Panoramica del programma e scelte progettuali

Il programma è organizzato in sei classi: Mezzo, che prevede due sottoclassi quali Autovettura e Furgone, Archivio, Prenotazione, ArchivioPrenotazioni e GestioneAutonoleggio, ossia il *main*.

Oggetti istanza della classe Mezzo o di una delle sue sottoclassi sono inseriti nella classe Archivio attraverso l'inizializzazione di una variabile d'istanza di quest'ultima, ossia un vettore di oggetti di tipo Mezzo

Lo stesso avviene per oggetti di tipo Prenotazione, gestiti dalla classe ArchivioPrenotazioni

É stato scelto di creare sia la classe Prenotazione che Mezzo poiché sono state considerate due entità logicamente separate. Vengono poi legate dal fatto che Mezzo è una variabile d'istanza della classe Prenotazione.

Il *main* risulta essere così la classe che utilizza gli oggetti di tutte le altre classi realizzate. Volutamente i controlli sulle variabili d'istanza non vengono mai fatti nel *main*: infatti tale responsabilità è delegata esclusivamente alle classi interessate per una questione di riusabilitá del codice. Queste classi restituiscono poi *feedback* attraverso eccezioni e valori di ritorno, che vengono gestiti a questo punto dal *main*.

Manuale d'uso del programma

Il programma si apre chiedendo di inserire il nome del registro, se non è ancora presente viene avvertito l'utente che il file sarà creato al primo salvataggio.

Viene quindi mostrato il menù principale con i possibili comandi.
(I seguenti paragrafi presentano tutti i comandi del programma seguendo il menù dell'interfaccia).

```
[A]ggiungi un mezzo all'archivio
```

Se si desidera aggiungere un nuovo mezzo viene chiesto se questo sarà un Furgone o un'Autovettura, a questo punto si procede con la compilazione dei campi richiesti in base alla scelta dell'utente. Il mezzo non viene quindi inizializzato fintantoché tutti i campi non siano stati completati.
In un primo momento all'utente vengono indicate le possibili opzioni fra cui scegliere. É stata comunque gestita la possibilitá che l'utente inserisca parametri non corretti delegandone la responsabilità al costruttore delle classi in questione.
Nel momento in cui i parametri sono stati raccolti attraverso input dell'utente e si cercherà di stanziare l'oggetto, il costruttore indicherà eventuali errori che, gestiti nel main, l'utente potrà correggere

Una volta inizializzato il mezzo, questo viene aggiunto all'archivio, e se l'operazione è andata a buon fine, viene restituito un valore positivo al *main*.

```
[R]imuovi un mezzo dall'archivio
```

Prima di rimuovere un mezzo, viene fatto un controllo nell'archivio delle prenotazioni se non siano presenti delle prenotazioni per il mezzo in questione. Se il mezzo risulta essere già prenotato allora non sarà possibile rimuoverlo dall'archivio dei mezzi dell'autonoleggio, in caso contrario sì.

[V]isualizza le informazioni di tutti i mezzi a disposizione
Se l'archivio dei mezzi gestiti dall'autonoleggio risulta non essere vuoto allora sarà possibile visualizzare tutti i mezzi presenti.

[P]renota un mezzo
Per prenotare un mezzo è necessario inserire una data valida, una targa valida, ossia di un mezzo presente nell'archivio, e il nome a cui intestare la prenotazione. A questo punto la prenotazione verrà inserita nell'archivio delle prenotazioni.

[M]ostra mezzi, per categoria, disponibili in una data a scelta
Inserendo una data valida e selezionando la tipologia di mezzo desiderata, verrà restituito al *main* una lista di mezzi, qualora questi risultino non essere prenotati quel tal giorno. Se l'archivio non dovesse contenere mezzi di quella categoria o se tutti i mezzi di questa dovessero risultare essere prenotati, allora il programma restituirà un messaggio d'errore.

[C]ancellazione della prenotazione
Dopo aver controllato che la prenotazione esista effettivamente all'interno dell'archivio delle prenotazioni, sarà possibile eliminarla. Se l'operazione sarà andata a buon fine, verrà ritornato al *main* un *feedback* positivo.

[I]ndicare il mezzo per visualizzare tutte le relative prenotazioni
Effettuato i controlli relativi all'esistenza del mezzo all'interno dell'archivio dei mezzi gestiti dall'autonoleggio e la presenza di prenotazioni dello stesso nell'archivio delle prenotazioni, queste verranno passate al *main* che le stamperà.

[S]alvataggio su file
Con questo comando è possibile salvare le modifiche, qualora queste siano state apportate.

[U]scita
Analogamente al comando precedente, sarà possibile salvare le modifiche prima di uscire dal programma. In questo caso, però, sarà l'utente a gestire questa opzione: infatti, si potrà decidere anche di abbandonare la sessione senza apportare alcuna modifica ai file.

