ESAME DI FONDAMENTI DI INFORMATICA T-2 del 23/7/2020
Proff. E. Denti – R. Calegari – A. Molesini
Tempo a disposizione: 3 ore

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"

Le ***Rapidissime Ferrovie di Dentinia*** (RFD), che gestiscono una antica rete ferroviaria a vapore fra alcune città, hanno chiesto lo sviluppo di una nuova funzionalità nella pre-esistente applicazione per la ricerca dei percorsi fra stazioni della loro rete: oltre ai percorsi possibili con relativo chilometraggio, desiderano che sia calcolato anche il *tempo di percorrenza*

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

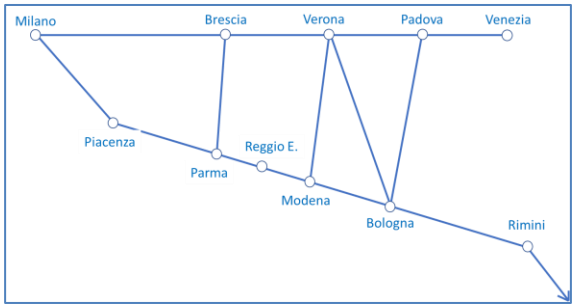
Ogni *linea ferroviaria* è descritta da una sequenza di *stazioni,* caratterizzate ciascuna dal nome del luogo (che può contenere spazi) e dalla *progressiva chilometrica* (ossia, la distanza dal capolinea iniziale). In alcune stazioni, dette *punti di interscambio* (*hub*), si intersecano due o più linee: ciò consente di offrire ai viaggiatori anche soluzioni di viaggio con *al più un cambio intermedio.* Non si considerano soluzioni con due o più cambi, perché scomode.

Si dice *segmento* una tratta orientata fra due qualsiasi stazioni *della stessa linea*. Un segmento è *semplice* se non esistono al suo interno stazioni intermedie, ossia non può essere ulteriormente spezzato in sotto-segmenti.

ESEMPIO: il segmento Parma-Milano, che è cosa distinta dal segmento Milano-Parma (che descriverebbe un percorso nella direzione opposta), *non* è semplice, essendo presenti al suo interno molte stazioni intermedie.

Un *percorso* fra due stazioni si dice *diretto* se non prevede cambi (ed è quindi costituito da un unico segmento), *indiretto* altrimenti: in questo secondo caso il percorso è costituito da *due o più segmenti consecutivi* (ossia tali che la stazione terminale di ciascuno coincide con quella di inizio del successivo).

ESEMPIO: fra Parma e Milano, nella rete in figura, sono possibili due percorsi: uno diretto (Parma-Milano sulla linea Bologna-Milano) e uno indiretto (Parma-Brescia sulla linea omonima + Brescia-Milano sulla linea Venezia-Milano).



Ovviamente, la lunghezza di un percorso è pari alla *somma delle lunghezze dei segmenti* che lo costituiscono, a loro volta pari alla differenza fra le progressive chilometriche delle rispettive stazioni di estremità.

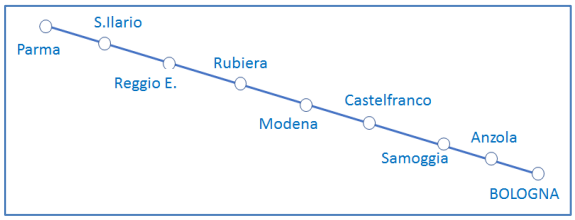
ESEMPIO: il percorso fra Lodi (km 183,80 della linea Bologna-Milano) e Rimini (km 111,04 della linea Bologna-Lecce) è di 294,84 km e comprende due segmenti (Lodi-Bologna e Bologna-Rimini).

Per calcolare i tempi di percorrenza occorre conoscere la *velocità* ammessa sui *singoli tratti* di linea. Per semplicità, si suppone che essa sia *costante all'interno di un singolo segmento semplice*: si conviene perciò che *ogni stazione* sia associata alla *velocità* da mantenere *nel tratto di linea fino alla stazione successiva*.

