ESAME DI FONDAMENTI DI INFORMATICA T-2 del 25/7/2024
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*

Compiti *non compilabili*, o che *non passino almeno 2/3 dei test* o siano *lontani da 18/30* NON SARANNO CORRETTI e causeranno la verbalizzazione del giudizio "RESPINTO".

La Repubblica di Dentinia ha deciso di adottare, per le proprie elezioni, il sistema elettorale australiano, basato sull'approccio di *voto singolo trasferibile.* Si chiede di sviluppare un'applicazione che consenta, in ogni collegio uninominale, di determinare il vincitore, applicando l'algoritmo di scrutinio sotto specificato.

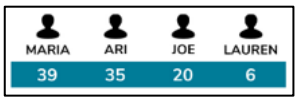
DESCRIZIONE DEL DOMINIO DEL PROBLEMA

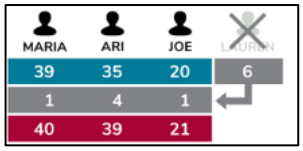
Nel sistema elettorale australiano per la House of Representatives il territorio è diviso in tanti collegi uninominali: ogni collegio elegge un solo rappresentante. A differenza però di altri sistemi, in questo caso si chiede all'elettore di *numerare in ordine di preferenza tutti i candidati,* da 1 (il più gradito) a N (il meno gradito): a lato un esempio di scheda.

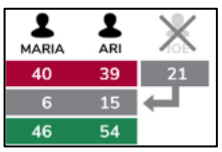
Per essere eletto, un candidato deve avere la *maggioranza assoluta* (50%+1) dei voti: la particolarità di questo sistema è che, *laddove nessuno abbia tale maggioranza in prima battuta,* anziché prevedere più turni o ballottaggi, *si conteggiano le seconde, terze, quarte,… preferenze,* agendo per fasi successive.

Ciò richiede uno specifico **algoritmo di scrutinio** a più fasi:

1. In prima battuta, si considerano solo le prime preferenze: se un candidato raggiunge la maggioranza assoluta viene eletto, altrimenti si passa alla fase successiva.
2. Se nessuno ha la maggioranza assoluta, si provvede a eliminare dalla competizione il candidato meno votato e trasferire le sue schede agli altri, guardando le seconde preferenze. Se, così facendo, qualcuno raggiunge la maggioranza assoluta, egli/ella viene eletto/a; altrimenti, si itera il procedimento, eliminando via via sempre il candidato meno votato e trasferendo agli altri le sue schede, guardando le seconde, terze, quarte preferenze, etc. Salvo il caso di parità, con N candidati il massimo numero di iterazioni è N-2: a quel punto rimangono sicuramente in gioco solo due candidati e quindi uno di essi raggiunge per forza la maggioranza assoluta, e viene eletto.
3. In caso di parità fra due o più candidati, la scelta su quale candidato eliminare viene effettuata secondo regole varie: qui si supporrà che avvenga a sorteggio (quindi, si potrà eliminare uno qualsiasi dei candidati a pari merito).







ESEMPIO: si supponga che vi siano 4 candidati (Maria, Ari, Joe, Lauren) e 100 elettori, e che il conteggio delle prime preferenze sia quello illustrato a lato. Poiché nessuno raggiunge la maggioranza assoluta di 51 voti, si procede eliminando la candidata meno votata (Lauren) e attribuendo le sue 6 schede agli altri, in base alle seconde preferenze indicate su tali schede. Se, supponiamo, 4 di queste sono per Ari e 1 a testa per gli altri due, la situazione è quella illustrata a fianco. Poiché nessuno raggiunge ancora la maggioranza assoluta, occorre iterare il procedimento: si elimina il meno votato dei tre candidati rimasti (Joe) trasferendo i suoi 21 voti agli altri in base alle successive preferenze indicate in quelle 21 schede. Se 6 di queste sono per Maria e 15 per Ari, il risultato è quello a lato: stavolta un candidato (Ari) ha la maggioranza assoluta ed è eletto.

