ESAME DI FONDAMENTI DI INFORMATICA T-2 del 12/6/2023
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"

L'Associazione Consumatori di Dentinia ha richiesto lo sviluppo di un'applicazione che calcoli il ritardo di un treno secondo diversi criteri, per poter poi scegliere il più adeguato alle proprie rilevazioni di qualità del servizio.

L'applicazione deve offrire all'utente la possibilità di operare secondo tre diversi criteri:

- considerare solo il ritardo (in minuti) alla stazione finale del percorso del treno;
- considerare la media dei ritardi (in minuti) in ogni stazione del percorso del treno;
- calcolare la percentuale di stazioni in cui il treno è stato "puntuale" lungo il suo percorso.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

A Dentinia, purtroppo, a volte i treni viaggiano in ritardo: per questo motivo, l'Associazione Consumatori desidera poter confrontare diversi possibili criteri per valutare la qualità del servizio offerto. A tal fine, occorre innanzitutto etichettare ogni treno con una "misura" della sua qualità, che può essere definita in modi diversi:

1) considerando solo il ritardo (in minuti) alla stazione finale del percorso del treno;
2) considerando la media dei ritardi (in minuti) in ogni stazione del percorso del treno;
3) calcolando, invece, la percentuale di stazioni in cui il treno è stato "puntuale" lungo il suo percorso, intendendosi per "puntuale" che è giunto in quella stazione *entro una certa soglia* (in minuti) rispetto all'orario previsto.

Il viaggio di ogni treno è monitorato da un sistema automatico, che salva su un apposito file (di testo) l'elenco delle stazioni servite, riportando per ciascuna sia l'orario previsto, sia quello effettivo, di arrivo e partenza del treno.

ESEMPIO: la tabella seguente riporta i dati di monitoraggio di un possibile treno

Secondo il criterio 1), il treno risulta giunto a destinazione con 14 minuti di ritardo.

Secondo il criterio 2), calcolando i ritardi in arrivo in ogni stazione e facendo la media, risultano invece ben 24 minuti di ritardo.

Secondi il criterio 3), infine, occorre in primis stabilire la soglia entro cui considerare un treno "puntuale": assumendo 5 minuti, il treno in questione risulta in ritardo nel 100% delle stazioni e lo stesso accade assumendo una soglia di 10 o 15 minuti; assumendo invece 20 minuti, risulta in ritardo nel 56% delle stazioni, che si riducono al 6% assumendo 40 minuti; e così via.

stazione	arrivo previsto	arrivo effettivo	partenza prevista	partenza effettiva
MILANO CENTRALE	--	--	13:20	13:54
MILANO LAMBRATE	13:26	13:59	13:27	14:01
MILANO ROGOREDO	13:31	14:05	13:32	14:07
LODI	13:46	14:21	13:47	14:22
CASALPUSTERLENGO	13:57	14:34	13:58	14:36
PIACENZA	14:12	14:45	14:14	14:48
FIORENZUOLA	14:26	14:58	14:27	14:59
FIDENZA	14:35	15:06	14:37	15:07
PARMA	14:58	15:18	15:00	15:19
SANT'ILARIO	15:07	15:25	15:08	15:27
REGGIO EMILIA	15:18	15:36	15:20	15:38
RUBIERA	15:27	15:43	15:28	15:45
MODENA	15:36	15:54	15:38	15:57
CASTELFRANCO EMILIA	15:45	16:02	15:46	16:04
SAMOGGIA	15:51	16:09	15:52	16:09
ANZOLA	15:56	16:13	15:57	16:14
BOLOGNA CENTRALE	16:10	16:24	--	--

Ogni file di testo contiene i dati di monitoraggio di uno specifico treno (ne sono forniti due).