ESEMPIO: sulla linea Bologna-Milano, il primo segmento semplice (Bologna Centrale-Anzola dell'Emilia) potrebbe avere una velocità ammessa di 120 km/h, mentre il successivo potrebbe averne una diversa (es. 140 km/h) e così via, di stazione in stazione, fino a Milano Centrale (termine linea).

Il tempo di percorrenza di un dato percorso si ottiene quindi *sommando i tempi di percorrenza* dei *segmenti semplici* che lo compongono, così da considerare tutti gli eventuali cambiamenti di velocità lungo la linea.

ESEMPIO: nel caso a lato, il tempo di percorrenza della tratta Bologna Centrale-Modena si ottiene sommando i tempi parziali dei segmenti semplici Bologna Centrale-Anzola, Anzola-Samoggia, Samoggia-Castelfranco e Castelfranco-Modena, a cui possono essere associate velocità diverse: NON prendendo per buona la velocità iniziale di Bologna fino a Modena!



CONVENZIONE: poiché una linea è descritta nel verso dal capolinea iniziale (progressiva km 0,0) al capolinea finale, le stazioni costituiscono una *sequenza ordinata in ordine di progressiva chilometrica crescente*. Di conseguenza, *anche le velocità ammesse* si intendono "da quella stazione alla successiva" ***nello stesso verso***.

ESEMPIO: la linea Bologna-Milano è descritta partendo da Bologna Centrale (progressiva km 0,0) e terminando a Milano Centrale (progressiva km 216,18): perciò, la velocità associata alla stazione di Bologna Centrale si intende valida fino ad Anzola, quella associata ad Anzola si assume valida fino a Samoggia, e così via.

Di ciò bisogna tenere conto quando si calcolano i tempi di percorrenza: la velocità da assumere nel tratto da A a B, infatti, *potrebbe non essere quella associata alla stazione A*, *ma quella associata alla stazione B*, se la linea è descritta nel verso opposto, "da B verso A" (ossia la progressiva chilometrica di A è maggiore di quella di B).

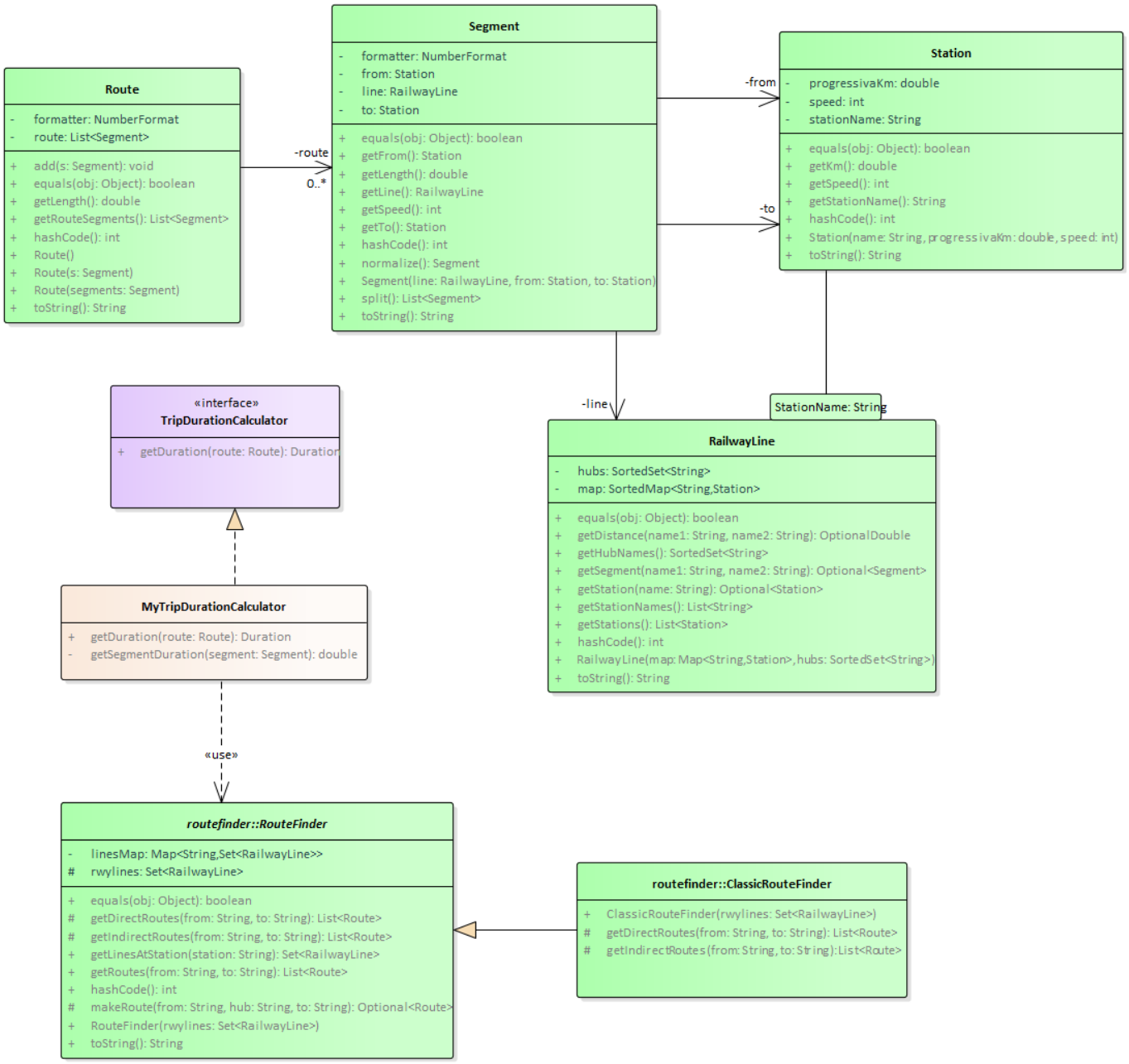
ESEMPIO: la velocità ammessa nella tratta Modena (progressiva km 36,93) - Castelfranco (progressiva km 25,01) della linea Bologna-Milano **non è** quella associata alla stazione di Modena, ma quella associata alla stazione di Castelfranco, appunto perché la linea Bologna-Milano è descritta partendo da Bologna, non da Milano. La velocità associata a Modena è quella valida fino alla "sua" stazione "successiva", ossia fino a Rubiera (progressiva 49,59); e così via proseguendo (quella di Rubiera copre il segmento fino a Reggio Emilia, etc.).

