Università Politecnica delle Marche – Facoltà di Ingegneria

INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

OOP1617Gruppo08

APPLICAZIONE PER LA GESTIONE DI UN AUTONOLEGGIO

RELAZIONE

ABBADINI LORENZO, COMPAGNONI PAOLO, SCISCI VALERIO

Sommario

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.

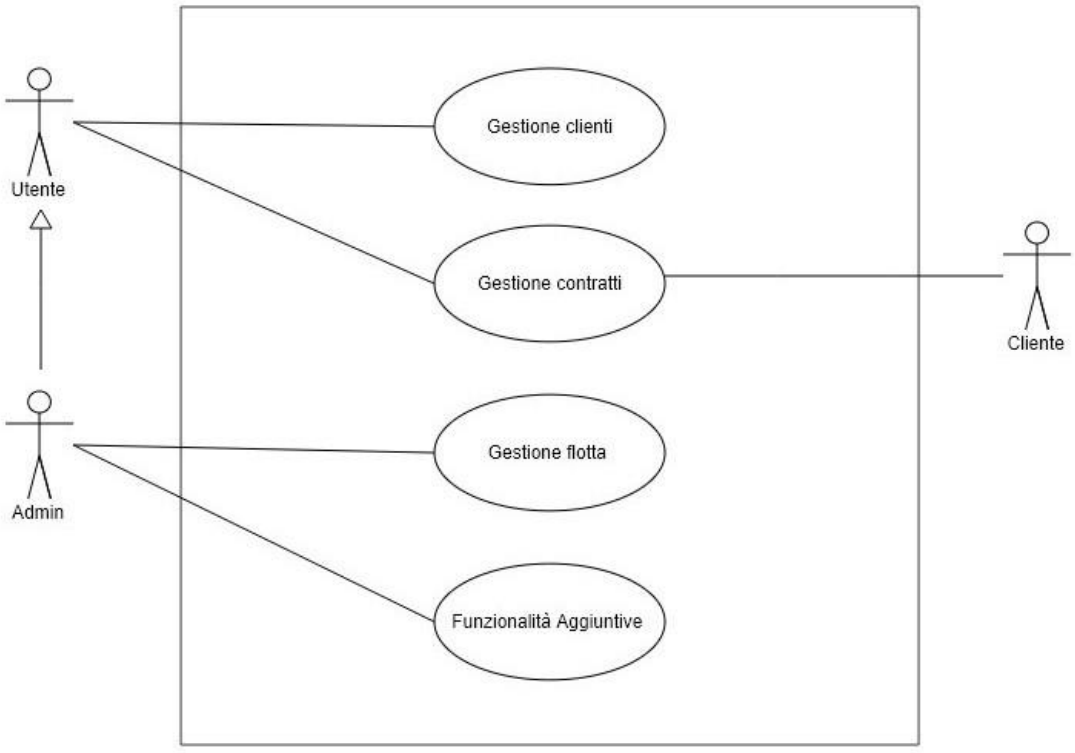
Gli attori del diagramma dei casi d'uso sono:

- Amministratore
- Cliente
- Utente standard

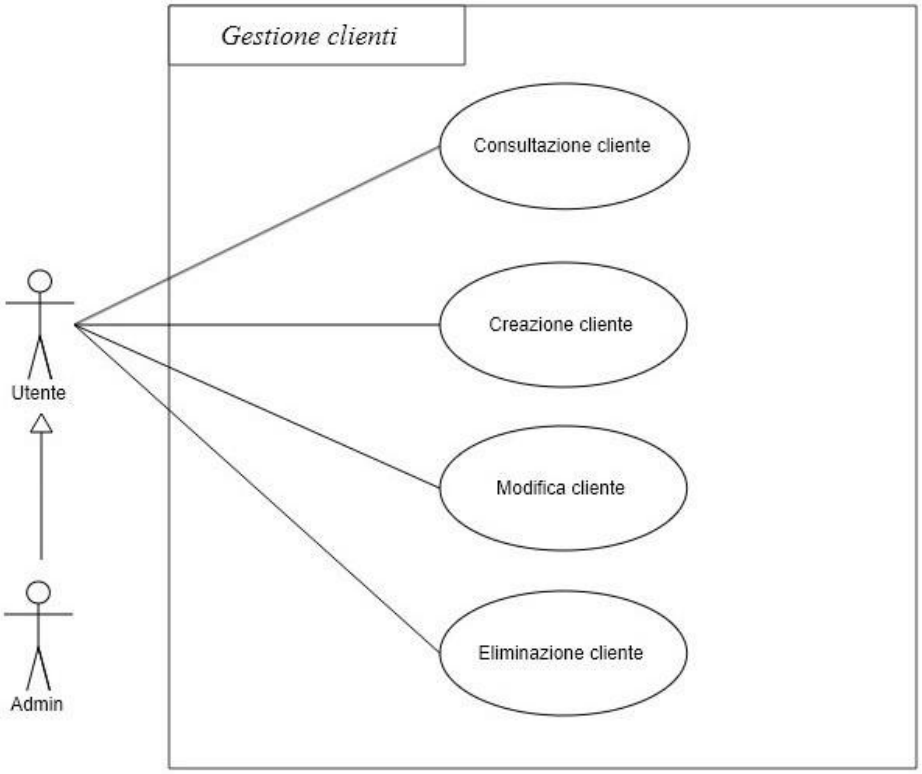
L'amministratore è il titolare dell'attività. I casi d'uso ad esso associati rappresentano le principali funzionalità fornite dall'applicazione.

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

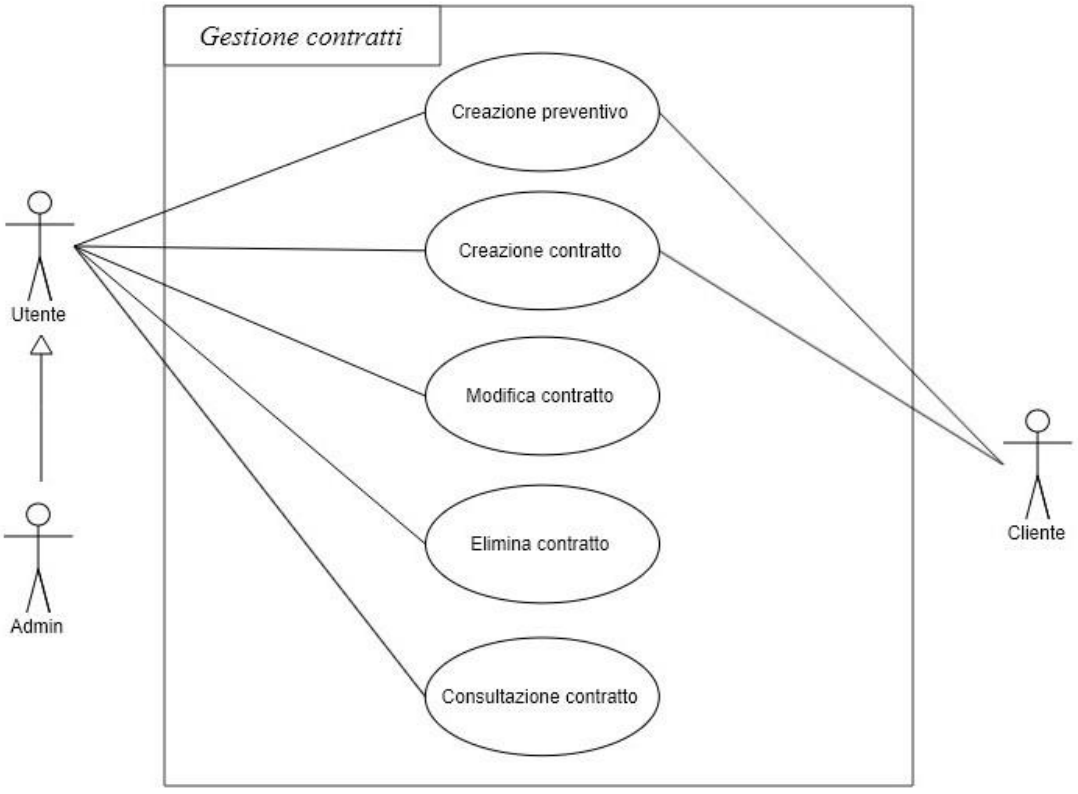
Al cliente è associato un solo caso d'uso. Come evidenziato nella sezione Gestione contratti, esso può soltanto richiedere un preventivo o stipulare un contratto di noleggio.



