ESAME DI FONDAMENTI DI INFORMATICA T-2 del 16/06/2015
Proff. Enrico Denti – Gabriele 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)

Nella repubblica di Dentinia i servizi ferroviari sono offerti da due operatori concorrenti, *Sferraglia* e *ZannoRails.* L'autorità nazionale dei trasporti ha tuttavia stabilito che essi debbano offrire ai viaggiatori un'applicazione unica per la ricerca dei collegamenti, in grado anche di proporre, a richiesta, soluzioni con treni di entrambe le compagnie.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA.

Ogni città dotata di stazione ha un nome univoco che può contenere spazi (es. "Reggio Emilia"). Se la città ha un'unica stazione, il nome della città e il nome della stazione coincidono, altrimenti ogni stazione ha il proprio nome specifico, distinto da quello della città (es. "Firenze" indica la città, mentre le stazioni si chiamano "Firenze S.M.N.", "Firenze C.M.", "Firenze Rifredi", etc.).

Ogni treno è descritto da un identificativo univoco che lo caratterizza completamente, composto da:

- un carattere che rappresenta l'operatore ("S"=*Sferraglia*, "Z"=*ZannoRails*)
- un numero univoco (dispari per i treni circolanti in una direzione, pari per quelli circolanti nella direzione opposta).

Ogni treno inoltre specifica:

- le stazioni servite
- l'orario di arrivo e partenza in ogni stazione (vedi nota)
- i giorni della settimana in cui si effettua (1=lunedì,…, 7=domenica)

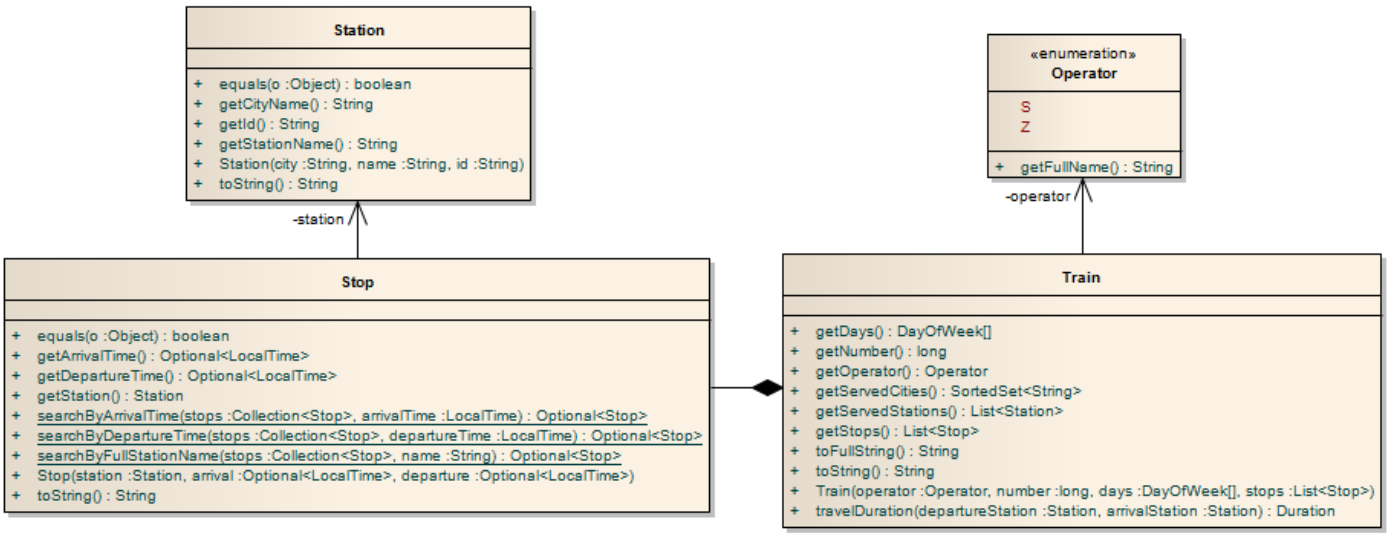
Quindi, qualora siano necessarie variazioni in alcuni giorni (es. fermate extra solo nel weekend, orari diversi nei giorni feriali e festivi, etc.), si definiscono treni distinti, con numeri differenziati.