La rete delle ***Rapidissime Ferrovie di Dentinia***, costituita da *sette linee*, è quella illustrata sopra: ogni linea è descritta in un singolo file di testo, di estensione "**.line**", il cui formato è riportato più oltre.

TEMPO STIMATO PER SVOLGERE L'INTERO COMPITO: 1h50 – 2h20

Parte 1 (punti: 20)

Modello dei dati (package rfd.model) **[TEMPO STIMATO: 20-30 minuti]** *(punti: 9)*



SEMANTICA:

a) la classe ***Station*** (fornita) rappresenta una stazione, caratterizzata da nome, progressiva chilometrica (numero reale in km) e velocità ammessa (numero intero in km/h) "fino alla stazione successiva" della linea di cui fa parte.

b) la classe ***Segment*** (fornita) rappresenta un segmento di una data *linea* (***RailwayLine***) compreso fra due ***Station*** distinte (anche non adiacenti): sono presenti i classici metodi per recuperare gli elementi, produrre un'opportuna stringa descrittiva, nonché *equals* e *hashcode*. Da segnalare i seguenti metodi di particolare interesse:

- *normalize*: produce il corrispondente segmento "normalizzato", orientato nel verso della linea
  ESEMPIO: normalizzando il segmento Modena-Bologna viene restituito il segmento Bologna-Modena, che è orientato secondo il verso crescente della linea Bologna-Milano (ovviamente, se il segmento iniziale è già normalizzato, viene restituito intoccato)

- *split*: suddivide un segmento nella lista dei suoi segmenti semplici, elencati *sempre e comunque* in ordine normalizzato (anche se il segmento iniziale non lo era).
  ESEMPIO: splittando il segmento Modena-Bologna verrà restituita la lista di segmenti semplici [Bologna-Anzola, Anzola-Samoggia, Samoggia-Castelfranco, Castelfranco-Modena], rispettando cioè *sempre e comunque* l'ordinamento della linea Bologna-Milano.