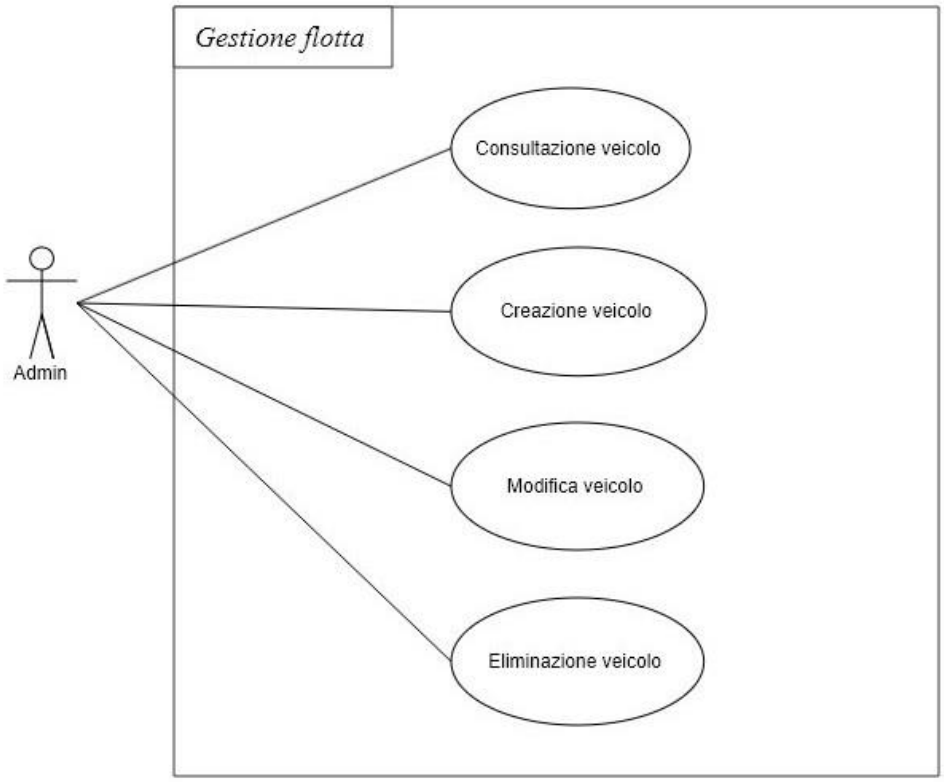
Gestione clienti



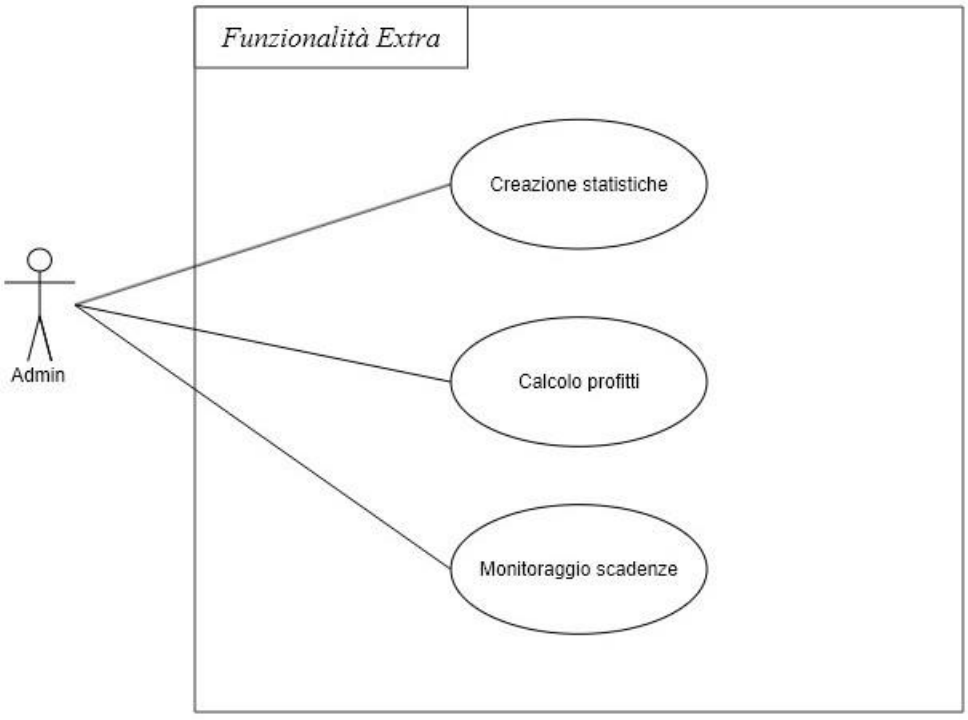
Gestione contratti



Gestione flotta



Funzionalità extra



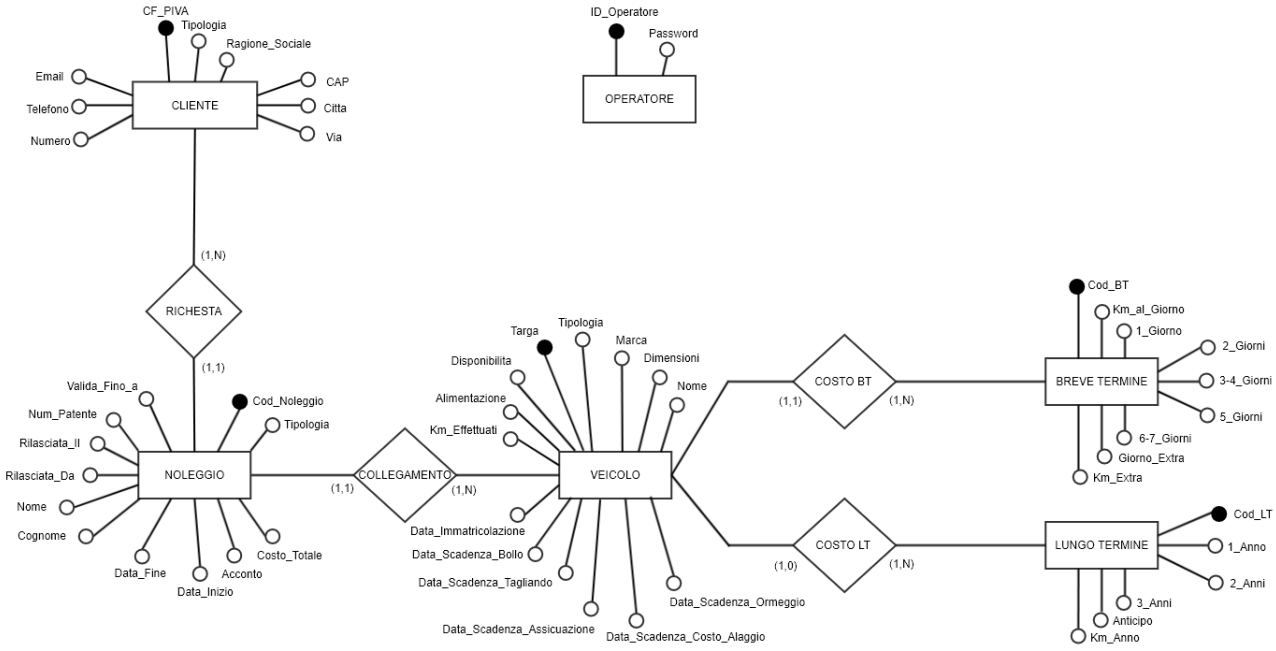
Progettazione del database

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.

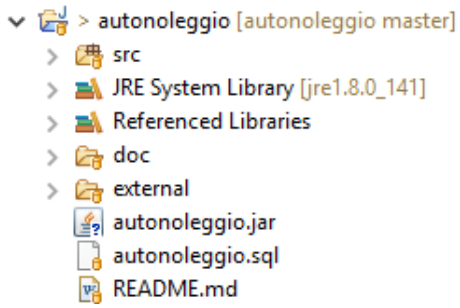
L’entità Breve Termine rappresenta il tariffario per i noleggi a breve termine.

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.

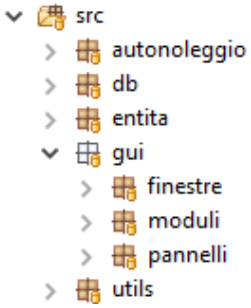
Struttura del progetto



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"



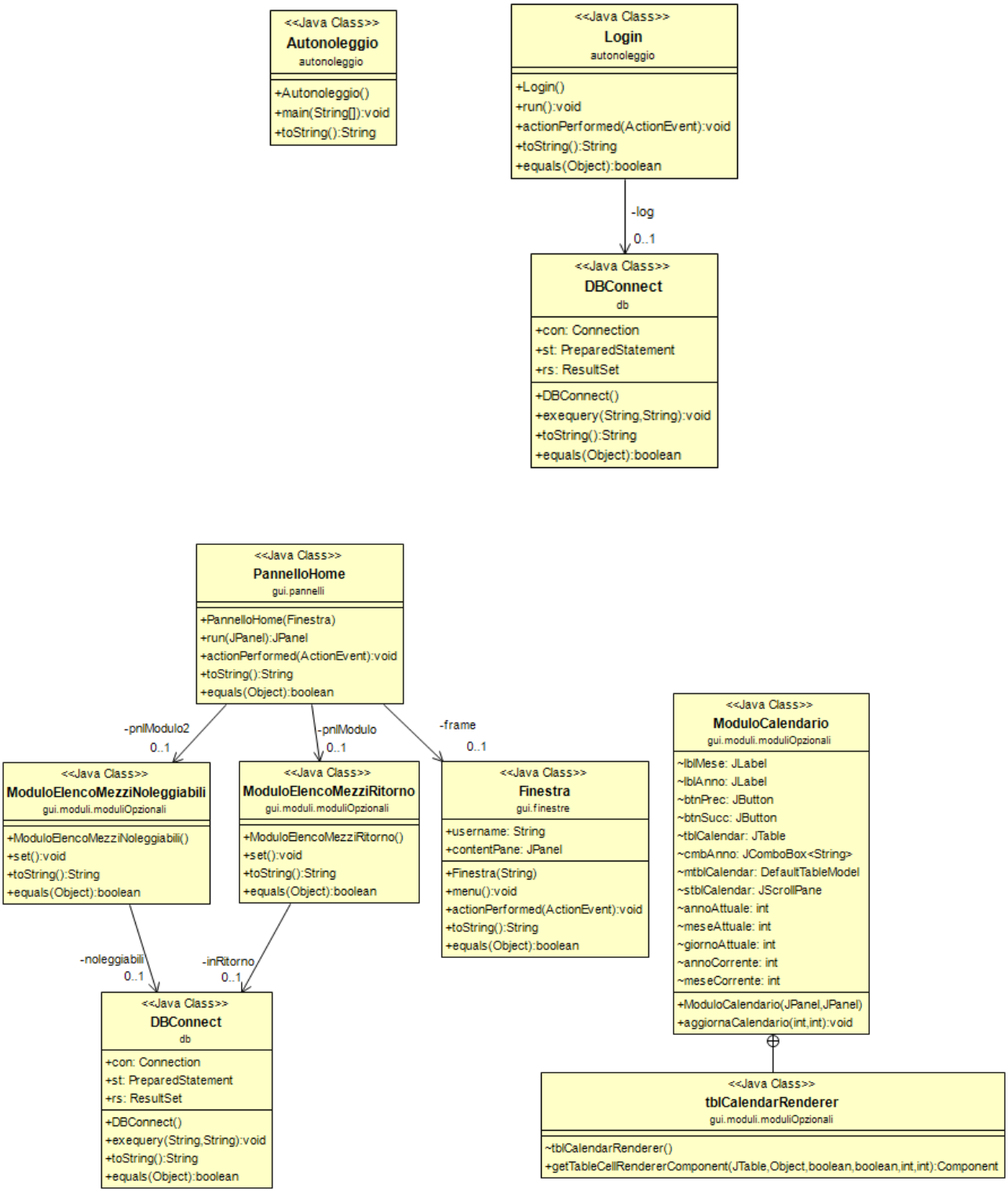
La cartella src contiene tutto il codice sorgente organizzato in package:

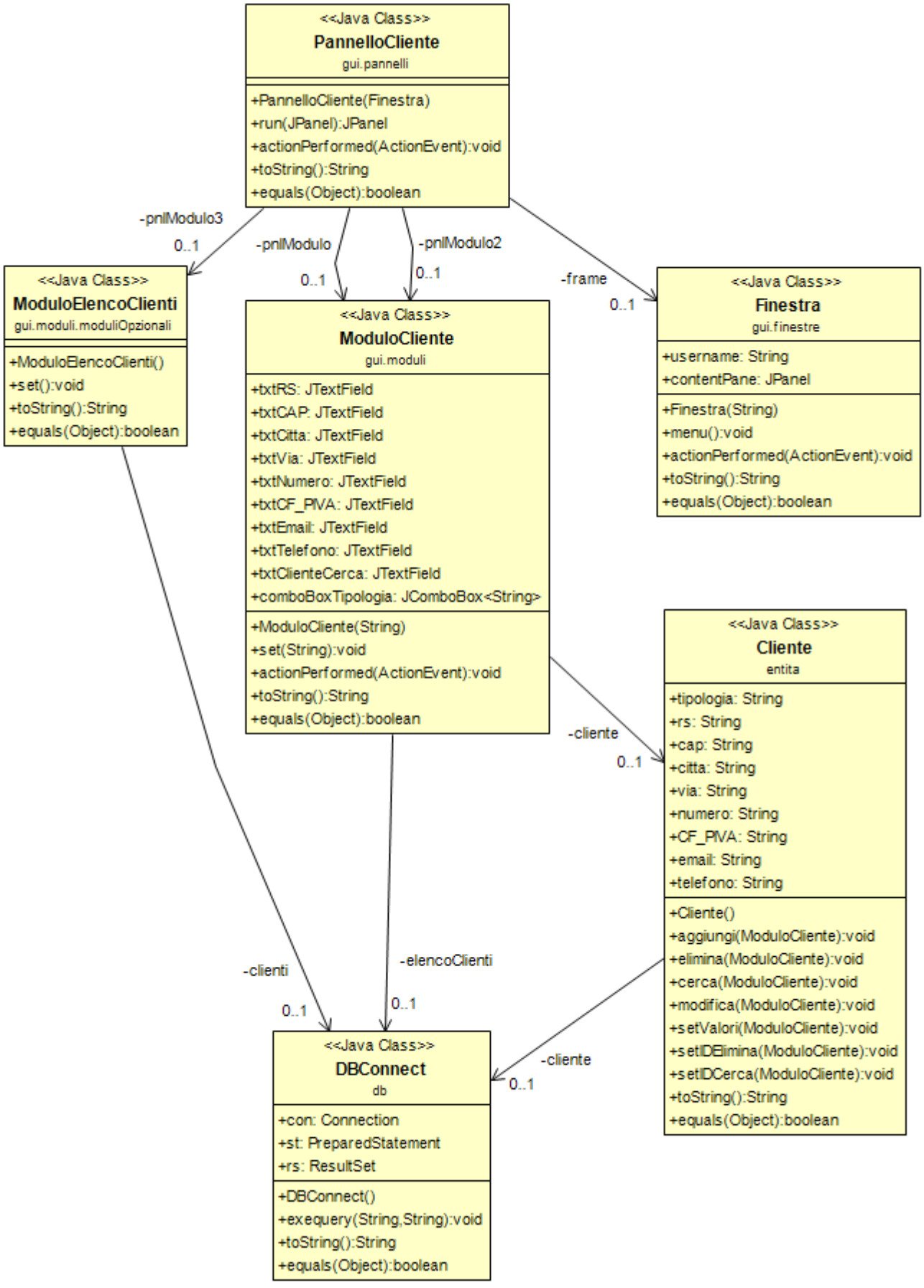
- autonoleggio contiene la classe necessaria per avviare l'applicazione e la classe per il login
- db contiene la classe necessaria per effettuare la connessione al database
- entita contiene le classi che mappano le entità del database
- gui contiene le classi che implementano l'interfaccia grafica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