TEMPO STIMATO PER SVOLGERE L'INTERO COMPITO: 2h15 – 3h

PARTE 1 – Modello dei dati: Punti 14 [TEMPO STIMATO: 60-80 minuti]

PARTE 2 – Persistenza: Punti 7 [TEMPO STIMATO: 35-45 minuti]

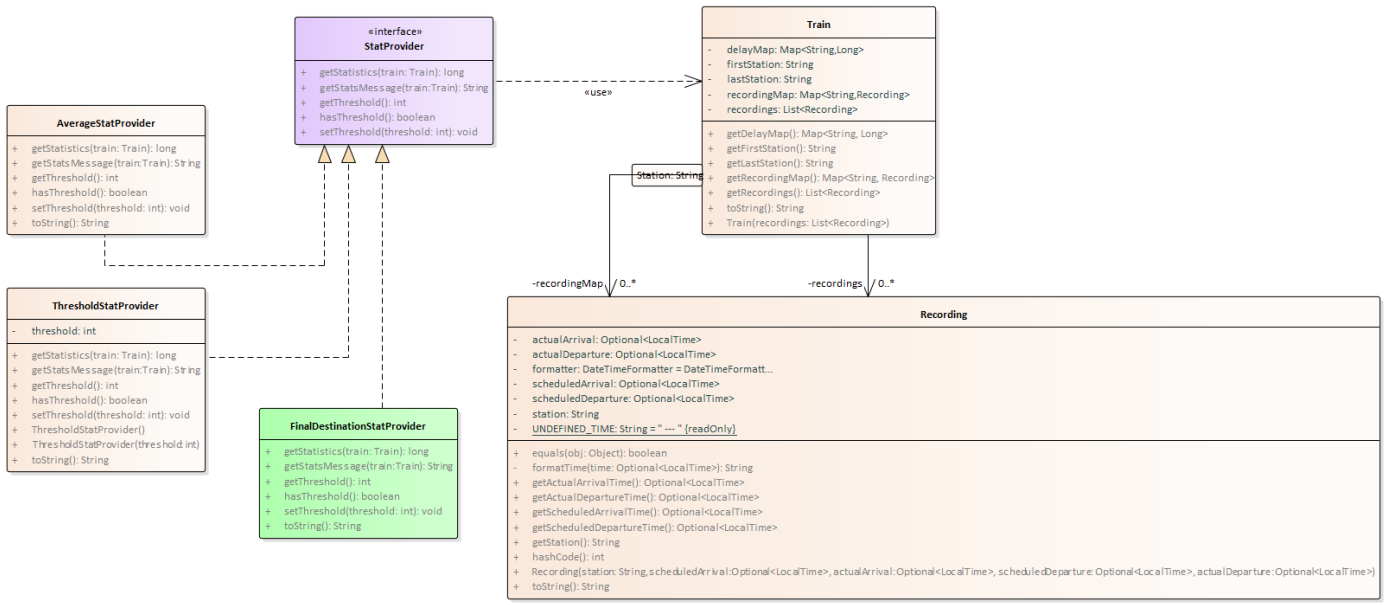
PARTE 3 – Grafica: Punti 9 [TEMPO STIMATO: 40-55 minuti]

JAVAFX – Parametri run configuration nei LAB

```
--module-path "C:\applicativi\moduli\javafx-sdk-19\lib"
--add-modules javafx.controls
```

Parte 1 – Modello dei dati (punti: 13)

Package: trainstats.model **[TEMPO STIMATO: 60-70 minuti]**



SEMANTICA:

a) La classe ***Recording*** (da completare nelle verifiche di precondizioni del costruttore) rappresenta una rilevazione del treno in una data stazione: a tal fine il costruttore riceve il *nome della stazione*, *l'orario di arrivo previsto, l'orario di arrivo effettivo, l'orario di partenza previsto* e *l'orario di partenza effettivo*. Poiché gli orari, in alcune stazioni, potrebbero non essere tutti presenti, gli argomenti sono tutti **Optional<LocalTime>**. È compito del costruttore verificare che

- nessuno di essi sia null
- gli optional empty siano solo per le coppie di orari previste (ossia, o entrambi gli orari di arrivo, o entrambi quelli di partenza): ogni altro uso di optional empty dev'essere considerato illegale
- l'orario di partenza previsto non sia antecedente all'orario di arrivo previsto
- l'orario di partenza reale non sia antecedente né all'orario di arrivo previsto, né all'orario di partenza previsto.

