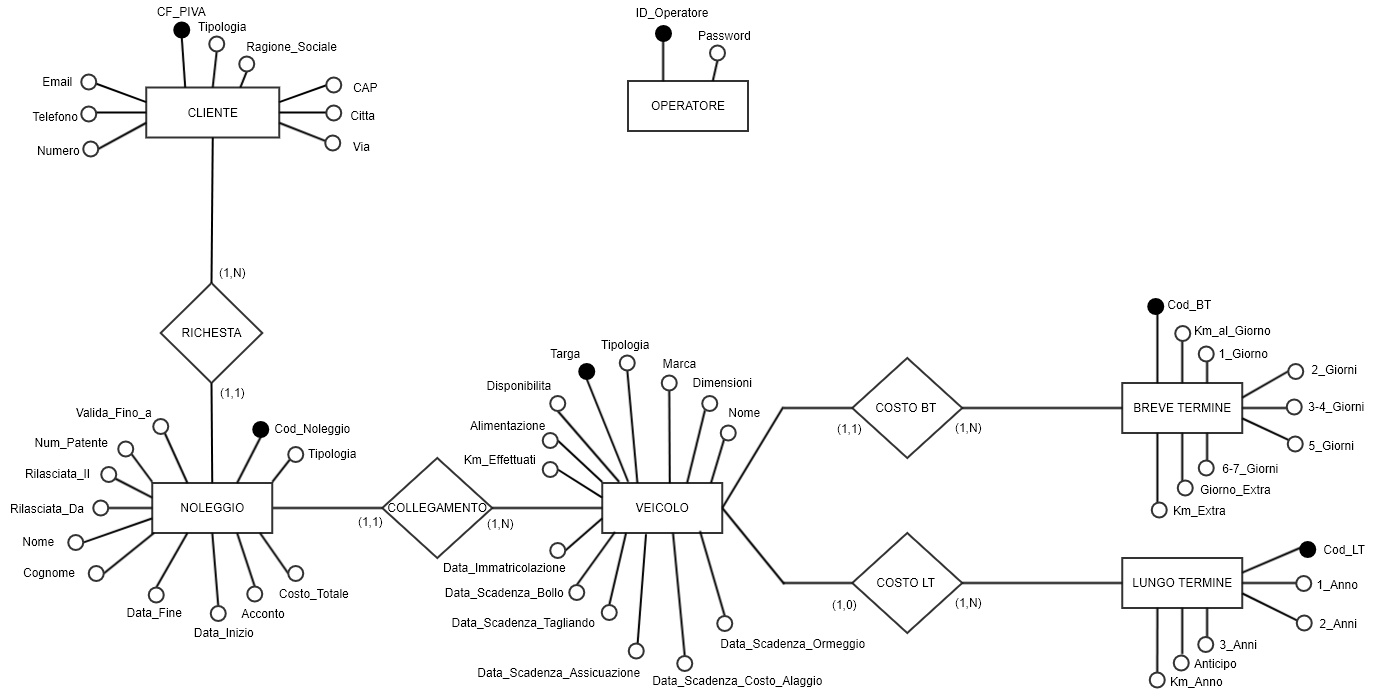
## Progettazione del database

Come già visto nel Manuale di Prima Esecuzione e nel Manuale di Utilizzo, per il corretto funzionamento dell’applicazione è necessaria la presenza di un database. In particolare si fa uso dell’applicazione multipiattaforma XAMPP contenente il DBMS MySQL.

In fase di progettazione è stata adottata la strategia top-down. Il lavoro ha seguito le seguenti fasi:

* analisi del dominio di interesse ed individuazione delle entità principali
* costruzione dello scheletro dello schema concettuale
* specializzazione dello schema concettuale
* ristrutturazione dello schema concettuale

Lo schema E-R ottenuto al termine della progettazione è il seguente.



L’entità Operatore rappresenta gli utenti che utilizzeranno il software. Ciascun utente è caratterizzato da un username (ID\_Operatore) e da una password.

L’entità Cliente rappresenta i clienti dell’autonoleggio; per ciascuno di essi vengono memorizzati tutti i dati anagrafici.

L’entità Noleggio rappresenta i contratti di noleggio; per ciascuno di essi vengono memorizzati anche i dati anagrafici del conducente, che in generale può essere diverso dal cliente che ha stipulato il contratto.

L’entità Veicolo rappresenta i mezzi posseduti dall’autonoleggio; per ciascuno di essi vengono memorizzati tutti i dati di interesse. Il campo Disponibilità indica se un veicolo è impegnato in operazioni di manutenzione.

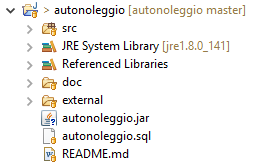
L’entità Breve Termine rappresenta il tariffario per i noleggi a breve termine.

L’entità Lungo Termine rappresenta il tariffario per i noleggi a lungo termine.

### Note

Un preventivo può essere scartato oppure trasformato in un contratto di noleggio a tutti gli effetti. Questo non rende necessario tenere uno storico dei preventivi calcolati; per questo motivo, nello schema E-R non è presente una entità Preventivo.

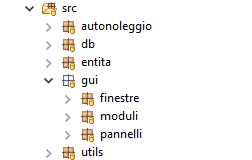
# **Struttura del progetto**



La cartella principale del progetto contiene tre sottocartelle:

* doc contiene il javaDoc
* external contiene le librerie importate all’interno del programma in fase di configurazione
  + car per l’icona dell’applicazione
  + jcalendar per la gestione delle date all’interno dell’applicazione
  + mysqlconnector per la connettività al database
  + seaglasslookandfeel per il look and feel dell’applicazione
* source contiene il codice sorgente dell’applicazione

## Struttura della cartella “src”



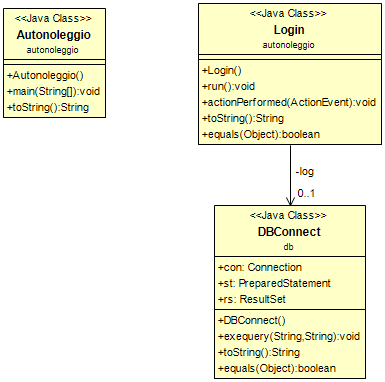
La cartella src contiene tutto il codice sorgente organizzato in package:

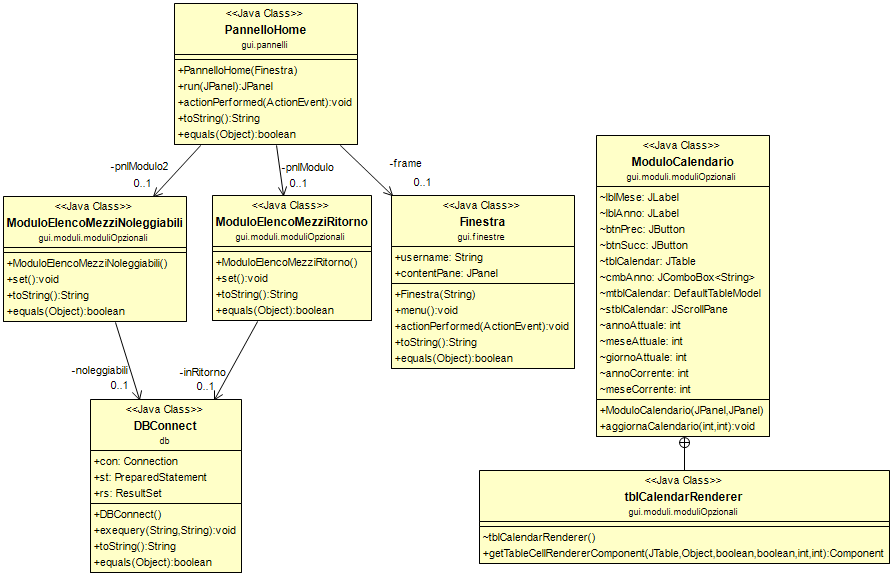
* autonoleggio contiene la classe necessaria per avviare l’applicazione e la classe per il login
* db contiene la classe necessaria per effettuare la connessione al database
* entita contiene le classi che mappano le entità del database
* gui contiene le classi che implementano l’interfaccia grafica dell’applicazione ed è organizzato nei seguenti sottopackage:
  + finestre
  + moduli
  + pannelli
* utils contiene classi che implementano funzionalità necessarie per l’applicazione

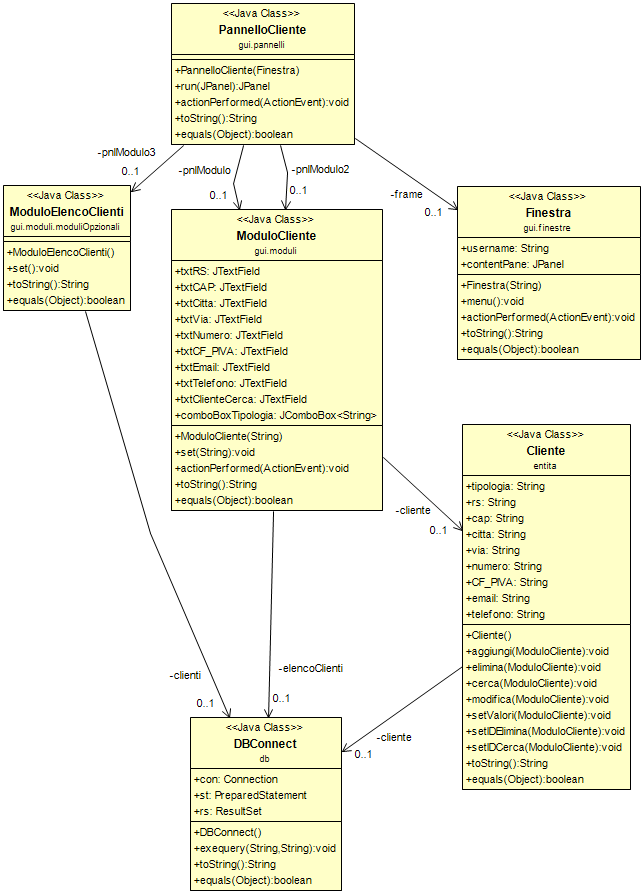
## Diagramma delle classi

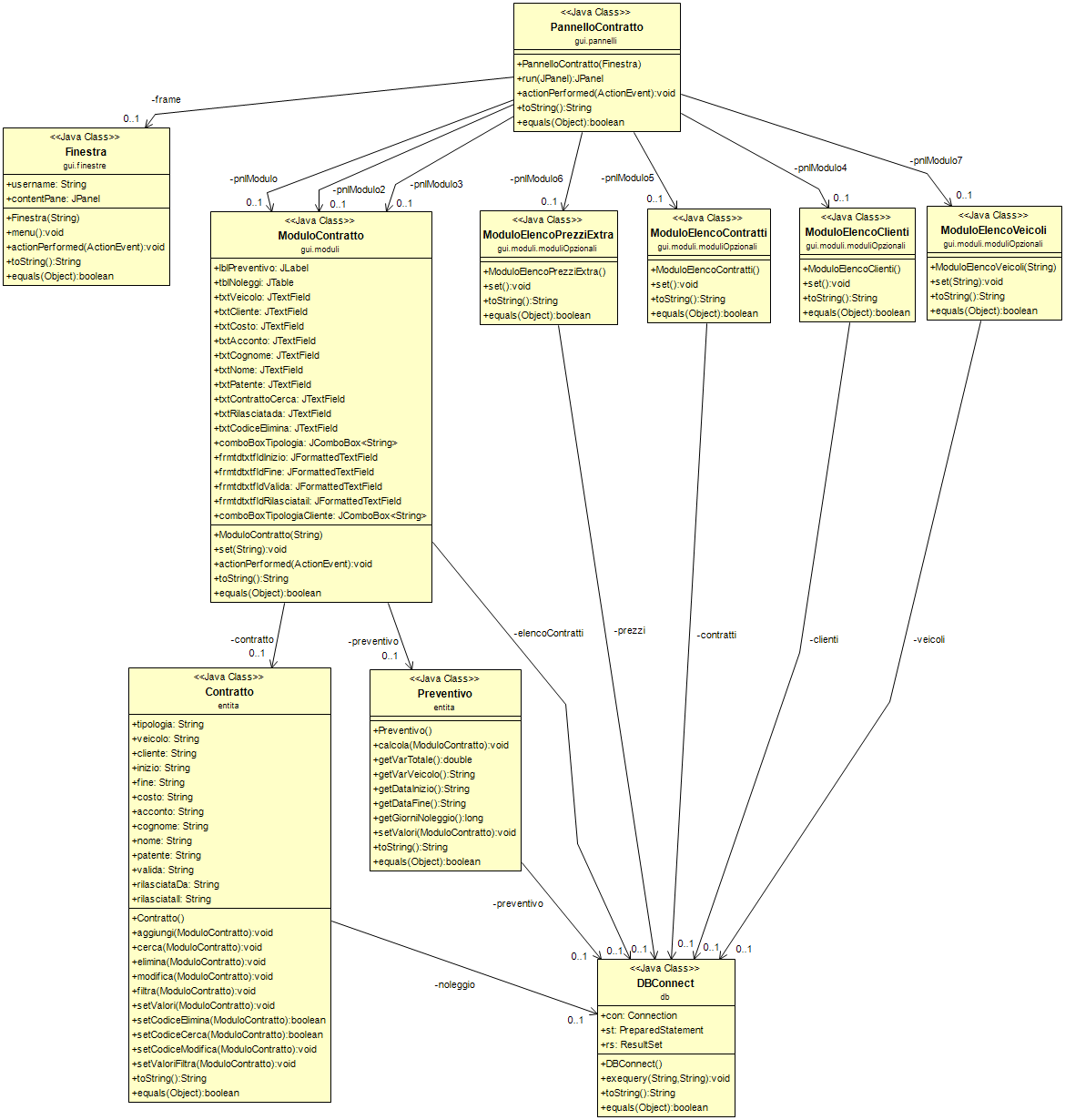
Il diagramma descrive le caratteristiche delle classi che compongono l’applicazione e le relazioni statiche esistenti tra di loro.