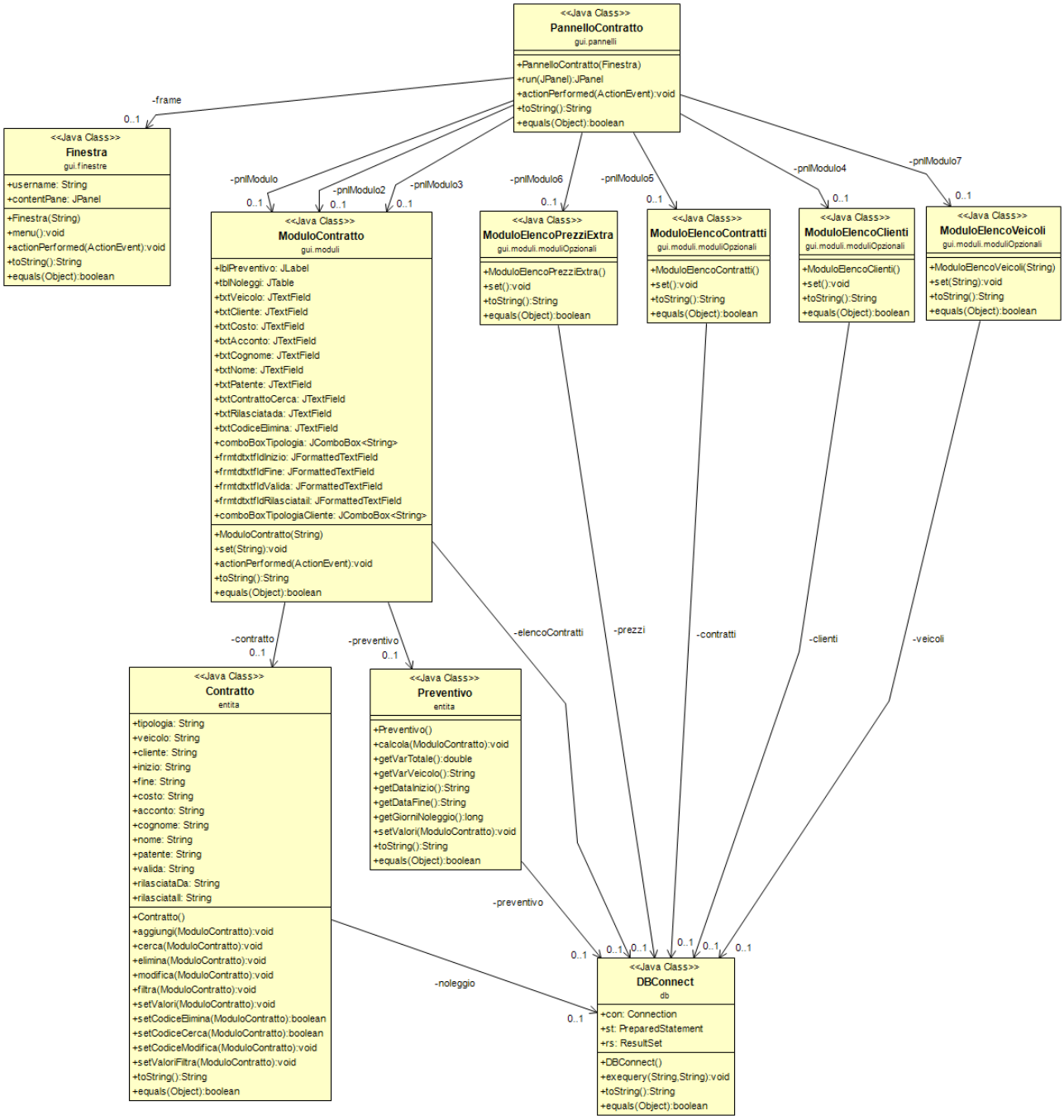
Diagramma delle classi

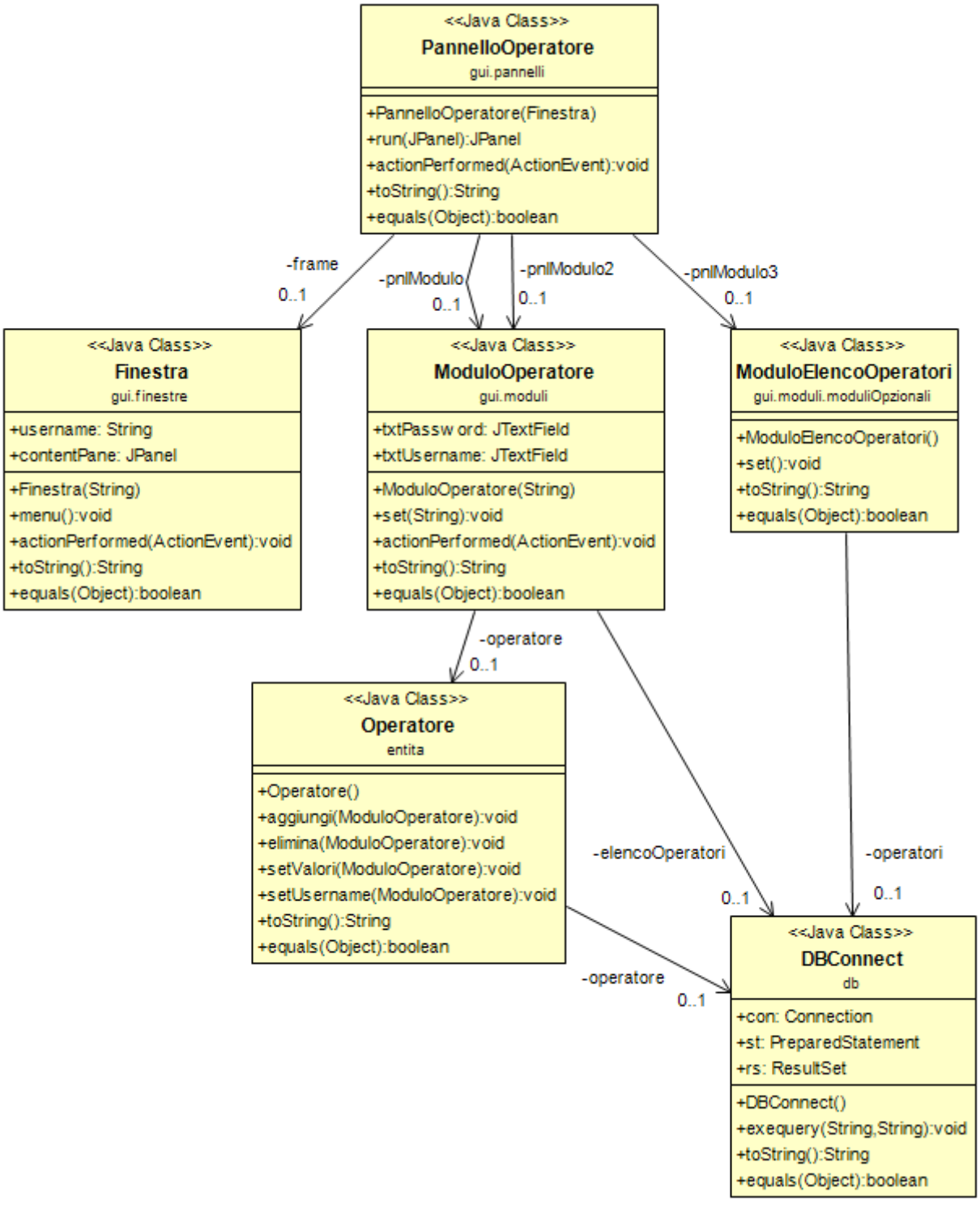
Il diagramma descrive le caratteristiche delle classi che compongono l'applicazione e le relazioni statiche esistenti tra di loro.

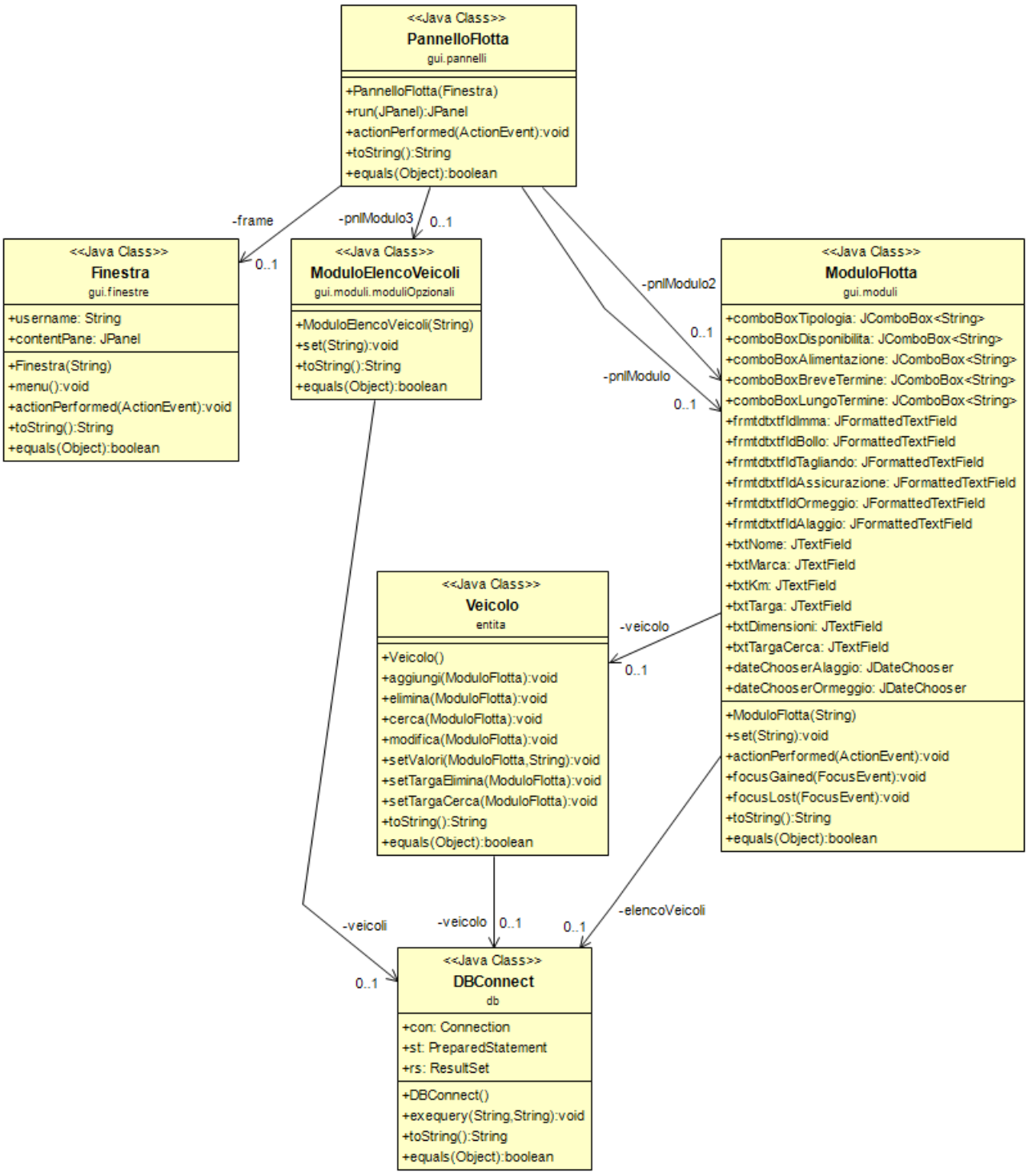
Come fatto per il diagramma dei casi d'uso, si è scelto di suddividere il diagramma delle classi in più parti, in modo tale da facilitarne la lettura.