La classe definisce i vari accessor, nonché un'opportuna *toString* e idonee definizioni di *equals* e *hashcode*.

b) La classe ***Train*** (da completare nel costruttore) rappresenta un treno inteso come sequenza di rilevazioni: tale sequenza non può essere nulla e, poiché il percorso minimo va da una stazione iniziale a una finale, deve

sempre contenere almeno due stazioni. È compito del costruttore non solo verificare gli argomenti, ma anche predisporre tutte le strutture dati, come di seguito specificato:

- **una mappa <String, Recording>**, restituita poi dal metodo *getRecordingMap*, che associa a ogni stazione la corrispondente rilevazione;
- **una mappa <String, Long>**, restituita poi dal metodo *getDelayMap*, che associa a ogni stazione il corrispondente ritardo all'arrivo.
  IMPORTANTE: se il treno è arrivato in anticipo, il ritardo va considerato zero (mai negativo!)

La classe definisce inoltre vari accessor, nonché un'opportuna *toString*; in particolare, i due metodi *getFirstStation* e *getLastStation* restituiscono il nome rispettivamente della stazione iniziale e finale.

c) L'interfaccia ***StatProvider*** (fornita) dichiara i metodi per comunicare con qualunque fornitore di statistiche. Il metodo principale è *getStatistics* che, dato un ***Train***, restituisce il valore della corrispondente misura sotto forma di intero long. Il metodo *getStatsMessage* è analogo ma restituisce il risultato all'intero di un'apposita stringa descrittiva. Gli altri tre metodi servono a gestire l'eventuale soglia, per quei fornitori che supportano tale concetto: in particolare, il metodo *hasThreshold* restituisce true se quel certo provider supporta il concetto di soglia, false altrimenti. Nel caso la supporti, la coppia di metodi *setThreshold / getThreshold* permette di impostare/recuperare la soglia in minuti; ove invece quel certo provider non supporti tale concetto, i due metodi dovranno lanciare ***UnsupportedOperationException***.

d) La classe ***FinalDestinationStatProvider*** (fornita) implementa ***StatProvider*** nel caso del criterio 1) del Dominio del Problema, ossia quello in cui il ritardo da considerare sia quello di arrivo alla stazione finale.

e) Le due classi ***AverageStatProvider*** e ***ThresholdStatProvider*** **(da completare)** implementano ***StatProvider*** nei casi dei criteri 2) e 3) del Dominio del Problema, ovvero quelli in cui, rispettivamente, si debba considerare la media dei ritardi di arrivo o, al contrario, la percentuale di arrivo "puntuali" (ossia entro soglia).
Per entrambe, il metodo da implementare è solo *getStatistics*.

Parte 2 – Persistenza (punti: 8)

Package: trainstats.persistence **[TEMPO STIMATO: 35-45 minuti]**

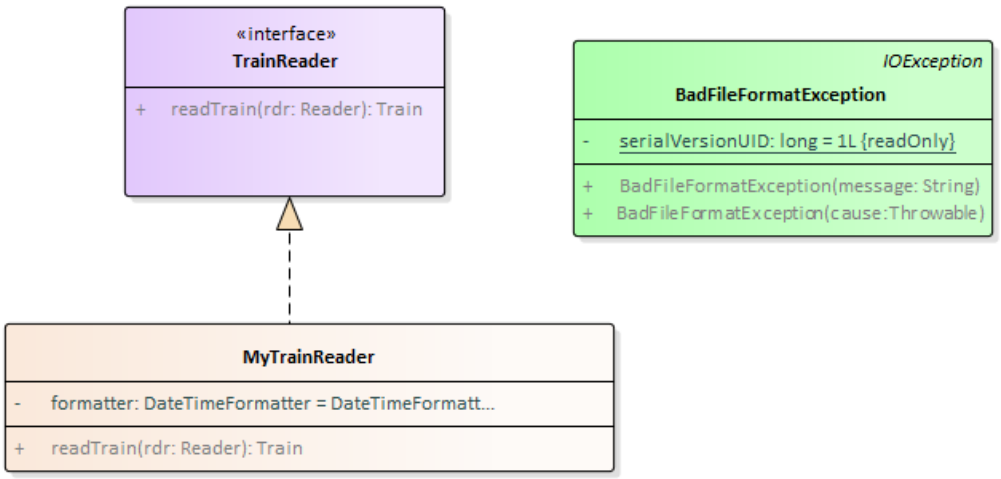
Una serie di file di testo contiene in ciascun file i dati di monitoraggio di un singolo treno.