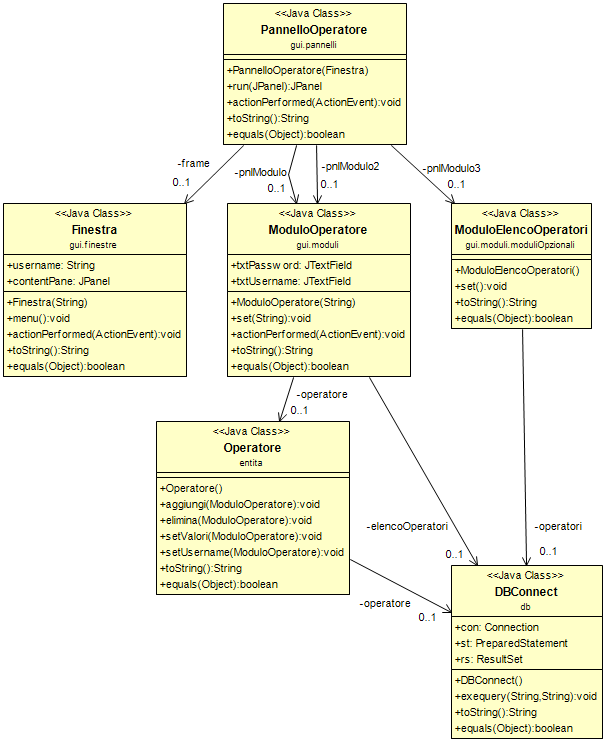
Come fatto per il diagramma dei casi d’uso, si è scelto di suddividere il diagramma delle classi in più parti, in modo tale da facilitarne la lettura.

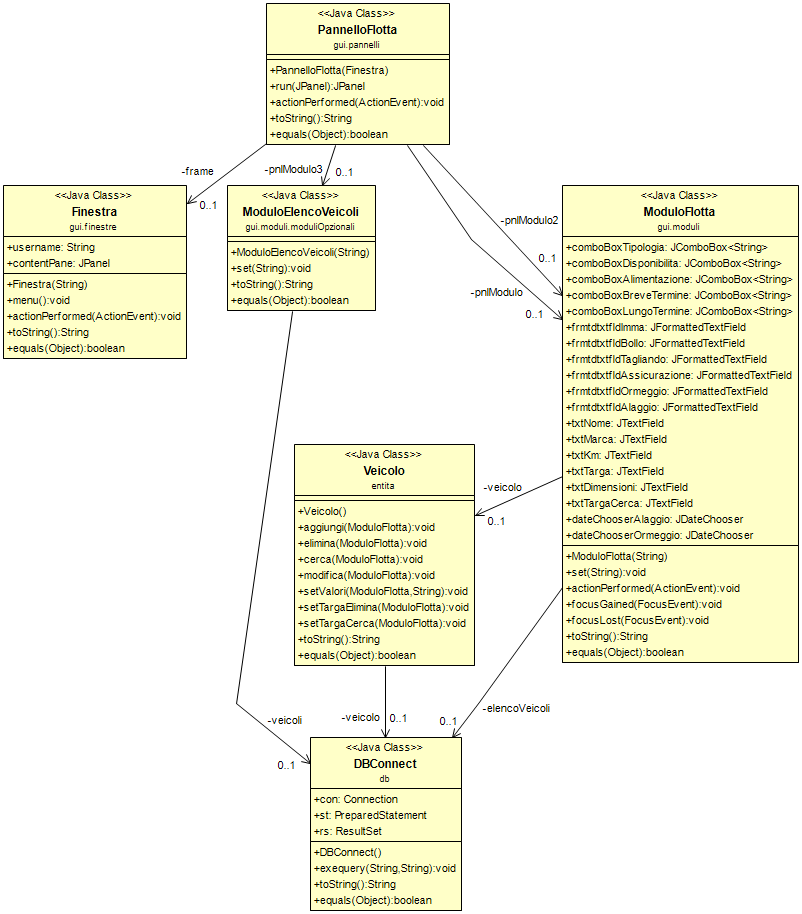


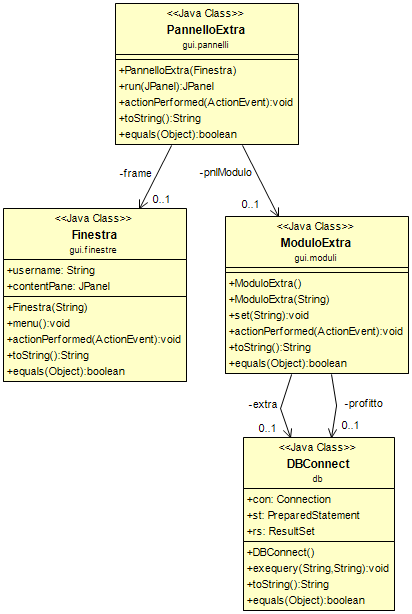


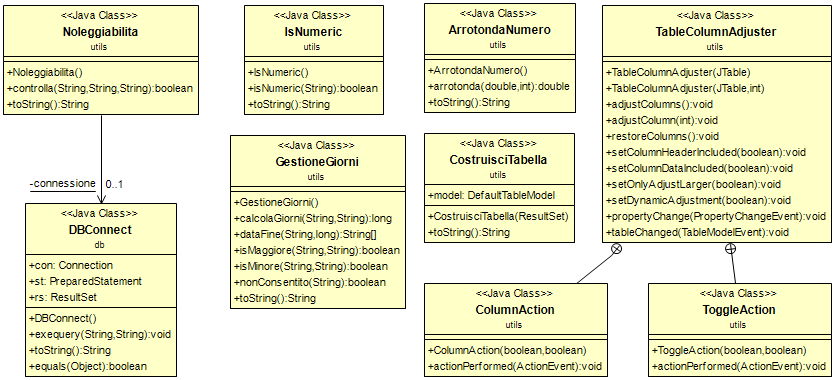


****









# **Implementazione**

A questo punto si analizzeranno nel dettaglio le classi contenute in ciascuno dei package.