NB: l'orario di partenza nella stazione finale e l'orario di arrivo nella stazione di origine sono ovviamente indefiniti.

I dati sono rappresentati in due file di testo: Stazioni.txt contiene l'elenco delle città servite, associate alle rispettive stazioni, mentre Treni.txt contiene l'elenco di tutti i treni, di entrambi gli operatori, previsti in orario.

Parte 1 – Modello dei dati *(namespace dentiniarail.model)* — Punti: 8

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



a) la classe ***Station*** (fornita) rappresenta una stazione, caratterizzata dal nome della città e dal nome specifico di quella stazione; il metodo ***getId*** restituisce l'identificatore univoco della stazione (ad esempio, "VE01"), mentre i metodi ***getCityName*** e ***getStationName*** restituiscono rispettivamente il nome della città e il nome *specifico* della stazione; ***toString*** restituisce invece il nome *completo* della stazione, ottenuto concatenando nome città e nome stazione con un spazio intermedio [ESEMPIO: se ***getCityName*** e ***getStationName*** restituiscono rispettivamente "Bologna" e "Centrale", ***toString*** restituisce "Bologna Centrale"];

b) l'enumerativo ***Operator*** (fornito) rappresenta un operatore ferroviario, caratterizzato dal suo simbolo (il valore dell'enumerativo) e dal suo nome completo, restituito dal metodo ***getFullName***;

c) la classe ***Stop*** (fornita) rappresenta una fermata: a tal fine è composta da una ***Station*** e dagli orari di arrivo e partenza del treno, rappresentati come Optional poiché possono risultare indefiniti per la stazione iniziale/finale del viaggio; sono disponibili metodi accessor a tali proprietà. Sono altresì forniti due metodi statici di utilità (***searchByStation***, ***searchByFullStationName***) per cercare una fermata in una collezione di fermate in base a alla stazione desiderata.

d) la classe ***Train*** **(da realizzare)** rappresenta un treno con le sue proprietà: il costruttore ha quattro argomenti – l'operatore (di tipo ***Operator***), il numero del treno (un ***long***), i giorni di effettuazione (un array di ***DayOfWeek***) e le fermate (una lista di ***Stop***) – e lancia ***IllegalArgumentException*** se uno o più di essi è assurdo (operatore nullo, numero treno non positivo, array dei giorni o lista delle fermate nulli o vuoti). Il metodo ***getServedCities*** restituisce l'insieme dei nomi delle città servite dal treno in ordine alfabetico e senza ripetizioni, mentre ***getServedStations*** restituisce la lista delle stazioni (***Station***) servite dal treno, secondo la sequenza naturale del percorso. Il metodo ***travelDuration(Station partenza, Station arrivo)*** calcola invece la durata del viaggio fra le due stazioni date, lanciando ***IllegalArgumentException*** in caso di parametri nulli, o se una o entrambe le stazioni non sono servite da quel treno o se il treno circola in senso opposto (ovvero, l'orario di arrivo è antecedente all'orario di partenza: per semplicità non si tiene conto di treni che viaggiano a cavallo della mezzanotte).

e) Il metodo ***toString*** deve produrre una descrizione sintetica del treno, della forma:

```
ZannoRails 9902 BOLOGNA Centrale (6.18) / TORINO P.S. (8.33)
```

mentre il metodo ***toFullString*** deve produrre una descrizione completa, della forma:

```
ZannoRails 9902 da BOLOGNA Centrale (06:18) a TORINO P.S. (08:33), circola lun-mar-mer-gio-ven
ferma a: REGGIO EMILIA AV Mediopadana (06:41/06:43), MILANO Rogoredo (07:21/07:23), MILANO Porta
Garibaldi (07:41/07:44), MILANO Rho Fiera Expo (07:53/07:55)
```

SUGGERIMENTO: per formattare il nome del giorno con tre lettere ("`lun`", etc), usare un ***DateTimeFormatter*** creato con la specifica `DateTimeFormatter.ofPattern("E", Locale.ITALY)`, chiamando poi il metodo ***format*** passandogli come argomento il giorno (***DayOfWeek***) da formattare.

