ESAME DI FONDAMENTI DI INFORMATICA T-2 del 15/02/2012
Proff. E. Denti – G. Zannoni
Tempo a disposizione: 4 ore MAX

NB: il candidato troverà nell'archivio ZIP scaricato da Esamix anche il software "Start Kit"

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)

Gli studenti che sostengono questo esame come completamento a seguito di opzione (6 cfu anziché 12) devono sostenere la sola prima parte (model); come main potranno utilizzare ConsoleTesterX6.java.

Al fine di limitare il traffico in centro, il comune di Roncofritto ha richiesto lo sviluppo di un'applicazione per controllare e rendere a pagamento gli accessi al centro storico, commisurando il pagamento al tempo di permanenza nell'area ZTL. A tal fine ha installato opportuni varchi sulle strade di entrata o uscita dal centro: poiché a Roncofritto tutte le strade sono a senso unico, ogni varco di controllo registra veicoli o solo in entrata, o solo in uscita.

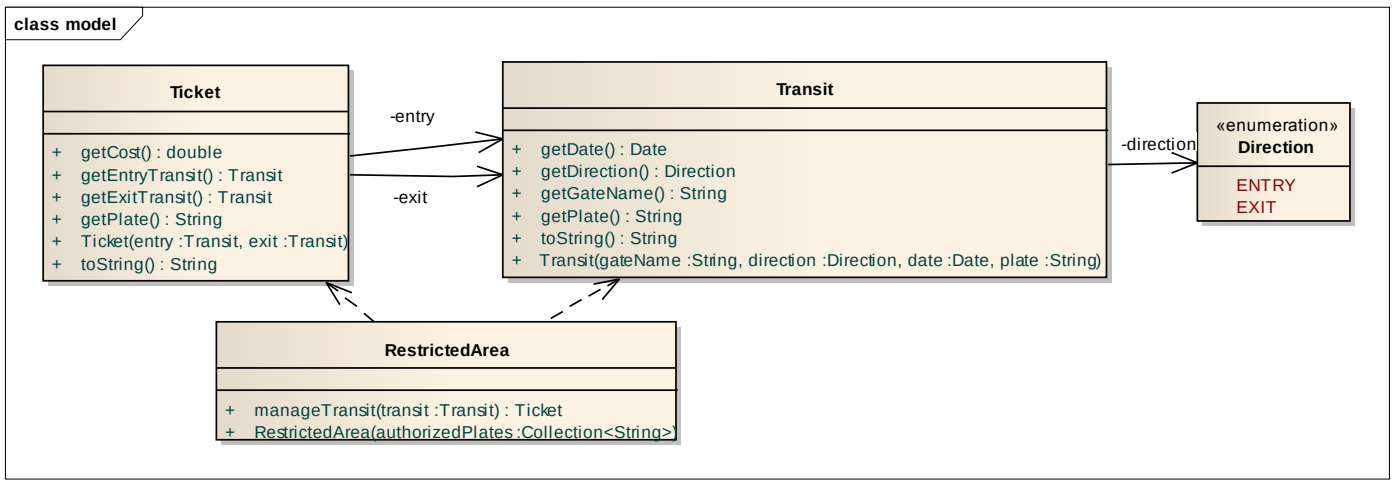
DESCRIZIONE DEL DOMINIO DEL PROBLEMA.
L'accesso al centro storico è libero soltanto per i veicoli autorizzati; per tutti gli altri l'ingresso è a pagamento. La tariffa da pagare dipende dal tempo di permanenza in centro storico: di fatto, il sistema fattura i minuti fra l'entrata e l'uscita (analogamente a una telefonata o a un parcheggio). Fisicamente, il sistema è costituito da un insieme di varchi che registrano le targhe dei veicoli in transito, con data e ora del transito stesso. Ogni *Varco* è caratterizzato da una identificativo univoco e dalla direzione in cui opera ("entrata" o "uscita"). Quando un veicolo non autorizzato attraversa un varco, il transito viene registrato: all'uscita viene emessa una *Nota di addebito* proporzionale al tempo di permanenza. La tariffa è uguale tutti i giorni della settimana e pari a € 3 l'ora (ad esempio, se si è entrati al martedì mattina alle 9 e si esce al mercoledì pomeriggio alle 15.30, saranno addebitate 30h ½ per un totale di 91,50 €).

I file di testo autorizzati.txt e transiti.txt contengono rispettivamente le targhe dei veicoli autorizzati e l'elenco dei transiti registrati, con le rispettive proprietà, nei formati più oltre specificato.