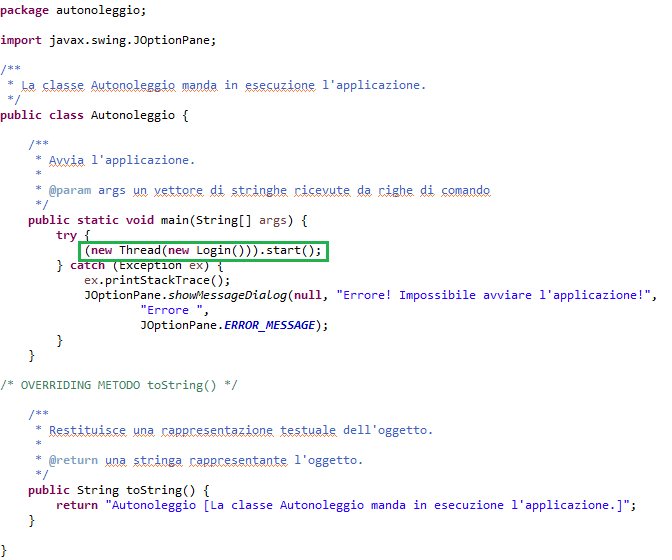
## Package “autonoleggio”

Il package autonoleggio contiene la classe per avviare l’applicazione e la classe per gestire il login.

C:\Users\Lorenzo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Cattura.png

### Autonoleggio

La classe Autonoleggio contiene l’implementazione del metodo main.



La riga di codice evidenziata merita una spiegazione più dettagliata.

Viene richiamato il costruttore public Thread (Runnable target) il quale prende come parametro l’oggetto il cui metodo run verrà richiamato quando verrà mandato in esecuzione il thread.

Nello specifico viene allocato un oggetto di tipo Thread a partire da un oggetto di tipo Login; questo è possibile in quanto la classe Login implementa l’interfaccia Runnable.

Infine viene richiamato il metodo start del thread, il quale causa l’esecuzione del metodo run dell’oggetto Login.

#### Note

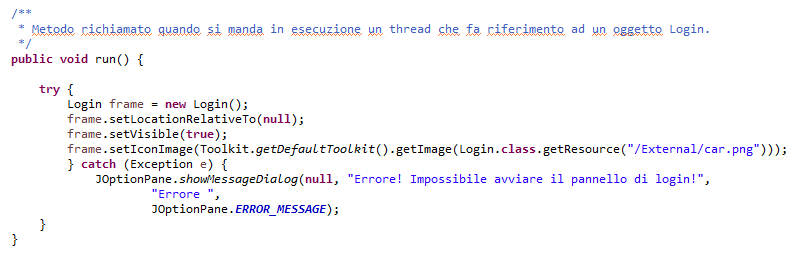
Un’interfaccia è simile ad una classe astratta che dichiara solo metodi astratti senza fornirne un’implementazione: è compito di un’altra classe implementare tutti i suoi metodi astratti.

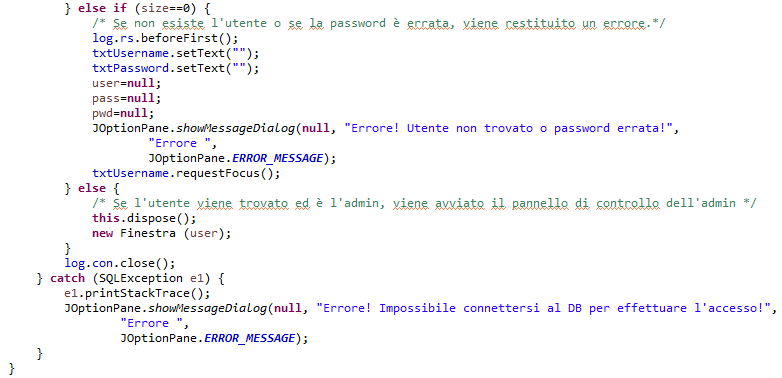
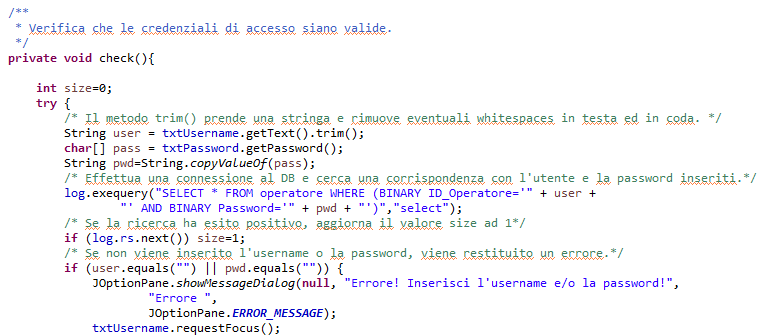
Nel caso analizzato si fa riferimento all’interfaccia Runnable che dichiara soltanto un metodo astratto: il metodo run.

### Login

La classe Login si occupa della completa gestione del login: crea il form per il login e verifica che le credenziali di accesso inserite siano valide.

Il metodo run a cui si è fatto riferimento in precedenza.



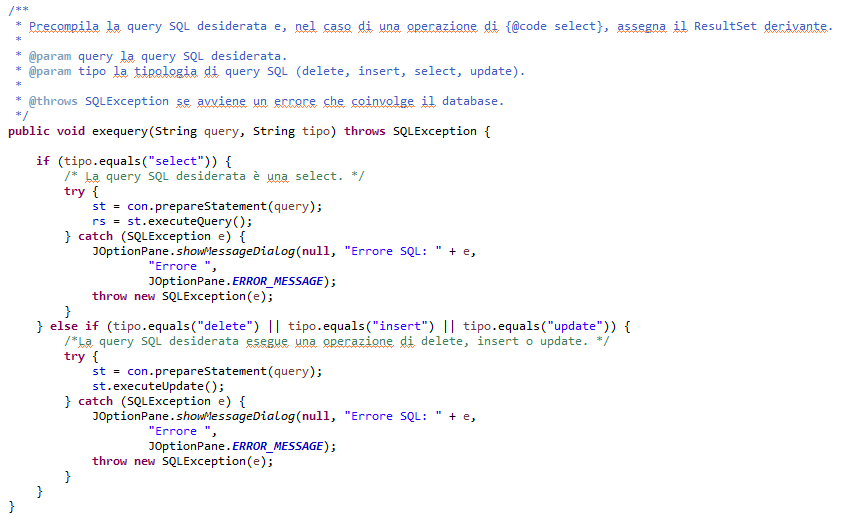
Il metodo check verifica che le credenziali d’accesso inserite nel form per il login siano valide.   
  
  


## Package “db”

Il package db contiene la classe per effettuare la connessione al database.

### DBConnect

La classe DBConnect stabilisce la connessione con il database e precompila le query SQL.



Il metodo exequery prende due stringhe come parametri: la prima rappresenta la query da precompilare; la seconda specifica il tipo di operazione eseguito dalla query.

