ESAME DI FONDAMENTI DI INFORMATICA T-2 del 06/02/2020
Proff. E. Denti – R. Calegari – A. Molesini
Tempo a disposizione: 4 ore MAX

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)

NB: l'archivio ZIP da consegnare deve contenere l'intero progetto Eclipse

Si ricorda che compiti *non compilabili* o *palesemente lontani da 18/30* NON SARANNO CORRETTI e causeranno la verbalizzazione del giudizio "RESPINTO"

Per le elezioni del Governatore di Dentinia la legge elettorale prevede l'uso di un sistema proporzionale, ma lascia la possibilità di introdurre uno *sbarramento* che limiti la frammentazione, escludendo le liste che non ottengono una certa *percentuale minima* di voti. Si vuole sviluppare un'applicazione che simuli l'attribuzione dei seggi al variare, appunto, del livello di sbarramento fra un minimo di 0 (assenza di qualunque sbarramento) e un massimo del 10%.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Nei sistemi proporzionali i seggi sono attribuiti in proporzione ai voti ottenuti da ciascuno. In particolare, nel sistema ***D'Hondt*** (il più diffuso) l'attribuzione dei seggi avviene *dividendo i voti ottenuti da ogni lista per i numeri naturali 1,2,3,4…* e inserendo poi i risultati di tutte le liste in un'unica graduatoria decrescente: i seggi si attribuiscono quindi seguendo tale graduatoria.

ESEMPIO

Si supponga di dover assegnare 20 seggi ai partiti A, B, C, D, E che abbiano ottenuto rispettivamente 9.100.000, 7.200.000, 4.880.000, 4.100.000, 1.320.000 voti. Si dividono quindi tali voti per i numeri naturali da 1 a 20 (caso peggiore in cui un singolo partito prenda tutti i seggi), ottenendo i seguenti valori:

A) 9.100.000, 4.550.000, 3.033.333, 2.275.000, 1.820.000, 1.516.666, 1.300.000, 1.137.500, 1.011.111, 910.000, …
B) 7.200.000, 3.600.000, 2.400.000, 1.800.000, 1.440.000, 1.200.000, 1.028.571, 900.000, 800.000, 720.000, …
C) 4.880.000, 2.440.000, 1.626.666, 1.220.000, 976.000, 813.333, 697.142, 610.000, 542.222, 488.000, …
D) 4.100.000, 2.050.000, 1.366.666, 1.025.000, 820.000, 683.333, 585.714, 512.500, 455.555, 410.000, …
E) 1.320.000, 1.160.000, 773.333, 580.000, 464.000, 386.666, 331.428, 290.000, 257.777, 232.000, …

Per assegnare i 20 seggi in palio occorre ora prendere i 20 valori migliori, ossia quelli evidenziati in giallo: pertanto, la lista A ottiene 7 seggi, la lista B 5 seggi, la lista C 4 seggi, la lista D 3 seggi e la lista e 1 seggio soltanto.