c) la classe ***RailwayLine*** (fornita) rappresenta una linea ferroviaria intesa come insieme di ***Station***, che il costruttore si aspetta di ricevere sotto forma di *mappa* indicizzata per *nome della stazione*. Il secondo argomento del costruttore è l'insieme ordinato dei *nomi* delle stazioni della linea che possono fungere da *punti di interscambio*. La classe offre svariati metodi per:
   - recuperare la lista dei nomi delle stazioni (metodo *getStationNames*) / delle stazioni come oggetti (metodo *getStations*), nonché l'insieme ordinato dei nomi degli hub (metodo *getHubNames*)
   - recuperare una ***Station***, se esiste, dato il suo nome (metodo *getStation*): restituisce un optional
   - calcolare la distanza fra due ***Station*** distinte, se esistono (metodo *getDistance*)
   - recuperare il ***Segment***, se esiste, corrispondente alla tratta compresa fra due ***Station*** distinte (metodo *getSegment*)
   - emettere una stringa descrittiva (metodo *toString*)

d) la classe ***Route*** (fornita) rappresenta un *percorso* per un viaggiatore, inteso come *sequenza di* ***Segment***. I costruttori consentono di costruire la ***Route*** in tre situazioni tipiche (inizialmente vuota, inizialmente con un solo segmento, o a partire da un numero variabile di segmenti): è comunque possibile aggiungere via via altri segmenti *in coda* alla sequenza, tramite il metodo *add*. Opportuni accessor consentono di recuperare i vari elementi: anche in questo caso sono presenti *equals*, *hashcode* e *toString*.

e) La classe astratta ***RouteFinder*** (fornita sotto forma di libreria JAR, senza sorgente) rappresenta l'entità in grado di cercare percorsi fra due città date (metodo *getRoutes*): a tal fine riceve l'insieme di ***RailwayLine*** che rappresenta la rete ferroviaria. Il metodo *getLinesAtStation* restituisce l'insieme – eventualmente vuoto - delle ***RailwayLine*** che servono una data stazione. I due metodi protetti *getDirectRoutes* e *getIndirectRoutes* (stessi argomenti di *getRoutes*) sono destinati a essere implementati da sottoclassi concrete: in questa classe la loro implementazione si limita a lanciare ***UnsupportedOperationException*** a fini di test. Infine, il metodo protetto *makeRoute* costruisce, se esiste, il percorso indiretto fra due città date passando per l'hub intermedio specificato.

f) La classe concreta ***ClassicRouteFinder*** (fornita anch'essa sotto forma di libreria JAR, senza sorgente) estende opportunamente ***RouteFinder***.

g) L'interfaccia ***TripDurationCalculator*** (fornita) dichiara il metodo *getDuration* che calcola la durata di un percorso (***Route***).

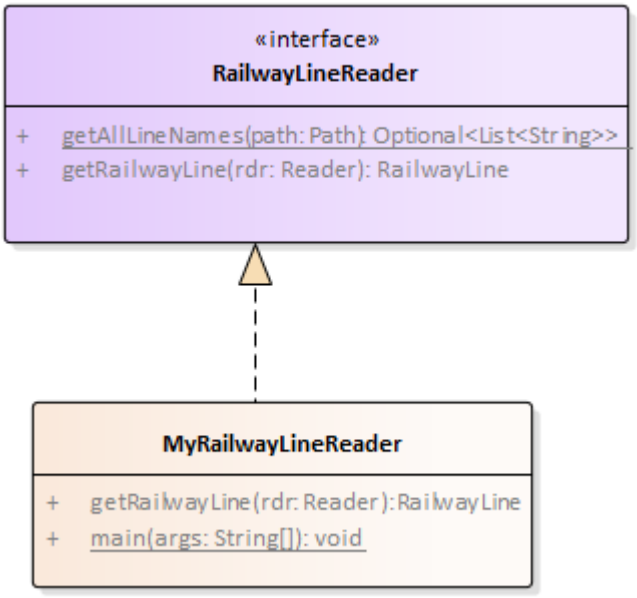
h) la classe ***MyTripDurationCalculator*** (**da realizzare**) concretizza tale interfaccia implementando *getDuration* nel modo spiegato nel *Dominio del Problema,* ovvero: ***(punti: 9)*** **[TEMPO STIMATO: 20-30 minuti]**
   - recupera i segmenti del percorso
   - li normalizza
   - li splitta nei segmenti semplici che li compongono
   - utilizza questi ultimi per il calcolo del tempo di percorrenza (tempo = spazio / velocità), assumendo come velocità quella ammessa nel segmento semplice considerato.