NB: il candidato non deve preoccuparsi del fatto che vi siano più file da leggere, in quanto tale aspetto è già gestito nel main (fornito): la classe da sviluppare deve leggere, come sempre, un singolo file.

Ogni riga descrive la fermata del treno in una stazione, specificando nell'ordine il *nome della stazione*, *l'orario di arrivo previsto, l'orario di arrivo effettivo, l'orario di partenza previsto* e *l'orario di partenza effettivo*. Questi valori sono separati uno dall'altro da uno o più caratteri "punto e virgola" (';'): non è dato sapere se vi siano spazi intorno, prima, davanti ai vari campi. Tutti gli orari sono in formati italiano SHORT.
Nel caso della stazione iniziale, naturalmente, gli orari di arrivo sono indefiniti; lo stesso accade nell'ultima stazione per gli orari di partenza. Al loro posto, per convenzione, è indicata una sequenza di almeno due trattini.

ESEMPIO

```
MILANO CENTRALE;--   ;--   ;13:20;13:54
MILANO LAMBRATE;13:26;13:59;13:27;14:01
MILANO ROGOREDO;13:31;14:05;13:32;14:07
...
ANZOLA  ;15:56;16:13;15:57;16:14
BOLOGNA C.LE;16:10;16:24;--   ;--
```



SEMANTICA:

a) L'interfaccia ***TrainReader*** (fornita) dichiara il metodo *readTrain* che carica da un apposito Reader (già aperto) i dati necessari, restituendo un ***Train*** perfettamente configurato. Il metodo lancia:

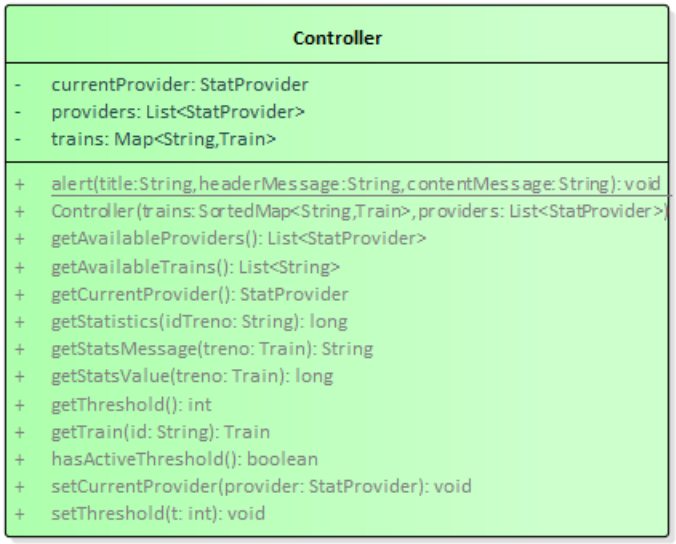
- ***IllegalArgumentException*** con opportuno messaggio d'errore in caso di argomento (reader) nullo;
- ***BadFileFormatException*** con messaggio d'errore appropriato in caso di problemi nel formato del file (mancanza/eccesso di elementi, errori nel formato degli orari, etc.).
  NB: a questo riguardo, si tenga conto che il costruttore di **Recording** lancia **IllegalArgumentException** in una serie di situazioni relative ad argomenti "illogici" (v. dettagli alle pagine precedenti): tale eccezione dovrà pertanto essere sostituita da un'opportuna ***BadFileFormatException.***
- una ***IOException*** in caso di altri problemi di I/O.

b) La classe ***MyTrainReader*** (**da realizzare**) implementa ***TrainReader*** secondo le specifiche sopra descritte.

Parte 3 (punti: 9)

Package: trainstats.controller *(punti 0)*

Il Controller è organizzato secondo il diagramma UML seguente:



SEMANTICA:

La classe ***Controller*** (fornita) riceve in fase di costruzione la mappa dei treni disponibili (la cui chiave è il codice univoco del treno, ottenuto a sua volta dal nome del file) e la mappa dei fornitori di statistica disponibili (la chiave è il loro nome univoco). Al suo interno mantiene l'informazione su quale **StatProvider** sia correntemente selezionato.

È ovviamente fornita un'ampia serie di metodi per recuperare tutte le informazioni utili, nonché per impostare lo **StatProvider** corrente e recuperarlo.

