ESAME DI FONDAMENTI DI INFORMATICA T-2 del 14/9/2022
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 *palesemente lontani da 18/30* NON SARANNO CORRETTI e causeranno la verbalizzazione del giudizio "RESPINTO"

La società di gestione di un parcheggio sotterraneo di una grande città ha richiesto un'applicazione per consentire agli utenti di pre-calcolare il costo della sosta, secondo lo schema tariffario descritto di seguito.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Il sistema tariffario adottato dalla società deve coniugare diversi aspetti:

- consentire anche soste molto brevi (anche di soli 15 minuti), senza tuttavia che esse risultino in importi troppo bassi, che porterebbero a un utilizzo improprio ("mordi e fuggi") dello spazio di sosta
- disincentivare soste di troppe ore consecutive, evitando però di allontanare clienti con importi troppo elevati
- incentivare comunque anche le soste medio-lunghe, di 12 o più ore, con una tariffazione ad hoc.

A questo fine, è stato elaborato il seguente schema:

- nelle prime 12 ore, la tariffazione procede a intervalli di 15 minuti, del costo di 0,80 € ciascuno (=3,20€/ora); il primo intervallo di 15 minuti, tuttavia, viene eccezionalmente tariffato 2,00 € (anziché 0,80 €)
- è previsto un tetto giornaliero (price cap) di 35,00 €, che si raggiunge entro le prime 12 ore: raggiunta tale cifra, il costo resta costante fino a 24 ore di sosta
- se la sosta dura più di 24 ore, i periodi successivi alle prime 24 ore sono tariffati a blocchi indivisibili di 12 ore, del costo di 16,00 € ciascuno.

ESEMPI

- sosta entro i 15': 2,00 €
- sosta da 16' e fino a 30': 2,80 € (slot iniziale di 15' da 2,00 € + slot di 15' a 0,80 €)
- sosta di 1 ora: 4,40 € (slot iniziale di 15' da 2,00 € + 3 slot di 15' a 0,80 € ciascuno)
- sosta di 1 ora e 1 minuto: 5,20 € (slot iniziale di 15' da 2,00 € + 4 slot di 15' a 0,80 € ciascuno)

…

- sosta di 12 ore: 35,00 € (price cap raggiunto)
- sosta di 13,14,…24 ore: sempre 35,00 € (price cap raggiunto)
- sosta di 24 ore e 1 minuto: 51,00 € (35,00 € per le prime 24 ore + slot indivisibile di 12 ore da 16,00€)
- sosta di 25,26,.. 36 ore: sempre 51,00 € (35,00 € per le prime 24 ore + slot indivisibile di 12 ore da 16,00€)
- sosta di 36 ore e 1 minuto: 67,00 € (35,00 € per le prime 24 ore + 2 slot indivisibili di 12 ore da 16,00€ ciascuno)

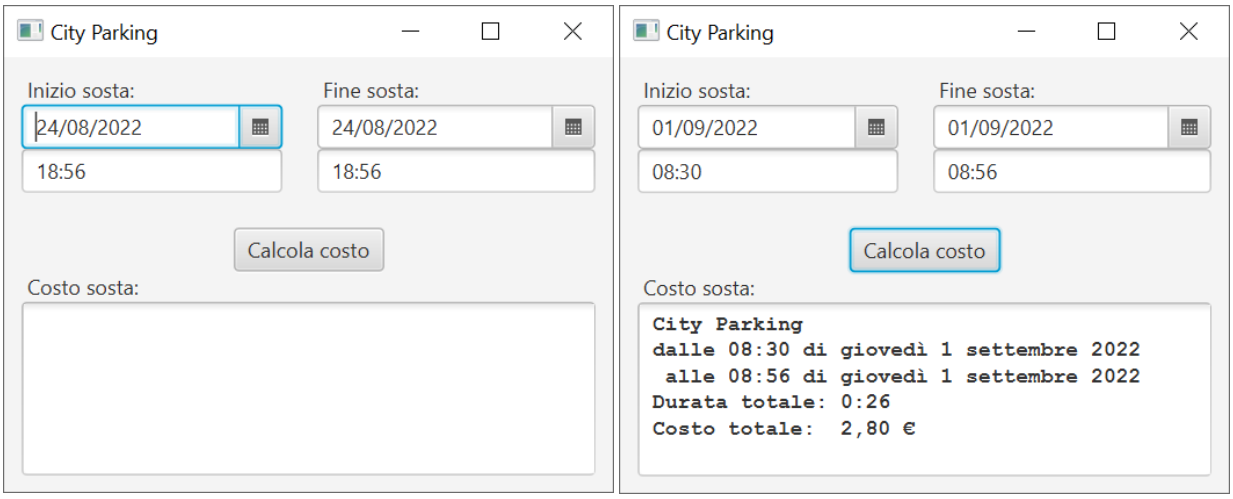
Il file **Tariffa.txt** contiene il sistema tariffario sopra descritto, opportunamente generalizzato in modo da parametrizzare durata degli slot e costi**,** nel formato descritto più oltre.