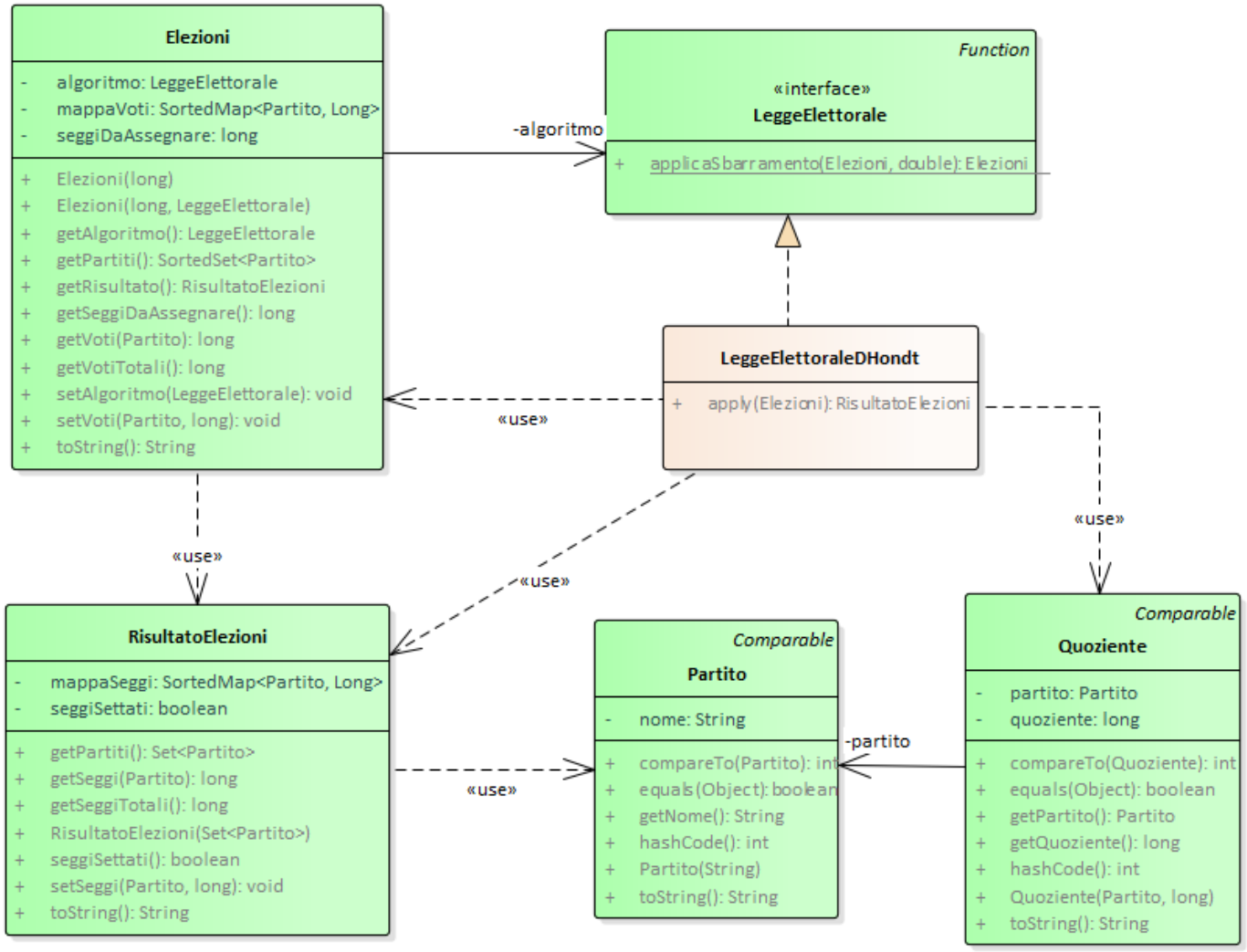
Per attenuare l'effetto di parcellizzazione che i sistemi proporzionali tendono a determinare, spesso si introduce uno **sbarramento,** che esclude dal riparto i partiti che non raggiungono una data percentuale di voti.

Nel caso ad esempio di uno **sbarramento al 5%**, le liste che non raggiungono la soglia di 1.330.000 voti (sopra, la lista E) non partecipano al riparto: pertanto, il suo seggio viene assegnato al successivo in graduatoria (la lista B, in azzurro).

Il file di testo Voti.txt contiene i risultati delle elezioni (nel formato dettagliato più oltre): la prima riga indica il numero di seggi da assegnare, la seconda i voti ottenuti da ciascun partito, separati da virgole.