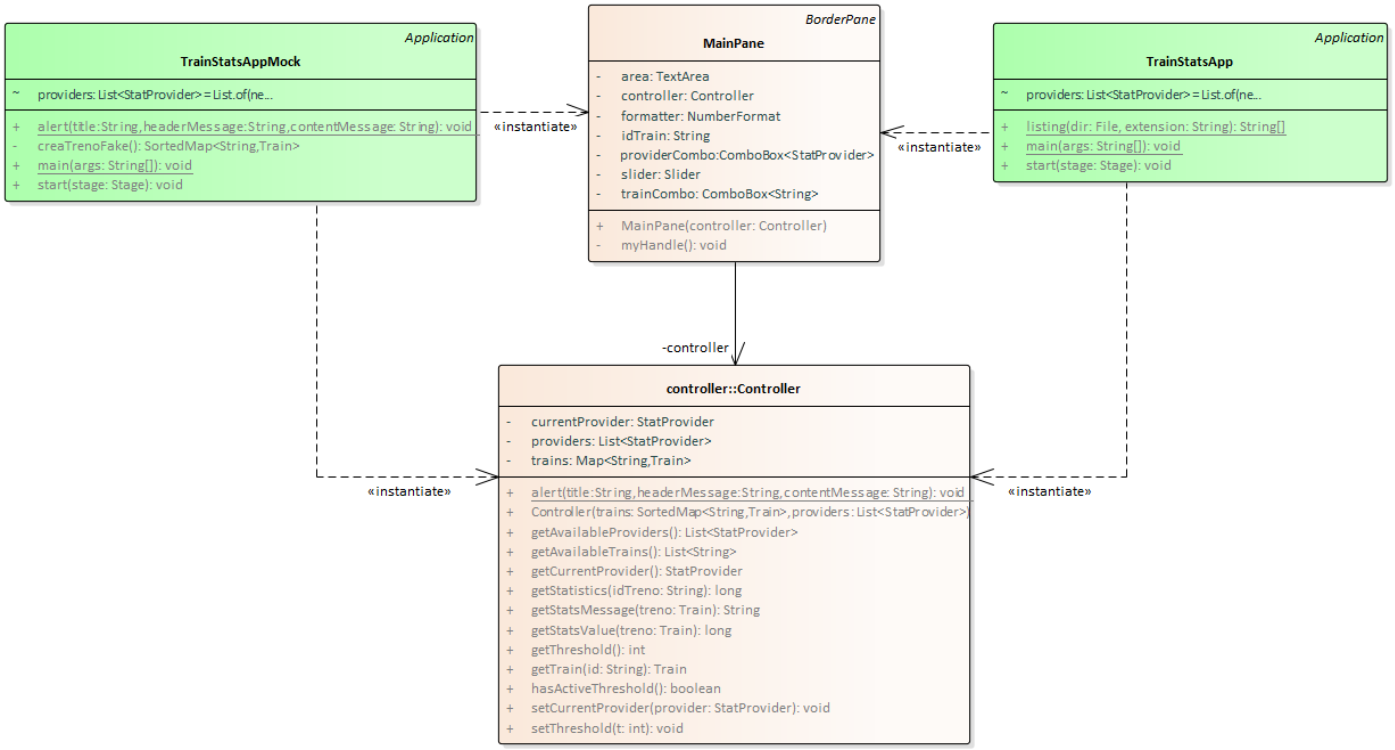
La terna di metodi *hasActiveThreshold* / *setThreshold* / *getThreshold* consente, rispettivamente, di sapere se lo **StatProvider** corrente abbia il concetto di soglia e, nel caso, di impostarla/recuperarla.

Infine, il metodo statico *alert*, utilizzabile anche dal MainPane, consente di far comparire all'utente, ove occorra, una finestra di dialogo con opportuno messaggio d'errore.

Package: trainstats.ui **[TEMPO STIMATO: 40-55 minuti]** *(punti 9)*

La classe ***TrainStatsApp*** (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 ***TrainStatsAppMock***.