Da notare che, dei 21 voti di Joe, 20 erano suoi fin dall'inizio (prima preferenza), mentre 1 era un voto originariamente per Lauren, trasferito a Joe (seconda preferenza) quando Lauren è stata eliminata. Pertanto, eliminando Joe, quella scheda va ora assegnata al terzo candidato in ordine di preferenza; e così via. In generale, quando si elimina un candidato e si trasferiscono le sue schede ad altri, *ogni scheda va assegnata al candidato successivo, in ordine di preferenza, a quello a cui la scheda è attualmente assegnata*.

Una serie di file di testo, nel formato descritto più oltre, contiene le schede votate dagli elettori in vari collegi. L'applicazione lavorerà volta per volta su *uno solo* di essi, che verrà scelto dall'utente tramite l'interfaccia grafica.

Scopo dell'applicazione è consentire all'utente di a) scegliere il file e caricare le relative schede, b) effettuare i calcoli per determinare il vincitore e c) poter vedere, a richiesta, il dettaglio (log) delle operazioni di scrutinio, così da poter capire come si sia giunti alla determinazione del risultato. Le immagini in calce illustrano il look & feel della GUI.

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

PARTE 1 – Modello dei dati:	Punti 16	[TEMPO STIMATO: 100-120 minuti]
PARTE 2 – Persistenza:	Punti 10	[TEMPO STIMATO: 25-40 minuti]
PARTE 3 – Grafica:	Punti 4	[TEMPO STIMATO: 10-20 minuti]

NUMERO MINIMO DI TEST CON SUCCESSO PERCHÉ IL COMPITO SIA CORRETTO

Considerando solo Scrutinio e MySchedeReader	11 su 16
Considerandoli tutti:	22 su 32

JAVAFX – Parametri run configuration nei LAB

```
--module-path "C:\applicativi\moduli\javafx-sdk-21.0.2\lib"
--add-modules javafx.controls
```

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?

Parte 1 – Modello dei dati (punti: 16)

Package: australia.elections.model **[TEMPO STIMATO: 100-120 minuti]**

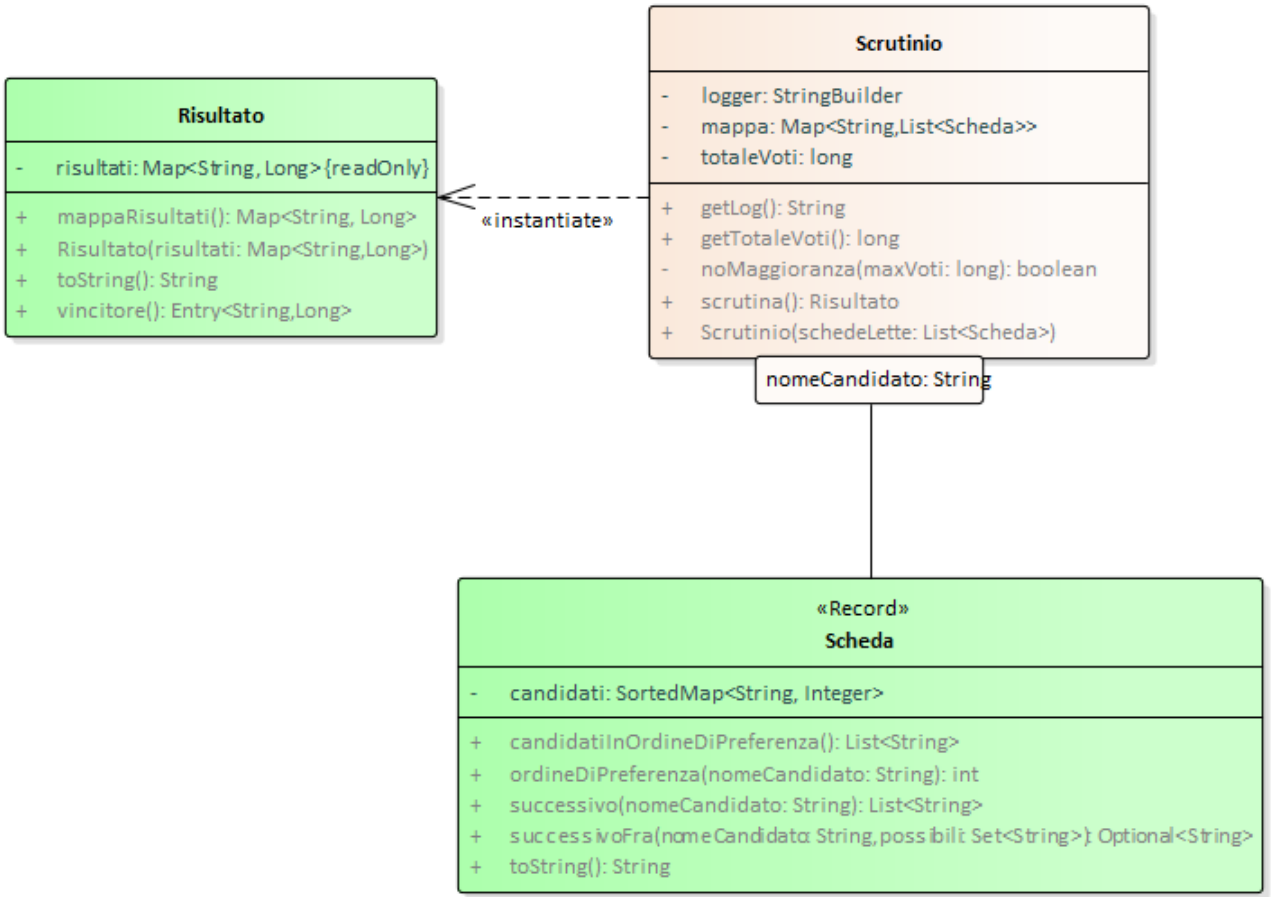
Conviene organizzare il modello dei dati su una struttura a mappa, che associ ogni candidato alla lista delle sue schede: la figura illustra come sarebbe la situazione iniziale nel caso dell'esempio sopra descritto.

