ESAME DI FONDAMENTI DI INFORMATICA T-2 del 15/1/2025
Proff. E. Denti – R. Calegari – A. Molesini
Tempo a disposizione: 3h30

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)
NOME CARTELLA PROGETTO: CognomeNome-matricola (es. RossiMario-0000123456)
NOME ZIP DA CONSEGNARE: CognomeNome-matricola.zip (es. RossiMario-0000123456.zip)
NOME JAR DA CONSEGNARE: CognomeNome-matricola.jar (es. RossiMario-0000123456.jar)

Si devono consegnare DUE FILE: *l'intero progetto Eclipse* e *il JAR eseguibile*

Si ricorda che compiti *non compilabili*, o che *non passino almeno 2/3 dei test* o siano *palesemente lontani da 18/30* NON SARANNO CORRETTI e causeranno la verbalizzazione del giudizio "RESPINTO".

È richiesto di sviluppare un'applicazione per il preventivo del costo di un noleggio auto. L'applicazione deve innanzitutto proporre all'utente le città in cui è possibile noleggiare l'auto, indi – dopo che la città sarà stata scelta – popolare un'apposita combo con le agenzie disponibili in tale città. Dovrà poi consentire all'utente di inserire giorno e ora di ritiro e di riconsegna dell'auto, il tipo di auto e di specificare se desideri lasciare l'auto in una città diversa (non specificata). Acquisiti tali dati, dovrà innanzitutto verificare che l'agenzia di noleggio prescelta sia aperta nel giorno e ora indicati come inizio del noleggio, e se sì calcolare e mostrare a video il preventivo corrispondente.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

In ogni ***città*** possono essere presenti più ***agenzie***, ognuna identificata da una descrizione (nome) univoca: ogni agenzia è caratterizzata dai suoi ***orari di apertura***, distinti per giorni infrasettimanali (da lunedì a venerdì), sabato, domenica.

Un orario di apertura può avere tre forme:
- La costante CHIUSO
- Uno ***slot orario***, della forma HH:MM-HH:MM (orario minimo: 00:00; orario massimo: 23:59)
- Due slot orari separati da "/", secondo il pattern HH:MM-HH:MM/HH:MM-HH:MM, per esprimere orari di apertura non continuati, con pausa intermedia (tipicamente, mattino/pomeriggio).

Il tipo di auto può essere A (mini), B (piccole), C (medie), D (lusso).

Il sistema tariffario prevede, per ogni tipo di auto:
- Una *tariffa giornaliera*, da applicarsi ogni 24h di noleggio (es. dalle 15 di un giorno alle 14:59 del giorno successivo)
- Una *tariffa speciale weekend*, omnicomprensiva, per noleggi compresi fra venerdì e domenica (inclusi)
- Una *sovrattassa fissa*, detta di "drop off", per noleggi con *riconsegna in una città diversa* da quella di ritiro.

Per calcolare il costo del noleggio l'applicazione deve innanzitutto calcolare la durata in giorni del noleggio: eventuali frazioni comportano l'addebito di un'intera giornata ulteriore (ad esempio, un noleggio di 1 giorno e 1 minuto deve essere tariffato come 2 giorni). L'importo base si ottiene quindi moltiplicando la durata in giorni per la tariffa giornaliera. Nel caso in cui il noleggio ricada interamente in un weekend (inizio e termine fra venerdì e domenica dello stesso fine settimana), si potrà applicare la tariffa speciale prevista, se più conveniente rispetto alla tariffa standard. Al costo così calcolato dovrà infine essere aggiunta la sovrattassa di drop-off, nel caso di riconsegna in altra città.

Il file di testo agencies.txt contiene, nel formato descritto più oltre, l'elenco delle agenzie disponibili. L'applicazione dovrà:

a) leggere tale file e caricare i dati nelle opportune strutture-dati interne
b) permettere all'utente, tramite la GUI, di introdurre tutti i dati necessari per il preventivo e mostrare il risultato del calcolo, specificando anche in dettaglio la durata in giorni, se sia stata applicata la tariffa weekend e se sia stata aggiunta la sovrattassa di drop-off; in caso di azioni errate, la GUI dovrà avvisare l'utente tramite appositi dialoghi.

TEMPO STIMATO PER SVOLGERE L'INTERO COMPITO: 2h15 – 3h

PARTE 1 – Modello dei dati: Punti 5 [TEMPO STIMATO: 20-30 minuti]

PARTE 2 – Persistenza: Punti 10 [TEMPO STIMATO: 40-60 minuti]

PARTE 3 – Grafica: Punti 15 [TEMPO STIMATO: 75-90 minuti]

NUMERO MINIMO DI TEST CON SUCCESSO PERCHÉ IL COMPITO SIA CORRETTO

Considerando solo OpeningTime, i due reader e il controller 40 su 56

Considerandoli tutti: 56 su 83

JAVAFX – Parametri run configuration nei LAB

```
--module-path "C:\applicativi\moduli\javafx-sdk-21.0.2\lib"
--add-modules javafx.controls
```

