ESAME DI FONDAMENTI DI INFORMATICA T-2 del 14/06/2011

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: matricola-CognomeNome (es. 0000123456-RossiMario)

La pluripremiata compagnia ferroviaria *DentiTrain* ha richiesto un'applicazione per il calcolo dei percorsi fra due città servite della propria rete.

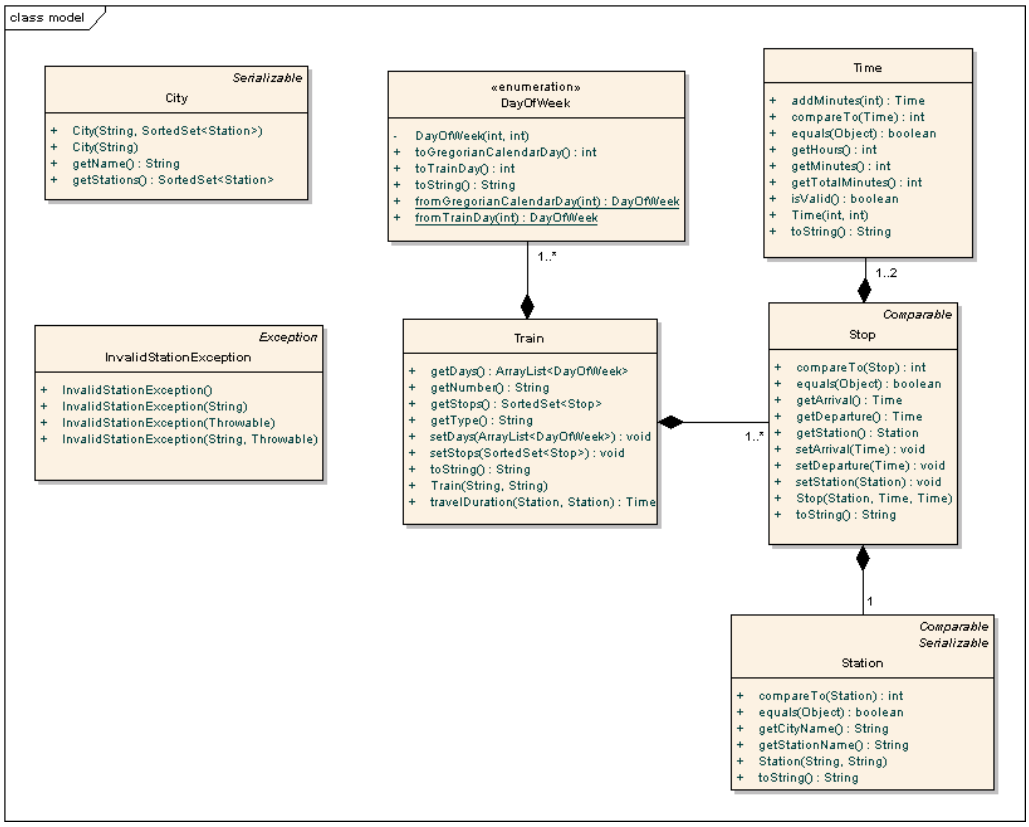
> DESCRIZIONE DEL DOMINIO DEL PROBLEMA.
> Ogni città dotata di stazione è identificata da un nome univoco (che può contenere spazi – es. "Reggio Calabria"); se la città ha un'unica stazione, il nome della città vale per entrambe, mentre in caso contrario ogni stazione ha il proprio nome, distinto da quello della città (es. "Firenze" indica la città, le cui stazioni sono "Firenze S.M.N.", "Firenze C.M.", "Firenze Rifredi", "Firenze Statuto", etc.).
> Ogni treno è descritto da un identificativo (tipologia del treno + numero identificativo univoco), i giorni di effettuazione (1=lunedì,…, 7=domenica) e una serie di nomi di stazioni con i rispettivi orari di partenza e arrivo; l'orario di partenza è ovviamente indefinito per la stazione di arrivo, e lo stesso vale per l'orario di arrivo nella stazione di origine.
> Le tipologie di treni ammessi da *DentiTrain* sono regionale ("R"), interregionale ("IR"), intercity ("IC"), razzo ("RZ").

Le descrizioni dei treni previsti in orario sono disponibili nel file di testo Trains.txt, mentre il file StationCities.dat ospita una mappa (serializzata) contenente l'elenco dei nomi di città con le relative stazioni.