Nel caso di una operazione di select:

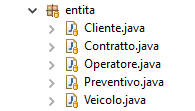
1. la query passata come argomento viene precompilata e memorizzata nella variabile st di tipo PreparedStatement
2. viene eseguita la query precompilata e viene memorizzato il relativo resultSet nella variabile rs di tipo ResultSet

Nel caso di una operazione di insert, delete o update (operazioni che non generano un resultSet):

1. la query passata come argomento viene precompilata e memorizzata nella variabile st di tipo PreparedStatement
2. viene eseguita la query precompilata

## Package “entita”

Il package entita contiene le classi (Cliente, Contratto, Operatore, Veicolo) che mappano le entità presenti nel database e la classe che si occupa del calcolo del preventivo.

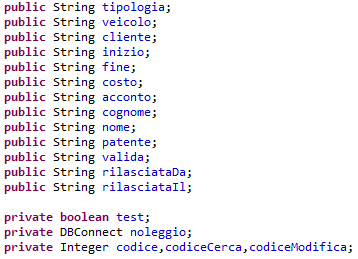


Le classi che mappano le entità presenti nel database seguono tutte la stessa logica implementativa; per questo motivo viene analizzata soltanto una di esse: la classe Contratto.

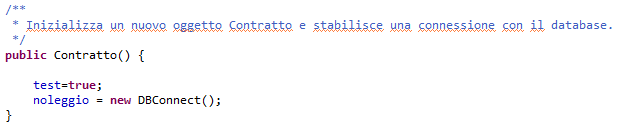
Viceversa la classe Preventivo merita una trattazione separata.

### Contratto

Le proprietà del contratto rispecchiano gli attributi dell’entità Contratto presente nel database.

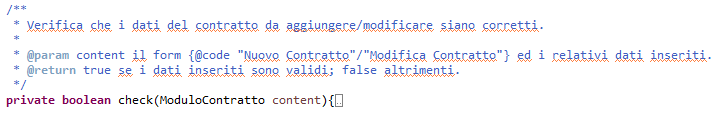


Il costruttore ha il solo compito di inizializzare una variabile di controllo e stabilire una connessione con il database. Entrambe saranno utilizzate dai metodi della classe.

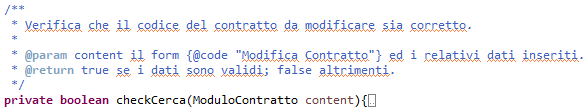


Tra i metodi della classe Contratto alcuni sono dedicati alla verifica della correttezza dei dati immessi nei diversi form dell’applicazione.

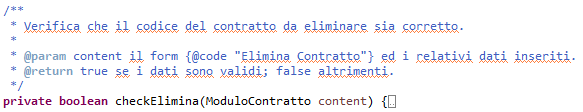
Il metodo check verifica che i dati del contratto da aggiungere/modificare siano corretti. In particolare il metodo controlla che siano rispettati i vari formati e che siano compilati tutti i campi obbligatori.



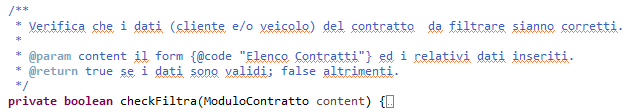
Il metodo checkCerca verifica che il codice del contratto da modificare sia corretto.



Il metodo checkElimina verifica che il codice del contratto da eliminare sia corretto.

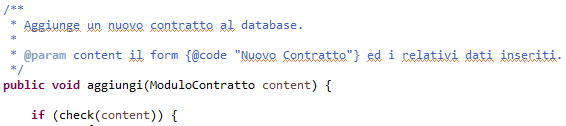


Il metodo checkFiltra verifica che il Codice Fiscale (o la Partita IVA) del cliente e/o la Targa del veicolo associati al contratto da filtrare siano corretti.

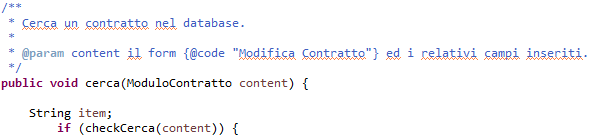


Altri metodi svolgono le principali funzioni relative all’entità Contratto, coinvolgendo direttamente il database nelle loro operazioni.

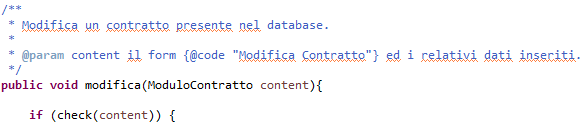
Il metodo aggiungi verifica che il cliente e il veicolo da noleggiare siano presenti nel database. In caso affermativo, inserisce il nuovo contratto di noleggio.



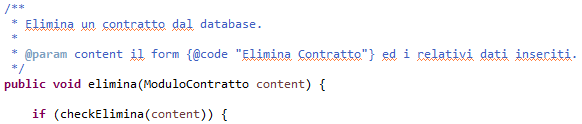
Il metodo cerca verifica se all’interno del database esiste un contratto con il codice di noleggio desiderato.



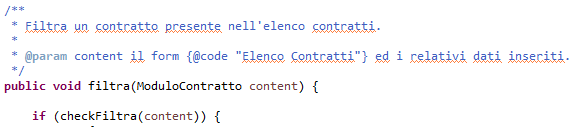
Il metodo modifica aggiorna i dati relativi ad un contratto contenuto nel database.



Il metodo elimina rimuove dal database il contratto con il codice di noleggio desiderato.

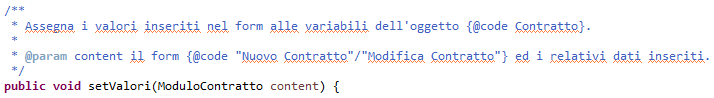


Il metodo filtra cerca i contratti di noleggio relativi al cliente e/o al veicolo desiderato.

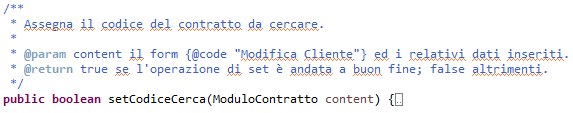


I metodi set consentono di impostare i valori delle proprietà dall’esterno della classe.

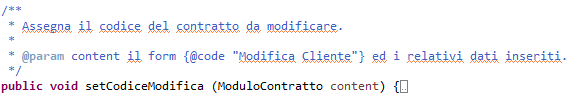
Il metodo setValori assegna i valori inseriti nel form alle variabili dell’oggetto Contratto.



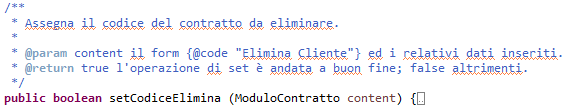
Il metodo setCodiceCerca assegna il codice del contratto da cercare alla variabile codiceCerca.



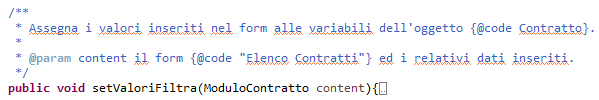
Il metodo setCodiceModifica assegna il codice del contratto da modificare alla variabile codiceModifica.