Candidato	Lista schede
Maria →	*Lista schede di Maria (39)*
Ari →	*Lista schede di Ari (35)*
Joe →	*Lista schede di Joe (20)*
Lauren →	*Lista schede di Lauren (6)*

Ovviamente, la lunghezza di ogni lista indica il numero di voti attribuiti in un dato momento a ogni candidato.

Quando si procede a eliminare un candidato le sue schede devono quindi essere spostate nelle altre liste, in base alla preferenza successiva indicata su ogni scheda.

Le classi che esprimono ciò sono le seguenti:



- Il record ***Scheda*** (fornito) rappresenta la scheda elettorale, espressa da una mappa <stringa,intero> che associa a ogni candidato il corrispondente numero di preferenza (a lato un esempio di scheda per Maria).

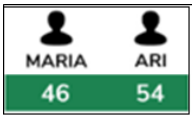
Candidato	Ordine di preferenza
Maria →	1
Ari →	4
Joe →	2
Lauren →	3

  - Il costruttore verifica che la mappa ricevuta non sia null né vuota (lanciando rispettivamente ***NullPointerException*** o ***IllegalArgumentException*** in caso contrario) e che i numeri specificati siano tutti distinti e compresi fra 1 e il numero di candidati (anche in questo caso, lanciando ***IllegalArgumentException*** in caso di violazione)
  - il metodo *candidatiInOrdineDiPreferenza* restituisce la lista dei candidati in

ordine di preferenza, dal più gradito (n°1) al meno gradito (n° N)

- il metodo *ordineDiPreferenza(String nomeCandidato)* restituisce la posizione di quel candidato nell'ordine di preferenza, sotto forma di intero compreso fra 1 e N: lancia ***NullPointerException*** se il nome del candidato è nullo, o ***IllegalArgumentException*** se esso è vuoto o un tale candidato non esiste nella scheda
- il metodo *successivo(String nomeCandidato)* restituisce il nome del candidato successivo a quello dato, secondo l'ordine di preferenza: se il candidato era ultimo e non ha successivi, viene restituito ***Optional.empty*** mentre se il nome del candidato è nullo o vuoto o inesistente vengono lanciate le stesse eccezioni del caso precedente
- il metodo *successivoFra(String nomeCandidato, Set<String> possibili)* restituisce il nome del candidato successivo a quello dato *considerando come possibili solo i candidati citati nel secondo argomento* (ciò è utile nella 2° fase dell'algoritmo, per saltare i candidati già eliminati)
- un'apposita *toString* emette una rappresentazione stampabile della scheda.