Descrizione delle classi e dei metodi

Nelle prossime pagine procederò illustrando tutte le classi, eccetto il *main*, e i loro rispettivi metodi.

Classe Mezzo

Public Mezzo(String t, String m) throws IllegalArgumentException:

Costruttore del mezzo a cui viene passata la targa e il modello, se le stringhe passate sono vuote viene invece lanciata un'eccezione (gestita nel *main*). Le variabili d'istanza vengono poi inizializzate in maiuscolo.

Public String getTarga(): ritorna la targa in stringa.

Public String getModello(): ritorna il modello del mezzo in stringa.

Public String toString(): ridefinizione del metodo *toString*, ritorna la stringa con la targa e modello.

Public boolean equals(Object o): ridefinizione del metodo *equals*: il confronto si basa sulla targa.

Classe Autovettura, sottoclasse della Classe Mezzo

Public Autovettura(String t, String m, String posti, String alimentazione, String tipologia) throws IllegalArgumentException:
Costruttore della classe Autovettura, dopo aver verificato che i posti siano 2, 4 o 5, che la tipologia dell'alimentazione sia diesel o benzina, che l'autovettura sia un'utilitaria, media o berlina, viene inizializzato l'oggetto. In caso contrario viene lanciata un'eccezione (gestita nel *main*).

Public String getNumPosti(): ritorna il numero di posti.

Public String getAlimentazione(): ritorna la tipologia dell'alimentazione in formato stringa.

Public String getTipologiaAuto(): ritorna la tipologia dell'autovettura in formato stringa.

Public String toString(): ritorna una stringa contenente le informazioni relative alla targa, modello, il numero di posti, la tipologia dell'alimentazione e dell'autovettura.

Classe Furgone, sottoclasse della Classe Mezzo

Public Furgone(String targa, String modello, String pat, String d) throws IllegalArgumentException:
Costruttore della classe Furgone. Dopo aver verificato che la tipologia della patente sia B o C (indipendentemente che sia scritta in minuscolo o maiuscolo) e che la dimensione del furgone sia piccola, media o grande, viene inizializzato l'oggetto. Se qualche campo non passa il controllo, allora viene lanciata un'eccezione (gestita nel *main*).

Public String getPatente(): ritorna la patente in formato stringa. Questa scelta è stata fatta per facilitare la gestione delle eccezioni di input nel *main*.

Public String getDimensioni(): ritorna la dimensione del furgone in formato stringa.

Public String toString(): ridefinizione del metodo *toString*. Ritorna in formato stringa le informazioni relative alla targa, modello, tipologia di patente e dimensione del furgone.

<u>Classe Archivio</u> serve per la gestione dei mezzi

Public Archivio(String nomefile):
Costruttore della classe Archivio. Sono inizializzate le variabili d'istanza *nomefile* e *modificato*.
Sempre all'interno del costruttore, recuperandone il valore attraverso il file di input, viene inizializzato il vettore *mezziGestiti* con le informazioni relative ai mezzi salvati.

Public boolean aggiungiNuovoMezzo(Mezzo m): metodo che permette di aggiungere un nuovo mezzo al vettore dei mezzi gestiti.
Prima viene scansionato il vettore, se il mezzo passato come argomento corrisponde ad un elemento contenuto all'interno del vettore allora viene ritornato false, in caso contrario il mezzo viene aggiunto al vettore e la variabile d'istanza *modificato* viene settata a *true*, in modo tale tener traccia delle modifiche apportate durante la sessione e se è necessario quindi fare la sovrascrittura del file di output alla chiusura del programma o al comando salvataggio.

Public boolean rimuoviMezzo(String targa): metodo che serve per rimuovere un mezzo dall'archivio. Passando al metodo la targa, viene in un primo momento verificata la corrispondenza di questa con quella di un qualche elemento del vettore *mezziGestiti*. Se effettivamente esiste, viene memorizzata la posizione di tale elemento nel vettore e in un secondo momento viene eliminato. Come nel caso precedente la variabile d'istanza *modificato* è settata a *true*.

Public Mezzo selezionaElemento(String t): metodo che serve per restituire un mezzo specifico. In primis viene inizializzato un mezzo a *null*, se poi la targa passata come argomento al metodo trova corrispondenza nel vettore *mezziGestiti* allora il metodo ritorna al *main* un mezzo altrimenti ritorna *null*.

Public Vector<Mezzo> elencoMezziDisposizione(): metodo che ritorna il vettore *mezziGestiti*.

Public Vector <Mezzo> selezionaPerCategoria(String c): è creato un vettore vuoto di mezzi, che sarà successivamente restituito al *main*, questo è poi riempito con la tipologia del mezzo (che viene passata attraverso l'argomento al metodo): si effettua così una selezione nel vettore *mezziGestiti*: se è richiesto la lista delle autovetture il vettore vuoto sarà popolato con le autovetture presenti nell'archivio; stessa cosa per i furgoni. Se all'interno di *mezziGestiti* non sono presenti mezzi di quella tipologia allora verrà ritornato al *main null*.