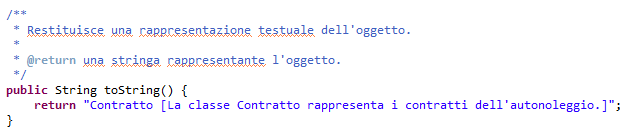


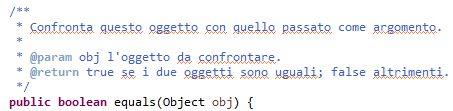
Il metodo setCodiceElimina assegna il codice del contratto da eliminare alla variabile codiceElimina.



Il metodo setValoriFiltra assegna i valori inseriti nel form alle variabili cliente e veicolo.

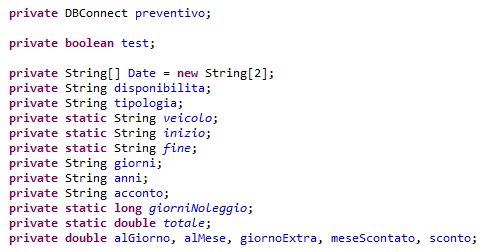
Infine sono presenti i metodi che sovrascrivono l’equals ed il toString ereditati dalla classe Object.



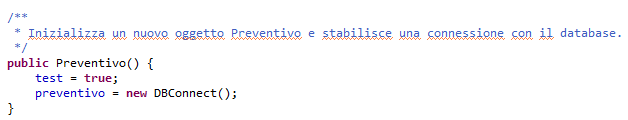


### Preventivo

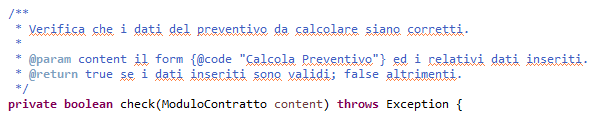
Il preventivo possiede le seguenti proprietà.



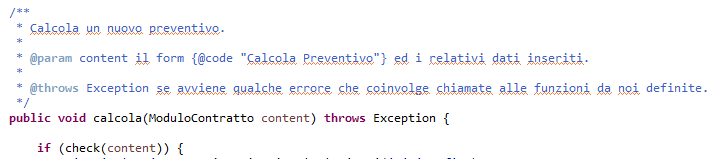
Il costruttore ha il solo compito di inizializzare una variabile di controllo e stabilire una connessione con il database. Entrambe saranno utilizzate dai metodi della classe.



Il metodo check verifica che i dati del preventivo siano corretti. In particolare il metodo controlla che siano rispettati i vari formati e che siano compilati tutti i campi. Inoltre esso richiama il metodo controlla della classe Noleggiabilita, per verificare se il veicolo desiderato è noleggiato per il periodo indicato.

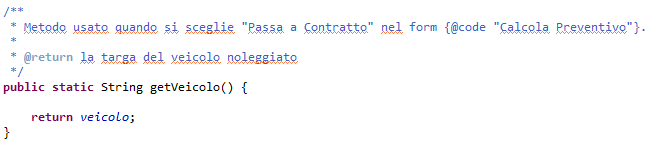


Il metodo calcola richiama il metodo calcolaGiorni della classe GestioneGiorni per calcolare il numero di giorni di noleggio. Inoltre calcola il costo totale del noleggio, tenendo conto di eventuali sovrapprezzi per kilometraggio illimitato e conducente under 25.

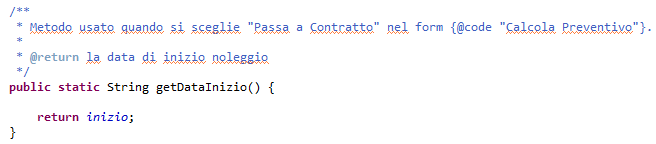


I metodi get consentono di leggere i valori delle proprietà dall’esterno della classe.

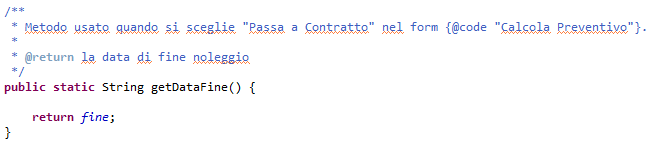
Il metodo getVeicolo restituisce il valore della variabile veicolo.



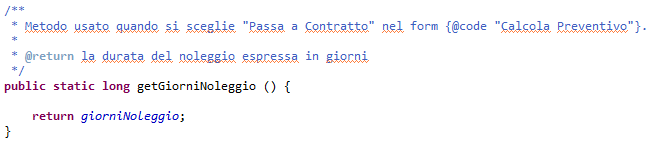
Il metodo getDataInizio restituisce il valore della variabile inizio.



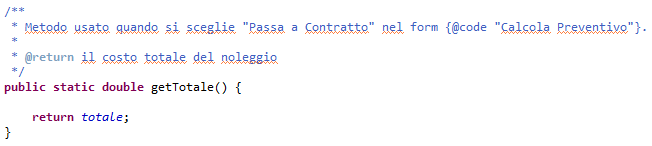
Il metodo getDataFine restituisce il valore della variabile fine.



Il metodo getGiorniNoleggio restituisce il valore della variabile giorniNoleggio.

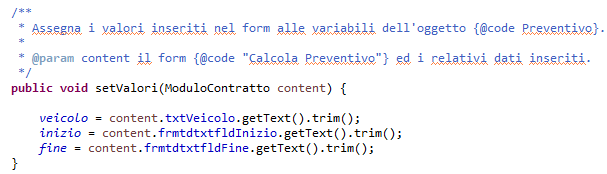


Il metodo getTotale restituisce il valore della variabile totale.

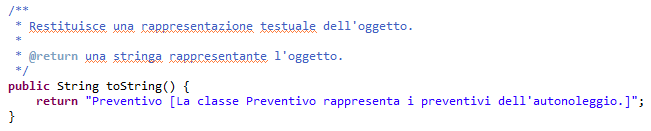


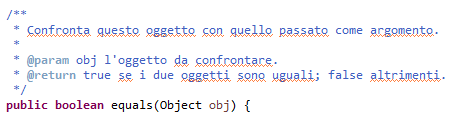
I metodi set consentono di impostare i valori delle proprietà dall’esterno della classe.

Il metodo setValori assegna i valori inseriti nel form alle variabili dell’oggetto Preventivo.



Infine sono presenti i metodi che sovrascrivono l’equals ed il toString ereditati dalla classe Object.





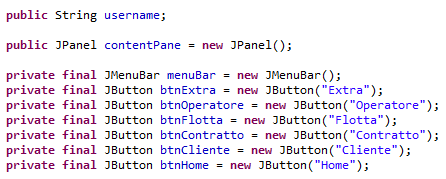
## Package “finestre”

Il package finestre contiene la classe che implementa il frame dell’applicazione.

### Finestra

La classe Finestra costruisce il frame dell’applicazione e crea i pulsanti che costituiscono il menu principale.