- La classe ***Risultato*** (fornita) rappresenta il risultato dello scrutinio, espresso come mappa <stringa,long> che associa a ogni candidato i voti ottenuti nell'ultimo turno di esecuzione dell'algoritmo.



  - Il costruttore verifica che la mappa ricevuta non sia null né vuota (lanciando ***NullPointerException*** o ***IllegalArgumentException*** in caso contrario)
  - il metodo *vincitore* restituisce il nome del vincitore
  - un'apposita *toString* emette una rappresentazione stampabile del risultato

- La classe ***Scrutinio*** (da completare nella parte algoritmica) incorpora la logica di scrutinio: la gestione del log mantiene traccia dei principali passaggi svolti.
  - il costruttore riceve la lista di schede su cui operare: verifica, al solito, che tale lista non sia nulla né vuota, lanciando nel caso le opportune eccezioni, e costruisce la tabella (mappa) iniziale contenente, per ogni candidato, la lista delle *schede in cui il candidato è la prima preferenza* (fase 1 dell'algoritmo); inizializza inoltre i valori del totale voti e lo status del logger interno (uno ***StringBuilder***)
  - gli accessor *getTotaleVoti* e *getLog* restituiscono rispettivamente il totale dei voti espressi (su cui si basa a sua volta il metodo privato ***noMaggioranza*** per verificare il raggiungimento o meno della maggioranza assoluta) e la stringa di log che elenca le operazioni svolte
  - il metodo ***scrutina*** *(da realizzare)* incorpora la logica di scrutinio descritta nel Dominio del Problema. Pertanto, esso deve agire iterativamente come segue:
    - in primis estrarre il numero di voti del candidato più votato (*) e verificare se abbia o meno la maggioranza assoluta: se ce l'ha, il procedimento termina
    - se non ce l'ha, identificare il candidato meno votato (*), recuperare le sue schede, eliminarlo e ri-assegnare tali schede, una ad una, a un altro candidato, seguendo l'ordine di preferenza indicato su ogni singola scheda
      SUGGERIMENTO: a tal fine, conviene invocare su ogni scheda l'apposito metodo *successivoFra*, a cui passare come nomi possibili i soli candidati ancora in lizza: il risultato sarà il nome del candidato a cui assegnare tale scheda.
    - (*) in caso di parità la scelta può essere effettuata secondo un criterio qualunque.

    Al termine del ciclo, il metodo sintetizza e restituisce il ***Risultato*** dello scrutinio.

    Durante le varie operazioni il metodo deve trascrivere sul log i momenti salienti, ovvero:
    - a ogni iterazione, lo stato della mappa voti e del massimo dei voti
    - ogni rimozione di candidato

- al termine, la mappa voti finale col risultato, che mostra il vincitore

Ciò consentirà alla fine di mostrare sulla GUI come si è giunti alla determinazione del risultato (vedere figure in calce).

Parte 2 – Persistenza (punti: 10)

Package: australia.elections.persistence **[TEMPO STIMATO: 25-40 minuti]**

Vari file di testo elencano le schede votate in diversi collegi: il reader dovrà però preoccuparsi di leggere un singolo file, che sarà scelto dall'utente tramite la GUI.

Ogni file elenca, una per riga, le schede votate in quel collegio, nel seguente formato:

- tutte le righe hanno *eguale struttura* e devono contenere lo *stesso numero di elementi* (separati da virgole) e gli *stessi nomi* di candidati
- tutte le righe alternano un nome (che non può essere costituito di soli caratteri numerici) e un numero (che è costituito ovviamente di soli caratteri numerici)

```
Maria, 1, Ari, 3, Joe, 2, Lauren, 4, Juliet, 5, Fred, 6
Maria, 1, Ari, 3, Joe, 2, Lauren, 4, Juliet, 5, Fred, 6
Maria, 1, Ari, 2, Joe, 3, Lauren, 4, Juliet, 6, Fred, 5
Maria, 1, Ari, 6, Joe, 3, Lauren, 4, Juliet, 2, Fred, 5
Maria, 1, Ari, 2, Joe, 4, Lauren, 6, Juliet, 3, Fred, 5
Maria, 1, Ari, 6, Joe, 2, Lauren, 3, Juliet, 4, Fred, 5
```