Cose da ricordare

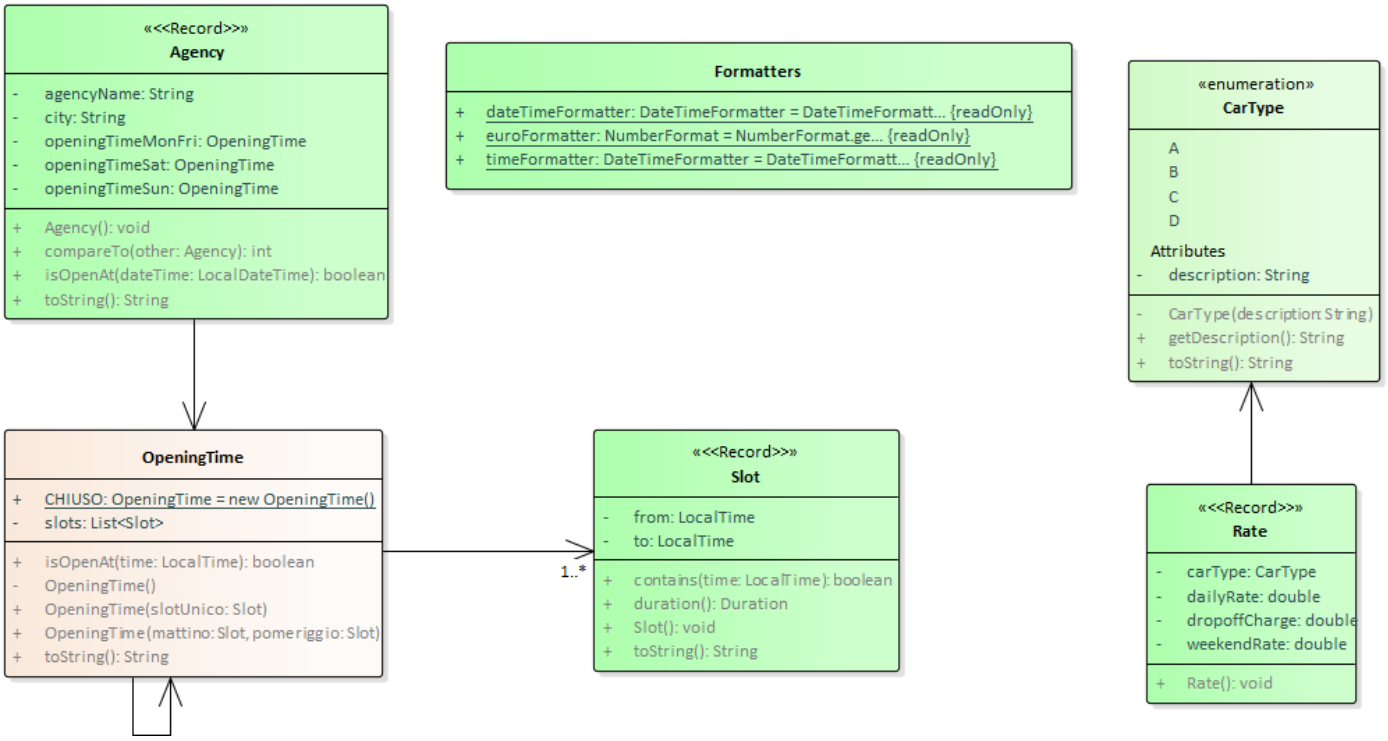
- salva costantemente il tuo lavoro: l'informatica a volte può essere "subdolamente ostile"..
- in particolare: se ora compila e stai per fare modifiche, salva la versione attuale (non si sa mai)

Checklist di consegna

- Hai fatto un **JAR eseguibile**, che contenga cioè l'indicazione del main?
- Hai controllato che **si compili** e **ci sia tutto**? [NB: non includere il PDF del testo]
- Hai **rinominato** IL PROGETTO, lo ZIP e il JAR esattamente come richiesto?
- Hai **chiamato** la cartella del progetto esattamente come richiesto?
- Hai fatto un **unico file ZIP (NON .7z, rar o altri formati)** contenente l'intero progetto? In particolare, ti sei assicurato di aver incluso tutti i file .java (e non solo i .class)?
- **Hai consegnato DUE file distinti, ossia lo ZIP col progetto e il JAR eseguibile?**
- Su EOL, hai **premuto** il tasto "CONFERMA" per inviare il tuo elaborato?

Parte 1 – Modello dei dati (punti: 5)

Package: autonoleggio.model **[TEMPO STIMATO: 20-30 minuti]**



SEMANTICA:

- L'enumerativo ***CarType*** definisce semplicemente i quattro tipi di auto, con i relativi metodi accessori
- La classe ***Formatters*** contiene alcuni formattatori per valuta (euro), ora, data e ora in standard italiano
- Il record ***Slot*** specifica una fascia oraria di apertura, da un certo orario a un altro. Il costruttore effettua i necessari controlli sugli argomenti in ingresso. Oltre a un'apposita *toString*, sono forniti i seguenti metodi:
  - *contains* verifica se l'orario fornito come argomento sia contenuto nell'intervallo specificato dallo slot
  - *duration* restituisce la durata dell'intervallo stesso
- Il record ***Agency*** specifica i dati di un'agenzia di autonoleggio (nome città, nome univoco agenzia, orari di apertura rispettivamente da lunedì a venerdì, il sabato, la domenica): il costruttore effettua accurati controlli sugli argomenti in ingresso, mentre un'apposita *compareTo* garantisce la confrontabilità (e quindi l'ordinamento) per città e in subordine per nome agenzia e un'adeguata *toString* emette una rappresentazione sintetica dell'agenzia stessa (città + nome agenzia). Il metodo *isOpenAt* consente di verificare se l'agenzia sia aperta nel giorno e ora indicato, fornito come istanza di ***LocalDateTime***.
- Il record ***Rate*** specifica i dati di una tariffa (tipo di auto, tariffa giornaliera, tariffa weekend, costo di drop-off): il costruttore effettua i necessari controlli sugli argomenti in ingresso.
- La classe ***OpeningTime*** (da completare) rappresenta l'orario di apertura di un'agenzia. Come specificato nel Dominio del Problema, esso può assumere tre forme:
  - la costante pubblica statica ***CHIUSO***
  - un unico slot orario (es. per esprimere agenzie aperte con orario continuato, o solo mattina/solo pom.)
  - due slot orari (es. per esprimere agenzie aperte mattina e pomeriggio, con pausa intermedia): in questo caso, *il primo slot deve precedere interamente il secondo.*

  La costante CHIUSO è costruita da un costruttore privato (fornito), mentre **gli altri due costruttori (da fare)** devono gestire il caso del singolo slot e dei due slot, rispettivamente. Entrambi devono verificare che gli

argomenti forniti non siano nulli, emettendo nel caso ***NullPointerException*** con apposito messaggio; il costruttore a due argomenti deve anche verificare che il secondo slot sia successivo al primo.