La classe Finestra presenta le seguenti proprietà.



La variabile username conterrà l’username dell’utente loggato.

La variabile contentPane di tipo JPanel costituisce il pannello contenuto all’interno del frame dell’applicazione.

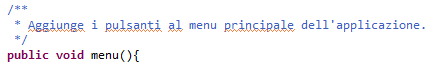
La variabile menuBar di tipo JMenuBar costituisce il menu principale dell’applicazione.

Le restanti variabili di tipo JButton costituiscono i pulsanti contenuti nel menu.

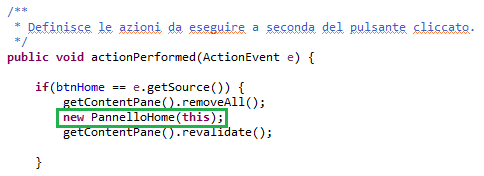
Il costruttore inizializza il frame dell’applicazione e memorizza l’username dell’utente nella variabile username. Inoltre richiama il metodo menu ed il costruttore della classe PannelloHome passando come parametro l’oggetto di tipo Finestra appena inizializzato.



Il metodo menu aggiunge i pulsanti al menu principale dell’applicazione. In particolare disabilita i pulsanti btnFlotta, btnOperatore e btnExtra se l’utente loggato non è l’admin.

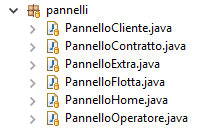


Il metodo actionPerformed richiama il costruttore del generico pannello a seconda del pulsante cliccato. Nella figura sottostante viene mostrato il caso del pulsante “Home”.



## Package “pannelli”

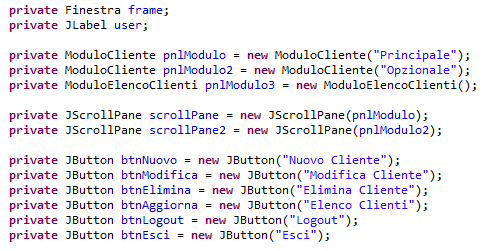
Il package Pannelli contiene le classi che modificano opportunamente il pannello contentPane contenuto nel frame dell’applicazione.



Le classi che modificano il pannello contentPane seguono tutte la stessa logica implementativa; per questo motivo viene analizzata soltanto una di esse: la classe PannelloCliente.

### PannelloCliente

La classe PannelloCliente modifica il pannello contentPane inserendo tutti i componenti grafici relativi alla sezione cliente dell’applicazione.  
  
La classe PannelloCliente presenta le seguenti proprietà.



La variabile frame di tipo Finestra conterrà il frame dell’applicazione.

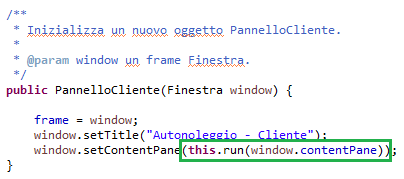
La variabile user di tipo JLabel costituisce l’etichetta che conterrà l’username dell’utente loggato.

Le variabili di tipo JScrollPane costituiscono i “pannelli scorrevoli” contenuti nel contentPane.

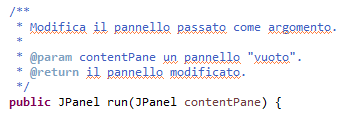
Le variabili pnlModulo, pnlModulo2 e pnlModulo3 costituiscono i moduli intercambiabili che saranno opportunamente caricati all’interno dei “pannelli scorrevoli”.

Le restanti variabili di tipo JButton costituiscono i pulsanti contenuti nel sottomenu cliente, il pulsante per eseguire il logout e quello per uscire dall’applicazione.

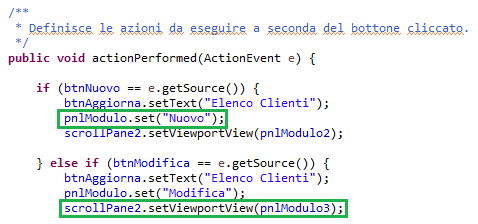
Il costruttore inizializza un nuovo oggetto PannelloCliente copiando il riferimento della variabile window passata come argomento nella variabile frame; a questo punto le due variabili fanno riferimento allo stesso oggetto ed è indifferente lavorare sull’una o sull’altra variabile. Inoltre il costruttore imposta il titolo del frame e richiama il metodo run sull’oggetto PannelloCliente appena inizializzato, passando come parametro il pannello contentPane contenuto all’interno del frame dell’applicazione.



Il metodo run prende come parametro un pannello e lo restituisce dopo averlo opportunamente modificato.



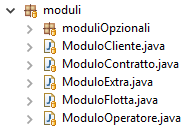
Il metodo actionPerformed definisce le azioni da eseguire a seconda del pulsante cliccato. Nella figura sottostante viene mostrato il caso dei pulsanti “Nuovo Cliente” e “Modifica Cliente”.



Innanzitutto viene impostato il testo del pulsante btnAggiorna. Viene poi richiamato il metodo set dell’oggetto pnlModulo per modificare opportunamente il modulo principale. Infine viene cambiato il modulo visualizzato nel “pannello scorrevole” scrollPane2.

## Package “moduli”

Il package moduli contiene le classi che implementano i moduli che vengono caricati nei “pannelli scorrevoli” dell’applicazione.

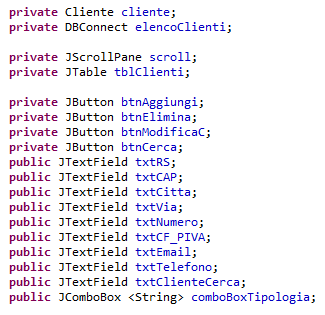


Le classi visibili nell’elenco implementano i moduli principali. Esse seguono tutte la stessa logica implementativa; per questo motivo viene analizzata soltanto una di esse: la classe ModuloCliente.

Il package moduliOpzionali contiene le classi che implementano i moduli opzionali. Esse seguono una logica implementativa differente e quindi meritano una trattazione separata.

### ModuloCliente

La classe ModuloCliente presenta le seguenti proprietà.



La variabile cliente contiene il riferimento ad un oggetto di tipo Cliente.

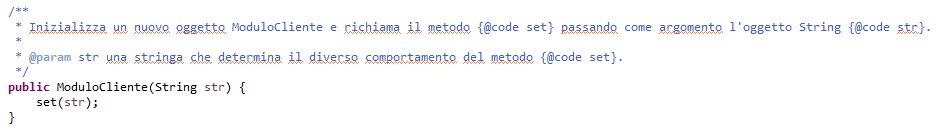
La variabile elencoClienti contiene il riferimento ad un oggetto di tipo DBConnect.

La variabile tblClienti di tipo JTable costituisce la tabella che conterrà l’elenco dei clienti.

La variabile scroll di tipo JScrollPane costituisce il “pannello scorrevole” che conterrà la tabella tblClienti.