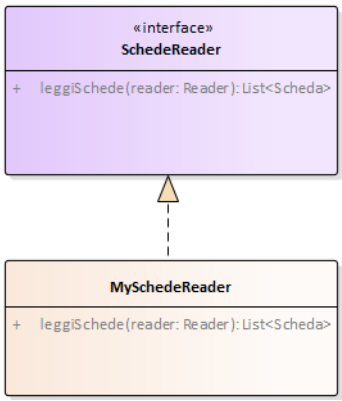
SEMANTICA:

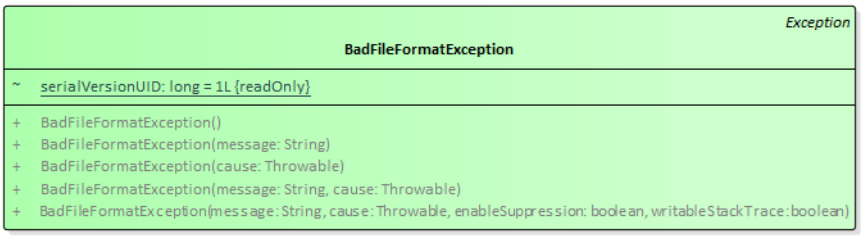
a) L'interfaccia ***SchedeReader*** (fornita) dichiara:

- il metodo *leggiSchede,* che legge dal ***Reader*** (già aperto) fornito i dati necessari e restitisce una lista di ***Scheda*** perfettamente configurato; esso lancia ***BadFileFormatException*** con opportuno messaggio d'errore in caso di problemi nel formato del file o ***IOException*** in caso di altri problemi di I/O

b) La classe ***MySchedeReader*** (da completare) implementa ***SchedeReader*** secondo le specifiche sopra descritte.

- non c'è alcun costruttore
- Il metodo *leggiSchede* legge le righe, memorizzando numero di elementi e nomi della prima riga per usarli poi come confronto per le righe successive, e le elabora estraendo i dati per costruire le singole schede, popolando quindi la lista da restituire.

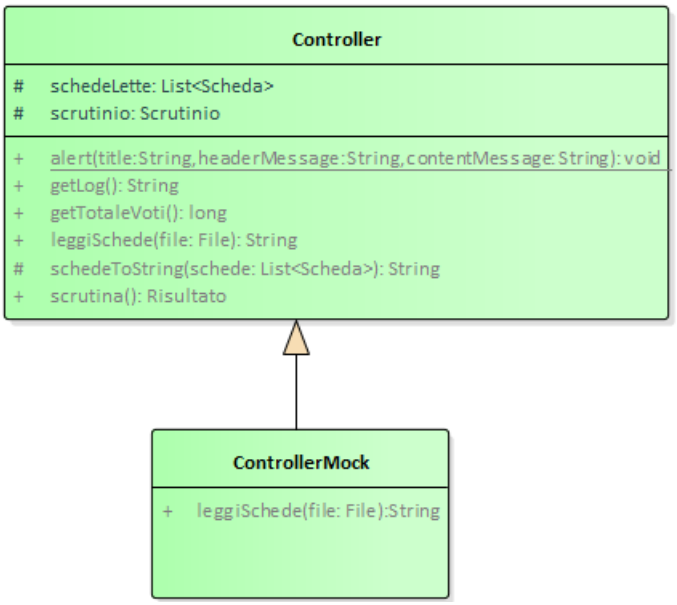




Parte 3 (punti: 4)

Package: australia.elections.controller *(punti 0)*

Il Controller funge semplicemente da front-end verso l'analizzatore, ed è quindi è organizzato come segue:



SEMANTICA:

La classe ***Controller*** (fornita) non ha costruttore e definisce i seguenti metodi:

- *leggiSchede* legge, tramite il reader, il ***File*** fornito e memorizza la lista di schede così ottenuta nel proprio stato interno; quindi, istanzia e configura lo ***Scrutinio***, che effettuerà poi i calcoli. Restituisce intanto le schede lette, sotto forma di stringa pronta per la visualizzazione o stampa.
- *scrutina* attiva lo scrutinio e ne restituisce il ***Risultato***
- *getTotaleVoti* e *getLog* restituiscono le stesse informazioni restituite da ***Scrutinio***.