L'applicazione dovrà mostrare a video il costo previsto della sosta, nel formato illustrato nelle figure seguenti.

PARTE 1 – Modello dei dati:	Punti 10	[TEMPO STIMATO: 40-50 minuti]
PARTE 2 – Persistenza:	Punti 8	[TEMPO STIMATO: 30-40 minuti]
PARTE 3 – Grafica:	Punti 12	[TEMPO STIMATO: 50-60 minuti]

TEMPO STIMATO PER SVOLGERE L'INTERO COMPITO: 2h – 2h30

ESEMPI DI CALCOLO DEL COSTO DELLA SOSTA MOSTRATO A VIDEO



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

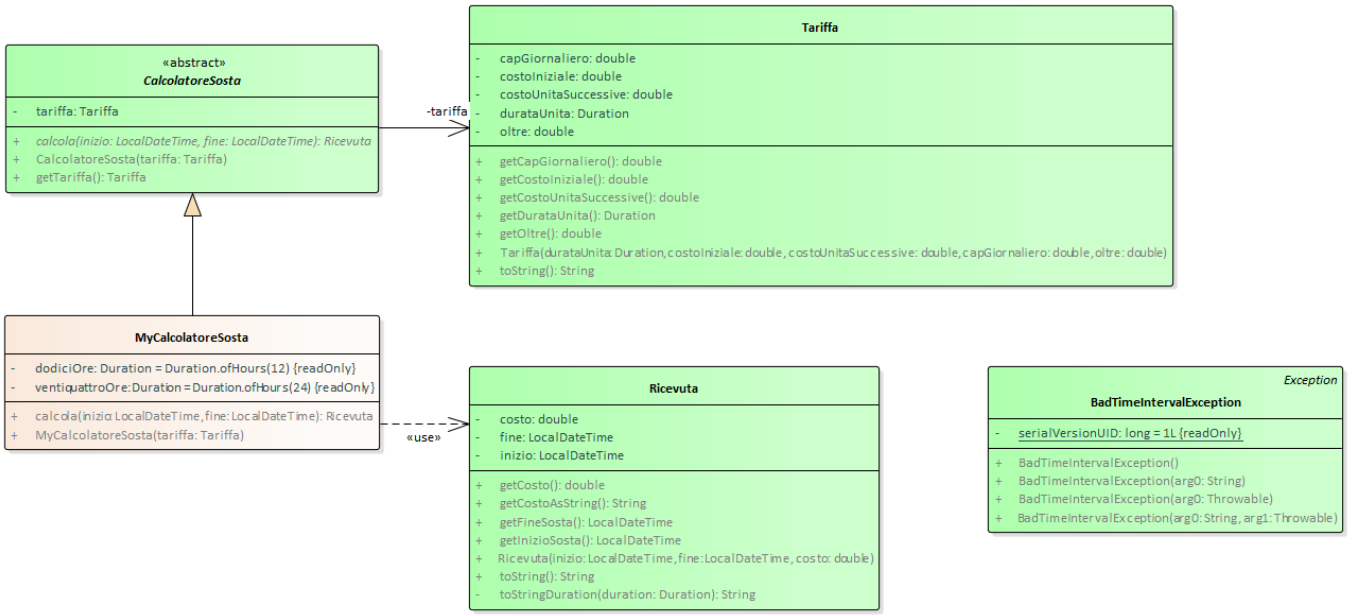
- Hai fatto un **JAR eseguibile**, che contenga cioè l'indicazione del main?
- Hai controllato che **si compili** e **ci sia tutto**? [NB: non includere il PDF del testo]
- Hai **rinominato** IL PROGETTO, lo ZIP e il JAR esattamente come richiesto?
- Hai **chiamato** la cartella del progetto esattamente 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: 10