Parte 1 (punti: 19)

Modello dei dati (namespace ztl.model) *(punti: 13)*

Il modello dei dati deve essere organizzato secondo il diagramma UML di seguito riportato.



SEMANTICA:

a) l'enumerativo ***Direction*** (fornito nello start kit) rappresenta le direzioni di funzionamento dei varchi: i valori possibili sono ENTRY ed EXIT

b) la classe ***Transit*** (fornita nello start kit) rappresenta il passaggio da un varco di un veicolo di targa nota: è caratterizzato da data/ora, targa del veicolo (*plate*), varco interessato e relativa direzione, e fornisce i necessari metodi accessor;

c) la classe ***RestrictedArea*** **(da realizzare)** rappresenta la zona a traffico limitato ed è caratterizzata dall'insieme dei veicoli autorizzati ad accedervi e da un elenco di transiti, memorizzati in una mappa avente come chiave la targa

del veicolo. Tale mappa deve contenere in ogni istante i transiti dei veicoli attualmente all'interno della ZTL, ossia entrati ma non ancora usciti. Coerentemente, il costruttore riceve come argomento la collezione di targhe autorizzate. L'unico altro metodo pubblico è ***manageTransit***, che analizza un transito e se necessario genera e restituisce la nota di addebito (***Ticket***) creata. Questo metodo incapsula l'algoritmo di controllo vero e proprio e agisce come segue:

- se il transito passato come argomento riguarda un veicolo autorizzato, lo ignora;
- se invece riguarda un veicolo non autorizzato :
  - in entrata, memorizza il transito nella mappa transiti;
  - in uscita, rimuove il corrispondente transito in entrata dalla mappa transiti e produce la corrispondente nota di addebito (***Ticket***), passandole i due transiti di ingresso e uscita.

d) la classe ***Ticket*** **(da realizzare)** rappresenta la nota di addebito ed è caratterizzata dalla serie di proprietà dettagliate nel diagramma UML, con i relativi metodi accessor (di sola lettura) e a un opportuno metodo ***toString***. È compito del costruttore verificare 1) che i due ***Transit*** ricevuti siano relativi allo stesso veicolo, 2) che il primo sia in ingresso e il secondo in uscita, 3) che il primo sia temporalmente precedente al secondo. Se queste tre precondizioni sono vere, il costruttore calcola la durata della permanenza e il conseguente importo; altrimenti, deve lanciare ***IllegalArgumentException*** con opportuno messaggio testuale esplicativo della situazione.

Lo Start Kit contiene anche i test (da includere nel progetto) per verificare il funzionamento di questa classe.

Persistenza (namespace ztl.persistence) *(punti 6)*

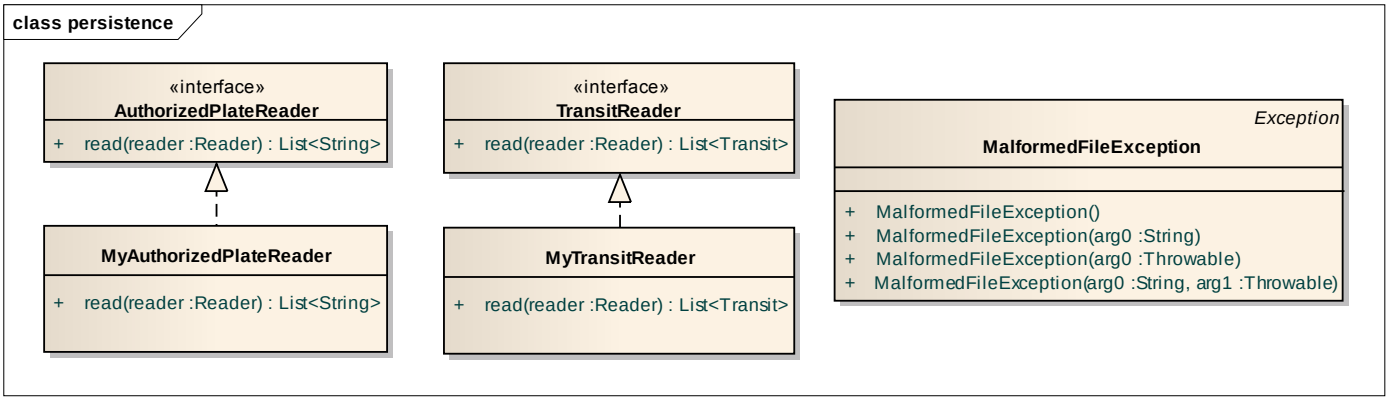
Il file di testo autorizzati.txt contiene le targhe dei veicoli autorizzati, separate da virgole, spazi, tabulazioni, a capo.