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



L'interfaccia grafica si presenta come segue:

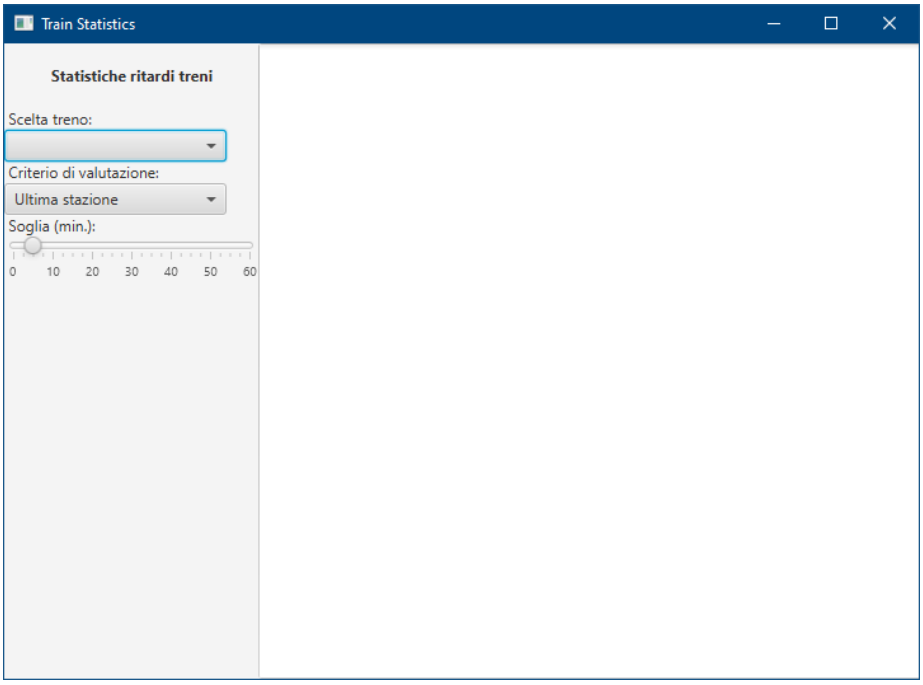
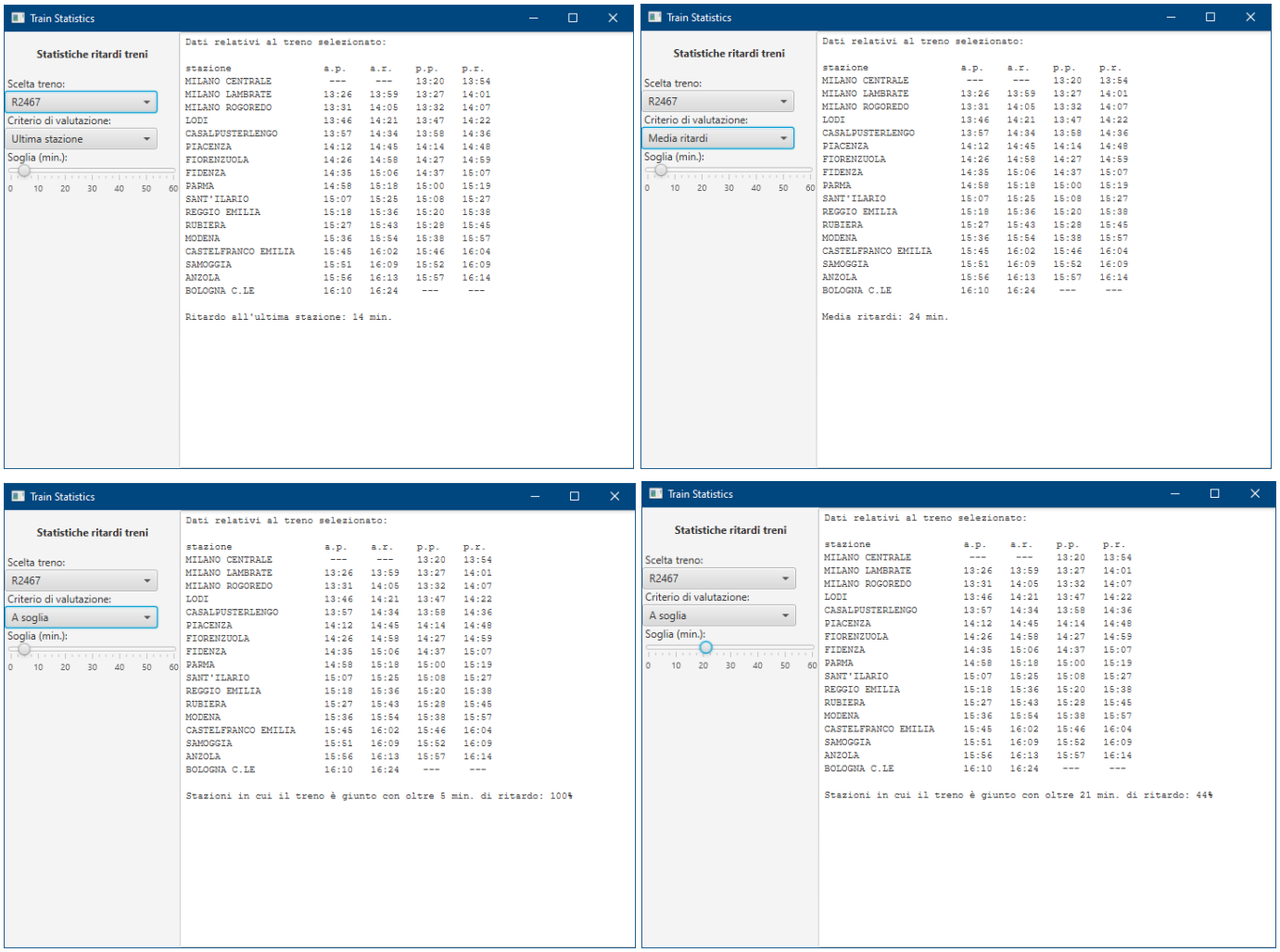