Parte 2 – Persistenza *(namespace dentiniarail.persistence)* Punti: 10

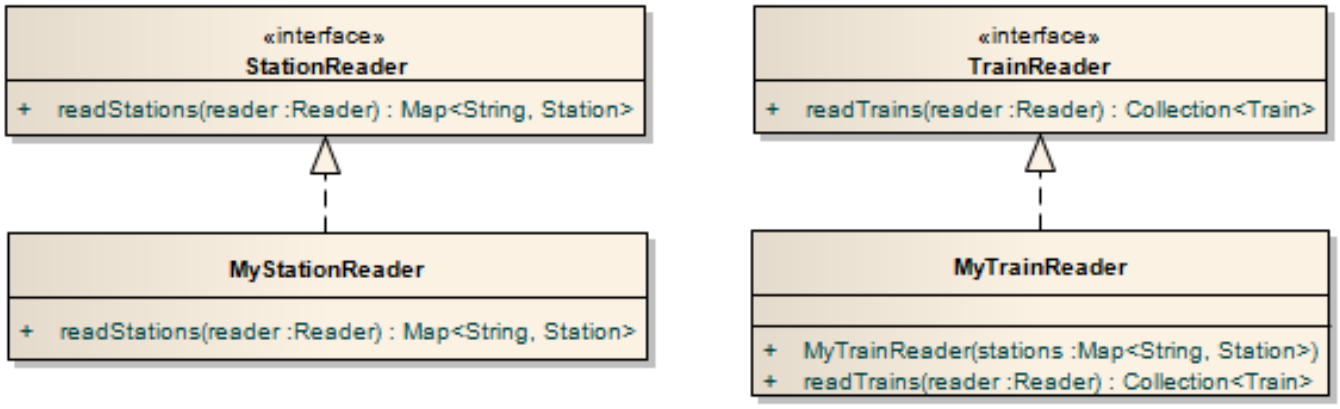
Come sopra anticipato, il file di testo Stazioni.txt contiene l'elenco delle città servite da uno o entrambi gli operatori, associate alle rispettive stazioni. Ogni riga contiene il nome della città (che può includere spazi), una tabulazione e poi l'elenco delle stazioni associate (che possono includere spazi *ma non ripetono il nome della città*), separate da virgole: ogni stazione è anche associata a un identificativo univoco, separato dal nome della stazione da una barra.

```
FIRENZE        S.M.N./FI01, Campo Marte/FI02, Rifredi/FI03
VENEZIA        S.Lucia/VE01, Mestre/VE02
BOLOGNA        Centrale/BO00
…
```

Il file di testo Treni.txt contiene l'elenco di tutti i treni, di entrambi gli operatori, previsti in orario. Ogni treno è descritto da una riga, strutturata in una serie di campi separati da tabulazioni:

- identificativo univoco del treno (il primo carattere indica l'operatore, il resto è numerico)
- giorni di effettuazione (1=lunedì,…, 7=domenica; i giorni di non effettuazione sono semplicemente omessi)
- identificativi univoci delle stazioni servite, ciascuna con i rispettivi orari di arrivo e partenza espressi nella forma `hh.mm` (fanno eccezione la stazione di origine e la stazione finale, in cui l'orario rispettivamente di arrivo o di partenza è indefinito e perciò sostituito dalla stringa "`--`"). Stazioni e orari sono separati fra loro da barre, mentre i blocchi "stazione/orarioArrivo/orarioPartenza" sono separati fra loro da virgole.

```
Z9902   12345    BO00/--/6.18,RE01/6.41/6.43,MI02/7.21/7.23,MI01/7.41/7.44,MI04/7.53/7.55,TO01/8.33/--
S9570   12345    BO00/--/11.23,RE01/11.42/11.44,MI02/12.22/12.24, ... ,TO01/13.35/13.37,TO00/13.45/--
…
```