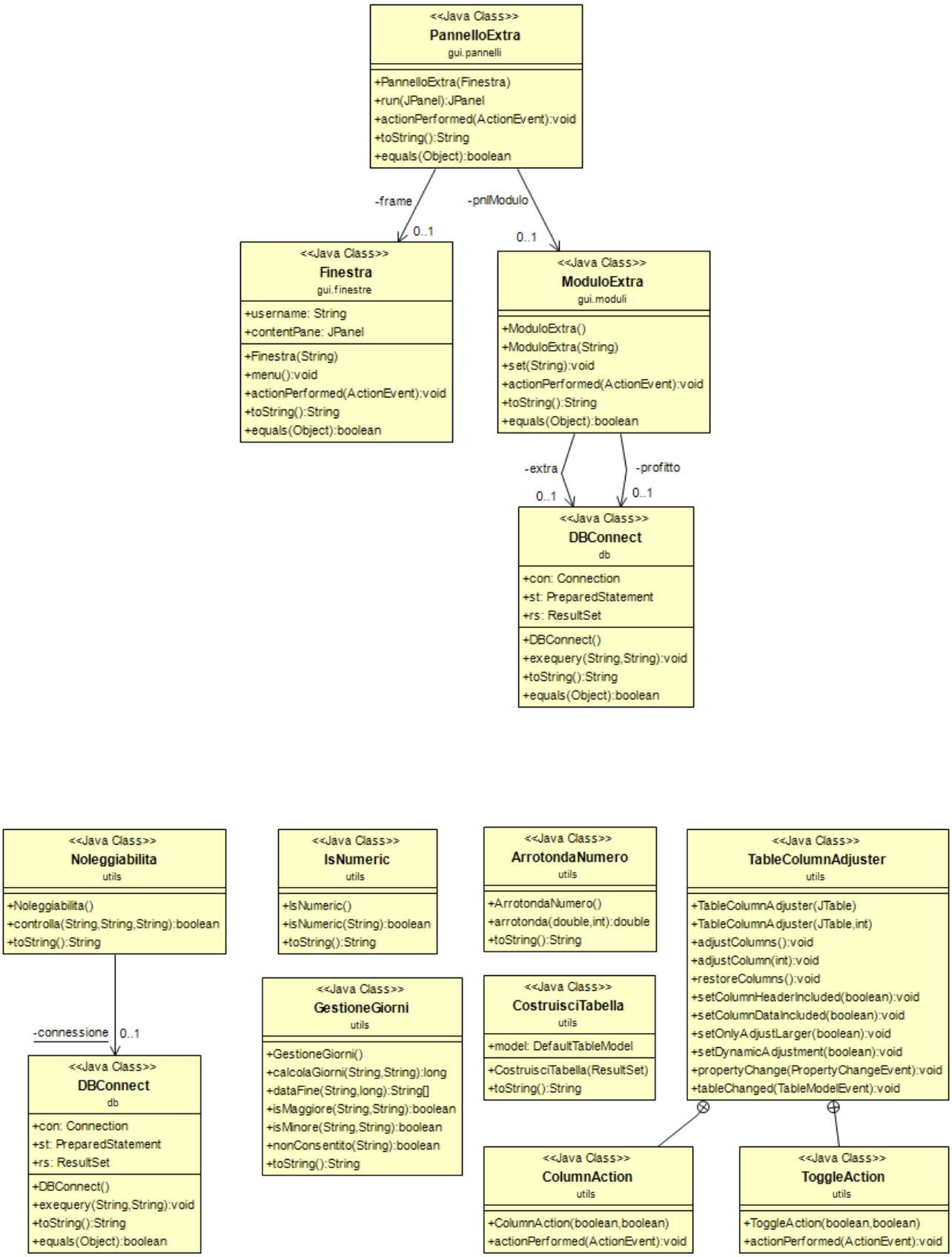



<<Java Class>>
PannelloCliente
gui.pannelli
+PannelloCliente(Finestra)
+run(JPanel):JPanel
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
-pnlModulo3
0..1
-pnlModulo
0..1
-pnlModulo2
0..1
-frame
0..1
<<Java Class>>
ModuloElencoClienti
gui.moduli.moduliOpzionali
+ModuloElencoClienti()
+set():void
+toString():String
+equals(Object):boolean
<<Java Class>>
ModuloCliente
gui.moduli
+txtRS: JTextField
+txtCAP: JTextField
+txtCitta: JTextField
+txtVia: JTextField
+txtNumero: JTextField
+txtCF_PIVA: JTextField
+txtEmail: JTextField
+txtTelefono: JTextField
+txtClienteCerca: JTextField
+comboBoxTipologia: JComboBox<String>
+ModuloCliente(String)
+set(String):void
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
<<Java Class>>
Finestra
gui.finestre
+username: String
+contentPane: JPanel
+Finestra(String)
+menu():void
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
-cliente
0..1
<<Java Class>>
Cliente
entita
+tipologia: String
+rs: String
+cap: String
+citta: String
+via: String
+numero: String
+CF_PIVA: String
+email: String
+telefono: String
+Cliente()
+aggiungi(ModuloCliente):void
+elimina(ModuloCliente):void
+cerca(ModuloCliente):void
+modifica(ModuloCliente):void
+setValori(ModuloCliente):void
+setIDElimina(ModuloCliente):void
+setIDCerca(ModuloCliente):void
+toString():String
+equals(Object):boolean
-clienti
0..1
-elencoClienti
0..1
<<Java Class>>
DBConnect
db
+con: Connection
+st: PreparedStatement
+rs: ResultSet
+DBConnect()
+exequery(String,String):void
+toString():String
+equals(Object):boolean
-cliente
0..1


<<Java Class>>
PannelloContratto
gui.pannelli
+PannelloContratto(Finestra)
+run(JPanel):JPanel
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
-frame
0..1
<<Java Class>>
Finestra
gui.finestre
+username: String
+contentPane: JPanel
+Finestra(String)
+menu():void
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
-pnlModulo
-pnlModulo2
-pnlModulo3
-pnlModulo4
-pnlModulo5
-pnlModulo6
-pnlModulo7
0..1
<<Java Class>>
ModuloContratto
gui.moduli
+lblPreventivo: JLabel
+tblNoleggi: JTable
+txtVeicolo: JTextField
+txtCliente: JTextField
+txtCosto: JTextField
+txtAcconto: JTextField
+txtCognome: JTextField
+txtNome: JTextField
+txtPatente: JTextField
+txtContrattoCerca: JTextField
+txtRilasciatada: JTextField
+txtCodiceElimina: JTextField
+comboBoxTipologia: JComboBox<String>
+frmtdtxtfldInizio: JFormattedTextField
+frmtdtxtfldFine: JFormattedTextField
+frmtdtxtfldValida: JFormattedTextField
+frmtdtxtfldRilasciatail: JFormattedTextField
+comboBoxTipologiaCliente: JComboBox<String>
+ModuloContratto(String)
+set(String):void
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
<<Java Class>>
ModuloElencoPrezziExtra
gui.moduli.moduliOpzionali
+ModuloElencoPrezziExtra()
+set():void
+toString():String
+equals(Object):boolean
<<Java Class>>
ModuloElencoContratti
gui.moduli.moduliOpzionali
+ModuloElencoContratti()
+set():void
+toString():String
+equals(Object):boolean
<<Java Class>>
ModuloElencoClienti
gui.moduli.moduliOpzionali
+ModuloElencoClienti()
+set():void
+toString():String
+equals(Object):boolean
<<Java Class>>
ModuloElencoVeicoli
gui.moduli.moduliOpzionali
+ModuloElencoVeicoli(String)
+set(String):void
+toString():String
+equals(Object):boolean
-contratto
-preventivo
-elencoContratti
-prezzi
-contratti
-clienti
-veicoli
<<Java Class>>
Contratto
entita
+tipologia: String
+veicolo: String
+cliente: String
+inizio: String
+fine: String
+costo: String
+acconto: String
+cognome: String
+nome: String
+patente: String
+valida: String
+rilasciataDa: String
+rilasciatall: String
+Contratto()
+aggiungi(ModuloContratto):void
+cerca(ModuloContratto):void
+elimina(ModuloContratto):void
+modifica(ModuloContratto):void
+filtra(ModuloContratto):void
+setValori(ModuloContratto):void
+setCodiceElimina(ModuloContratto):boolean
+setCodiceCerca(ModuloContratto):boolean
+setCodiceModifica(ModuloContratto):void
+setValoriFiltra(ModuloContratto):void
+toString():String
+equals(Object):boolean
<<Java Class>>
Preventivo
entita
+Preventivo()
+calcola(ModuloContratto):void
+getVarTotale():double
+getVarVeicolo():String
+getDataInizio():String
+getDataFine():String
+getGiorniNoleggio():long
+setValori(ModuloContratto):void
+toString():String
+equals(Object):boolean
-preventivo
-noleggio
<<Java Class>>
DBConnect
db
+con: Connection
+st: PreparedStatement
+rs: ResultSet
+DBConnect()
+exequery(String,String):void
+toString():String
+equals(Object):boolean


<<Java Class>>
PannelloOperatore
gui.pannelli
+PannelloOperatore(Finestra)
+run(JPanel):JPanel
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
-frame
0..1
-pnlModulo
0..1
-pnlModulo2
0..1
-pnlModulo3
0..1
<<Java Class>>
Finestra
gui.finestre
+username: String
+contentPane: JPanel
+Finestra(String)
+menu():void
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
<<Java Class>>
ModuloOperatore
gui.moduli
+txtPassword: JTextField
+txtUsername: JTextField
+ModuloOperatore(String)
+set(String):void
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
<<Java Class>>
ModuloElencoOperatori
gui.moduli.moduliOpzionali
+ModuloElencoOperatori()
+set():void
+toString():String
+equals(Object):boolean
-operatore
0..1
<<Java Class>>
Operatore
entita
+Operatore()
+aggiungi(ModuloOperatore):void
+elimina(ModuloOperatore):void
+setValori(ModuloOperatore):void
+setUsername(ModuloOperatore):void
+toString():String
+equals(Object):boolean
-elencoOperatori
0..1
-operatori
0..1
-operatore
0..1
<<Java Class>>
DBConnect
db
+con: Connection
+st: PreparedStatement
+rs: ResultSet
+DBConnect()
+exequery(String,String):void
+toString():String
+equals(Object):boolean


<<Java Class>>
PannelloFlotta
gui.pannelli
+PannelloFlotta(Finestra)
+run(JPanel):JPanel
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
-frame
0..1
-pnlModulo3
0..1
<<Java Class>>
Finestra
gui.finestre
+username: String
+contentPane: JPanel
+Finestra(String)
+menu():void
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
<<Java Class>>
ModuloElencoVeicoli
gui.moduli.moduliOpzionali
+ModuloElencoVeicoli(String)
+set(String):void
+toString():String
+equals(Object):boolean
-pnlModulo2
0..1
-pnlModulo
0..1
<<Java Class>>
ModuloFlotta
gui.moduli
+comboBoxTipologia: JComboBox<String>
+comboBoxDisponibilita: JComboBox<String>
+comboBoxAlimentazione: JComboBox<String>
+comboBoxBreveTermine: JComboBox<String>
+comboBoxLungoTermine: JComboBox<String>
+frmtdtxtfldImma: JFormattedTextField
+frmtdtxtfldBollo: JFormattedTextField
+frmtdtxtfldTagliando: JFormattedTextField
+frmtdtxtfldAssicurazione: JFormattedTextField
+frmtdtxtfldOrmeggio: JFormattedTextField
+frmtdtxtfldAlaggio: JFormattedTextField
+txtNome: JTextField
+txtMarca: JTextField
+txtKm: JTextField
+txtTarga: JTextField
+txtDimensioni: JTextField
+txtTargaCerca: JTextField
+dateChooserAlaggio: JDateChooser
+dateChooserOrmeggio: JDateChooser
+ModuloFlotta(String)
+set(String):void
+actionPerformed(ActionEvent):void
+focusGained(FocusEvent):void
+focusLost(FocusEvent):void
+toString():String
+equals(Object):boolean
<<Java Class>>
Veicolo
entita
+Veicolo()
+aggiungi(ModuloFlotta):void
+elimina(ModuloFlotta):void
+cerca(ModuloFlotta):void
+modifica(ModuloFlotta):void
+setValori(ModuloFlotta,String):void
+setTargaElimina(ModuloFlotta):void
+setTargaCerca(ModuloFlotta):void
+toString():String
+equals(Object):boolean
-veicolo
0..1
-veicoli
0..1
-veicolo
0..1
0..1
-elencoVeicoli
<<Java Class>>
DBConnect
db
+con: Connection
+st: PreparedStatement
+rs: ResultSet
+DBConnect()
+exequery(String,String):void
+toString():String
+equals(Object):boolean