Package: cityparking.model

[TEMPO STIMATO: 40-50 minuti]



SEMANTICA:

a) la classe ***Tariffa*** (fornita) rappresenta il sistema tariffario come descritto nel dominio del problema: i vari *accessor* consentono di recuperare le proprietà

b) la classe ***Ricevuta*** (fornita) rappresenta la ricevuta di parcheggio: contiene il costo e i vari elementi che hanno contribuito a determinarlo, recuperabili tramite appositi *accessor*

c) la classe astratta ***CalcolatoreSosta*** (fornita) rappresenta il componente che effettua il calcolo del costo della sosta sulla base di una data tariffa. Il metodo *calcola*, astratto, è destinato a catturare lo specifico algoritmo di calcolo: data e ora di inizio/fine sosta sono forniti tramite due oggetti ***LocalDateTime***. Se l'orario di inizio sosta non precede quello di fine sosta, il metodo deve lanciare l'apposita ***BadTimeIntervalException*** (fornita), con adeguato messaggio d'errore.

d) la classe ***MyCalcolatoreSosta*** **(da completare)** estende la precedente implementando *calcola* secondo l'algoritmo descritto nel dominio del problema.

Parte 2 – Persistenza

Punti: 8

Package: cityparking.persistence

[TEMPO STIMATO: 30-40 minuti]

Il file di testo **Tariffa.txt** contiene la descrizione del sistema tariffario, opportunamente parametrizzata. Il file contiene sempre e solo 5 righe, che descrivono, esattamente in quest'ordine:

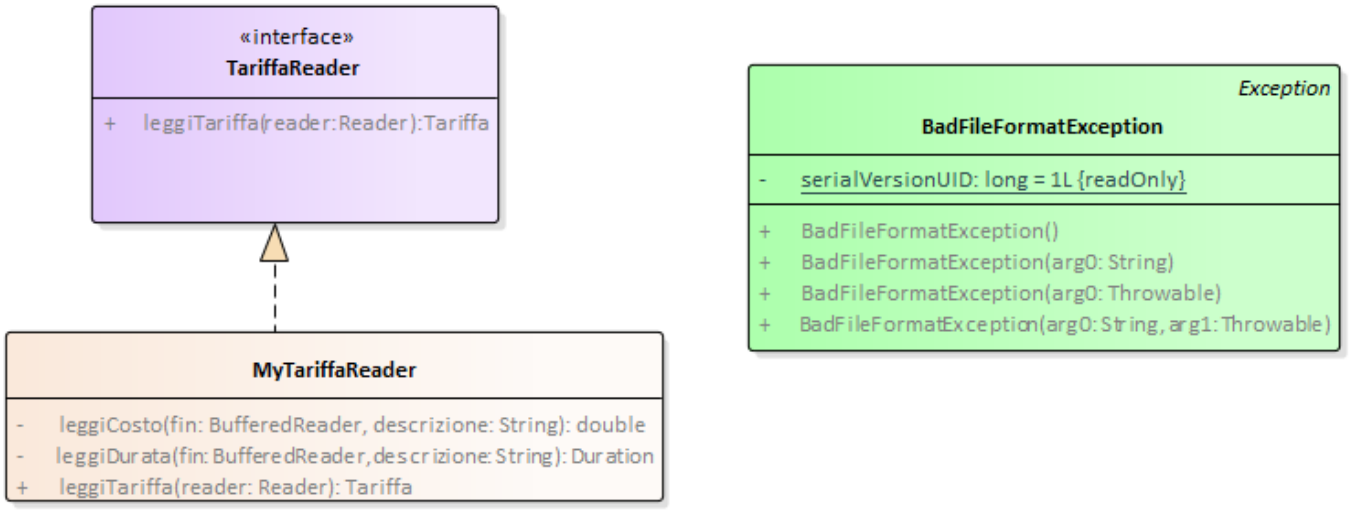
- la durata dell'unità di tariffazione (tipicamente, 15 minuti)
- il costo della prima unità
- il costo della succcessive unità
- il price cap giornaliero, valido entro le prime 24 ore
- il costo dei periodi indivisibili da 12h successivi