Il file di testo transiti.txt contiene i transiti registrati: ogni riga rappresenta un transito e contiene nell'ordine l'identificativo univoco del varco, la direzione del transito ("entrata"/"uscita"), la targa del veicolo, la data (nel formato GG/MM/AAAA) e ora (nel formato HH.MM) del passaggio, separate da spazi e/o tabulazioni.

```
ESEMPIO DEL FILE transiti.txt
ViaNovelli entrata ED999AP 10/01/2012 12.34
VialeTimavo uscita ED999AP 10/01/2012 14.20
LargoChigi entrata MP333BO 11/01/2012 20.34
ViaOmbrosa uscita  MP333BO 12/01/2012 6.55
LargoChigi entrata MP333BO 12/01/2012 21.05
ViaOmbrosa uscita  MP333BO 13/01/2012 7.35
ViaNovelli entrata GZ666RR 13/01/2012 20.04
VialeSerio uscita  GZ666RR 16/01/2012 06.56
…
```

Le interfacce ***AuthorizedPlateReader*** e ***TransitReader*** (fornite) dichiarano rispettivamente i metodi ***readPlates*** e ***readTransits*** che leggono da un ***Reader*** rispettivamente una lista di ***Plate*** e di ***Transit***.

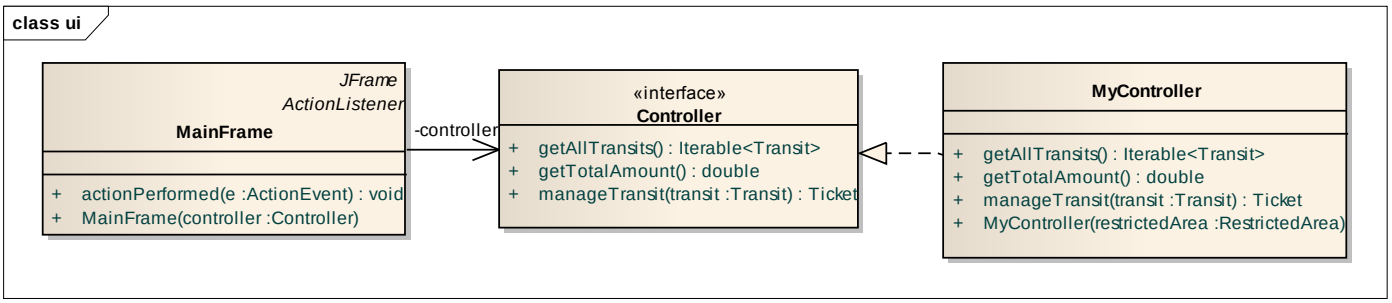
Le classi ***MyAuthorizedPlateReader*** e ***MyTransitReader*** (**da realizzare**) implementano tali interfacce.



Lo Start Kit contiene anche i test (da includere nel progetto) per verificare il funzionamento di questa classe.

Parte 2 (punti: 11)

Controller (namespace ztl.ui) *(punti 4)*



La classe ***MyController*** (**da realizzare**) implementa l'interfaccia ***Controller*** (fornita): il costruttore riceve in ingresso un'istanza di ***RestrictedArea*** e carica il file dei transiti; può rilanciare le eccezioni eventualmente sollevate dai metodi di lettura (IOException o MalformedFileException). Gli altri metodi pubblici sono ***manageTransit***, ***getAllTransits*** e ***getTotalAmount:*** il primo semplicemente delega la gestione del transito all'omonimo metodo della ***RestrictedArea*** incapsulata, provvedendo di suo solo all'aggiornamento del totale incassato; il secondo restituisce la collezione dei transiti caricati dal costruttore, mentre il terzo restituisce il totale incassato fino a quel momento.

Interfaccia utente (namespace ztl.ui) *(punti 7)*

L'interfaccia utente deve essere simile (non necessariamente identica) all'esempio mostrato in figura.

La classe ***GuiMain*** (fornita nello start kit) contiene il main di partenza dell'applicazione.