- Il metodo *isOpenAt* **(da realizzare)** deve verificare se l'agenzia sia aperta all'orario fornito come argomento, distinguendo e gestendo opportunamente i tre casi
- Un'apposita *toString* **(da realizzare)** deve emettere la rappresentazione stampabile adeguata, costituita o dalla stringa "chiuso" o da uno o più slot, eventualmente separati da '/'.

Parte 2 – Persistenza (punti: 10)

Package: autonoleggio.persistence **[TEMPO STIMATO: 40-60 minuti]**

Il file di testo agencies.txt contiene l'elenco delle agenzie disponibili, nel formato sotto descritto e illustrato negli esempi in calce. Esso specifica un'agenzia per ogni riga, tramite una serie di campi separati da virgole. Nell'ordine sono riportati: città, descrizione agenzia, orario di apertura per i giorni infrasettimanali da lunedì a venerdì, orario di apertura del sabato, orario di apertura della domenica. Gli orari di apertura hanno la seguente forma:

- una delle tre stringhe "lun-ven", "sab", "dom", seguita da uno o più spazi
- poi, o la parola "chiuso"
- oppure un singolo slot orario, della forma HH:MM-HH:MM, senza spazi intermedi
- oppure due slot orari, ciascuno della forma HH:MM-HH:MM, separati da "/", senza spazi intermedi

```
BOLOGNA, Aeroporto Marconi, lun-ven 08:00-23:59, sab 08:00-23:59, dom 08:00-23:59
BOLOGNA, Stazione centrale, lun-ven 09:00-13:00/14:00-18:00, sab chiuso, dom chiuso
MODENA, centro, lun-ven 08:30-12:00/15:00-18:00, sab chiuso, dom chiuso
...
```

SEMANTICA:

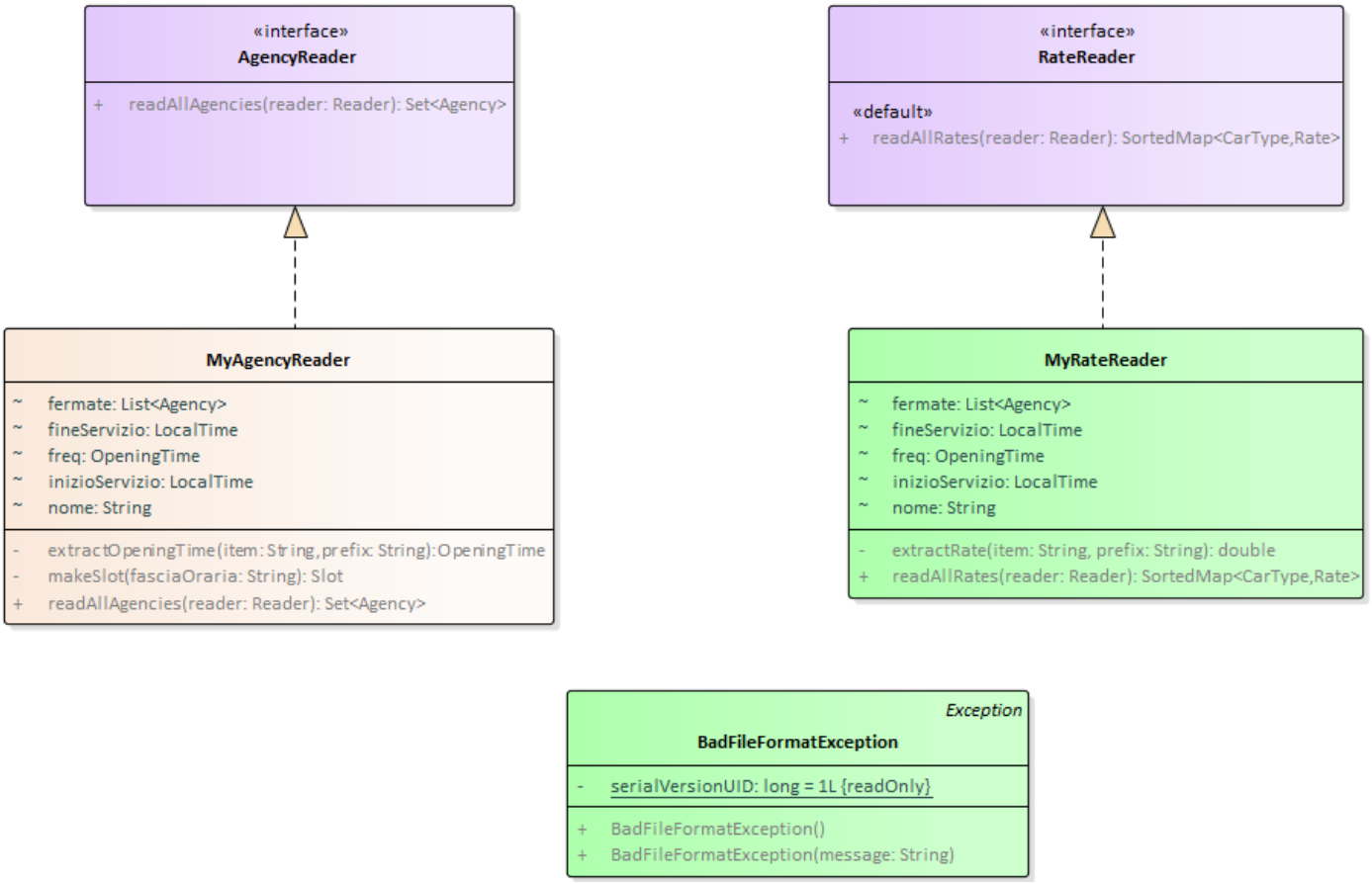
a) L'interfaccia ***AgencyReader*** (fornita) dichiara il metodo *readAllAgencies,* che legge dal ***Reader*** fornito (già aperto) i dati e restituisce un ***Set<Agency>*** contenente tanti oggetti ***Agency*** quante le righe lette dal file; lancia ***NullPointerException*** nel caso di reader nullo, ***BadFileFormatException*** con opportuno messaggio d'errore in caso di problemi nel formato del file, o ***IOException*** in caso di altri problemi di I/O.