Il formato è illustrato nell'esempio seguente:

```
Durata unità = 15m
Costo iniziale = 2,00 €
Unità successive = 0,80 €
Cap giornaliero = 35,00 €
12h successive = 16,00 €
```

Si noti che:

- tutte le righe hanno la forma ***descrizione = valore***
- la descrizione può essere qualsiasi e può contenere spazi e ogni altro carattere diverso da '='
- per le righe che specificano costi, il valore è un prezzo in euro, formattato secondo lo standard italiano
- per la prima riga, che specifica una durata in minuti, il valore ha la forma di un numero intero, seguito senza spazi intermedi dal carattere '**m**'



SEMANTICA:

a) L'eccezione ***BadFileFormatException*** (fornita) esprime l'idea di file formattato in modo scorretto

b) L'interfaccia ***TariffaReader*** (fornita) dichiara il metodo *leggiTariffa,* che legge da un *Reader* (ricevuto come argomento) i dati del sistema tariffario, configurando e restituendo l'opportuno oggetto ***Tariffa***

c) La classe ***MyTariffaReader*** (**da realizzare**) implementa ***TariffaReader***: non prevede costruttori e implementa il metodo ***leggiTariffa*** in accordo alle specifiche sopra definite. In caso di problemi di I/O deve propagare l'opportuna ***IOException,*** in caso di ***Reader*** nullo ***IllegalArgumentException*** e in caso di altri problemi di formato del file ***BadFileFormatException***, il cui messaggio dettagli l'accaduto.

   Al fine di modularizzare opportunamente la lettura, *leggiTariffa* si appoggia a due metodi privati:

   - *leggiDurata* **(da implementare),** che gestisce la prima riga
   - *leggiCosto* **(da implementare)** che gestisce le altre righe

   essi restituendo l'opportuno oggetto o lanciano ***BadFileFormatException*** se necessario.

   Il metodo *leggiTariffa* deve effettuare specifici controlli sul formato delle varie righe, in particolare:

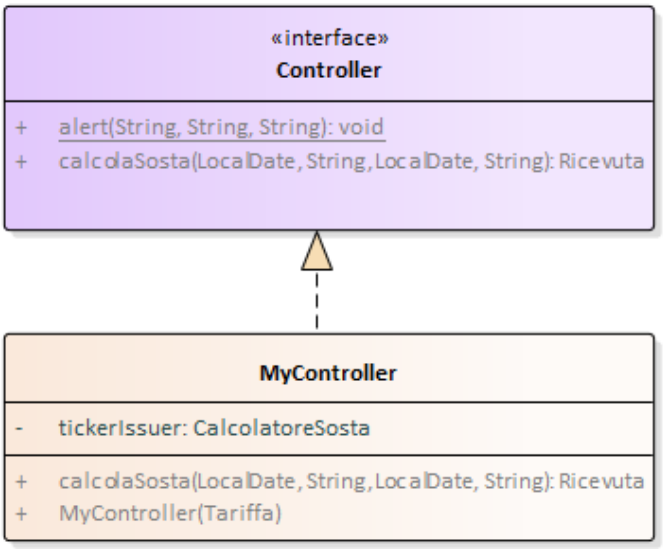
   i. verificando che la descrizione (che precede il simbolo '=') sia esattamente quella richiesta

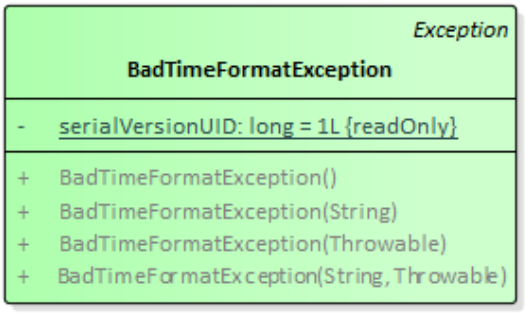
   ii. che il valore fornito sia di formato e valore coerenti con quanto richiesto, e specificatamente che la durata dell'unità non sia negativa o nulla, e che i vari costi non siano negativi, infiniti o NaN.