1. L'interfaccia ***StationReader*** (fornita) dichiara il metodo ***readStations*** che legge da un reader tutte le stazioni e restituisce una ***Map<String, Station>*** che associa la stazione al suo identificatore univoco.
2. La classe ***MyStationReader*** (fornita) implementa tale interfaccia, lanciando ***IOException*** in caso di problemi di accesso al file o incapsulando in una ***BadFileFormatException*** gli eventuali problemi di formato (nome stazione vuoto, formato stazione o identificativo non rispettato).
3. L'interfaccia ***TrainReader*** (fornita) dichiara i metodi per leggere da un reader un insieme di ***Train***: ***readTrains*** restituisce l'elenco dei treni letti sotto forma di ***Collection<Train>.***
4. La classe ***MyTrainReader*** (**da realizzare**) implementa tale interfaccia, lanciando ***IOException*** in caso di problemi di accesso al file o incapsulando in una ***BadFileFormatException*** gli eventuali problemi di formato (operatore mancante o non valido, numero treno mancante o non numerico, giorni di effettuazione mancanti o non numerici nel range 1-7, identificativo stazione sconosciuto). Il costruttore di ***MyTrainReader*** riceve in ingresso la mappa ***Map<String, Station>*** (restituita dal metodo ***readStations*** di ***StationReader***), necessaria per ricavare la stazione a partire dal suo identificativo (ovvero, ad esempio, per poter risalire da "TO01" a "TORINO P.S.").

Parte 3 – GUI *(namespace dentiniarail.ui)* (punti: 12)

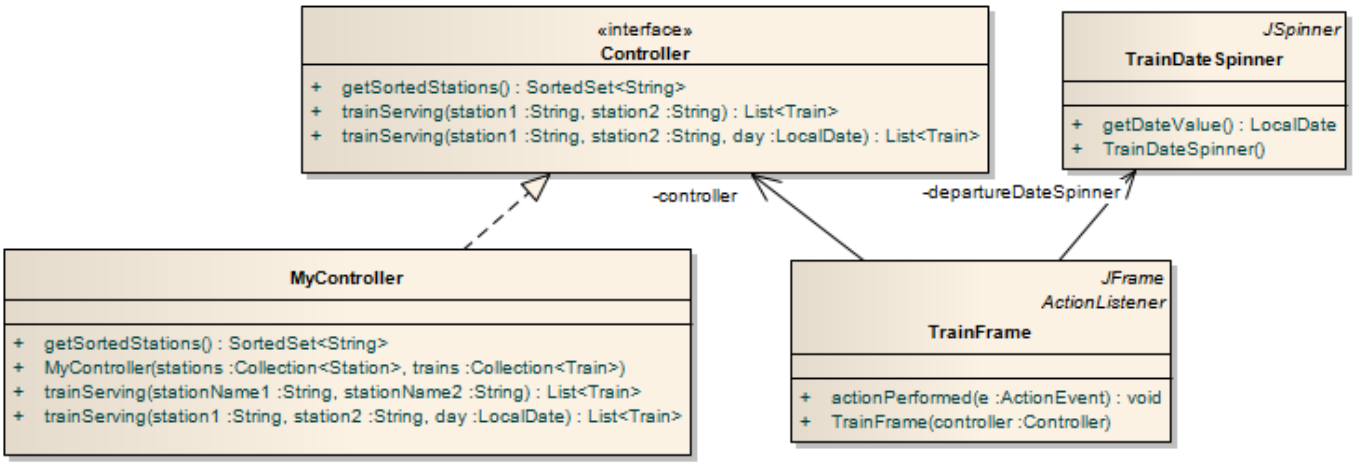
Controller *(punti 6)*

L'interfaccia ***Controller*** dichiara i seguenti metodi:

- il metodo ***trainServing(String station1, String station2)*** cerca i treni disponibili fra due stazioni *in qualunque direzione circolino* (ossia includendo sia i treni da ***station1*** a ***station2*** sia quelli da ***station2*** a ***station1***) gli argomenti ricevuti sono i *nomi completi* delle due stazioni;

