ESAME DI FONDAMENTI DI INFORMATICA T-2 dell' 11/6/2019
Proff. E. Denti, R. Calegari, A. Molesini – Tempo: 4 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)

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"

A supporto dei genitori che devono mettere a letto i bambini raccontando ogni sera una favola diversa, è stato richiesto lo sviluppo dell'applicazione *Il Favoliere,* in grado di sintetizzare automaticamente brevi favole a partire da alcuni elementi noti.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Tutte le favole seguono uno schema generale prestabilito, che si compone di quattro parti

- un *esordio* ("C'era una volta…) che introduce i *personaggi*
- lo *scenario* in cui essi si trovano
- una fase di *azione* in cui essi agiscono
- infine, una *conclusione* che porta all'inevitabile lieto fine.

I personaggi possono essere *positivi* (principi, cavalieri, principesse…) o *negativi* (orchi, lupi, personaggi malvagi vari): ogni scenario contiene sempre *almeno due personaggi positivi e uno negativo*, che si confrontano nell'azione – con ovvia vittoria finale dei buoni, altrimenti i bimbi non dormono e non fanno dormire neanche i genitori.

A seconda del numero di personaggi, della complessità dello scenario e del tipo di azione, la favola può essere più o meno adatta a bambini di diversa *fascia d'età* (piccolissimo, piccolo, grandicello, grande) e *impressionabilità* (per nulla impressionabile, poco impressionabile, impressionabile, molto impressionabile). La *complessità* dello scenario e la *durezza* dell'azione sono espresse tramite due *indici numerici* da 1 (minimo) a 5 (massimo).

Dev'essere garantito il rispetto di due vincoli chiave:

- i bambini *piccolissimi* non devono essere esposti a scenari di *complessità* superiore a 2, i *piccoli* non devono essere esposti a scenari di *complessità* superiore a 3

- i bambini *impressionabili* non devono essere esposti ad azioni di *durezza* superiore a 3, quelli *molto impressionabili* non devono essere esposti ad azioni di *durezza* superiore a 2

Il genitore specifica nella GUI solo la fascia d'età e il grado di impressionabilità del bambino: al resto pensa il sistema.

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..)

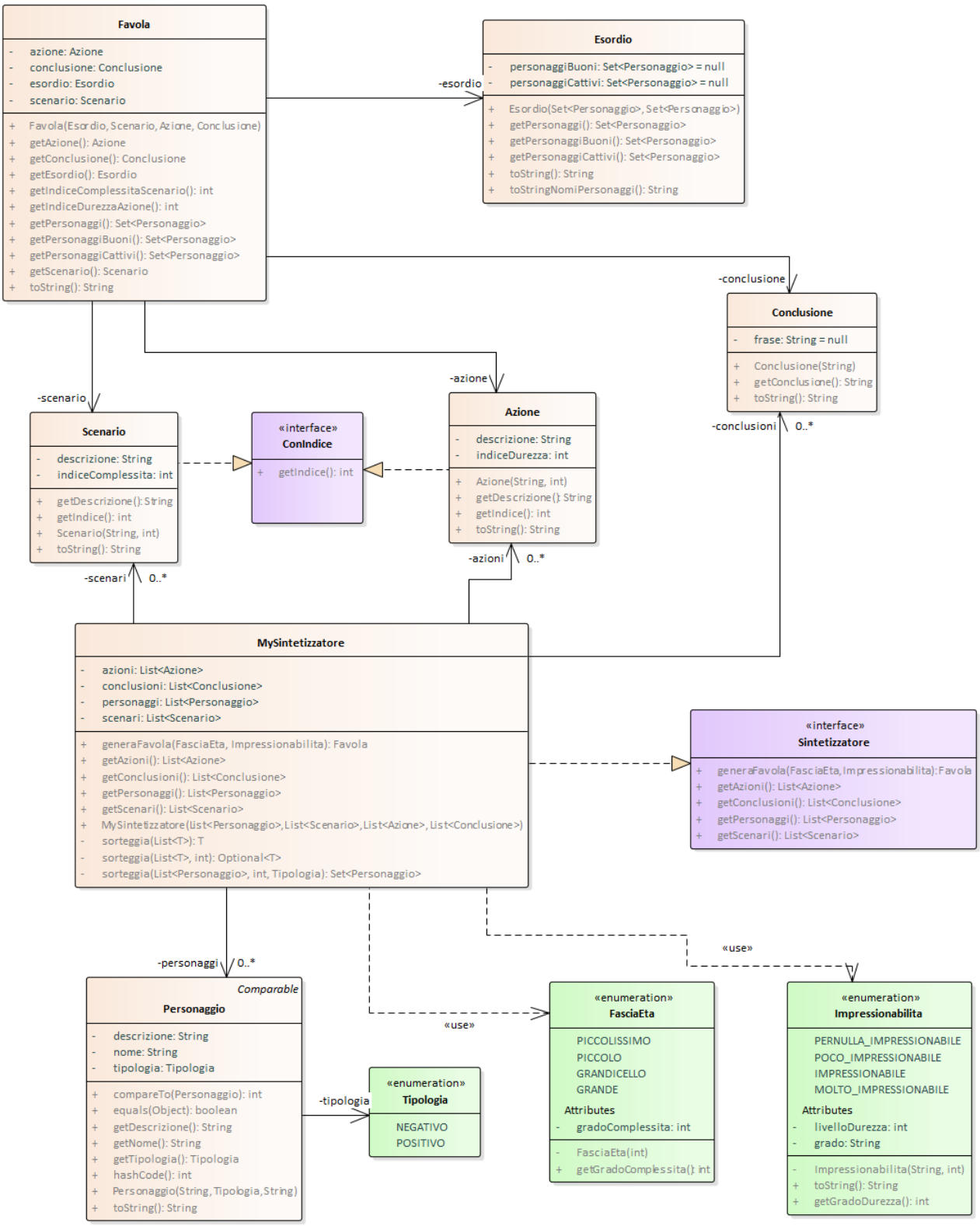
Checklist di consegna

- Hai fatto un unico file ZIP 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?
- Dopo aver caricato il file su Esamix, hai premuto il tasto "CONFERMA", ottenendo il messaggio "Hai concluso l'esame"?