Parte 1 (punti: 17)

Dati (namespace trains.model) *(punti: 6)*

Il modello dei dati deve essere organizzato secondo il diagramma UML riportato in Figura.



a) la classe ***Station*** (fornita nello Start Kit) rappresenta una stazione, caratterizzata dal nome della città in cui si trova e dal suo nome specifico di stazione; il metodo ***getCityName*** restituisce il nome della città, il metodo ***getStationName*** restituisce il nome specifico della stazione, mentre ***toString*** restituisce il nome completo della stazione stessa, composto dalla concatenazione dei due [ESEMPIO: per la stazione "Bologna Centrale" (nome completo), i due metodi ***getCityName*** e ***getStationName*** restituiscono rispettivamente "Bologna" e "Centrale"];

b) la classe ***City*** (fornita nello Start Kit) rappresenta una città, caratterizzata dal suo nome univoco e dall'insieme delle sue stazioni; il metodo ***getStations*** restituisce le stazioni della città sotto forma di ***SortedSet<Station>***;

c) la classe ***Time*** (fornita nello Start Kit) rappresenta un orario (ore, minuti) di cui fornisce le relative proprietà;

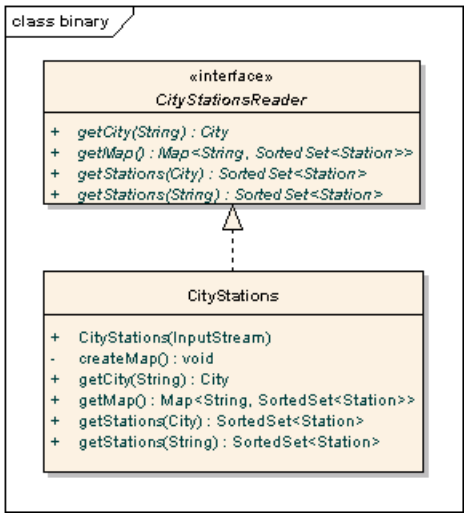
d) la classe ***Stop*** (fornita nello Start Kit) rappresenta una "fermata del treno" composta da una ***Station*** e dai rispettivi orari di arrivo e partenza; sono disponibili opportuni metodi accessor a tali proprietà;

e) l'enumerativo ***DayOfWeek*** (fornito nello Start Kit) rappresenta un giorno della settimana: ad ogni valore dell'enumerativo sono associati sia la costante intera che rappresenta il giorno della settimana nel contesto del ***GregorianCalendar***, sia il valore intero che rappresenta il giorno della settimana nel nostro contesto; sono disponibili opportuni metodi per effettuare le conversioni del caso;

f) la classe ***Train*** **(da realizzare)** rappresenta la descrizione di un treno con tutte le proprietà sopra dettagliate; oltre ai metodi per accedervi, deve offrire un metodo ***travelDuration*** che, date due stazioni, *calcoli la durata del viaggio fra esse* sotto forma di opportuna istanza di ***Time***, ovvero lanci l'eccezione ***InvalidStationException*** (fornita nello Start Kit) se una o entrambe le stazioni passate non sono servite da tale treno o se presentano errori.

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

Persistenza (namespace trains.persistence.binary e trains.persistence.text) *(punti 11)*



1) Il file binario StationCities.dat contiene istanze serializzate di ***City***, opportunamente precedute da un intero che ne indica il numero complessivo. Nel diagramma UML a lato, l'interfaccia ***CityStationsReader*** (fornita nello Start Kit) è implementata dalla classe ***CityStations*** (**da realizzare**) che provvede, nel proprio costruttore, a caricare tali istanze dal file, rilanciando ***IOException*** o lanciando ***InvalidDataException*** (fornita nello Start Kit) in caso di problemi; essa offre inoltre svariati metodi: più specificatamente, ***getStations***, di cui devono essere previste due versioni overloaded, restituisce, data una istanza di ***City*** (versione 1), ovvero dato il nome della città (versione 2), l'insieme ordinato delle stazioni in essa presenti, sotto forma di ***SortedSet<Station>***; dualmente, il metodo ***getCity***, restituisce, dato il nome completo di una stazione, l'istanza di ***City*** corrispondente. È esplicitamente richiesto che tali metodi *non distinguano fra minuscole e maiuscole.*