  [SUGGERIMENTO: sfruttare il metodo ***Stop***.***searchByFullStationName*** per cercare la fermata fra quelle del treno]
- il metodo ***trainServing(String station1, String station2, LocalDate data)*** opera come sopra ma restringendo la ricerca ai soli treni che circolano *nella data specificata*, sfruttata per ricavare il giorno della settimana corrispondente (ad esempio, se la data è il 16.6.2015, si cercheranno i soli treni circolanti il martedì)
  [SUGGERIMENTO: sfruttare il metodo precedente e filtrare i risultati]
- il metodo ***getSortedStations*** che restituisce l'insieme di tutte le stazioni ordinate alfabeticamente.

La classe ***MyController*** (**da realizzare**) implementa tale interfaccia: il costruttore riceve in ingresso la collezione delle stazioni e la collezione dei treni, ovvero rispettivamente ***Collection<Station>*** e ***Collection<Train>***.



Interfaccia utente (namespace dentiniarail.ui) *(punti 6)*

L'interfaccia utente deve essere simile (non necessariamente identica) all'esempio mostrato nelle figure seguenti.

La classe ***TrainFrame*** **(da realizzare)** realizza la finestra principale, il cui scopo è cercare un treno fra due stazioni in una data specificata. A tal fine, due caselle a discesa (combo) sono pre-popolate con tutte le stazioni: l'utente deve scegliere le stazioni di partenza e arrivo, selezionare la data e premere il pulsante CERCA (Fig. 1).

In risposta, il sistema recupera la lista di treni circolanti fra tali stazioni nella data specificata (tramite il metodo ***trainServing/3*** del controller), *esclude quelli che circolano nella direzione opposta* a quella desiderata (***trainServing*** li restituisce tutti, ma per filtrarli basta verificare che l'orario di partenza sia anteriore a quello di arrivo) e li mostra nell'area di testo (ad esempio, in Fig. 2 sono mostrati i treni circolanti il 26 maggio 2015, un martedì, mentre in Fig. 3 e Fig. 4 sono mostrati i treni circolanti un sabato o una domenica).

["PIANO B": se per motivi di tempo non si è riusciti a completare il metodo ***trainServing/3*** del controller, ma si è realizzato il fratello minore ***trainServing/2*** (senza data), utilizzare quest'ultimo nella GUI per proseguire]

L'area di testo deve utilizzare il font "Courier New", grassetto, 12 punti. Per ogni viaggio si devono mostrare operatore e numero del treno, le stazioni di partenza e arrivo con i rispettivi orari e la durata del viaggio in minuti.

È fornita già pronta nello start kit la classe ***trains.ui.TrainDateSpinner*** che rappresenta un "date picker", ossia un componente grafico per scegliere una data (vedere figure): il metodo ***getDateValue*** restituisce la data attualmente selezionata sotto forma di opportuna ***LocalDate.***