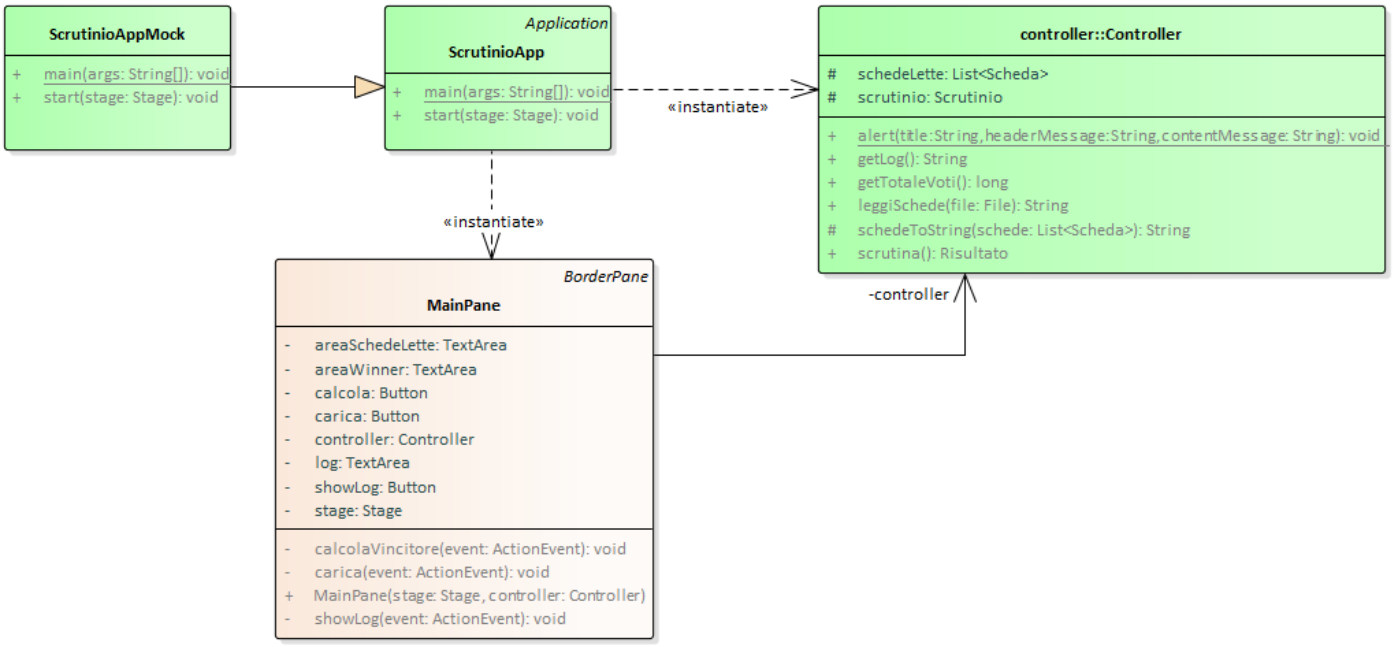
Il metodo statico *alert*, utilizzabile dal ***MainPane*** quando è attiva la grafica, consente di far comparire all'utente una finestra di dialogo che mostri il messaggio d'errore specificato.

La sottoclasse ***ControllerMock*** (fornita) è utilizzata esclusivamente da ***ScrutinioAppMock*** (fornita, vedere sotto) per simulare la presenza dei dati letti da file: pertanto, non deve essere considerata in questo compito.

Package: australian.elections.ui **[TEMPO STIMATO: 10-20 minuti]** *(punti 4)*

La classe ***ScrutinioApp*** (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 ***ScrutinioAppMock*** (che si avvale del controller ausiliario ***ControllerMock***).