Parte 3 - Grafica — Punti: 12

Package: cityparking.controller — **[TEMPO STIMATO: 15-20 minuti]** ***(punti 6)***

L'interfaccia ***Controller*** (fornita) dichiara il metodo *calcolaSosta,* che riceve data e ora di inizio/fine sosta: suo compito è, tramite il ***CalcolatoreSosta,*** ottenere e restituire l'opportuna ***Ricevuta.***





A differenza del ***CalcolatoreSosta***, il cui metodo *calcola* riceve data e ora di inizio/fine sosta come ***LocalDateTime***, il metodo *calcolaSosta* del ***Controller*** riceve quattro argomenti: due ***LocalDate***, che forniscono il giorno di inizio/fine sosta, e due *stringhe della forma HH:MM*, che forniscono la corrispondente ora di inizio/fine sosta.

È compito del metodo *calcolaSosta* verificare che le due stringhe abbiano il corretto formato, lanciando in caso contrario l'apposita ***BadTimeFormatException*** (fornita).

NB: si ricorda che l'analoga verifica relativa al fatto che l'istante di inizio sosta preceda quello di fine sosta è già svolta da ***CalcolatoreSosta***, che lancia in tal caso l'apposita ***BadTimeIntervalException*** (fornita).

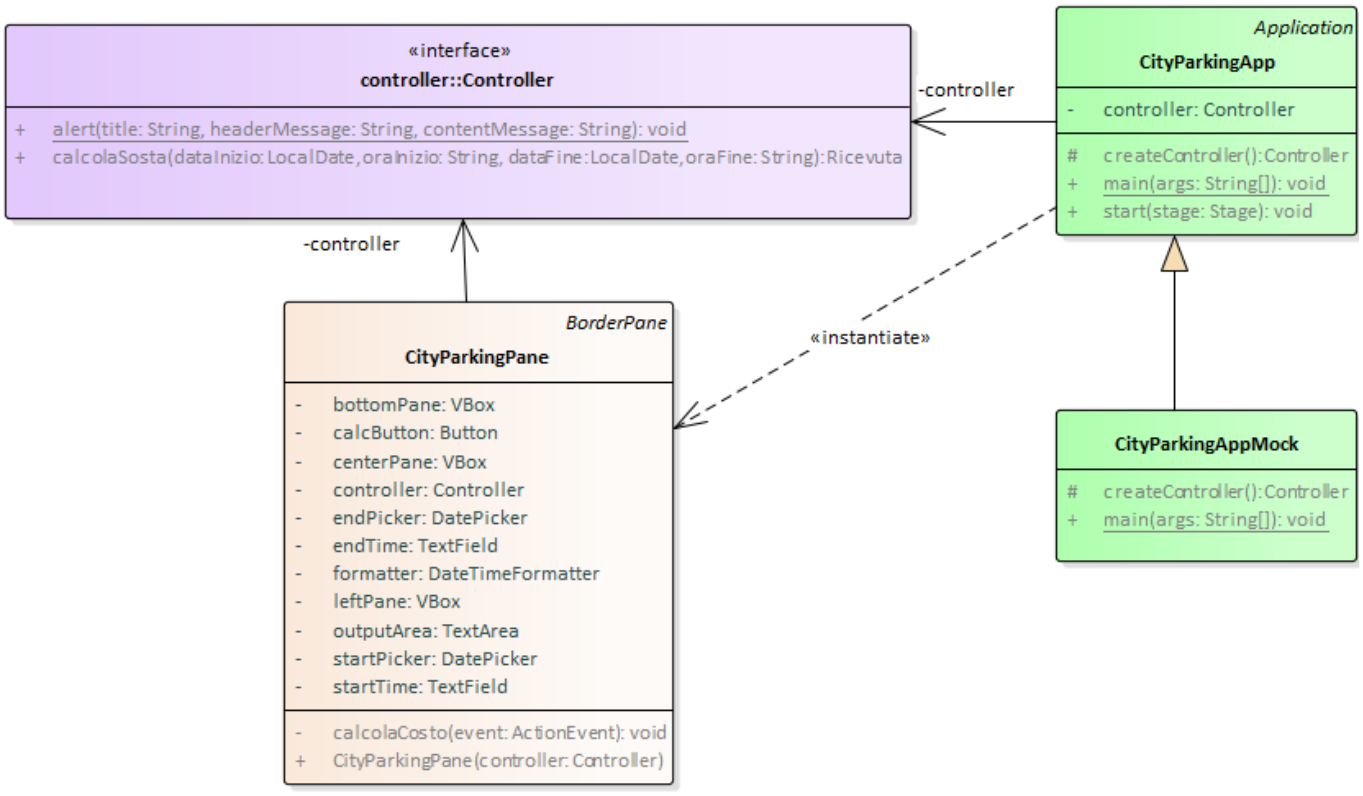
La classe ***MyController*** **(da realizzare)** concretizza ***Controller*** implementando il metodo *calcolaSosta* come sopra specificato: la ***Tariffa*** su cui lavorare dev'essere passata all'atto della costruzione.