Parte 1 (punti: 20)

Dati (package dentinia.governor.model) *(punti: 9)*



SEMANTICA:

a) la classe ***Partito*** (fornita) rappresenta un partito, caratterizzato dal suo nome;

b) la classe ***Elezioni*** (fornita) mantiene i dati relativi ai voti ottenuti da ogni partito: il costruttore riceve il numero di seggi da assegnare ed eventualmente l'algoritmo da usare per il calcolo del risultato (se non specificato, rimane null). Gli accessor *setVoti/getVoti* operano sui voti di un dato partito in modo controllato, mentre gli accessor *setAlgoritmo/getAlgoritmo* impostano/restituiscono l'algoritmo di calcolo da usare; ovviamente, i metodi *getPartiti, getVotiTotali* e *getSeggiDaAssegnare* restituiscono le proprietà omonime. Il metodo *toString* restituisce una stringa multi-linea con partiti, voti e seggi perfettamente formattata.
Il metodo *getRisultato* effettua il calcolo dei seggi con l'algoritmo precedentemente impostato: il risultato è un oggetto di tipo ***RisultatoElezioni***, che associa a ogni partito i seggi corrispondenti. Se l'algoritmo è null, restituisce un'istanza di ***RisultatoElezioni*** con tutti i seggi azzerati.

c) la classe ***RisultatoElezioni*** (fornita) mantiene i dati relativi ai seggi ottenuti da ciascun partito: il costruttore riceve un set di ***Partito*** e imposta i seggi tutti a zero. Gli accessor *setSeggi/getSeggi*, *getSeggiTotali* permettono di recuperare le corrispondenti proprietà. Diversamente dal caso precedente, il metodo *toString* restituisce una mera rappresentazione interna utile solo a fini di debugging. Infine, Il metodo *seggiSettati* restituisce un boolean che indica se i seggi sono stati impostati dopo la costruzione iniziale.

d) l'interfaccia ***LeggeElettorale*** (fornita) è un semplice alias per il tipo funzione da ***Elezioni*** a ***RisultatoElezioni***. Contiene però il metodo statico *applicaSbarramento*, che applica alle ***Elezioni*** date lo sbarramento specificato, restituendo una nuova istanza di ***Elezioni*** in cui i voti dei partiti sotto soglia sono azzerati.

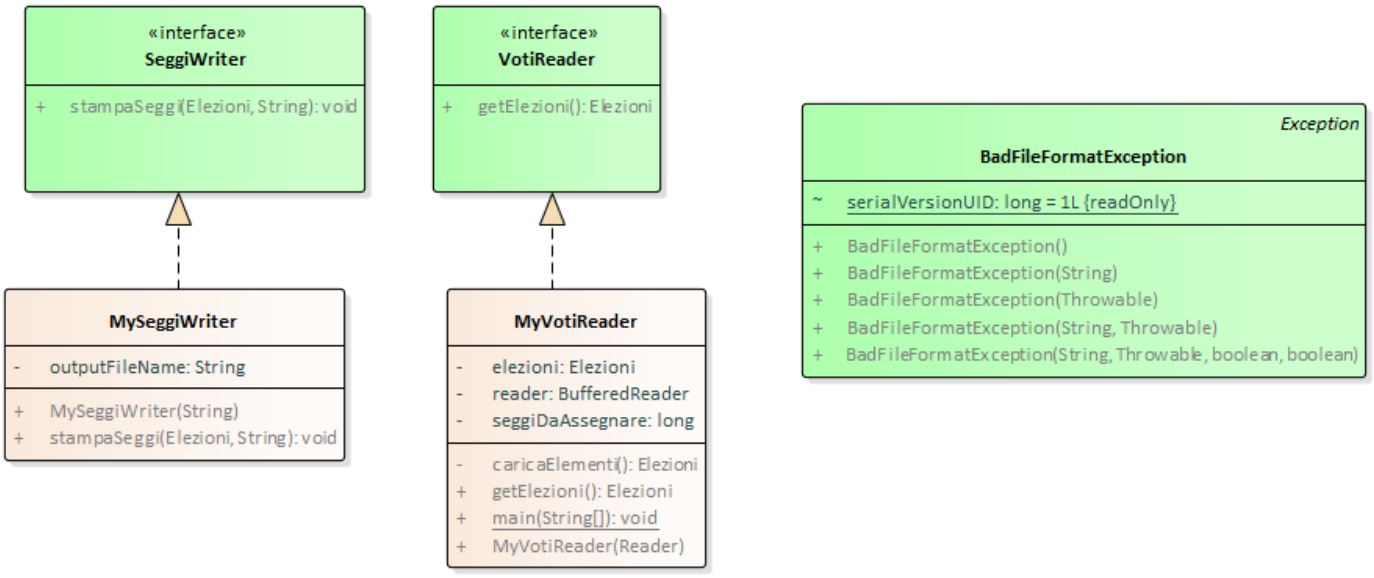
e) la classe ***Quoziente*** (fornita) rappresenta una coppia <***Partito***, valore>, utile per rappresentare i valori ottenuti nell'algoritmo D'Hondt: per tale motivo, è comparabile in senso decrescente sul valore (intero). Ovviamente, il costruttore riceve i due elementi della coppia, e due accessor consentono di recuperarli.