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



- a sinistra vi sono i pulsanti “Carica file” e “calcola”, ognuno seguito dalla relativa mini-area di testo, rispettivamente per la visualizzazione delle schede caricate e per il risultato del calcolo (ossia, il vincitore);
- a destra vi è invece il solo pulsante “Show log”, seguito dall’area di testo di maggiore dimensione destinata a mostrare i dettagli (log) delle operazioni di scrutinio.

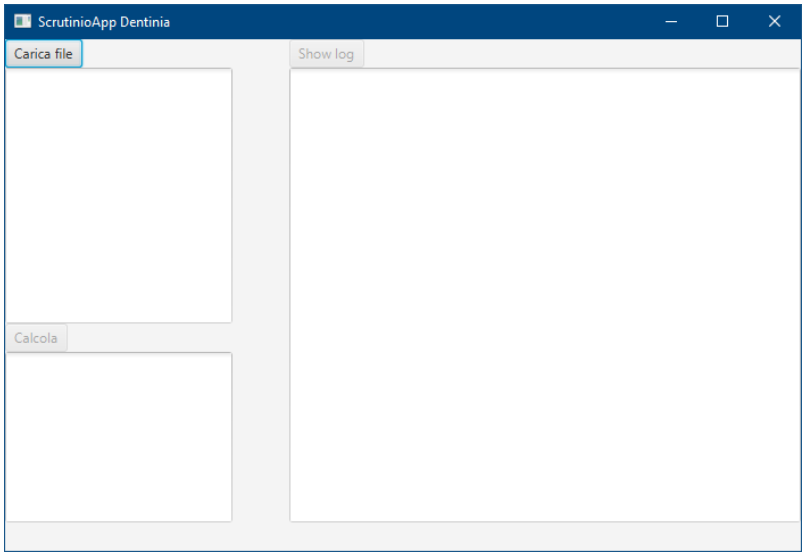


Fig. 1: vista generale: a sinistra i controlli per il caricamento del file e la visualizzazione delle schede (in alto), per il calcolo e la visualizzazione del risultato (in basso); a destra, i controlli per la visualizzazione dei dettagli (log) delle operazioni di scrutinio. Da notare che inizialmente il solo pulsante abilitato è quello relativo al caricamento del file.

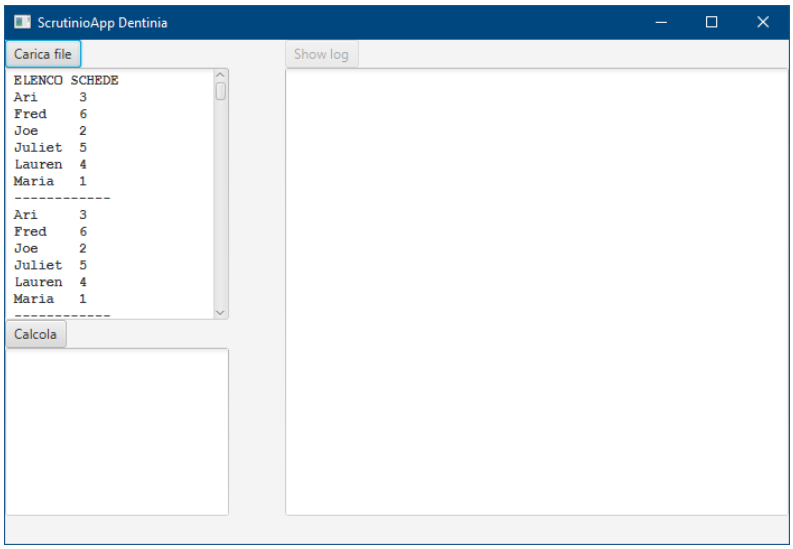


Fig. 2: la GUI dopo aver premuto il pulsante “Carica file”: notare che il pulsnte “Calcola” è ora abilitato.