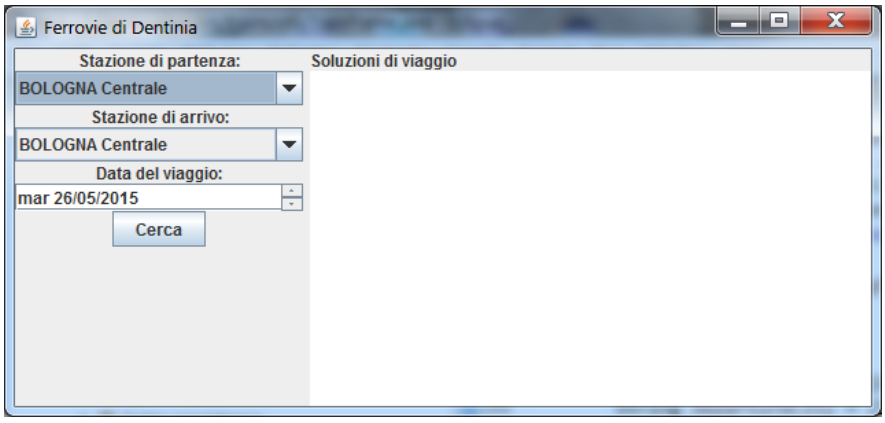


Fig 1

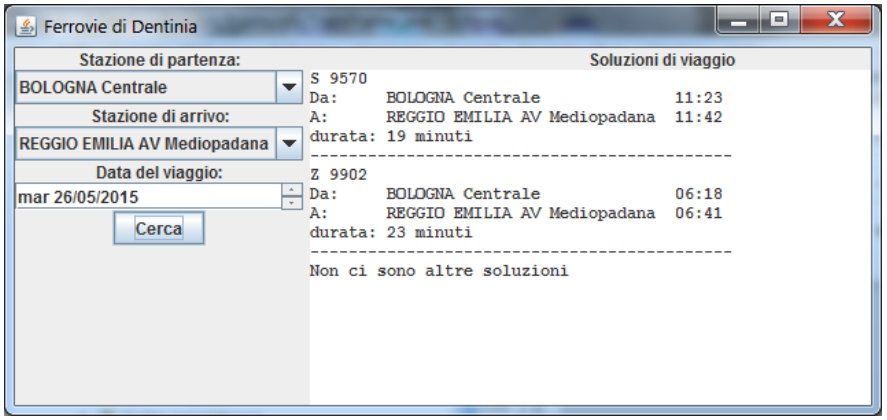


Fig 2

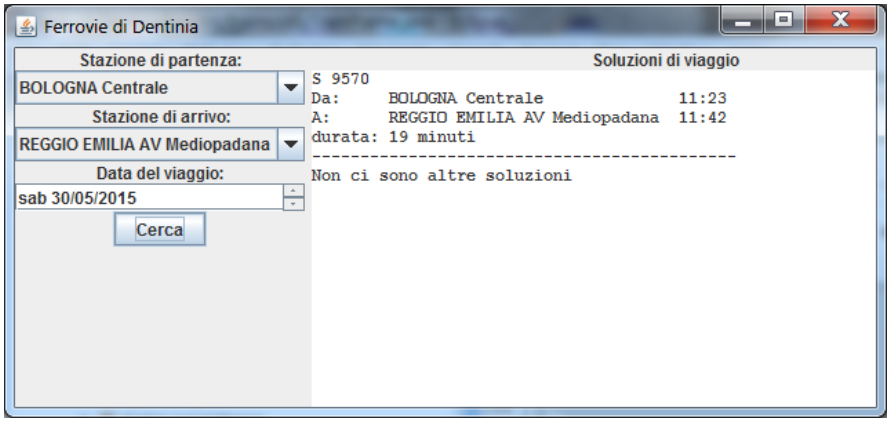


Fig 3

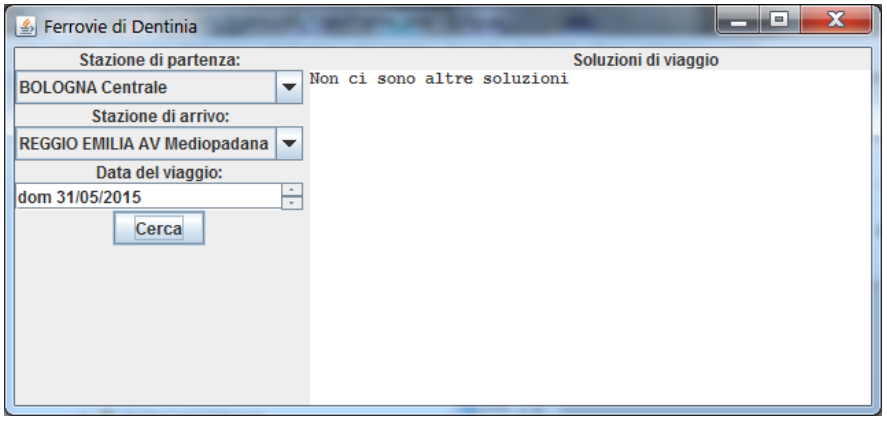


Fig 4