b) La classe ***MyAgencyReader*** (da implementare) implementa ***AgencyReader***:

- non c'è alcun costruttore
- Il metodo *readAllAgencies* (da realizzare) deve fare uso del metodo privato *extractOpeningTime* per estrarre dalla sottostringa, formattato come sopra descritto, un orario di apertura. Nel caso di reader nullo dev'essere lanciata apposita ***NullPointerException*** con adeguato messaggio d'errore.
- il metodo ausiliario *extractOpeningTime (da realizzare)* riceve la sottostringa (ad esempio, "`lun-ven 08:00-23:59`", "`dom 08:00-23:59`", "`sab chiuso`", "`lun-ven 09:00-13:00/14:00-18:00`", etc.) e il prefisso da considerare (uno dei tre "`lun-ven`", "`sab`", "`dom`") e restituisce l'oggetto ***OpeningTime*** corrispondente, perfettamente configurato, catturando eventuali ***IllegalArgumentException*** da rilanciare esternamente come ***BadFileFormatException***. Si avvale a sua volta del metodo privato ausiliario *makeSlot*.
- il metodo ausiliario *makeSlot (da realizzare)* riceve la sottostringa che rappresenta una fascia oraria nel formato HH:MM-HH:MM e restituisce l'oggetto ***Slot*** corrispondente, adeguatamente configurato; verifica che la stringa sia correttamente formattata, catturando eventuali ***DateTimeParseException*** e/o ***IllegalArgumentException***, da rilanciare esternamente come ***BadFileFormatException***.

Le tariffe previste per i diversi tipi di auto sono elencate nel file rates.txt, che tuttavia non deve essere letto perché, per semplicità, i relativi dati di default sono già incapsulati nel codice dell'interfaccia ***RateReader*** seguente. Esso è quindi incluso nello start kit solo per completezza, per lo studente che voglia esercitarsi in futuro.

c) L'interfaccia ***RateReader*** (fornita) dichiara il metodo *readAllRates,* che legge dal ***Reader*** fornito (già aperto) i dati e restituisce una ***SortedMap<CarType,Rate>*** contenente tante ***Rate*** quante le righe lette dal file, ossia una per ogni tipo di auto; lancia ***BadFileFormatException*** in caso di problemi nel formato del file, o ***IOException*** in caso di altri problemi di I/O. Questa interfaccia contiene già un'implementazione di default (fasulla) di tale metodo, quindi NON deve essere implementata: è usata dall'***AutoNoleggioApp*** così com'è. [Lo studente che in futuro desideri esercitarsi potrà implementare la classe ***MyRateReader***, secondo l'usuale schema, modificando poi l'app principale affinché usi tale classe anziché l'implementazione di default fornita in ***RateReader***]

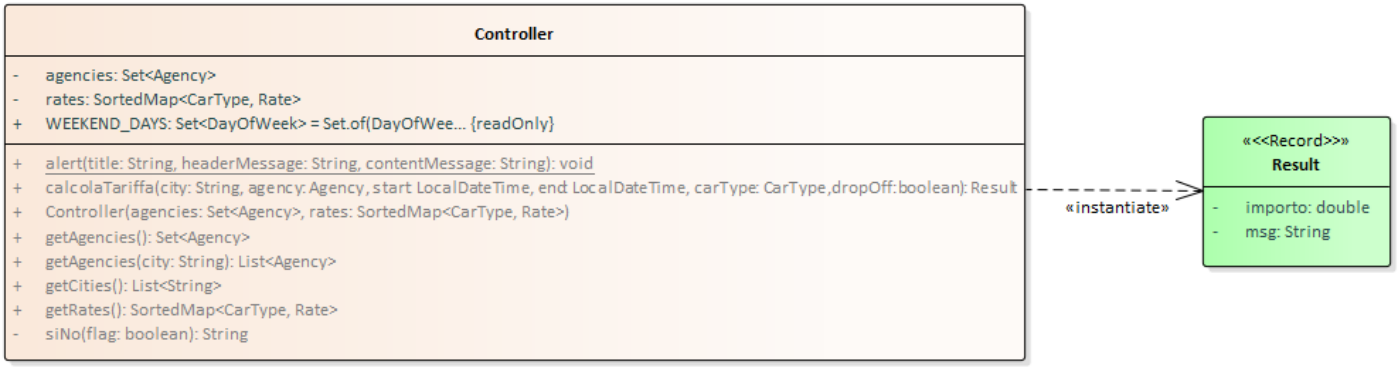


PARTE 3: SEGUE NELLA PAGINA SUCCESSIVA → → →

Parte 3 (punti: 15)

Package: autonoleggio.controller **[TEMPO STIMATO: 25-35 minuti]** ***(punti 6)***

Il controller costituisce il nucleo centrale di questa applicazione: crea e mantiene le strutture dati, e gestisce la logica di funzionamento. A tal fine è organizzato come segue:



SEMANTICA:

a) Il record ***Result*** (fornito) specifica la coppia (importo, messaggio), utile per accoppiare il risultato del calcolo a un messaggio esplicativo che spieghi come tale risultato è stato ottenuto.

b) La classe ***Controller*** **(da completare)** implementa già le funzionalità di base: rimane da realizzare solo il metodo *calcolaTariffa*, che deve implementare la logica descritta nel Dominio del Problema. Più precisamente:

- Il costruttore riceve il **Set<Agency>** e la **SortedMap<CarType,Rate>** restituite dalla persistenza: effettua i necessari controlli e memorizza internamente i riferimenti alle strutture dati ricevute.
- Oppurtuni metodi consentono di recuperare l'insieme delle agenzie (*getAgencies*), la mappa delle tariffe (*getRates*), la lista delle città (*getCities*) e la lista delle agenzie di una data città (*getAgencies(String city)*)
- Il metodo ***calcolaTariffa (da realizzare)*** riceve la città (una stringa), l'oggetto ***Agency*** che rappresenta l'agenzia, due ***LocalDateTime*** che rappresentano ora e giorno rispettivamente di inizio e di fine noleggio, un ***CarType*** che rappresenta il tipo di veicolo, e un ***boolean*** che specifica se debba essere applicata la sovrattassa di drop-off (vero = sì, falso = no). Il metodo deve:
  - Verificare accuratamente che tutti gli argomenti ricevuti siano non nulli, lanciando eventualmente una ***NullPointerException*** con apposito e specifico messaggio d'errore
  - Verificare che la città non sia una stringa vuota o blank, lanciando in caso ***IllegalArgumentException*** con apposito e dettagliato messaggio d'errore
  - Verificare che giorno e ora di fine noleggio non precedano quello di inizio noleggio, lanciando anche in tal caso ***IllegalArgumentException*** con apposito e dettagliato messaggio d'errore
  - Calcolare la durata del noleggio e applicare, in base ai giorni della settimana, la tariffa opportuna, ricordando che la tariffa speciale weekend va applicata solo se più conveniente rispetto alla tariffa standard; alla tariffa così ottenuta dovrà essere aggiunta la sovrattassa apposita nel caso di riconsegna in città diversa (drop-off).

  Il metodo deve restituire un'opportuna istanza di ***Result***, il cui messaggio abbia la forma:

  "`Giorni: N, tariffa weekend: XX, dropOff: XX`"

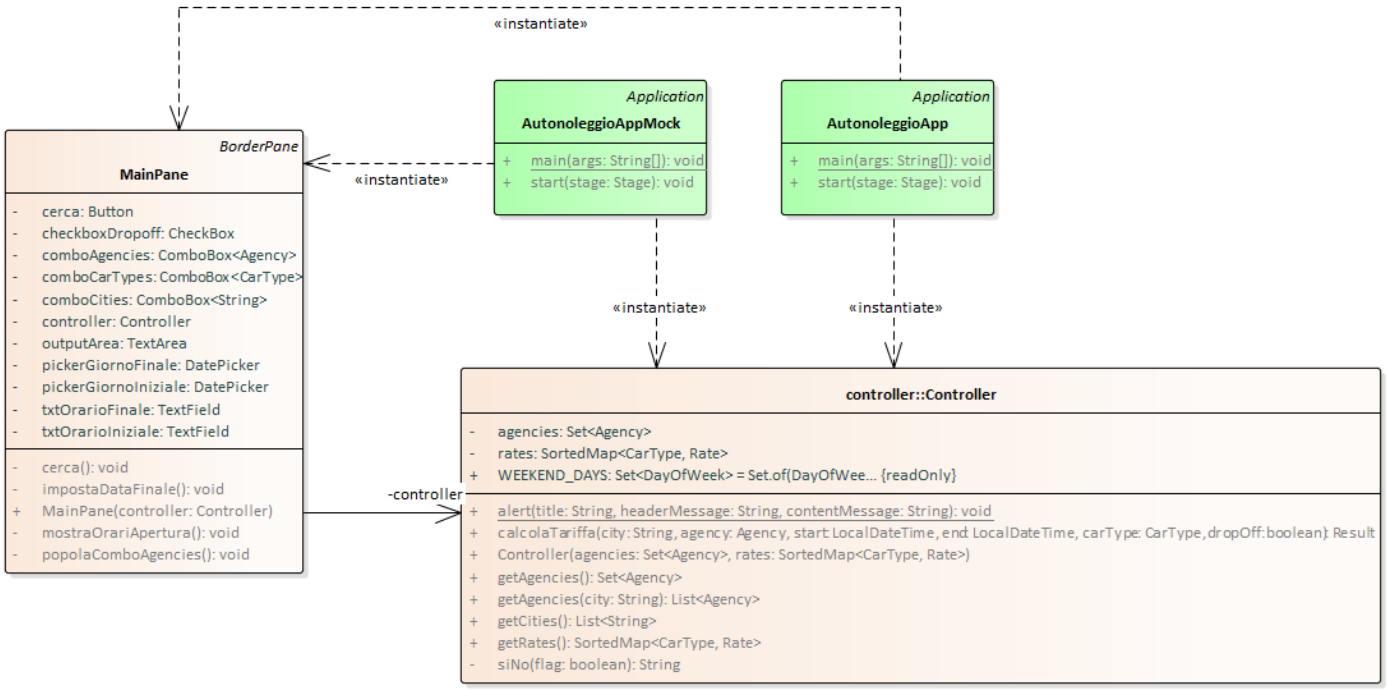
  dove *N* è il numero di giorni e *XX* è una delle due stringhe "`sì`" o "`no`".

Il metodo statico *alert*, utilizzabile dal ***MainPane*** quando è attiva la grafica, consente di far comparire all'utente una finestra di dialogo che mostri il messaggio d'errore specificato.