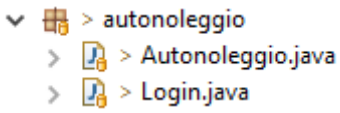

<<Java Class>>
PannelloExtra
gui.pannelli
+PannelloExtra(Finestra)
+run(JPanel):JPanel
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
-frame
0..1
-pnlModulo
0..1
<<Java Class>>
Finestra
gui.finestre
+username: String
+contentPane: JPanel
+Finestra(String)
+menu():void
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
<<Java Class>>
ModuloExtra
gui.moduli
+ModuloExtra()
+ModuloExtra(String)
+set(String):void
+actionPerformed(ActionEvent):void
+toString():String
+equals(Object):boolean
-extra
0..1
-profitto
0..1
<<Java Class>>
DBConnect
db
+con: Connection
+st: PreparedStatement
+rs: ResultSet
+DBConnect()
+exequery(String,String):void
+toString():String
+equals(Object):boolean
<<Java Class>>
Noleggiabilita
utils
+Noleggiabilita()
+controlla(String,String,String):boolean
+toString():String
-connessione
0..1
<<Java Class>>
DBConnect
db
+con: Connection
+st: PreparedStatement
+rs: ResultSet
+DBConnect()
+exequery(String,String):void
+toString():String
+equals(Object):boolean
<<Java Class>>
IsNumeric
utils
+IsNumeric()
+isNumeric(String):boolean
+toString():String
<<Java Class>>
GestioneGiorni
utils
+GestioneGiorni()
+calcolaGiorni(String,String):long
+dataFine(String,long):String[]
+isMaggiore(String,String):boolean
+isMinore(String,String):boolean
+nonConsentito(String):boolean
+toString():String
<<Java Class>>
ArrotondaNumero
utils
+ArrotondaNumero()
+arrotonda(double,int):double
+toString():String
<<Java Class>>
CostruisciTabella
utils
+model: DefaultTableModel
+CostruisciTabella(ResultSet)
+toString():String
<<Java Class>>
TableColumnAdjuster
utils
+TableColumnAdjuster(JTable)
+TableColumnAdjuster(JTable,int)
+adjustColumns():void
+adjustColumn(int):void
+restoreColumns():void
+setColumnHeaderIncluded(boolean):void
+setColumnDataIncluded(boolean):void
+setOnlyAdjustLarger(boolean):void
+setDynamicAdjustment(boolean):void
+propertyChange(PropertyChangeEvent):void
+tableChanged(TableModelEvent):void
<<Java Class>>
ColumnAction
utils
+ColumnAction(boolean,boolean)
+actionPerformed(ActionEvent):void
<<Java Class>>
ToggleAction
utils
+ToggleAction(boolean,boolean)
+actionPerformed(ActionEvent):void

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.



Autonoleggio

La classe Autonoleggio contiene l’implementazione del metodo main.

```
package autonoleggio;

import javax.swing.JOptionPane;

/**
 * La classe Autonoleggio manda in esecuzione l'applicazione.
 */
public class Autonoleggio {

    /**
     * Avvia l'applicazione.
     *
     * @param args un vettore di stringhe ricevute da righe di comando
     */
    public static void main(String[] args) {
        try {
            (new Thread(new Login())).start();
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(null, "Errore! Impossibile avviare l'applicazione!",
                    "Errore ",
                    JOptionPane.ERROR_MESSAGE);
        }
    }

/* OVERRIDING METODO toString() */

    /**
     * Restituisce una rappresentazione testuale dell'oggetto.
     *
     * @return una stringa rappresentante l'oggetto.
     */
    public String toString() {
        return "Autonoleggio [La classe Autonoleggio manda in esecuzione l'applicazione.]";
    }

}
```

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.

```
/**
 * Metodo richiamato quando si manda in esecuzione un thread che fa riferimento ad un oggetto Login.
 */
public void run() {

    try {
        Login frame = new Login();
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
        frame.setIconImage(Toolkit.getDefaultToolkit().getImage(Login.class.getResource("/External/car.png")));
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Errore! Impossibile avviare il pannello di login!",
                    "Errore ",
                    JOptionPane.ERROR_MESSAGE);
    }
}
```

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

```
/**
 * Verifica che le credenziali di accesso siano valide.
 */
private void check(){

    int size=0;
    try {
        /* Il metodo trim() prende una stringa e rimuove eventuali whitespaces in testa ed in coda. */
        String user = txtUsername.getText().trim();
        char[] pass = txtPassword.getPassword();
        String pwd=String.copyValueOf(pass);
        /* Effettua una connessione al DB e cerca una corrispondenza con l'utente e la password inseriti.*/
        log.exequery("SELECT * FROM operatore WHERE (BINARY ID_Operatore='" + user +
                "' AND BINARY Password='" + pwd + "')","select");
        /* Se la ricerca ha esito positivo, aggiorna il valore size ad 1*/
        if (log.rs.next()) size=1;
        /* Se non viene inserito l'username o la password, viene restituito un errore.*/
        if (user.equals("") || pwd.equals("")) {
            JOptionPane.showMessageDialog(null, "Errore! Inserisci l'username e/o la password!",
                    "Errore ",
                    JOptionPane.ERROR_MESSAGE);
            txtUsername.requestFocus();
        } else if (size==0) {
            /* Se non esiste l'utente o se la password è errata, viene restituito un errore.*/
            log.rs.beforeFirst();
            txtUsername.setText("");
            txtPassword.setText("");
            user=null;
            pass=null;
            pwd=null;
            JOptionPane.showMessageDialog(null, "Errore! Utente non trovato o password errata!",
                    "Errore ",
                    JOptionPane.ERROR_MESSAGE);
            txtUsername.requestFocus();
        } else {
            /* Se l'utente viene trovato ed è l'admin, viene avviato il pannello di controllo dell'admin */
            this.dispose();
            new Finestra (user);
        }
        log.con.close();
    } catch (SQLException e1) {
        e1.printStackTrace();
        JOptionPane.showMessageDialog(null, "Errore! Impossibile connettersi al DB per effettuare l'accesso!",
                "Errore ",
                JOptionPane.ERROR_MESSAGE);
    }
}
```

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.

```
/**
 * Precompila la query SQL desiderata e, nel caso di una operazione di {@code select}, assegna il ResultSet derivante.
 *
 * @param query la query SQL desiderata.
 * @param tipo la tipologia di query SQL (delete, insert, select, update).
 *
 * @throws SQLException se avviene un errore che coinvolge il database.
 */
public void exequery(String query, String tipo) throws SQLException {

    if (tipo.equals("select")) {
        /* La query SQL desiderata è una select. */
        try {
            st = con.prepareStatement(query);
            rs = st.executeQuery();
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Errore SQL: " + e,
                    "Errore ",
                    JOptionPane.ERROR_MESSAGE);
            throw new SQLException(e);
        }
    } else if (tipo.equals("delete") || tipo.equals("insert") || tipo.equals("update")) {
        /*La query SQL desiderata esegue una operazione di delete, insert o update. */
        try {
            st = con.prepareStatement(query);
            st.executeUpdate();
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Errore SQL: " + e,
                    "Errore ",
                    JOptionPane.ERROR_MESSAGE);
            throw new SQLException(e);
        }
    }
}
```

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:

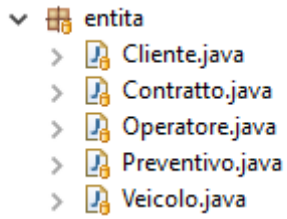
1. la query passata come argomento viene precompilata e memorizzata nella variabile st di tipo PreparedStatement
2. viene eseguita la query precompilata e viene memorizzato il relativo resultSet nella variabile rs di tipo ResultSet

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.

```
public String tipologia;
public String veicolo;
public String cliente;
public String inizio;
public String fine;
public String costo;
public String acconto;
public String cognome;
public String nome;
public String patente;
public String valida;
public String rilasciataDa;
public String rilasciataIl;

private boolean test;
private DBConnect noleggio;
private Integer codice,codiceCerca,codiceModifica;
```

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.

```
/**
 * Inizializza un nuovo oggetto Contratto e stabilisce una connessione con il database.
 */
public Contratto() {

    test=true;
    noleggio = new DBConnect();
}
```

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.

```
/**
 * Verifica che i dati del contratto da aggiungere/modificare siano corretti.
 *
 * @param content il form {@code "Nuovo Contratto"/"Modifica Contratto"} ed i relativi dati inseriti.
 * @return true se i dati inseriti sono validi; false altrimenti.
 */
private boolean check(ModuloContratto content){
```

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

```
/**
 * Verifica che il codice del contratto da modificare sia corretto.
 *
 * @param content il form {@code "Modifica Contratto"} ed i relativi dati inseriti.
 * @return true se i dati sono validi; false altrimenti.
 */
private boolean checkCerca(ModuloContratto content){
```

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

```
/**
 * Verifica che il codice del contratto da eliminare sia corretto.
 *
 * @param content il form {@code "Elimina Contratto"} ed i relativi dati inseriti.
 * @return true se i dati sono validi; false altrimenti.
 */
private boolean checkElimina(ModuloContratto content) {
```

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.

```
/**
 * Verifica che i dati (cliente e/o veicolo) del contratto  da filtrare sianno corretti.
 *
 * @param content il form {@code "Elenco Contratti"} ed i relativi dati inseriti.
 * @return true se i dati sono validi; false altrimenti.
 */
private boolean checkFiltra(ModuloContratto content) {
```

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.

```
/**
 * Aggiunge un nuovo contratto al database.
 *
 * @param content il form {@code "Nuovo Contratto"} ed i relativi dati inseriti.
 */
public void aggiungi(ModuloContratto content) {

    if (check(content)) {
```

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

```
/**
 * Cerca un contratto nel database.
 *
 * @param content il form {@code "Modifica Contratto"} ed i relativi campi inseriti.
 */
public void cerca(ModuloContratto content) {

    String item;
        if (checkCerca(content)) {
```

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

```
/**
 * Modifica un contratto presente nel database.
 *
 * @param content il form {@code "Modifica Contratto"} ed i relativi dati inseriti.
 */
public void modifica(ModuloContratto content){

    if (check(content)) {
```

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

```
/**
 * Elimina un contratto dal database.
 *
 * @param content il form {@code "Elimina Contratto"} ed i relativi dati inseriti.
 */
public void elimina(ModuloContratto content) {

    if (checkElimina(content)) {
```

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

```
/**
 * Filtra un contratto presente nell'elenco contratti.
 *
 * @param content il form {@code "Elenco Contratti"} ed i relativi dati inseriti.
 */
public void filtra(ModuloContratto content) {

    if (checkFiltra(content)) {
```

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.

```
/**
 * Assegna i valori inseriti nel form alle variabili dell'oggetto {@code Contratto}.
 *
 * @param content il form {@code "Nuovo Contratto"/"Modifica Contratto"} ed i relativi dati inseriti.
 */
public void setValori(ModuloContratto content) {
```

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

```
/**
 * Assegna il codice del contratto da cercare.
 *
 * @param content il form {@code "Modifica Cliente"} ed i relativi dati inseriti.
 * @return true se l'operazione di set è andata a buon fine; false altrimenti.
 */
public boolean setCodiceCerca(ModuloContratto content) {
```

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

```
/**
 * Assegna il codice del contratto da modificare.
 *
 * @param content il form {@code "Modifica Cliente"} ed i relativi dati inseriti.
 */
public void setCodiceModifica (ModuloContratto content) {
```

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

```
/**
 * Assegna il codice del contratto da eliminare.
 *
 * @param content il form {@code "Elimina Cliente"} ed i relativi dati inseriti.
 * @return true l'operazione di set è andata a buon fine; false altrimenti.
 */
public boolean setCodiceElimina (ModuloContratto content) {
```

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

```
/**
 * Assegna i valori inseriti nel form alle variabili dell'oggetto {@code Contratto}.
 *
 * @param content il form {@code "Elenco Contratti"} ed i relativi dati inseriti.
 */
public void setValoriFiltra(ModuloContratto content){
```