La classe ***MainFrame*** (**da realizzare**) realizza la finestra principale, ed è articolata in tre parti: due liste (JList) incorporate in pannelli a scorrimento (JScrollPane), e un pulsante centrale. La lista in alto mostra i transiti registrati, che il pulsante centrale ELABORA PROSSIMO TRANSITO permette di analizzare singolarmente, mentre la lista in basso mostra gli addebiti via via prodotti; un'etichetta sottostante mostra *il totale incassato dal Comune.*

Inizialmente (Fig. 1) la lista in alto è popolata coi dati letti da file, mentre quella inferiore è vuota. Ogni volta che si preme il pulsante viene analizzato un nuovo transito, che viene innanzitutto rimosso dalla lista in alto: se esso riguarda un veicolo autorizzato o l'entrata di un veicolo non autorizzato, non viene mostrato altro (Fig. 2), mentre nel caso di uscita di veicolo non autorizzato si calcola la durata in minuti della permanenza in centro, mostrando il corrispondente addebito nell'area in basso **unitamente al totale incassato fino a quel momento** (Figg. 3 e 4). Quando tutti i transiti sono stati elaborati, il pulsante viene definitivamente disabilitato (Fig. 4).

SUGGERIMENTO: per gestire facilmente le ***JList***, e in particolare la rimozione di elementi da esse, è opportuno crearle passando esplicitamente un ***DefaultListModel***, da crearsi preventivamente col costruttore predefinito: basterà quindi agire su quest'ultimo *come se fosse una lista* (dispone di metodi ***size***, ***add***, ***remove***..) per ottenere automaticamente l'effetto richiesto. **Un esempio è riportato in fondo a questa pagina**.

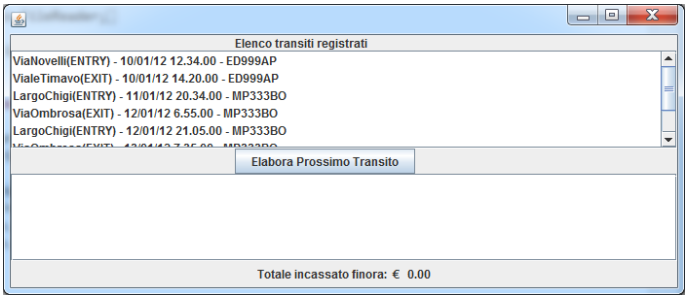


Fig. 1

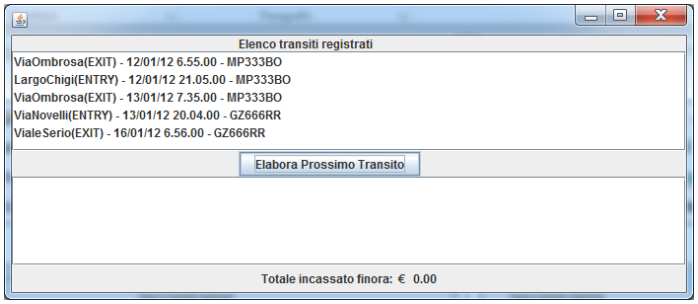


Fig. 2

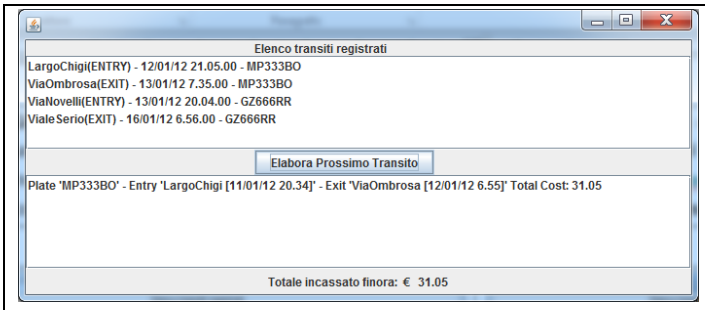


Fig. 3

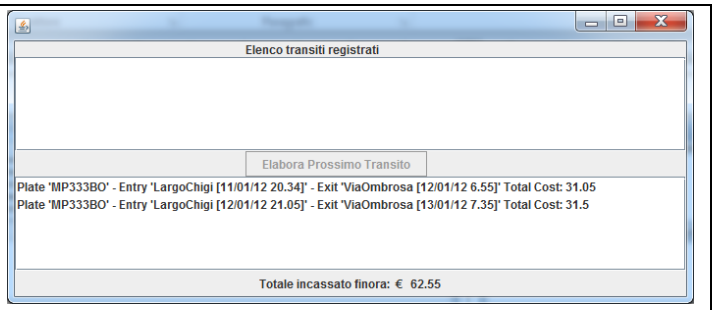


Fig. 4

Schema d'uso di *DefaultListModel* in *JList* (maggiori informazioni sulla documentazione online)

Creazione modello, riempimento iniziale, creazione lista:

```
DefaultListModel model = new DefaultListModel();
for (Element el : elements) model.addElement(el);
JList list = new JList(model);
```

Azioni sul modello (es. estrazione e/o rimozione elementi):

```
if (!model.isEmpty()) {
        Element element = (Element) model.get(0);
        model.remove(0);
}
```