NB: l'interfaccia ***Controller*** contiene anche il **metodo statico ausiliario** *alert*, utile per mostrare avvisi all'utente.

Package: cityparking.ui **[TEMPO STIMATO: 15-20 minuti]** *(punti 6)*

La classe ***CityParkingApp*** (fornita) costituisce l'applicazione JavaFX che si occupa di aprire i file, creare il controller e incorporare il pannello principale, ***CityParkingPane***. 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 ***CityParkingAppMock***.

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



L'interfaccia grafica si presenta come segue:

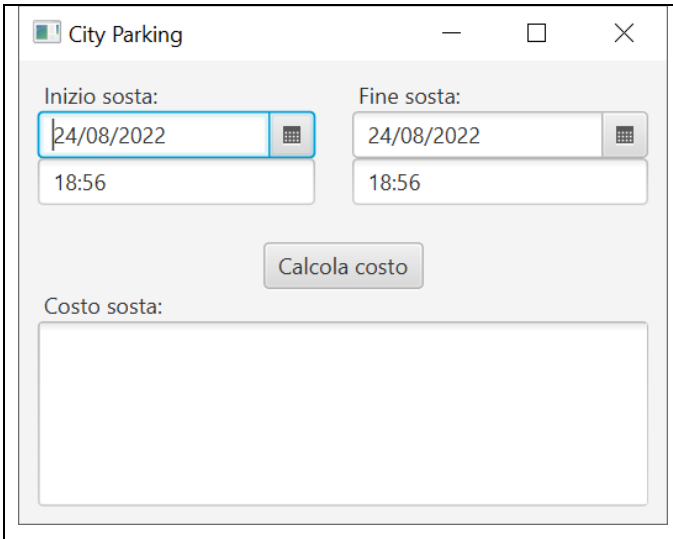


Fig. 1: la situazione iniziale della GUI

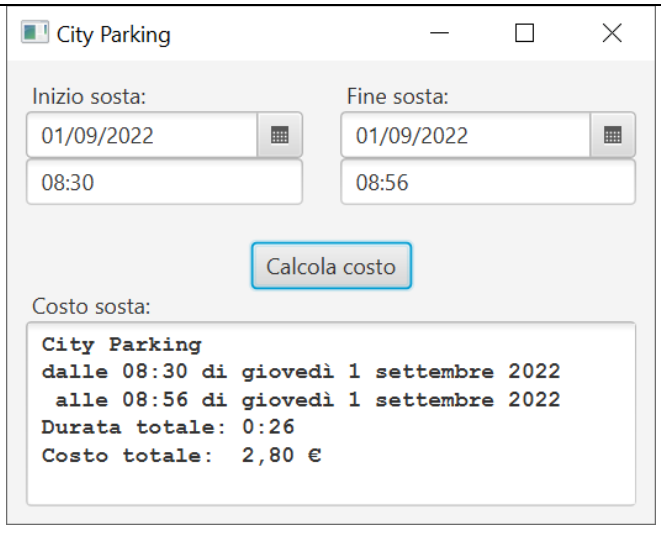


Fig. 2: un primo caso di funzionamento normale

Fig. 3: un altro caso di funzionamento normale

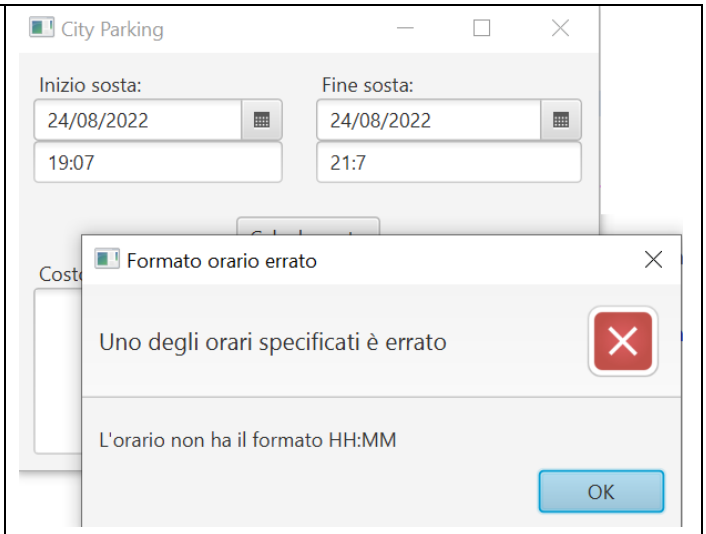


Fig. 4: errore: orario con formato errato

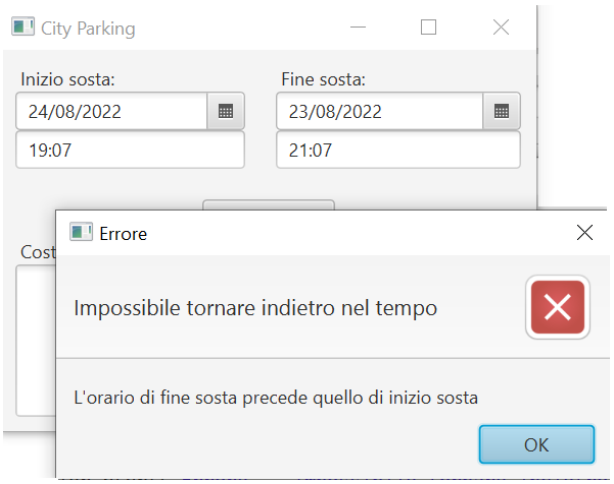


Fig. 5: errore: data/ora finale precedente quella iniziale

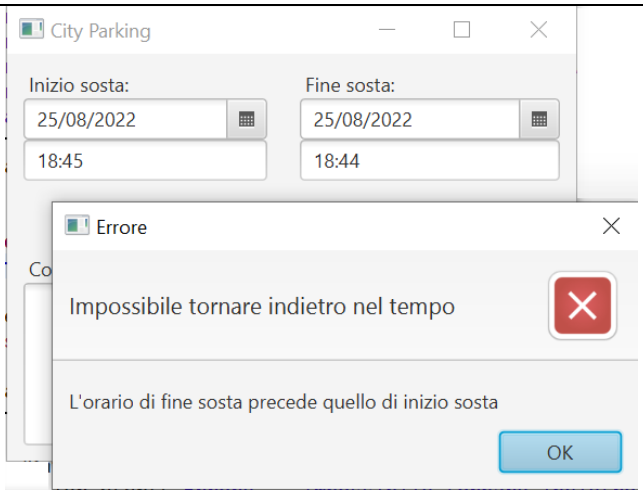


Fig. 6:

- in alto a sinistra, un ***DatePicker*** e un campo di testo consentono di specificare data e ora di inizio sosta
- in alto a destra, un ***DatePicker*** e un campo di testo consentono di specificare data e ora di fine sosta
- al centro, il pulsante *Calcola* effettua il calcolo del costo della sosta per il periodo sopra specificato: se l'istante di fine sosta precede quello di inizio sosta, o se gli orari non seguono il prescritto formato HH:MM *secondo l'uso italiano*, devono essere mostrati opportuni avvisi all'utente (Figg. 4-5).
- in basso, un'area di testo mostra la ricevuta di sosta calcolata, *usando come font il Courier New, 12 punti, grassetto.*

È richiesto che all'avvio dell'applicazione date e orari siano quelli relativi e data e ora correnti.

CityParkingPane è fornito *parzialmente realizzato*: è presente quasi tutta la parte strutturale, mentre sono da completare la configurazione dei campi di testo e la gestione dell'evento.

In particolare, la parte da completare riguarda:

1) il popolamento dei valori di default di date e ore iniziali, nel corretto formato
2) l'aggancio dell'opportuno *listener*
3) la logica di gestione dell'evento, *che deve catturare le eccezioni eventualmente lanciate dal controller, mostrando all'utente, in tali casi, i messaggi di errore sopra illustrati.*