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

```
/**
 * Restituisce una rappresentazione testuale dell'oggetto.
 *
 * @return una stringa rappresentante l'oggetto.
 */
public String toString() {
    return "Contratto [La classe Contratto rappresenta i contratti dell'autonoleggio.]";
}
```

```
/**
 * Confronta questo oggetto con quello passato come argomento.
 *
 * @param obj l'oggetto da confrontare.
 * @return true se i due oggetti sono uguali; false altrimenti.
 */
public boolean equals(Object obj) {
```

Preventivo

Il preventivo possiede le seguenti proprietà.

```
private DBConnect preventivo;

private boolean test;

private String[] Date = new String[2];
private String disponibilita;
private String tipologia;
private static String veicolo;
private static String inizio;
private static String fine;
private String giorni;
private String anni;
private String acconto;
private static long giorniNoleggio;
private static double totale;
private double alGiorno, alMese, giornoExtra, meseScontato, sconto;
```

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.

```
/**
 * Inizializza un nuovo oggetto Preventivo e stabilisce una connessione con il database.
 */
public Preventivo() {
    test = true;
    preventivo = new DBConnect();
}
```

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.

```
/**
 * Verifica che i dati del preventivo da calcolare siano corretti.
 *
 * @param content il form {@code "Calcola Preventivo"} ed i relativi dati inseriti.
 * @return true se i dati inseriti sono validi; false altrimenti.
 */
private boolean check(ModuloContratto content) throws Exception {
```

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.

```
/**
 * Calcola un nuovo preventivo.
 *
 * @param content il form {@code "Calcola Preventivo"} ed i relativi dati inseriti.
 *
 * @throws Exception se avviene qualche errore che coinvolge chiamate alle funzioni da noi definite.
 */
public void calcola(ModuloContratto content) throws Exception {

    if (check(content)) {
```

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

Il metodo getVeicolo restituisce il valore della variabile veicolo.

```
/**
 * Metodo usato quando si sceglie "Passa a Contratto" nel form {@code "Calcola Preventivo"}.
 *
 * @return la targa del veicolo noleggiato
 */
public static String getVeicolo() {

    return veicolo;
}
```

Il metodo getDataInizio restituisce il valore della variabile inizio.

```
/**
 * Metodo usato quando si sceglie "Passa a Contratto" nel form {@code "Calcola Preventivo"}.
 *
 * @return la data di inizio noleggio
 */
public static String getDataInizio() {

    return inizio;
}
```

Il metodo getDataFine restituisce il valore della variabile fine.

```
/**
 * Metodo usato quando si sceglie "Passa a Contratto" nel form {@code "Calcola Preventivo"}.
 *
 * @return la data di fine noleggio
 */
public static String getDataFine() {

    return fine;
}
```

Il metodo getGiorniNoleggio restituisce il valore della variabile giorniNoleggio.

```
/**
 * Metodo usato quando si sceglie "Passa a Contratto" nel form {@code "Calcola Preventivo"}.
 *
 * @return la durata del noleggio espressa in giorni
 */
public static long getGiorniNoleggio () {

    return giorniNoleggio;
}
```

Il metodo getTotale restituisce il valore della variabile totale.

```
/**
 * Metodo usato quando si sceglie "Passa a Contratto" nel form {@code "Calcola Preventivo"}.
 *
 * @return il costo totale del noleggio
 */
public static double getTotale() {

    return 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.

```
/**
 * Assegna i valori inseriti nel form alle variabili dell'oggetto {@code Preventivo}.
 *
 * @param content il form {@code "Calcola Preventivo"} ed i relativi dati inseriti.
 */
public void setValori(ModuloContratto content) {

    veicolo = content.txtVeicolo.getText().trim();
    inizio = content.frmtdtxtfldInizio.getText().trim();
    fine = content.frmtdtxtfldFine.getText().trim();
}
```

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

```java
/**
 * Restituisce una rappresentazione testuale dell'oggetto.
 *
 * @return una stringa rappresentante l'oggetto.
 */
public String toString() {
    return "Preventivo [La classe Preventivo rappresenta i preventivi dell'autonoleggio.]";
}


/**
 * Confronta questo oggetto con quello passato come argomento.
 *
 * @param obj l'oggetto da confrontare.
 * @return true se i due oggetti sono uguali; false altrimenti.
 */
public boolean equals(Object obj) {
```

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.

```
public String username;

public JPanel contentPane = new JPanel();

private final JMenuBar menuBar = new JMenuBar();
private final JButton btnExtra = new JButton("Extra");
private final JButton btnOperatore = new JButton("Operatore");
private final JButton btnFlotta = new JButton("Flotta");
private final JButton btnContratto = new JButton("Contratto");
private final JButton btnCliente = new JButton("Cliente");
private final JButton btnHome = new JButton("Home");
```

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.

```
/**
 * Inizializza un nuovo oggetto Finestra e memorizza l'oggetto String {@code user}
 *  passato come argomento nella variabile {@code username}.
 *
 * @param user l'username dell'utente che utilizza l'applicazione.
 */
public Finestra(String user) {

    username = user;
    try {
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.menu();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        this.setBounds(100, 100, 1050, (screenSize.height-100));
        this.setMinimumSize(new Dimension(1000,(screenSize.height-100)));
        this.setLocationRelativeTo(null);
        this.setIconImage(Toolkit.getDefaultToolkit().getImage(Login.class.getResource("/External/car.png")));
        this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                int scelta = JOptionPane.showConfirmDialog(
                        null,
                        "Si desidera uscire dall'applicazione?",
                        "Conferma uscita",
                        JOptionPane.YES_NO_OPTION);
                if (scelta == JOptionPane.YES_OPTION) {
                    System.exit(0);
                }
            }
        });
        new PannelloHome(this);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

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.

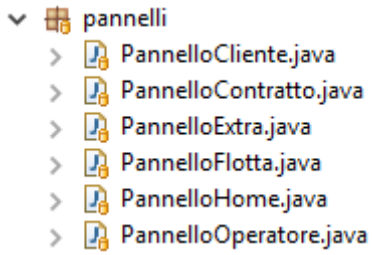
```
/**
 * Aggiunge i pulsanti al menu principale dell'applicazione.
 */
public void menu(){
```

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

```
/**
 * Definisce le azioni da eseguire a seconda del pulsante cliccato.
 */
public void actionPerformed(ActionEvent e) {

    if(btnHome == e.getSource()) {
        getContentPane().removeAll();
        new PannelloHome(this);
        getContentPane().revalidate();

    }
```

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.

```
private Finestra frame;
private JLabel user;

private ModuloCliente pnlModulo = new ModuloCliente("Principale");
private ModuloCliente pnlModulo2 = new ModuloCliente("Opzionale");
private ModuloElencoClienti pnlModulo3 = new ModuloElencoClienti();

private JScrollPane scrollPane = new JScrollPane(pnlModulo);
private JScrollPane scrollPane2 = new JScrollPane(pnlModulo2);

private JButton btnNuovo = new JButton("Nuovo Cliente");
private JButton btnModifica = new JButton("Modifica Cliente");
private JButton btnElimina = new JButton("Elimina Cliente");
private JButton btnAggiorna = new JButton("Elenco Clienti");
private JButton btnLogout = new JButton("Logout");
private JButton btnEsci = new JButton("Esci");
```

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.

```
/**
 * Inizializza un nuovo oggetto PannelloCliente.
 *
 * @param window un frame Finestra.
 */
public PannelloCliente(Finestra window) {

    frame = window;
    window.setTitle("Autonoleggio - Cliente");
    window.setContentPane(this.run(window.contentPane));
}
```

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

```
/**
 * Modifica il pannello passato come argomento.
 *
 * @param contentPane un pannello "vuoto".
 * @return il pannello modificato.
 */
public JPanel run(JPanel contentPane) {
```

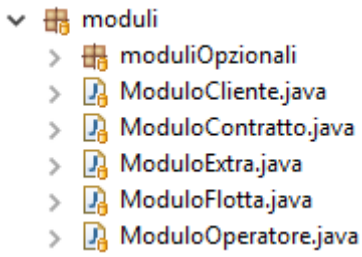
Il metodo actionPerformed definisce le azioni da eseguire a seconda del pulsante cliccato. Nella figura sottostante viene mostrato il caso dei pulsanti "Nuovo Cliente" e "Modifica Cliente".

```
/**
 * Definisce le azioni da eseguire a seconda del bottone cliccato.
 */
public void actionPerformed(ActionEvent e) {

    if (btnNuovo == e.getSource()) {
        btnAggiorna.setText("Elenco Clienti");
        pnlModulo.set("Nuovo");
        scrollPane2.setViewportView(pnlModulo2);

    } else if (btnModifica == e.getSource()) {
        btnAggiorna.setText("Elenco Clienti");
        pnlModulo.set("Modifica");
        scrollPane2.setViewportView(pnlModulo3);
```

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.

```
private Cliente cliente;
private DBConnect elencoClienti;

private JScrollPane scroll;
private JTable tblClienti;

private JButton btnAggiungi;
private JButton btnElimina;
private JButton btnModificaC;
private JButton btnCerca;
public JTextField txtRS;
public JTextField txtCAP;
public JTextField txtCitta;
public JTextField txtVia;
public JTextField txtNumero;
public JTextField txtCF_PIVA;
public JTextField txtEmail;
public JTextField txtTelefono;
public JTextField txtClienteCerca;
public JComboBox <String> comboBoxTipologia;
```

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.

```
/**
 * Inizializza un nuovo oggetto ModuloCliente e richiama il metodo {@code set} passando come argomento l'oggetto String {@code str}.
 *
 * @param str una stringa che determina il diverso comportamento del metodo {@code set}.
 */
public ModuloCliente(String str) {
    set(str);
}
```

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.

```java
/**
 * Si comporta in maniera differente a seconda dell'oggetto String che viene passato come argomento. <br><br>
 *
 * - Se viene passato "Principale", viene creato un modulo vuoto recante la scritta "Modulo Principale".<br>
 * - Se viene passato "Opzionale", viene creato un modulo vuoto recante la scritta "Modulo Opzionale".<br>
 * - Se viene passato "Nuovo", viene creato il form per aggiungere un cliente. <br>
 * - Se viene passato "Modifica", viene creato il form per modificare un cliente. <br>
 * - Se viene passato "Elimina", viene creato il form per eliminare un cliente. <br>
 * - Se viene passato "Elenca", viene generato l'elenco dei clienti contenuti nel database.
 *
 * @param str una stringa che determina cosa verrà mostrato a schermo.
 */
public void set(String str) {

    if (str.equals("Principale")) {



    } else if (str.equals("Opzionale")) {



    } else if (str.equals("Nuovo")) {



    } else if (str.equals("Modifica")) {



    } else if (str.equals("Elimina")) {



    } else if (str.equals("Elenca")) {

        /* Viene generato l'elenco dei clienti contenuti nel database. */
        elencoClienti = new DBConnect();
        this.removeAll();
        this.setBorder(BorderFactory.createTitledBorder("Elenco Clienti"));

        try {
            elencoClienti.exequery("SELECT * FROM cliente","select");

            tblClienti = new JTable();
            tblClienti.setModel(new CostruisciTabella(elencoClienti.rs).model);
            tblClienti.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
            TableColumnAdjuster tca = new TableColumnAdjuster(tblClienti);
            tca.adjustColumns();

            scroll = new JScrollPane(tblClienti);
            scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
            scroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
            scroll.setViewportView(tblClienti);

            elencoClienti.con.close();
        } catch(SQLException e) {
            JOptionPane.showMessageDialog(null, "Errore! Impossibile caricare l'elenco dei clienti!",
                "Errore ",
                JOptionPane.ERROR_MESSAGE);
        }


    }
}
```

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

```
/**
 * Definisce le azioni da eseguire a seconda del pulsante cliccato.
 */
public void actionPerformed(ActionEvent e) {

    if (btnAggiungi == e.getSource()) {
        cliente = new Cliente();
        cliente.setValori(this);
        cliente.aggiungi(this);

    }
```

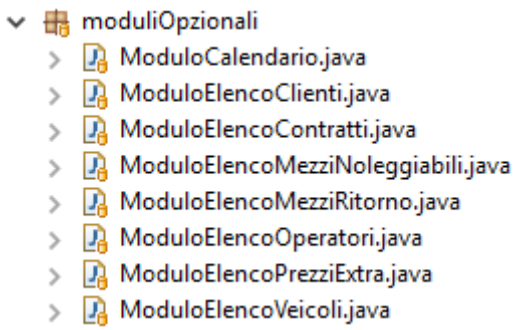
Innanzitutto viene istanziato un nuovo oggetto Cliente e viene assegnato il suo riferimento alla variabile cliente.