Le restanti variabili costituiscono gli elementi grafici presenti all’interno del modulo.

Il costruttore inizializza un nuovo oggetto ModuloCliente e richiama il metodo set con la stringa str passata come argomento.



Il metodo set modifica opportunamente il modulo a seconda della stringa str passata come argomento. In particolare se la stringa risulta:

* “Principale”, viene creato un modulo vuoto recante la scritta “Modulo Principale”
* “Opzionale”, viene creato un modulo vuoto recante la scritta “Modulo Opzionale”
* “Nuovo”, viene creato il form per aggiungere un cliente
* “Modifica”, viene creato il form per modificare un cliente
* “Elimina”, viene creato il form per eliminare un cliente
* “Elenca”, viene generato l’elenco dei clienti contenuti nel database

In particolare nella figura sottostante viene riportato il caso in cui la stringa risulta “Elenca”.

Innanzitutto viene istanziato un nuovo oggetto DBConnect e viene assegnato il suo riferimento alla variabile elencoClienti.

Viene poi eseguita una operazione di select sulla tabella cliente, richiamando il metodo exequery dell’oggetto elencoClienti.

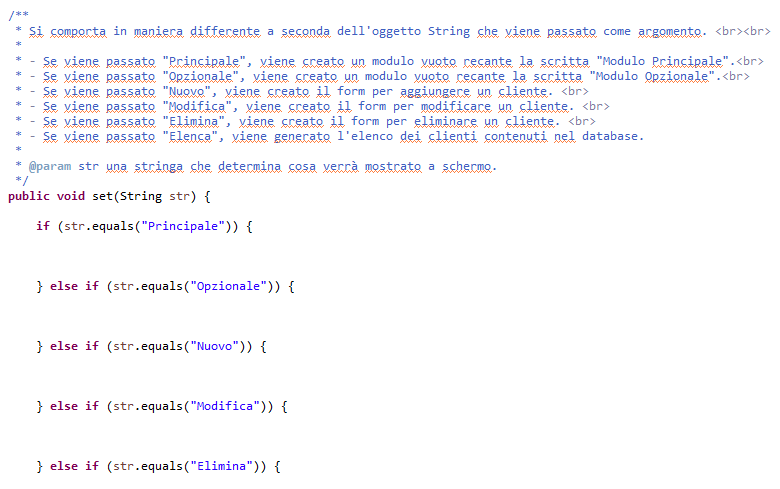
Viene creata la tabella tblClienti contente l’elenco dei clienti, passando il ResultSet della query al costruttore della classe CostruisciTabella.

Viene creato l’oggetto tca di tipo TableColumnAdjuster, passando la tabella tblClienti al costruttore della classe.

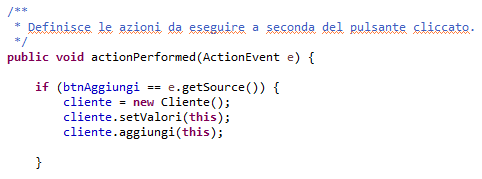
Viene richiamato il metodo adjustColumns dell’oggetto tca per ridimensionare in maniera opportuna la tabella.

Viene inserita la tabella tblClienti all’interno del “pannello scorrevole” scroll.

Infine viene chiusa la connessione al database.



Il metodo actionPerformed definisce le azioni da eseguire a seconda del pulsante cliccato. Nella figura sottostante viene mostrato il caso del pulsante “Aggiungi”.



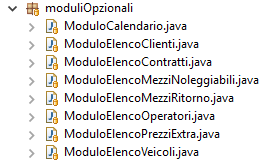
Innanzitutto viene istanziato un nuovo oggetto Cliente e viene assegnato il suo riferimento alla variabile cliente.

Viene richiamato il metodo setValori dell’oggetto cliente, passando come argomento il modulo.

Infine viene richiamato il metodo aggiungi dell’oggetto cliente, passando come argomento il modulo.

### Package “moduliOpzionali”

Il package moduliOpzionali contiene le classi che implementano i moduli opzionali.



Le classi del tipo ModuloElencoXXX seguono tutte la stessa logica implementativa; per questo motivo viene analizzata soltanto una di esse: la classe ModuloElencoClienti.

Viceversa la classe ModuloCalendario merita una trattazione separata.

#### ModuloElencoClienti

La classe ModuloElencoClienti presenta le seguenti proprietà.

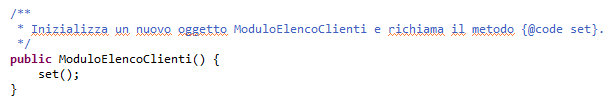
C:\Users\Lorenzo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\proprieta.png

La variabile tblClienti di tipo JTable costituisce la tabella che conterrà la tabella dei clienti.

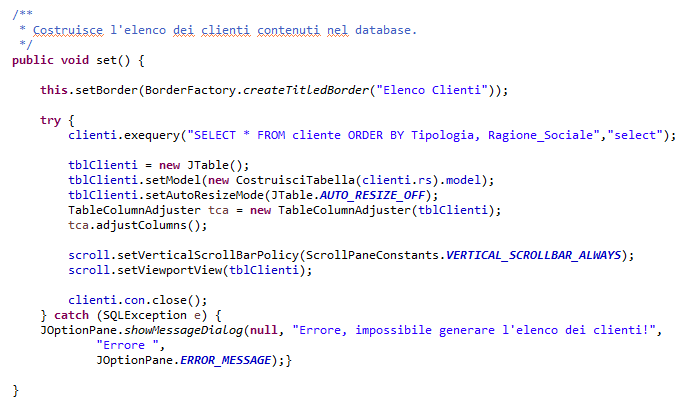
La variabile scroll di tipo JScrollPane costituisce il “pannello scorrevole” che conterrà la tabella tblClienti.

La variabile clienti contiene il riferimento ad un oggetto di tipo DBConnect.

Il costruttore inizializza un nuovo oggetto ModuloElencoClienti e richiama il metodo set.



Il metodo set costruisce l’elenco dei clienti contenuti nel database.



Viene eseguita una operazione di select sulla tabella cliente, richiamando il metodo exequery dell’oggetto clienti.

Viene creata la tabella tblClienti contente l’elenco dei clienti, passando il ResultSet della query al costruttore della classe CostruisciTabella.

Viene creato l’oggetto tca di tipo TableColumnAdjuster, passando la tabella tblClienti al costruttore della classe.

Viene richiamato il metodo adjustColumns dell’oggetto tca per ridimensionare in maniera opportuna la tabella.

Viene inserita la tabella tblClienti all’interno del “pannello scorrevole” scroll.

Infine viene chiusa la connessione al database.

#### ModuloCalendario

La classe ModuloCalendario presenta le seguenti proprietà.

# Bibliografia

Fowler, M. (2010). *UML Distilled. Guida rapida al linguaggio di modellazione standard.* Pearson.