Inoltre, al fine di rendere tali metodi efficienti, evitando costose ricerche sequenziali, è espressamente richiesto che ***CityStations*** provveda, durante la lettura del file, a *costruire e riempire una opportuna mappa* (restituita attraverso il metodo ***getMap***) che metta in corrispondenza ogni nome di città con l'insieme delle rispettive stazioni, come da figura di principio seguente:

Nome città (chiave)	Insieme di stazioni (valore)
BOLOGNA	Bologna Centrale, Bologna Corticella, Bologna S.Ruffillo, Bologna S.Lazzaro
PARMA	Parma
FIRENZE	Firenze S.M.N., Firenze Rifredi, Firenze Statuto, Firenze C.M.
RIMINI	Rimini, Rimini Miramare

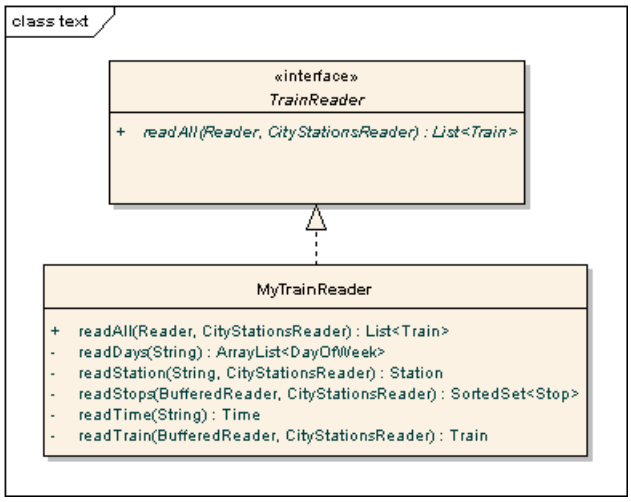
Conseguentemente, i metodi ***getStations(City)***, ***getStations(String)*** e ***getCity(String)*** dovranno essere necessariamente implementati facendo uso di tale mappa.

2) Il file di testo Trains.txt contiene l'orario dei treni programmati, nel seguente formato:
   - una riga iniziale "`#TRAIN`" contenente l'identificativo univoco del treno (costituito a sua volta dalla tipologia del treno + numero identificativo) e i giorni di effettuazione (1=lunedì,…, 7=domenica; i giorni di non effettuazione sono marcati da un trattino "-"); le tipologie possibili sono regionale ("R"), interregionale ("IR"), intercity ("IC"), razzo ("RZ").
   - nelle righe seguenti, un elenco di nomi di stazioni con i rispettivi orari di partenza e arrivo (una stazione per riga, seguita dai relativi orari di partenza e arrivo, separati da virgole); gli orari sono espressi nella forma `hh:mm`, ma lo '`0`' davanti alle ore è opzionale e potrebbe dunque sia essere, sia non essere presente. L'orario di partenza è indefinito per la stazione di arrivo, e lo stesso vale per l'orario di arrivo nella stazione di origine: al loro posto è convenzionalmente presente la dicitura `--:--`.
   - la riga finale "`#ENDTRAIN`".

Se il treno ha orari diversi, o ferma in stazioni diverse, in diversi giorni o periodi, ha numeri distinti ed è quindi descritto da record distinti (si vedano i due esempi IR 2069/2135 nella tabella sotto).

ESEMPI

```
#TRAIN R3961, 1234567          #TRAIN IR2069, 12345--         #TRAIN IR2135, -----67
Bologna centrale, 9:28,9:36    Bologna centrale,10;28,10:36   Bologna centrale,10;28,10:36
Castel S.Pietro, 9:51, 9:52    Castel S.Pietro, 10:51,10:52   Castel S.Pietro, 10:51,10:52
Imola, 10:00, 10:01            Imola, 11:00, 11:01            Imola, 11:00, 11:01
Castelbolognese, 10:06,10:07   Castelbolognese, 11:09,11:10   Castelbolognese, 11:06,11:08
Faenza, 10:11, 10:13           Faenza, 11:23, 11:25           Faenza, 11:13, 11:15
Forlì,  10:27, 10:29           Forlì,  11:41, 11:43           Forlì,  11:27, 11:29
Cesena, 10:42, 10:44           Cesena, 12:04, 12:06           Cesena, 11:40, 11:42
Rimini, 11:05, --:--           Rimini, 12:30, --:--           Rimini, 12:03, --:--
#ENDTRAIN                      #ENDTRAIN                      #ENDTRAIN
```