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.

```
private JTable tblClienti;
private JScrollPane scroll = new JScrollPane(tblClienti);
private DBConnect clienti = new DBConnect();
```

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.

```
/**
 * Inizializza un nuovo oggetto ModuloElencoClienti e richiama il metodo {@code set}.
 */
public ModuloElencoClienti() {
    set();
}
```

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

```
/**
 * Costruisce l'elenco dei clienti contenuti nel database.
 */
public void set() {

    this.setBorder(BorderFactory.createTitledBorder("Elenco Clienti"));

    try {
        clienti.exequery("SELECT * FROM cliente ORDER BY Tipologia, Ragione_Sociale","select");

        tblClienti = new JTable();
        tblClienti.setModel(new CostruisciTabella(clienti.rs).model);
        tblClienti.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        TableColumnAdjuster tca = new TableColumnAdjuster(tblClienti);
        tca.adjustColumns();

        scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        scroll.setViewportView(tblClienti);

        clienti.con.close();
    } catch (SQLException e) {
    JOptionPane.showMessageDialog(null, "Errore, impossibile generare l'elenco dei clienti!",
            "Errore ",
            JOptionPane.ERROR_MESSAGE);}

}
```

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.

```
static int annoAttuale, meseAttuale, giornoAttuale, annoCorrente, meseCorrente;

static JTable tblCalendar;
static JScrollPane stblCalendar;
static DefaultTableModel mtblCalendar;

static JLabel lblMese, lblAnno;
static JButton btnPrec, btnSucc;
static JComboBox <String> cmbAnno;
```

Le variabili intere annoAttuale (e annoCorrente), meseAttuale (e meseCorrente), giornoAttuale contengono rispettivamente l'anno, il mese ed il giorno corrispondenti alla data odierna.

La variabile tblCalendar di tipo JTable costituisce la tabella che conterrà il calendario.

La variabile stblCalendar di tipo JScrollPane costituisce il "pannello scorrevole" che conterrà la tabella tblCalendar.

La variabile mtblCalendar di tipo DefaultTableModel conterrà i valori delle celle della tabella tblCalendar sotto forma di vettore di vettori.

Il costruttore inizializza un nuovo oggetto ModuloCalendario a partire dai due pannelli passati come parametri. Inoltre richiama il metodo aggiornaCalendario passando come parametri le variabili meseAttuale e annoAttuale.

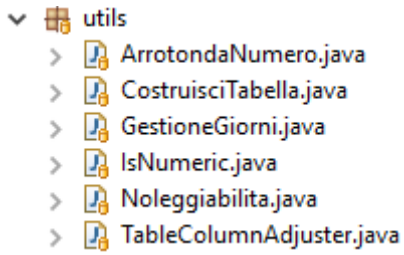
```
/**
 * Inizializza un nuovo oggetto ModuloCalendario.
 *
 * @param pane il container di {@code pnlCalendar}.
 * @param pnlCalendar il pannello in cui viene visualizzato il calendario.
 */
public ModuloCalendario(JPanel pane, JPanel pnlCalendar) {
```

Il metodo aggiornaCalendario riempe le celle della tabella tblCalendar con i giorni opportuni.

```
/**
 * Riempie il calendario con gli opportuni giorni.
 *
 * @param mese il mese corrente
 * @param anno l'anno corrente
 */
public static void aggiornaCalendario(int mese, int anno) {
```

Package “utils”

Il package utils contiente le classi



ArrotondaNumero

La classe ArrotondaNumero contiene metodi per gestire operazioni di arrotondamento di numeri.

Il metodo arrotonda arrotonda il numero decimale valore passato come argomento al numero di cifre dopo la virgola indicato dalla variabile posizioni.

```
/**
 *  Arrotonda un numero decimale.
 *
 *  @param valore il numero che si desidera arrotondare.
 *  @param posizioni il numero desiderato di cifre dopo la virgola.
 *  @return il valore dell'argomento arrotondato al numero di cifre indicato.
 */
public static double arrotonda(double valore, int posizioni) {
```

CostruisciTabella

La classe CostruisciTabella consente di costruire una tabella a partire dal ResultSet di una query SQL.

Essa presenta le seguenti proprietà.

```
public DefaultTableModel model;
```

La variabile model di tipo DefaultTableModel conterrà i valori delle celle della tabella sotto forma di vettore di vettori.

Il costruttore inizializza un nuovo oggetto CostruisciTabella e richiama il metodo costruisci passando come argomento il ResultSet rs.

```
/**
 * Inizializza un nuovo oggetto CostruisciTabella e richiama il metodo {@code costruisci}
 * passando come argomento il ResultSet di una query SQL.
 *
 * @param rs il ResultSet ottenuto da una query SQL.
 */
public CostruisciTabella(ResultSet rs){
    costruisci(rs);
}
```

Il metodo costruisci costruisce una tabella a partire dal ResultSet passato come argomento.

```
/*Costruisce una tabella a partire dal ResultSet di una query SQL. */

private void costruisci(ResultSet result){
```

GestioneGiorni

La classe GestioneGiorni contiene metodi per gestire date.

Il metodo calcolaGiorni calcola i giorni di noleggio a partire dalle date contenute nelle variabili dataInizio e dataFine sotto forma di stringhe.

```
/**
 * Calcola i giorni di noleggio.
 *
 * @param dataInizio la data di inizio del noleggio.
 * @param dataFine la data di fine del noleggio.
 * @return il numero di giorni del noleggio.
 *
 * @throws ParseException se avviene un errore durante il parsing delle stringhe passate come argomenti.
 */
public static long calcolaGiorni(String dataInizio, String dataFine) throws ParseException{
```

Il metodo dataFine calcola le date di fine noleggio in base ai periodi consentiti più vicini al numero di giorni di noleggio contenuto nella variabile giorniNoleggio.

```
/**
 * Calcola due possibili date di fine noleggio per un periodo corrispondente a 30, 60, 90, 120, 150, 180, 365, 730, 1095 giorni. <br><br>
 *
 * es. dataFine("2017-08-10", 65); <br><br>
 *
 *     data1 = "2017-10-09" corrispondente ad un periodo di noleggio di 60 giorni <br>
 *     data2 = "2017-11-08" corrispondente ad un periodo di noleggio di 90 giorni
 *
 * @param dataInizio la data di inizio del noleggio.
 * @param giorniNoleggio il numero di giorni del noleggio.
 * @return due possibili date di fine noleggio in relazione al valore del parametro giorniNoleggio.
 *
 * @throws ParseException se avviene un errore durante il parsing delle stringhe passate come argomenti.
 */
public static String[] dataFine(String dataInizio, long giorniNoleggio) throws ParseException {
```

Il metodo isMaggiore verifica se la data contenuta nella variabile dataInizio sotto forma di stringa viene dopo quella contenuta nella variabile dataFine.

```
/**
 * Verifica se la data di inizio noleggio viene dopo la data di fine noleggio.
 *
 * @param dataInizio la data di inizio noleggio.
 * @param dataFine la data di fine noleggio.
 * @return true se dataInizio viene dopo dataFine; false altrimenti.
 *
 * @throws ParseException se avviene un errore durante il parsing delle stringhe passate come argomenti.
 */
public static boolean isMaggiore(String dataInizio, String dataFine) throws ParseException {
```

Il metodo isMinore verifica se la data contenuta nella variabile dataInizio sotto forma di stringa viene prima di quella contenuta nella variabile dataFine.

```
/**
 * Verifica se la data di inizio noleggio viene prima della data di fine noleggio.
 *
 * @param dataInizio la data di inizio noleggio.
 * @param dataFine la data di fine noleggio.
 * @return true se dataInizio viene prima di dataFine; false altrimenti.
 *
 * @throws ParseException se avviene un errore durante il parsing delle stringhe passate come argomenti.
 */
public static boolean isMinore(String dataInizio, String dataFine) throws ParseException {
```

Il metodo nonConsentito verifica se la data contenuta nella variabile dataInizio sotto forma di stringa è antecedente alla data odierna.

```
/**
 * Verifica se la data di inizio noleggio è antecedente alla data odierna.
 *
 * @param dataInizio la data di inizio noleggio.
 * @return true se dataInizio è antecedente alla data odierna; false altrimenti.
 *
 * @throws ParseException se avviene un errore durante il parsing delle stringhe passate come argomenti.
 */
public static boolean nonConsentito(String dataInizio) throws ParseException {
```

IsNumeric

La classe IsNumeric contiene metodi per controllare se una stringa è numerica.

Il metodo isNumeric verifica se la stringa passata come argomento è numerica.

```
/**
 * Verifica se la stringa passata come argomento è numerica.
 *
 * @param string la stringa da analizzare.
 * @return true se la stringa è numerica; false altrimenti.
 */
public static boolean isNumeric(String string) {
```

Noleggiabilita

La classe Noleggiabilita permette di controllare se un veicolo è noleggiabile o meno.

Essa presenta le seguenti proprietà.

```
private static DBConnect connessione;
```

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

Il metodo controlla verifica se il veicolo indicato dalla variabile targa è noleggiabile per il periodo compreso tra le date contenute sotto forma di stringa nelle variabili inizio e fine.

```
/**
 * Verifica se un determinato veicolo è noleggiabile o meno.
 *
 * @param targa lo specifico veicolo che si intende noleggiare
 * @param inizio la data di inizio noleggio
 * @param fine la data di fine noleggio
 * @return true se il veicolo è noleggiabile; false altrimenti
 *
 * @throws SQLException se avviene un errore che coinvolge il database
 * @throws ParseException se avviene un errore durante il parsing delle stringhe passate come argomenti.
 */
public static boolean controlla(String targa, String inizio, String fine) throws SQLException, ParseException {
```

TableColumnAdjuster

La classe TableColumnAdjuster modifica la larghezza delle colonne di una tabella in base al contenuto.

Essa presenta le seguenti proprietà.

```
private int spacing;

private JTable table;
private Map<TableColumn, Integer> columnSizes = new HashMap<TableColumn, Integer>();

private boolean isColumnHeaderIncluded;
private boolean isColumnDataIncluded;
private boolean isOnlyAdjustLarger;
private boolean isDynamicAdjustment;
```

La variabile spacing conterrà il valore della larghezza delle colonne.

La variabile table conterrà la tabella da ridimensionare.

La variabile columnSizes conterrà una mappa costituita da coppie numero - larghezza di colonna.

Le restanti variabili di tipo boolean costituiscono variabili di controllo usate dai metodi della classe.

La classe TableColumnAdjuster possiede due costruttori.