Package: autonoleggio.ui **[TEMPO STIMATO: 50-55 minuti]** ***(punti 9)***

La classe ***AutonoleggioApp*** (fornita) costituisce l'applicazione JavaFX che si occupa di aprire i file, creare il controller e incorporare il ***MainPane***. Per consentire di collaudare la GUI anche in assenza / in caso di malfunzionamento della parte di persistenza, è possibile avviare l'applicazione mediante la classe ***AutonoleggioAppMock***.

L'interfaccia utente è illustrata nelle figure seguenti e segue il modello sotto illustrato*:*



- In alto vi sono le combo per la selezione della città (prima) e dell'agenzia (poi)
- Al centro vi sono i controlli per l'inserimento di data e ora di inizio e fine noleggio, del tipo di auto e per l'eventuale scelta di riconsegnare l'auto in una città diversa, nonché il pulsante CERCA che attiva il calcolo
- In basso vi è l'area di output, che mostra i messaggi destinati all'utente.

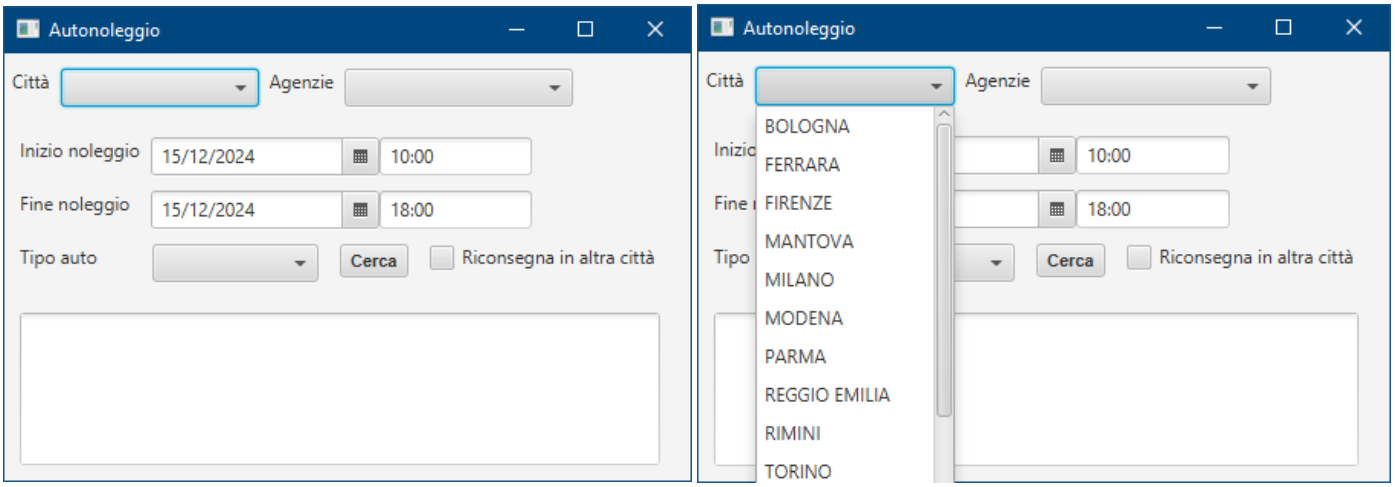


Fig. 1: situazione iniziale della GUI (a sinistra) ed elenco delle città (a destra).

Cliccando sulla combo città si può scegliere la città di proprio interesse: l'applicazione risponderà caricando nella combo agenzie le agenzie di quella specifica città (Fig. 2, sinistra). Selezionandone una, l'area di testo dovrà mostrare i rispettivi orari di apertura, correttamente formattati (Fig. 2, destra).

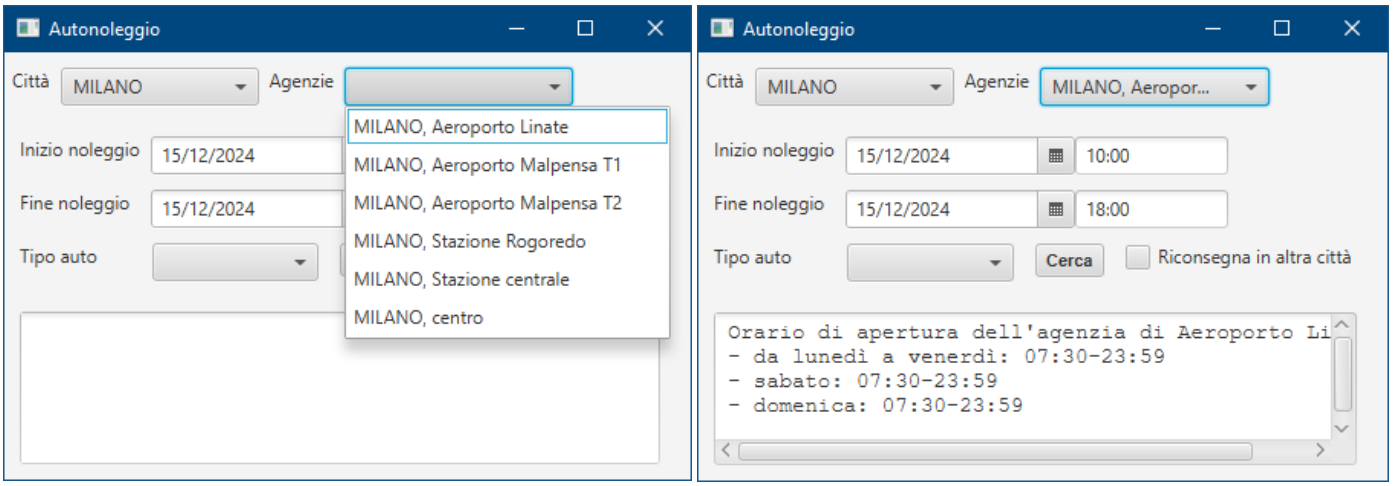


Fig. 2: selezione agenzie e visualizzazione orari di apertura dell'agenzia prescelta.