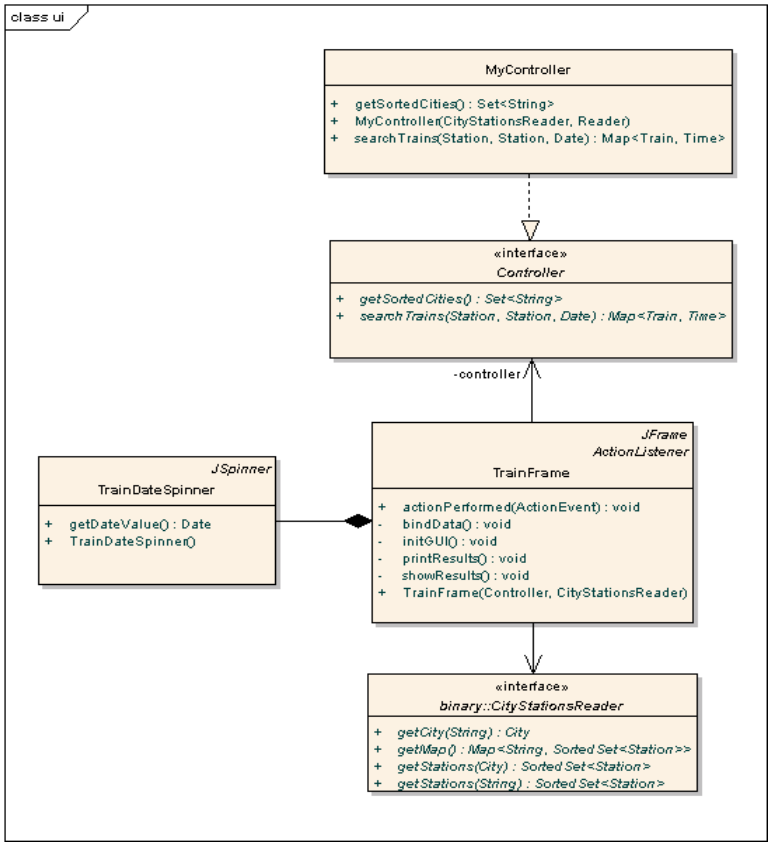


Nel diagramma UML a lato, l'interfaccia ***TrainReader*** (fornita nello StartKit), che dichiara i metodi utili per ottenere un insieme di ***Train***, dovrà essere implementata da ***MyTrainReader*** (**da realizzare**); quest'ultima dovrà lanciare ***IOException*** in caso di problemi di accesso al file, o ***BadFileFormatException*** (**da realizzare**) in caso di problemi di formato sul file. Il contratto stabilito da ***TrainReader*** prevede che venga restituita un'opportuna lista di ***Train.***

Lo Start Kit contiene anche i test relativi alle classi MyTrainReader e CityStations.

Parte 2 (punti: 13)

Controller (namespace trains.ui) *(punti 5)*



La classe ***MyController*** (**da realizzare**) implementa l'interfaccia ***Controller*** (fornita nello Start Kit); in particolare:

- il metodo ***getSortedCities*** restituisce una lista di nomi di città ordinate alfabeticamente;
- il metodo ***searchTrains*** cerca e restituisce una mappa dei treni disponibili con relativi tempi di percorrenza (espressi attraverso una istanza della classe ***Time***) fra le stazioni specificate nella data specificata (necessaria per conoscere il giorno della settimana).

→ → *Segue* → →

Interfaccia utente (namespace trains.ui) *(punti 8)*

L'interfaccia utente deve essere simile (non necessariamente identica) all'esempio mostrato in Fig. 1. La classe ***TrainFrame*** (**da realizzare**) realizza la finestra principale, che consente di cercare un treno fra due città (NON fra due specifiche stazioni!) in una certa data. A tal fine, sono offerte due caselle a discesa (combobox) da cui l'utente sceglie le città di partenza e arrivo; più sotto, un componente ***TrainDateSpinner*** (fornito nello Start Kit) consente di specificare la data del viaggio.

Premendo il pulsante FIND viene avviata la ricerca dei treni disponibili, tramite il metodo ***searchTrains*** del controller; *per ogni città si devono ovviamente considerare tutte le stazioni*. I risultati sono mostrati in una ***JTextArea***: per ogni viaggio, si riportano le stazioni di inizio e fine viaggio nonché la durata complessiva del viaggio stesso (Fig. 2).