Il primo inizializza un nuovo oggetto TableColumnAdjuster ed imposta la larghezza delle colonne della tabella table passata come argomento ad un valore di default.

```
/**
 * Inizializza un nuovo oggetto TableColumnAdjuster e imposta la larghezza
 * delle colonne della tabella {@code table} ad un valore di default.
 *
 * @param table la tabella da ridimensionare.
 */
public TableColumnAdjuster(JTable table) {
```

Il secondo inizializza un nuovo oggetto TableColumnAdjuster ed imposta la larghezza delle colonne della tabella table al valore desiderato contenuto nella variabile spacing passata come argomento.

```
/**
 * Inizializza un nuovo oggetto TableColumnAdjuster e imposta la larghezza
 * delle colonne della tabella {@code table} al valore {@code spacing} passato come argomento.
 *
 * @param table la tabella da ridimensionare.
 * @param spacing la larghezza delle colonne desiderata.
 */
public TableColumnAdjuster(JTable table, int spacing) {
```

Il metodo adjustColumns ridimensiona la larghezza di tutte le colonne della tabella.

```
/**
 * Ridimensiona la larghezza di tutte le colonne della tabella.
 */
public void adjustColumns() {
```

Il metodo adjustColumn ridimensiona la larghezza della colonna passata come argomento.

```
/**
 * Ridimensiona la larghezza della colonna passata come argomento.
 *
 * @param column la colonna da ridimensionare.
 */
public void adjustColumn(final int column) {
```

Il metodo getColumnHeaderWidth calcola la larghezza in base all'intestazione della colonna passata come argomento.

```
//Calcola la larghezza in base all'intestazione della colonna passata come argomento.
private int getColumnHeaderWidth(int column) {
```

Il metodo getColumnDataWidth calcola la larghezza in base all'elemento più largo della colonna passata come argomento.

```
//Calcola la larghezza in base all'elemento più largo della colonna passata come argomento.
private int getColumnDataWidth(int column) {
```

Il metodo getCellDataWidth calcola la larghezza della colonna in base all'elemento corrente.

```
//Calcola la larghezza della colonna in base all'elemento corrente.
private int getCellDataWidth(int row, int column) {
```

Il metodo updateTableColumn ridimensiona la colonna con la nuova larghezza calcolata.

```
//Ridimensiona la colonna con la nuova larghezza calcolata.
private void updateTableColumn(int column, int width) {
```

Il metodo restoreColumns ripristina la larghezza delle colonne ai loro precedenti valori.

```
/**
 * Ripristina la larghezza delle colonne ai loro precedenti valori.
 */
public void restoreColumns() {
```

Il metodo restoreColumn ripristina la larghezza della colonna passata come argomento al suo valore precedente.

```
//Ripristina la larghezza della colonna passata come argomento al suo valore precendente.
private void restoreColumn(int column) {
```

Strumenti di programmazione utilizzati

La programmazione orientata agli oggetti (OOP, Object Oriented Programming) è un paradigma di programmazione che permette di definire oggetti software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi.

Incapsulamento

L'incapsulamento è il meccanismo che collega il codice e i dati che manipola, mettendoli al sicuro da interferenze esterne e da utilizzi impropri. L'accesso al codice e ai dati è strettamente controllato mediante un'interfaccia ben definita. La potenza del codice incapsulato sta nel fatto che tutti sanno come accedere ad esso e possono utilizzarlo indipendentemente dai dettagli di implementazione, senza paura di effetti collaterali imprevisti.

In Java la base dell'incapsulamento è la classe. Esistono meccanismi per nascondere la complessità dell'implementazione all'interno della classe. È possibile contrassegnare ciascun metodo o variabile di una classe come privato (private) o pubblico (public).

L'interfaccia pubblica di una classe rappresenta tutto ciò che gli utenti esterni possono sapere. Ai metodi e ai dati privati può accedere solo codice membro della classe.

Ereditarietà

L'ereditarietà è uno dei concetti fondamentali nel paradigma di programmazione a oggetti. Essa consiste in una relazione che il linguaggio di programmazione, o il programmatore stesso, stabilisce tra due classi. Se la classe B eredita dalla classe A, si dice che B è una sottoclasse di A e che A è una superclasse di B. In generale, l'uso dell'ereditarietà dà luogo ad una gerarchia di classi.

Uno dei principali vantaggi dell'uso dell'ereditarietà (in particolare combinata con il polimorfismo) è il fatto di favorire il riuso del codice.

Un'altra ragione per usare l'ereditarietà è fornire ad una classe dati o funzionalità aggiuntive. Questa operazione è di solito chiamata estensione. L'estensione viene usata spesso quando non è possibile o conveniente aggiungere nuove funzionalità alla classe base.

All'interno del codice sorgente, le classi che implementano l'interfaccia grafica dell'applicazione fanno uso del meccanismo di estensione. Alcuni esempi di utilizzo sono i seguenti:

- la classe Login estende la classe JFrame del package javax.swing

```
public class Login extends JFrame implements ActionListener, Runnable {
```

- la classe Finestra estende la classe JFrame del package javax.swing

```
public class Finestra extends JFrame implements ActionListener {
```

- la classe PannelloContratto estende la classe JPanel del package javax.swing

```
public class PannelloCliente extends JPanel implements ActionListener {
```

- la classe ModuloContratto estende la classe JPanel del package javax.swing

```
public class ModuloContratto extends JPanel implements ActionListener {
```

Polimorfismo

Nel contesto della programmazione orientata agli oggetti, il termine polimorfismo si riferisce al fatto che una espressione il cui tipo sia descritto da una classe A può assumere valori di un qualunque tipo descritto da una classe B sottoclasse di A.

Ad esempio, nella classe GestioneGiorni viene assegnato un oggetto di tipo SimpleDateFormat ad una variabile di tipo DateFormat.

```
DateFormat fmt = new SimpleDateFormat("yyyy-MM-dd");
```

Inoltre si parla di polimorfismo anche per i metodi.

Dal punto di vista implementativo il polimorfismo per i metodi si ottiene utilizzando l'overload e l'override dei metodi stessi.

L'overload si basa sulla scrittura di più metodi identificati dallo stesso nome che però hanno in ingresso parametri di tipo e numero diverso.

L'override consiste in una vera e propria riscrittura di un certo metodo di una classe che è stata ereditata. Dunque l'override implica necessariamente ereditarietà.

Ad esempio, nella classe TableColumnAdjuster viene eseguito un overload sul costruttore.

```
public TableColumnAdjuster(JTable table) {

public TableColumnAdjuster(JTable table, int spacing) {
```

Classi statiche

Una classe statica è una classe della quale non è possibile istanziare oggetti; per questo essa contiene soltanto membri statici. Una classe statica è equivalente a una classe non statica con membri statici e costruttore privato, per impedire la creazione di istanze.

Un metodo statico (o metodo di classe) può essere invocato tramite il nome della classe a cui appartiene, senza dover utilizzare alcun oggetto.

Esempi di questa pratica all'interno del codice sorgente dell'applicazione sono i seguenti:

- il metodo isNumeric della classe IsNumeric richiamato nella classe Cliente

```
} else if ((!IsNumeric.isNumeric(cap) || cap.length() > 5 || cap.length() < 5) && !cap.equals("")) {
```

- il metodo calcolaGiorni della classe GestioneGiorni richiamato nella classe Preventivo

```
giorniNoleggio = GestioneGiorni.calcolaGiorni(inizio, fine);
```

- il metodo controlla della classe Noleggiabilita richiamato nella classe Preventivo

```
} else if (!Noleggiabilita.controlla(veicolo,inizio,fine)) {
```

Classi anonime

Una classe anonima è una classe "locale" senza un nome assegnato. Si tratta di una classe definita ed istanziata un'unica volta attraverso una singola espressione caratterizzata da una versione estesa della sintassi dell'operatore new.

Un esempio all'interno del codice sorgente dell'applicazione è il seguente.

```
txtPassword.addKeyListener(new KeyAdapter() {

        public void keyPressed(KeyEvent evt) {

            if (evt.getKeyCode()==KeyEvent.VK_ENTER) {
                log = new DBConnect();
                check();
            }
        }

        public void keyTyped(KeyEvent evt) {
        }

        public void keyReleased(KeyEvent evt) {
        }
});
```

In particolare viene implementata localmente la classe astratta KeyAdapter.

Una classe astratta è una classe che definisce una interfaccia senza implementarla completamente. Questo serve come base di partenza per generare una o più classi specializzate aventi tutte la stessa interfaccia di base. Prima che una classe derivata da una classe astratta possa essere istanziata essa ne deve implementare tutti i metodi astratti.

Nel caso precedente, vengono implementati tutti i metodi (keyPressed, keyTyped e keyReleased) della classe KeyAdapter.

Package Utilizzati

Il framework principale di Java mette a disposizione moltissimi package e classi pronti all'uso. Nel seguito verranno elencati quelli utilizzati nel codice sorgente dell'applicazione.

AWT

Il package java.awt contiene classi e metodi che consentono di creare e gestire finestre. Rappresenta inoltre la base su cui è realizzato Swing (Schildt, 2012). In particolare sono state utilizzate esclusivamente le classi relative alla gestione degli eventi, poiché l'interfaccia grafica è stata creata utilizzando Swing.

Beans

Il package java.beans contiene le classi e le interfacce che costituiscono il JavaBeans API. Un Java Bean è un componente software progettato per essere riutilizzato in diversi ambienti. Un Bean può essere visibile (es. un pulsante su un'interfaccia grafica) o invisibile all'utente (Schildt, 2012).

Nel codice sorgente dell'applicazione sono state utilizzate la classe PropertyChangeEvent e l'interfaccia PropertyChangeListener.

Math

Il package java.math contiene classi per eseguire calcoli con numeri interi a precisione arbitraria (BigInteger) e calcoli con numeri decimali a precisione arbitraria (BigDecimal) (Oracle, 2016).

Nel codice sorgente dell'applicazione sono state utilizzate la classe BigDecimal e l'enumerazione RoundingMode.

SQL

Il package java.sql fornisce le API per l'accesso e l'elaborazione di dati memorizzati in un database tramite l'utilizzo del linguaggio Java (Oracle, 2016). In particolare è stato utilizzato il driver JDBC.

Nel codice sorgente dell'applicazione sono state utilizzate le classi DriverManager, SQLException e le interfacce Connection, PreparedStatement, ResultSet.

Swing

Il package javax.swing contiene classi che forniscono componenti grafici più potenti e flessibili rispetto ad AWT (Schildt, 2012).

Text

Il package java.text contiene classi ed interfacce per la gestione di testi, date, numeri e messaggi in maniera indipendente dai diversi idiomi (Oracle, 2016).

Nel codice sorgente dell'applicazione sono state utilizzate le classi DateFormat e SimpleDateFormat.

Util

Il package java.util contiene classi che generano numeri pseudocasuali, gestiscono data e ora, osservano eventi, manipolano set di bit, suddividono le stringhe in token e gestiscono dati formattati. Esso contiene inoltre il Java Collections Framework, una sofisticata gerarchia di interfacce e classi per la gestione di gruppi di oggetti (Schildt, 2012).

Nel codice sorgente dell'applicazione sono state utilizzate le classi Calendar, Date, GregorianCalendar, HashMap, Timezone, Vector e l'interfaccia Map.

Strumenti software utilizzati

Per lo sviluppo dell'applicazione è stato utilizzato l'IDE Eclipse nella sua versione 4.7 "Oxygen".

Eclipse è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma distribuito dalla Eclipse Foundation. Una peculiarità di Eclipse è il suo essere incentrato sull'uso di plug-in, componenti software che permettono di estendere le funzionalità dell'IDE stesso. L'ambiente di sviluppo è interamente scritto in linguaggio Java, ma anziché basare la sua interfaccia grafica su Swing, si appoggia a SWT, librerie di nuova concezione che conferiscono ad Eclipse un'elevata reattività. (Wikipedia, 2017)

Per la gestione del database è stato utilizzato phpMyAdmin, raggiunto attraverso il servizio MySQL integrato in XAMPP.

Per il controllo versione (versioning) del codice è stato utilizzato GitHub.

GitHub è un servizio di hosting per progetti software. Il sito è utilizzato dagli sviluppatori open che caricano il codice sorgente dei loro programmi e lo rendono scaricabile dagli utenti. Questi ultimi possono interagire con lo sviluppatore tramite un sistema di issue tracking, pull request e commenti che permette di migliorare il codice del repository risolvendo bug o aggiungendo funzionalità (Wikipedia, 2017).

GitHub è completamente compatibile con l'IDE Eclipse. Per integrarlo all'interno dell'ambiente di sviluppo si è utilizzato il plug-in EGit (Eclipse Git Plug-in).

È possibile interagire con il servizio GitHub da terminale o da interfaccia grafica. Nello specifico si è scelto di utilizzare il software multipiattaforma GitKraken distribuito da axosoft, che fornisce una interfaccia grafica semplice ed intuitiva.

Bibliografia

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

Oracle. (2016). *Java™ Platform, Standard Edition 8*. Tratto da https://docs.oracle.com/javase/8/docs/api/

Schildt, H. (2012). *Java - La Guida Completa 8/ed.* McGraw-Hill.

Wikipedia. (2017). Tratto da https://it.wikipedia.org/wiki/Eclipse_(informatica)

Wikipedia. (2017). Tratto da https://en.wikipedia.org/wiki/GitHub