Persistenza (rfd.persistence) **[TEMPO STIMATO: 50-60 minuti]** ***(punti 11)***

Sono presenti vari file di testo con estensione "**.line**", uno per ogni linea ferroviaria, tutti formattati secondo lo stesso schema. Le righe sono tutte di identica lunghezza (è una specifica) e contengono nell'ordine:

- nei primi 8 caratteri, la *progressiva chilometrica* (un numero reale con due cifre decimali, formattato secondo le convenzioni italiane), con eventuali spazi davanti e/o dietro;

- in fondo, preceduta da uno spazio, la *velocità ammessa* (un numero intero)
- in mezzo, il *nome della stazione* (che può contenere spazi o qualunque altro carattere) seguito, per le stazioni che fungono da interscambio con altre linee, dalla parola "HUB", scritta con qualunque mix di maiuscole e/o minuscole.
  ATTENZIONE: non è garantito che prima della parola "HUB" vi siano spazi o altri separatori: è anche possibile che siano tutti consecutivi (ad esempio, "Torre CencelloHUB").




```
  0,00  Bologna Centrale          HUB 125
  2,63  Bologna San Vitale            170
  6,55  San Lazzaro di Savena         170
 13,01  Ozzano dell'Emilia            170
...
 96,22  Savignano sul Rubicone        140
101,27  Santarcangelo di Romagna      140
...
```

SEMANTICA:

a) L'interfaccia ***RailwayLineReader*** (fornita) dichiara il metodo *getRailwayLine,* che legge da un Reader (ricevuto come argomento) i dati di una singola linea ferroviaria, restituendo la corrispondente ***RailwayLine***. NB: l'interfaccia contiene anche il metodo statico *getAllLineNames* che restituisce la lista dei nomi di file di tipo ".line" (ossia, quelli che descrivono le linee ferroviarie) contenuti nella cartella passata come argomento. Tale metodo è invocato automaticamente dal main dell'applicazione (vedere Parte 2).

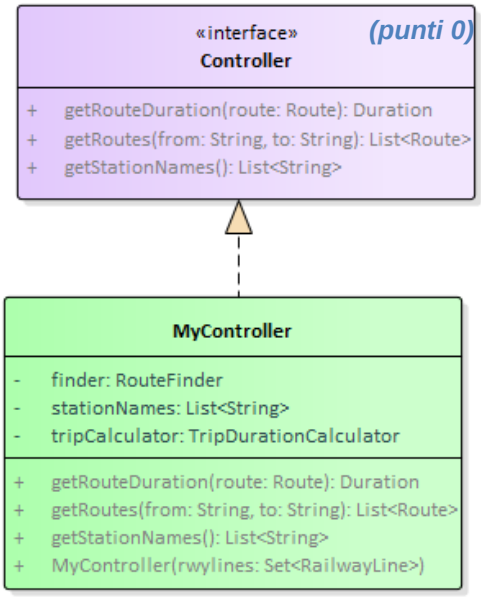
b) La classe ***MyRailwayLineReader*** (**da realizzare**) implementa ***RailwayLineReader***: non prevede costruttori, si limita a implementare il metodo *getRailwayLine* come sopra specificato. In caso di problemi di I/O deve essere propagata l'opportuna ***IOException***, mentre in caso di ***Reader*** nullo o altri problemi di formato dei file deve essere lanciata una opportuna ***IllegalArgumentException***, il cui messaggio dettagli l'accaduto.
In particolare, il reader deve verificare: 1) che la progressiva chilometrica abbia la forma di numero reale separato da virgola; 3) che la velocità sia un numero intero; 4) che il nome della stazione non sia vuoto né consista della sola parola "HUB".
SUGGERIMENTO: potrebbe essere comodo, in questo caso, sfruttare i metodi della classe String…

Parte 2

[TEMPO STIMATO: 40-50 minuti] **(punti: 10)**

Controller (rfd.controller)

Il Controller (fornito) è organizzato secondo il diagramma UML in figura.