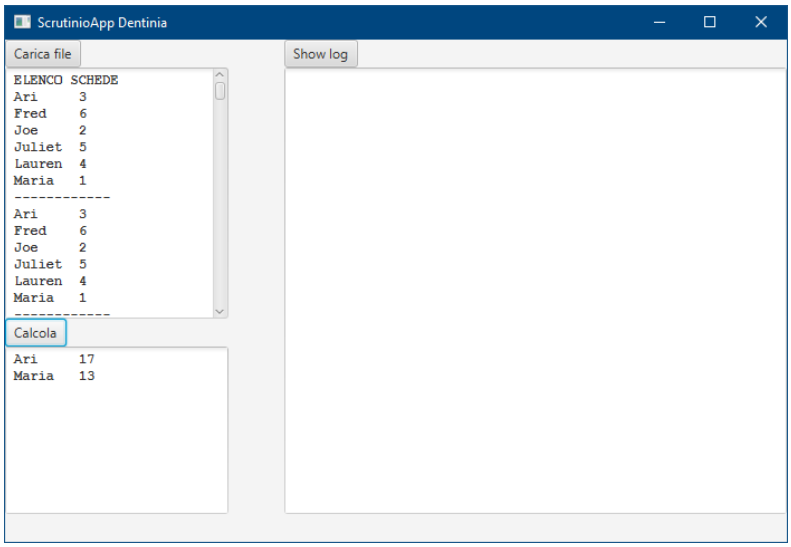


Fig. 3: la GUI dopo aver premuto il pulsante “Calcola” : notare che il pulsnte “Show log” è ora abilitato

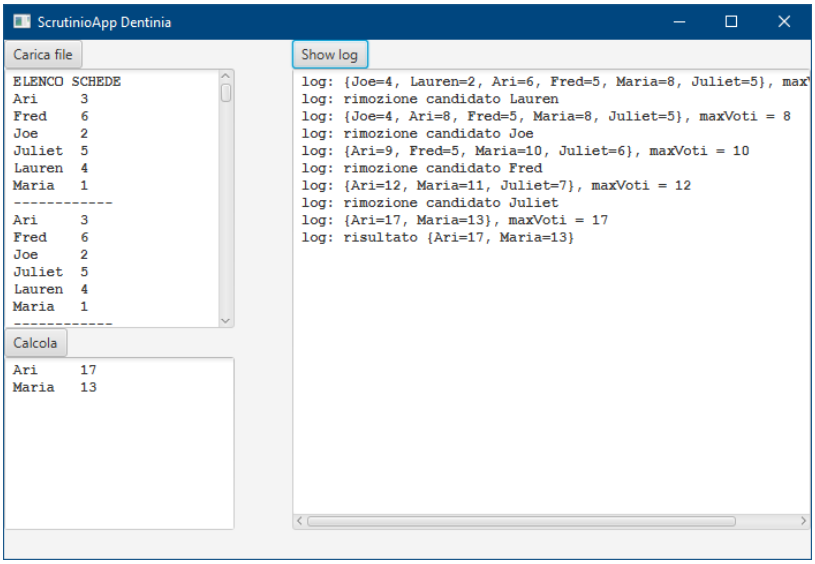


Fig. 4: la GUI dopo aver premuto il pulsante “Show log”. Per analizzare un altro file si può ripartire da capo riprendendo “Carica file”.

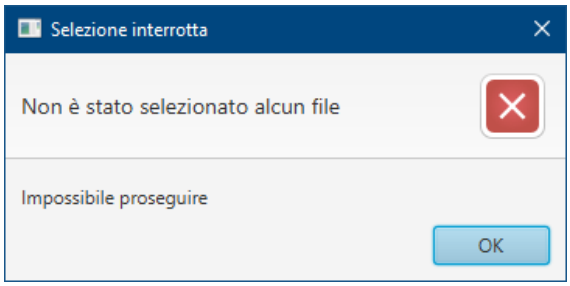


Fig. 5: messaggio d'errore nel caso si prema "Annulla" nella finestra (chooser) di scelta file, che compare premendo il pulsante "Carica file".

Il MainPane è fornito *parzialmente realizzato*: è presente tutta la parte strutturale, mentre rimangono da realizzare due gestori degli eventi (metodi privati *calcolaVincitore, showLog*)

In particolare:

- *calcolaVincitore* deve chiedere al ***Controller*** di effettuare lo scrutinio, abilitare il pulsante "Show log" e mostrare il vincitore nell'apposita area di testo in basso a sinistra.
- *showLog* deve chiedere al ***Controller*** i dettagli del log e mostrarli nell'area di testo a destra.

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?