Public boolean daSalvare(): metodo che ritorna il valore della variabile *modificato*. Quindi se è necessario sovrascrivere il file di output.

Public boolean salva(): metodo che scrive sul file di output, in base al valore restituito dal metodo *daSalvare()* e resetta la variabile *modificato* a false.

Public String toString(): ritorna il vettore *mezziGestiti* in formato stringa.

Classe Prenotazione serve per la creazione di nuove prenotazioni che verranno poi inserite nella classe ArchivioPrenotazioni.

Public Prenotazione(Mezzo m, String d, String n):
Costruttore della prenotazione. Le variabili d'istanza sono il mezzo, la data e il nome a cui è fatta la prenotazione.

Public Mezzo getMezzo(): ritorna il mezzo.

Public Mezzo getDataPrenotazione(): ritorna la data della prenotazione.

Public Mezzo getNomePrenotazione(): ritorna il nome a cui è stata fatta la prenotazione.

Public boolean equals(Object o): ridefinizione del metodo *equals*. Il confronto è fatto sulla base del mezzo e della data.

Public String toString(): **)**: ridefinizione del metodo *toString*. Ritorna in formato stringa le informazioni relative alla prenotazione: come il mezzo, chi prenota e per che data.

Classe ArchivioPrenotazioni serve per la gestione delle prenotazioni.

Public ArchivioPrenotazioni(String nomefile): sono inizializzate le variabili d'istanza *nomefile* e *modificato*. Sempre all'interno del costruttore, recuperandone il valore attraverso il file di input, viene inizializzato il vettore *listaPrenotazioni* con le informazioni relative prenotazioni registrate o che verranno salvati al primo salvataggio o alla chiusura del programma.

Public boolean checkPrenotazione(Mezzo m, String data): metodo che serve per controllare che il mezzo sia disponibile nella data richiesta. Al suo interno viene creato un oggetto di prenotazione (con una stringa vuota per quel che riguarda il nome di chi riserva la prenotazione), in quanto il controllo è fatto sulla data e il mezzo. Il valore di ritorno sarà *true* se non è stata trovata corrispondenza della prenotazione in questione nel vettore *listaPrenotazioni*, altrimenti *false* se la prenotazione risulta essere già stata inserita.

Public Vector<Mezzo> getCategoriaData(String data, String categoria, Vector<Mezzo> vm): metodo che restituisce la lista dei mezzi appartenenti ad una certa categoria in una certa data. In un primo momento viene creato un vettore vuoto, che sarà riempito con la tipologia di mezzi richiesti dall'utente. Se la data passata come parametro corrisponde a quella di un elemento presente nella *listaPrenotazioni*, e allo stesso tempo la targa del mezzo della stessa prenotazione coincide con quella passata al metodo, allora la variabile locale *reserved* viene settata a *true*. A questo punto è effettuato un controllo sulla tipologia dei mezzi che sono passati come argomento (questi sono i mezzi gestiti dall'archivio. Se la categoria richiesta è autovettura, allora vengono aggiunti alla lista di ritorno le autovetture che non sono *reserved*. La stessa operazione è fatta per i furgoni. Se la lista risulta essere vuota, allora viene restituito un valore nullo.

Public Prenotazione rimuoviPrenotazione(Mezzo m, String data): metodo che gestisce la rimozione della prenotazione. Restituisce un valore positivo se la rimozione è andata a buon fine, quindi se ha trovato la prenotazione all'interno della *listaPrenotazioni* e setta la variabile *modificato* a *true*. Altrimenti ritorna *false*, quindi se il vettore è vuoto o non è presente la prenotazione richiesta in esso.

Public Prenotazione visualizzaPrenotazione(Prenotazione p): metodo che restituisce la prenotazione richiesta, se questa esiste in *listaPrenotazioni*, altrimenti restituisce un valore nullo.

Public Vector<Prenotazione> cronologiaprenotazioniMezzo(String t): metodo che ritorna la lista di tutte le prenotazioni relative a un singolo mezzo. Qui viene creato un vettore di prenotazioni che sarà poi restituito al *main*, questo poi sarà popolato, qualora la *listaPrenotazioni* non sia vuota, con le prenotazioni che risultano avere la targa del mezzo in esso corrispondente a quella passata al metodo.

Public boolean daSalvare(): metodo che ritorna il valore della variabile *modificato*. Quindi se è necessario la sovrascrittura del file di output.

Public boolean salva(): metodo che scrive sul file di output, in base al valore restituito dal metodo *daSalvare()* e resetta la variabile *modificato* a false.