Premendo il pulsante PRINT, i risultati sono salvati in un file di testo Results.txt.

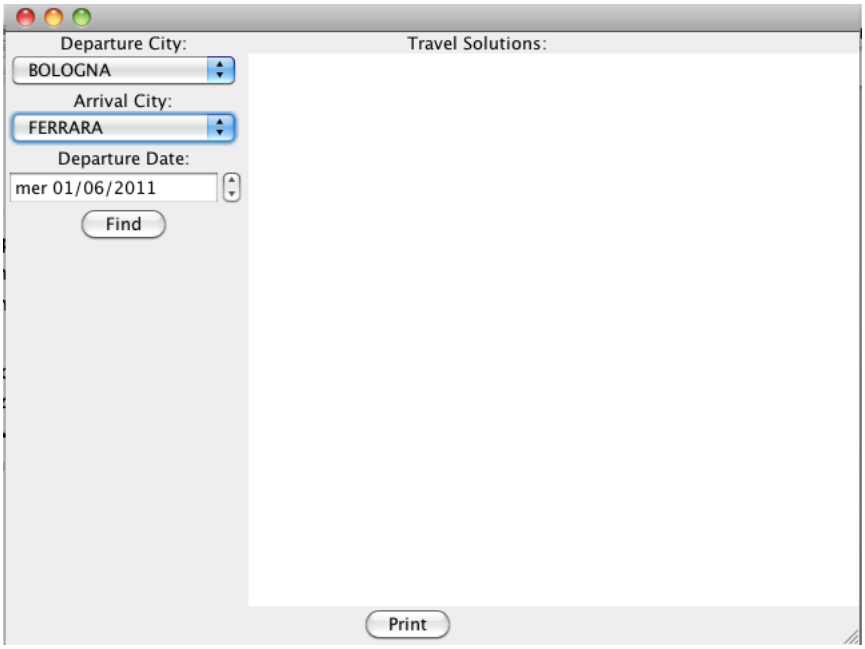


Fig.1

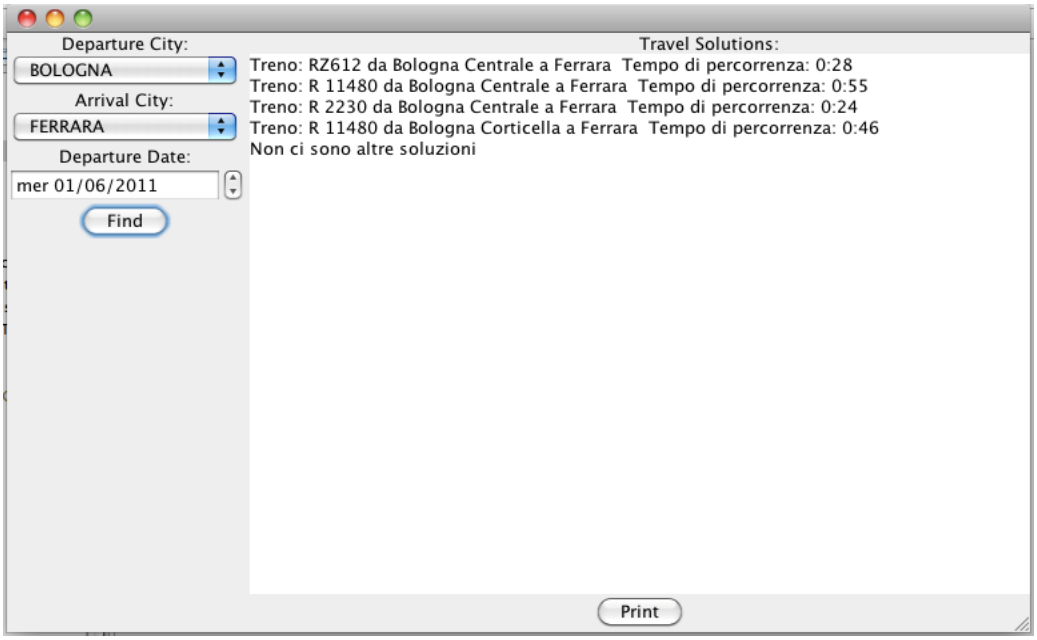


Fig.2