f) la classe ***LeggeElettoraleDHondt*** (**da realizzare**) implementa ***LeggeElettorale*** usando il metodo D'Hondt. Si prevede che utilizzi opportune istanze di ***Quoziente*** per rappresentare le divisioni successive dei voti di ogni partito per 1,2,3,4...

Persistenza (dentinia.governor.persistence) *(punti 11)*

Il file di testo Voti.txt contiene i risultati delle elezioni: la prima riga indica il numero di seggi totali da assegnare, nella forma "SEGGI" seguita da un intero, mentre la successiva riporta i voti ottenuti da ciascun partito, separati da virgole: ogni coppia partito/voti prevede prima il nome del partito (che può contenere spazi), poi i voti, formattati secondo l'uso italiano con il "." come separatore delle migliaia.

```
SEGGI 20
Lista A 9.100.000, Lista E 1.320.000, Lista C 4.880.000, Lista D 4.100.000, Lista B 1.500.000
```



SEMANTICA:

a) L'interfaccia ***VotiReader*** (fornita) dichiara il metodo *getElezioni*, che restituisce una istanza di ***Elezioni*** opportunamente popolata;

b) L'interfaccia ***SeggiWriter*** (fornita) dichiara il metodo *stampaSeggi* che stampa ***Elezioni*** utilizzando la *toString*, preceduta dalla data e ora corrente e dal messaggio ricevuto come argomento; la data e ora devono includere il nome del giorno (formato FULL, ma attenzione…. ☺)

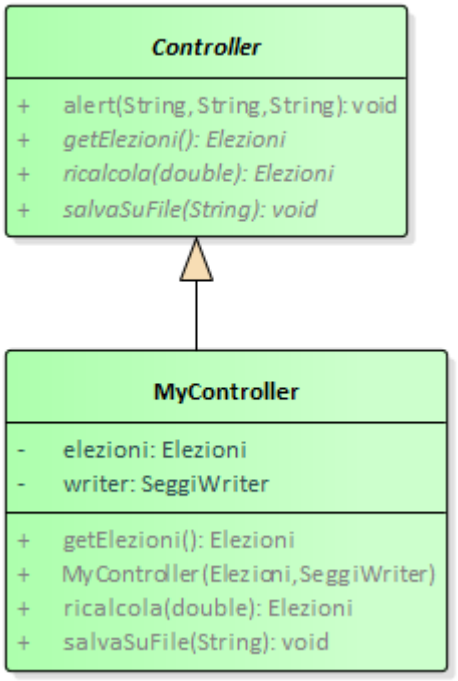
c) La classe ***MyVotiReader*** (**da realizzare**) implementa ***VotiReader***: il costruttore riceve in ingresso il **Reader** da cui leggere i dati, effettua la lettura e popola opportunamente l'entità ***Elezioni*** mantenuta al suo interno, che viene poi semplicemente restituita a ogni invocazione di *getElezioni*. In caso di problemi di I/O viene propagata l'opportuna ***IOException***, nel caso di Reader nullo deve invece essere lanciata una ***IllegalArgumentException***, infine entuali problemi nel formato del file devono essere incapsulati in un'opportuna ***BadFileFormatException***.