Fig. 1: la situazione iniziale della GUI, con combo treno vuota, criterio di valutazione di default e soglia 5 min.

- a sinistra c'è il pannello comandi, costituito da due combo (una con i codici dei treni, l'altra con i vari criteri di valutazione, ossia i vari ***StatProvider*** disponibili) e uno ***Slider*** (per impostare, se applicabile, la soglia, compresa fra 0 e 60 minuti);
- a destra, un'area di testo (inizialmente vuota e non scrivibile dall'utente) mostra i dettagli del treno scelto e i risultati della statistica calcolata dal provider prescelto.

Comportamento:

- la selezione di un treno dalla combo deve causare immediatamente il calcolo del ritardo col provider selezionato (inizialmente quello di default, ossia il criterio del ritardo valutato sull'ultima stazione) (Fig. 2)
- la selezione di un diverso provider dalla rispettiva combo deve causare immediatamente il ri-calcolo del ritardo secondo il nuovo criterio scelto (Fig. 3)
- la modifica della soglia dallo slider deve causare anch'essa il ri-calcolo del ritardo secondo il criterio attualmente selezionato: ovviamente, se il provider corrente non supporta il concetto di soglia, il risultato non cambierà, mentre se lo supporta si vedranno valori percentuali via via decrescenti all'aumentare della soglia (Figg. 4,5).



Figg. 2 / 3 / 4 / 5.

Il MainPane è fornito *parzialmente realizzato*: è presente quasi tutta l'impostazione strutturale, mentre sono da completare la configurazione di alcuni componenti e la gestione degli eventi.

In particolare, ***MainPane*** estende ***BorderPane*** e prevede:

1) a sinistra, una ***VBox*** per le varie combo, slider ed etichette ausiliarie
2) a destra, una ***VBox*** con la sola area di output.

La **parte da completare** riguarda:

1) la configurazione iniziale delle due combo e dello slider
2) l'aggancio dell'ascoltatore degli eventi (unico per i tre componenti), rappresentato dal metodo *myHandle*
3) la logica di gestione dell'evento, incapsulata nel metodo privato *myHandle*

In particolare, la gestione dell'evento principale, tramite *myHandle*, deve:

- recuperare dalla combo treni il nome del treno scelto e, sulla base di quello, recuperare l'oggetto ***Train*** corrispondente per le successive operazioni
- impostare come provider corrente nel ***Controller*** lo ***StatProvider*** selezionato nell'apposita combo
- recuperare dallo ***Slider*** il valore della soglia desiderata e impostarla nel ***Controller***
- emettere nell'area di testo i dati del treno e, subito sotto, l'esito del calcolo della corrispondente statistica (metodo *getStatsMessage* del ***Controller***).

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?