SEMANTICA:

a) L'interfaccia ***Controller*** (fornita) dichiara i metodi *getStationNames, getRoutes* e *getRouteDuration*.

b) La classe ***MyController*** (fornita) implementa tale interfaccia, fornendo:

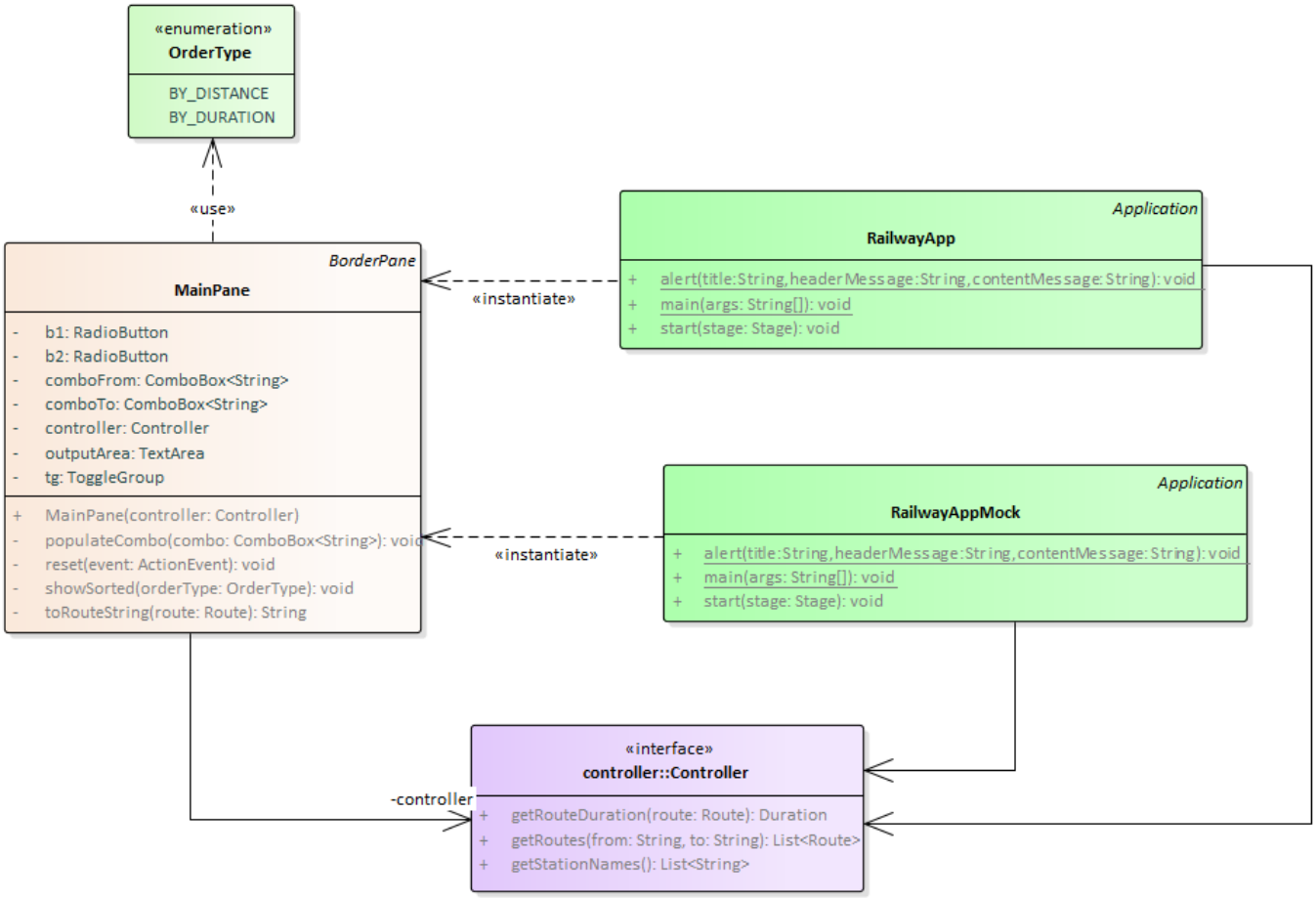
- il costruttore che, dall'insieme delle linee societarie, estrae l'elenco dei nomi delle stazioni, quindi crea internamente il ***RouteFinder*** e il ***TripDurationCalculator*** necessari;

- implementa i tre metodi delegando il lavoro ai rispettivi *finder* e *trip duration calculator*.