Scegliendo data e ora di inizio e fine noleggio, nonché il tipo di auto, è possibile farsi fare il preventivo premendo il pulsante "CERCA" (Fig.3, sinistra); si può facilmente vedere la differenza di costo nel caso di riconsegna in altra città, semplicemente selezionando l'apposita opzione (Fig. 3, destra).

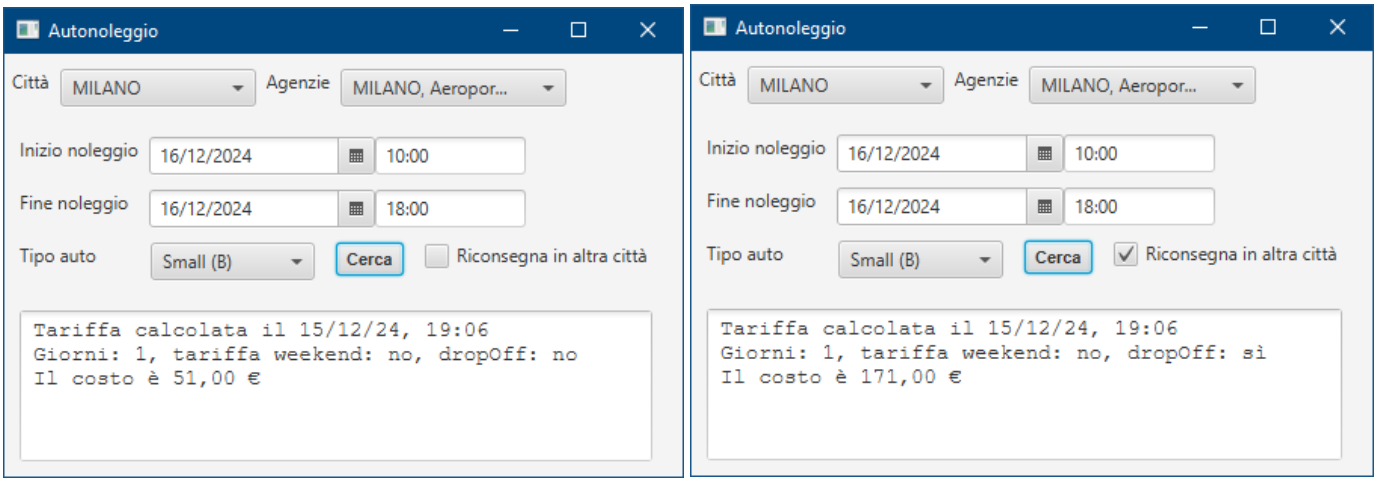


Fig. 3: calcolo del costo del noleggio (senza/con riconsegna in altra città)

Nel caso in cui il noleggio avvenga interamente in un weekend, dovrà essere applicata la tariffa apposita (Fig. 4)

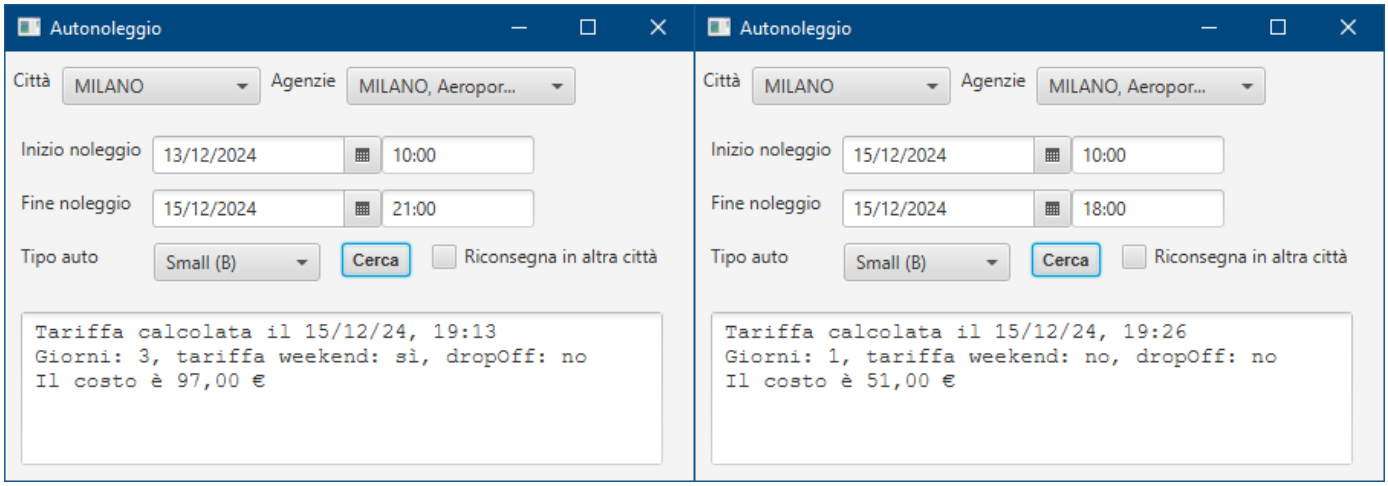


Fig. 4: caso di applicazione della tariffa weekend (sinistra) e di non applicazione perché non conveniente (destra)

L'applicazione deve segnalare errore, con apposito dialogo dotato di adeguato messaggio d'errore (Fig. 5), se:

- non è stata selezionata la città nell'apposita combo
- non è stata selezionata l'agenzia nell'apposita combo
- non sono state selezionate la data e/o l'ora di inizio e/o fine noleggio
- non è stato selezionato il tipo di auto nell'apposita combo
- l'agenzia di noleggio è chiusa nel giorno/ora di inizio noleggio (Fig 5, sotto).