d) la classe ***MySeggiWriter* (da realizzare)** implementa ***SeggiWriter***: il costruttore riceve in ingresso il nome del file su cui scrivere (il main dell'applicazione usa "Report.txt"). Il metodo *stampaSeggi* deve produrre una tabella analoga a quella mostrata in calce, in cui la prima riga contiene data e ora (nel formato indicato), la seconda una frase con l'indicazione del livello di sbarramento applicato, mentre dalla terza riga in poi sono riportati, uno per riga, i partiti coi rispettivi voti e seggi, separati da tabulazioni e formattati secondo l'uso italiano per le migliaia (vedi esempio). In caso di problemi di I/O viene propagata l'opportuna ***IOException***.

```
giovedì 16 gennaio 2020, 20:44:20
Metodo D'Hondt con sbarramento del 0.0%
Lista A                          Voti:    9.100.000   Seggi:    9
Lista B                          Voti:    1.500.000   Seggi:    1
Lista C                          Voti:    4.880.000   Seggi:    5
Lista D                          Voti:    4.100.000   Seggi:    4
Lista E                          Voti:    1.320.000   Seggi:    1
TOTALE                           Voti:   20.900.000   Seggi:   20
```

Parte 2 (punti: 10)

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



SEMANTICA:

a) La classe astratta ***Controller*** (fornita) dichiara l'interfaccia del controller (metodi *ricalcola, getVoti* e *salvaSuFile*) e implementa il metodo statico ausiliario *alert*, utile per mostrare avvisi all'utente.

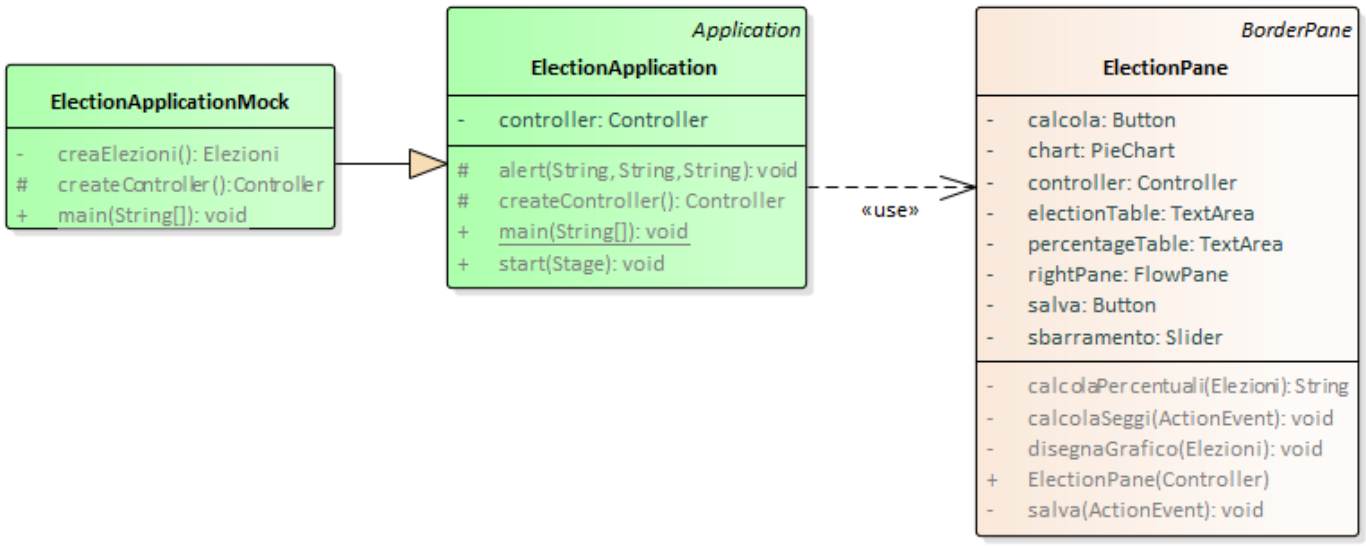
b) La classe ***MyController*** (**pure fornita**) completa l'implementazione realizzando:

- il costruttore a due argomenti (***Elezioni*** e ***SeggiWriter***) che memorizza gli argomenti in opportuni campi e imposta la legge elettorale D'Hondt come algoritmo da usare; tale istanza di ***Elezioni*** è quella che viene restituita dal metodo *getElezioni*;
- il metodo *ricalcola* riceve in ingresso lo sbarramento richiesto, lo applica (tramite ***LeggeElettorale***) e restituisce il risultato sotto forma di nuova istanza di ***Elezioni***;
- il metodo *salvaSuFile* opera solo se il controller dispone internamente di un oggetto ***Elezioni*** valido: in tal caso, semplicemente delega all'opportuno ***SeggiWriter*** la stampa effettiva, passandogli anche il messaggio personalizzato ricevuto come argomento.

Interfaccia utente (dentinia.governor.ui.javafx) *(punti 10)*

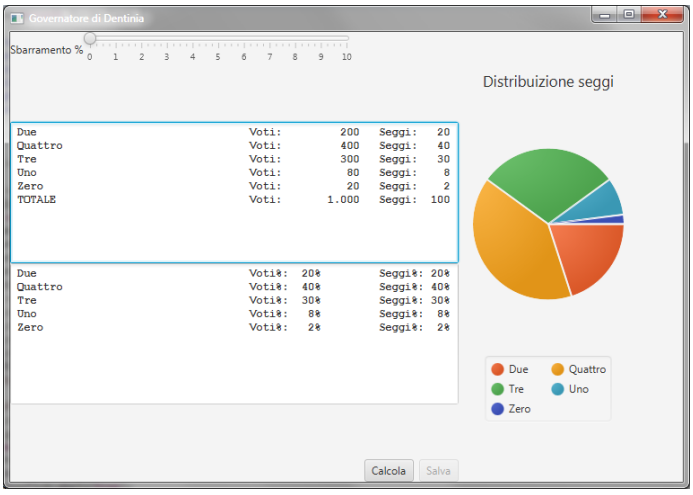
L'interfaccia utente deve essere simile (non necessariamente identica) all'esempio mostrato nelle figura seguenti.

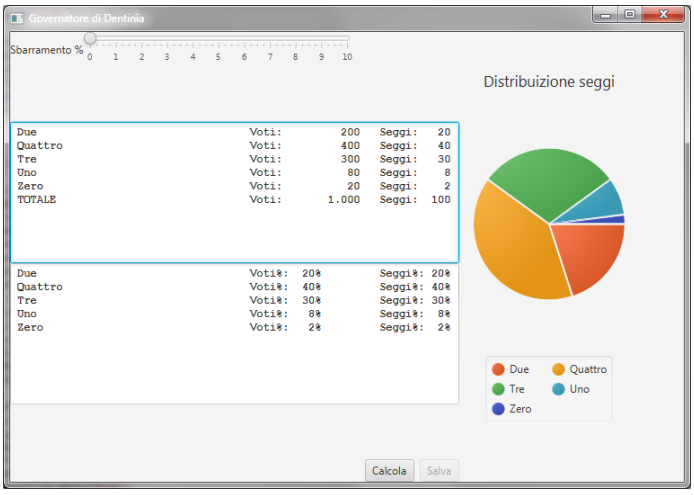
L'architettura segue il modello sotto illustrato:

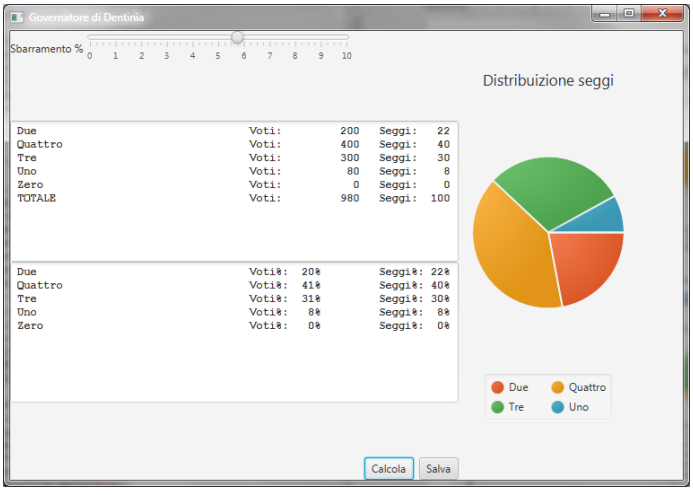


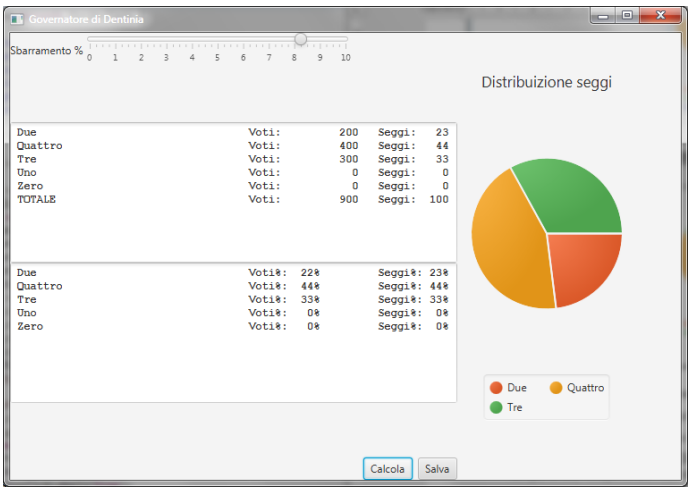
La classe ***ElectionApplication*** (fornita) costituisce l'applicazione JavaFX che si occupa di aprire il file, il controller e incorporare l'***ElectionPane*** (**da realizzare**). Per consentire di collaudare la GUI anche in assenza della parte di persistenza, è possibile avviare l'applicazione mediante la classe ***ElectionApplicationMock***.

L'interfaccia utente deve essere simile (non necessariamente identica) all'esempio mostrato nella figura seguente.









La classe ***ElectionPane*** (**da realizzare**), che estende ***BorderPane***, prevede:

1) in alto, uno ***Slider*** configurato da 0 a 10, passo 1, inizialmente a 0, per impostare lo sbarramento %
2) a sinistra, due ***TextArea*** una sotto l'altra: la prima mostra la situazione delle elezioni in termini di voti e seggi assegnati, mentre la seconda esprime le stesse informazioni in percentuale. All'inizio mostrano la situazione corrispondente all'assenza di sbarramento.
3) A destra, un ***PieChart*** (senza etichette) mostra anch'esso la situazione corrente, in forma grafica.
4) In basso, due pulsanti *Calcola* e *Salva,* di cui il secondo inizialmente disabilitato, controllano il funzionamento dell'applicazione. *Calcola* scatena il calcolo dei seggi, tenendo conto dello sbarramento prescelto e aggiornando tabelle e grafico a torta: dopo ogni ricalcolo ri-abilita il pulsante *Salva* per consentire il salvataggio su file dei nuovi risultati. Il pulsante *Salva* effettua il salvataggio dei risultati sul file, auto-disabilitandosi subito dopo.
   NB: mentre i dati della prima tabella sono facilmente ottenibili dal model, i secondi vanno calcolati!

Cose da ricordare

- salva costantemente il tuo lavoro, facendo ZIP parziali e consegne parziali (vale l'ultima)
- in particolare, se ora compila e stai per fare modifiche, salva la versione attuale (non si sa mai..)

Checklist di consegna

- Hai fatto un unico file ZIP **(non .7z!!!)** contenente l'intero progetto?
  In particolare, ti sei assicurato di aver incluso tutti i file .java (e non solo i .class)?
- Hai controllato che si compili e ci sia tutto ? [NB: non serve includere il PDF del testo]
- Hai rinominato IL PROGETTO esattamente come richiesto?
- Hai chiamato IL FILE ZIP esattamente come richiesto?
- Hai chiamato la cartella del progetto esattamente come richiesto?
- Dopo aver caricato il file su Esamix, hai premuto il tasto "CONFERMA", ottenendo il messaggio "Hai concluso l'esame"?