NB: la lista dei nomi di stazione restituita da *getStationNames* è già ordinata alfabeticamente.

Interfaccia utente (rfd.ui) [TEMPO STIMATO: 40-50 minuti] *(punti 10)*

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



La classe ***RailwayApp*** (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 ***RailwayAppMock***.

Entrambe le classi contengono anche il **metodo statico ausiliario *alert***, utile per mostrare avvisi all'utente.

Il MainPane è fornito *parzialmente realizzato*: è presente la parte strutturale, mentre manca la parte di popolamento combo e gestione degli eventi.

La classe ***MainPane*** (**da completare**) estende ***BorderPane*** e prevede:

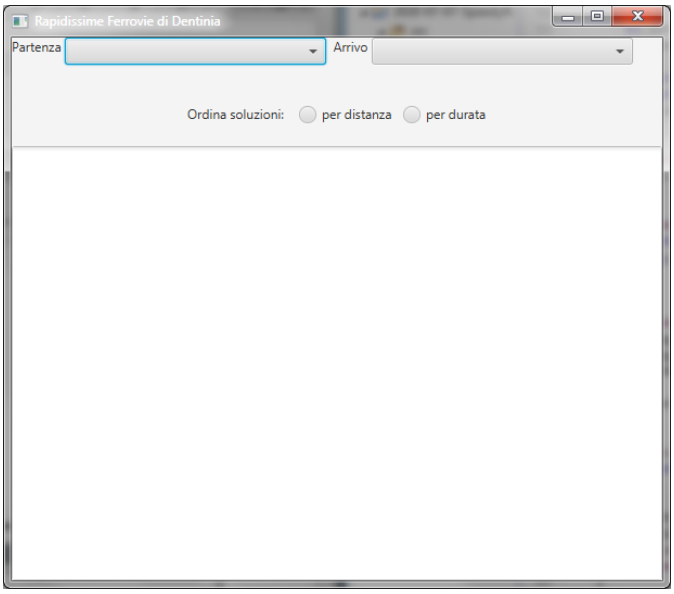
1) in alto, due ***ComboBox*** da popolare con *l'elenco alfabetico ordinato di tutte le stazioni di tutte le linee*
2) sotto, due ***RadioButton*** per scegliere il criterio di ordinamento dei percorsi, per distanza o per durata
3) in basso, una ***TextArea*** che mostra i percorsi (***Route***) trovati, nell'ordine richiesto.

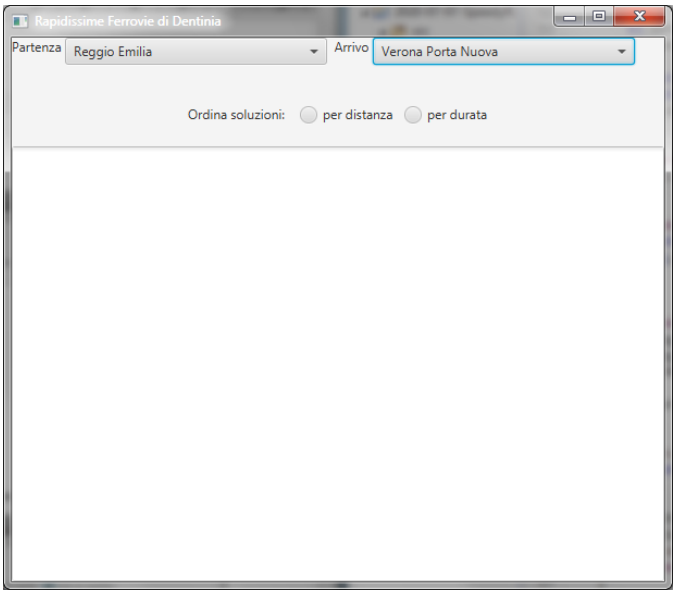
La **parte da completare** comprende l'uso e/o l'implementazione dei seguenti metodi:

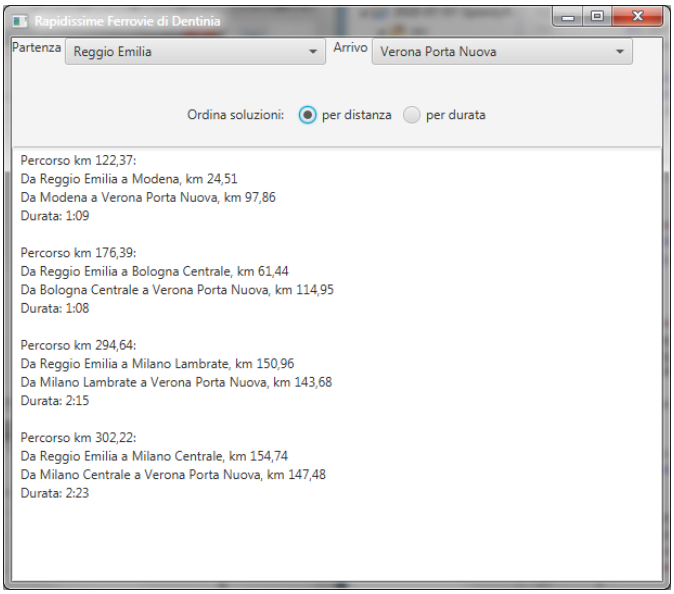
1) *populateCombo,* che popola la combo con *l'elenco alfabetico ordinato di tutte le stazioni*
2) *reset,* da chiamare in risposta all'evento di selezione delle combo, che riporta la GUI allo stato iniziale, svuotando la ***TextArea*** e riportando i due ***RadioButton*** allo stato iniziale di nessun pulsante selezionato
3) *showSorted,* da chiamare in risposta all'evento di pressione dei due ***RadioButton,*** ha come argomento di ingresso un valore dell'enumerativo ***OrderType*** che permette di discriminare l'ordinamento desiderato (per durata o per distanza chilometrica): il metodo reagisce all'evento ottenendo e mostrando l'elenco

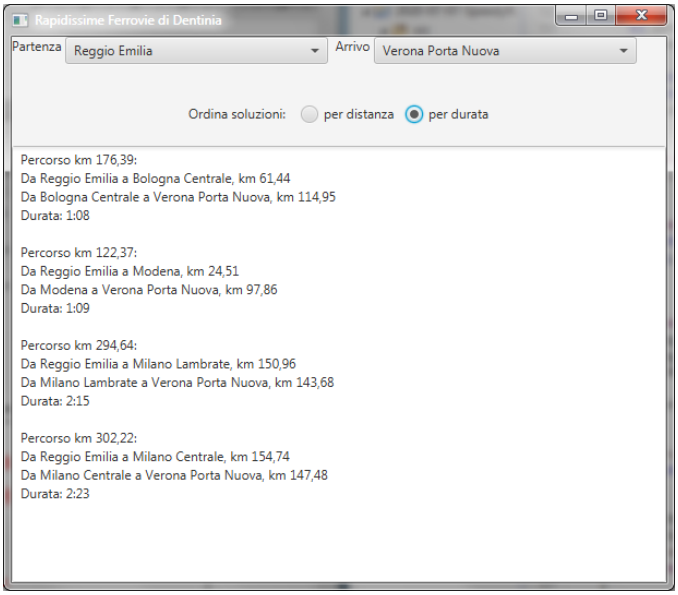
dei percorsi ordinandolo col giusto comparatore, secondo il criterio di confronto richiesto.
NB: si avvale del metodo ausiliario *toRouteString* per mostrare il percorso con la durata ben formattata.

4) *toRouteString,* data una ***Route***, produce una stringa nel previsto formato di uscita, concatenando alla *toString* di ***Route***, su riga separata, l'indicazione della durata del percorso nel formato `ore:minuti`, coi minuti su esattamente due cifre.









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?