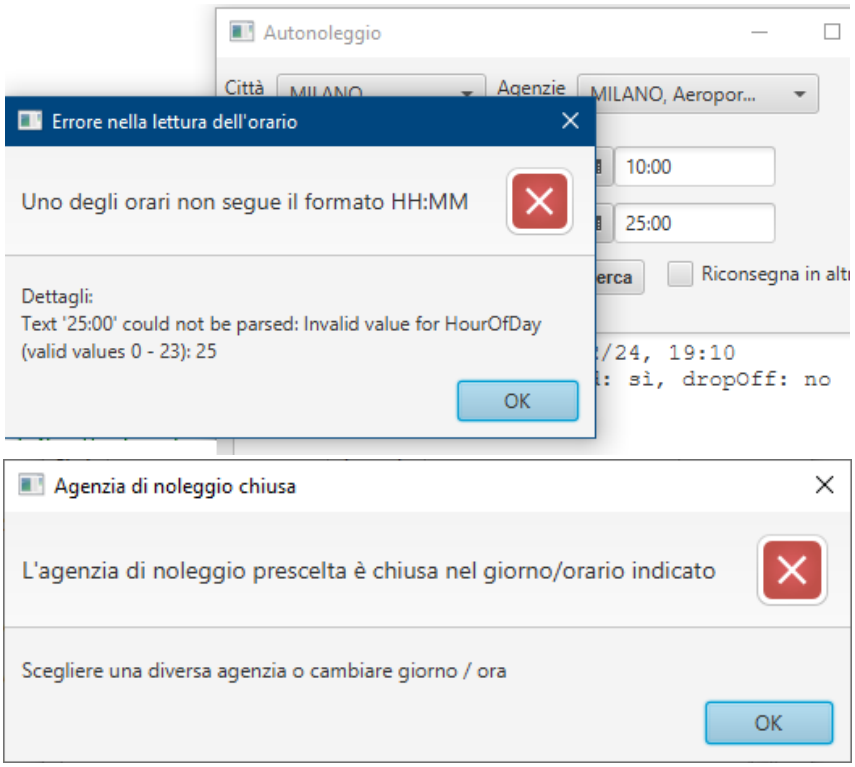


Fig. 5: segnalazioni di errore: sopra, orario illegale; sotto, agenzia chiusa al momento del ritiro dell'auto.

Il MainPane è fornito *parzialmente realizzato*: è presente tutta la parte strutturale, mentre rimangono da realizzare i gestori degli eventi (metodi privati *impostaDataFinale, popolaComboAgencies, mostraOrariApertura, cerca)*.

In particolare:

- *impostaDataFinale* deve impostare sul calendario relativo alla data di fine noleggio lo stesso giorno selezionato dall'utente nel calendario relativo alla data di inizio noleggio
- *popolaComboAgencies* deve recuperare la città selezionata nell'apposita combo e popolare, di conseguenza, *comboAgencies* con le sole agenzie relative alla città selezionata; in caso di errore nella selezione della città (valore nullo), dovrà emettere tramite *alert* apposito messaggio d'errore all'utente
- *mostraOrariApertura* deve recuperare l'agenzia selezionata e mostrare a video, nell'area di output, i relativi orari di apertura, nel formato mostrato in figura; in caso di errore nella selezione dell'agenzia (valore nullo), dovrà emettere tramite *alert* apposito messaggio d'errore all'utente
- *cerca* deve recuperare dalla GUI tutti i dati utili, controllarli accuratamente e quindi controllare se l'agenzia sia aperta nel giorno e ora indicati per il ritiro dell'auto: in caso positivo, provvede a invocare il metodo *calcolaTariffa* del ***Controller*** e mostrare a video, nell'area di output, il dettaglio del calcolo (ovvero il contenuto del messaggio interno a ***Result***) e il costo risultante, nel formato mostrato in figura. Nel caso l'agenzia sia chiusa nel momento indicato, dovrà essere emesso apposito messaggio d'errore, tramite il metodo *alert* del Controller (Fig. 5, sotto). Gli altri controlli da effettuare riguardano:

- la verifica che tutte le combo siano state correttamente selezionate (ossia, i valori da essere recuperati non siano nulli), emettendo nel caso, per ciascuno, un apposito messaggio d'errore tramite *alert* (Fig. 5, sopra);
- che l'orario iniziale e finale siano correttamente formattati nella forma HH:MM, ovvero che i valori di HH e MM siano corretti per un orario, emettendo nel caso, per ciascuno, un apposito messaggio d'errore tramite *alert* (Fig. 5, sopra).

Cose da ricordare

- salva costantemente il tuo lavoro: l'informatica a volte può essere "subdolamente ostile"…
- in particolare: se ora compila e stai per fare modifiche, salva la versione attuale (non si sa mai)

Checklist di consegna

- Hai fatto un **JAR eseguibile**, che contenga cioè l'indicazione del main?
- Hai controllato che **si compili** e **ci sia tutto**? [NB: non includere il PDF del testo]
- Hai **rinominato** IL PROGETTO, lo ZIP e il JAR esattamente come richiesto?
- Hai **chiamato** la cartella del progetto esattamente come richiesto?
- Hai fatto un **unico file ZIP (NON .7z, rar o altri formati)** contenente l'intero progetto? In particolare, ti sei assicurato di aver incluso tutti i file .java (e non solo i .class)?
- **Hai consegnato DUE file distinti, ossia lo ZIP col progetto e il JAR eseguibile?**
- Su EOL, hai **premuto** il tasto "CONFERMA" per inviare il tuo elaborato?