Parte 1 (punti: 20)

Dati (package favoliere.model) *(punti: 10)*

Il modello dei dati deve essere organizzato secondo il diagramma UML sotto riportato: da notare che **l'unica classe da realizzare è MySintetizzatore**, essendo tutte le altre fornite già pronte nello Start Kit.



SEMANTICA:

a) L'enumerativo ***FasciaEta*** (fornito) rappresenta le fasce d'età dei bimbi (piccolissimi, piccoli, grandicelli, grandi), unitamente al livello di complessità massima corrispondente: in altri termini, a ogni valore dell'enumerativo è associata la complessità massima accettabile per bambini di quell'età, recuperabile tramite apposito accessor

b) L'enumerativo ***Impressionabilita*** (fornito) rappresenta i diversi gradi di impressionabilità dei bambini (molto impressionabile, impressionabile, poco impressionabile, per nulla impressionabile) unitamente – anche qui – al livello di durezza massima associato, anch'esso recuperabile tramite apposito metodo accessor

c) L'enumerativo ***Tipologia*** (fornito) rappresenta le due tipologie di personaggi (negativo e positivo)

d) La classe ***Favola*** (fornita) rappresenta la composizione delle quattro parti descritte nel dominio nel problema, specificate da altrettante classi (***Esordio***, ***Scenario***, ***Azione***, ***Conclusione***): in particolare

- ***Esordio*** è composto di ***Personaggi***, buoni e cattivi, tutti distinti
- ***Scenario*** e ***Azione*** implementano l'interfaccia ***ConIndice***, che cattura il fatto di avere un indice numerico (ciò risulterà utile in fase di sintesi della favola, per trattare scenari e azioni in modo uniforme).

e) Le quattro classi ***Esordio***, ***Scenario***, ***Azione***, ***Conclusione*** specificano le rispettive parti della favola, alimentate ognuna tramite un opportuno ***SectionLoader<E>*** (vedere oltre per i dettagli)

f) L'interfaccia ***Sintetizzatore***, che rappresenta il sintetizzatore delle favole, è fornita nello start kit. A parte gli ovvi accessor, il metodo principale `generaFavola` genera una favola a partire dalla fascia d'età e dal livello di impressionabilità richiesti, lanciando ***NoSuchTaleException*** (fornita) in caso ciò non sia possibile (perché nessuno scenario o azione rispetta i vincoli minimi di complessità/durezza, o per mancanza di personaggi)

g) la classe ***MySintetizzatore*** **(da realizzare)** implementa ***Sintetizzatore*** concretizzando il particolare algoritmo di sintesi espresso dal dominio del problema, ovvero:

- ogni favola ha sempre due personaggi positivi e uno negativo, sorteggiati fra quelli disponibili
- lo scenario e l'azione sono sorteggiati fra quelli disponibili *del giusto grado di complessità/durezza*
- la conclusione è semplicemente sorteggiata fra quelle disponibili

SUGGERIMENTO: predisporre **tre funzioni ausiliarie** `sorteggia(…)` che fattorizzino il sorteggio delle diverse parti, sfruttando gli ***Optional*** per esprimere l'eventuale assenza di risultato nel caso con indice (vedere UML).

In particolare:

- `Set<Personaggio> sorteggia(List<Personaggio> lista, int n, Tipologia tipo)` sorteggia un numero ***n*** di personaggi di tipologia ***tipo*** dalla lista data (utile per il sorteggio di buoni e cattivi)
- `<T> T sorteggia(List<T> lista)` sorteggia un elemento dalla lista data (utile per le ***Conclusioni***)
- `<T extends ConIndice> Optional<T> sorteggia(List<T> lista, int upperBound)` sorteggia dalla lista data un elemento che abbia un indice inferiore all'***upperBound*** passato (utile per il sorteggio di ***Scenario*** e ***Azione***, che devono essere scelti entro un indice prestabilito).

Persistenza (package favoliere.persistence) *(punti 10)*

La persistenza è strutturata su quattro file di testo:

- Il file di testo Personaggi.txt contiene l'elenco dei personaggi (uno per riga): ogni riga specifica innanzitutto se il personaggio sia positivo o negativo, nella forma "**`POSITIVO:`**" o "**`NEGATIVO:`**", poi ne dà la denominazione e infine, dopo un ulteriore "**`:`**", la descrizione.

```
POSITIVO: il principe William : un principe alto e nobile, sempre in sella al fido Alfius, il suo cavallo bianco
NEGATIVO: Urcus : un orco molto cattivo e feroce
...
```

- Il file di testo Scenari.txt contiene l'elenco degli scenari (uno per riga): ogni riga contiene prima la descrizione, poi – nella forma "***#N***" – l'indice numerico di complessità (dove N è un intero da 1 a 5)

```
nel bosco incantato di Dentilandia #2
nelle prigioni dell'alta torre del paese di Dentinia #4
...
```

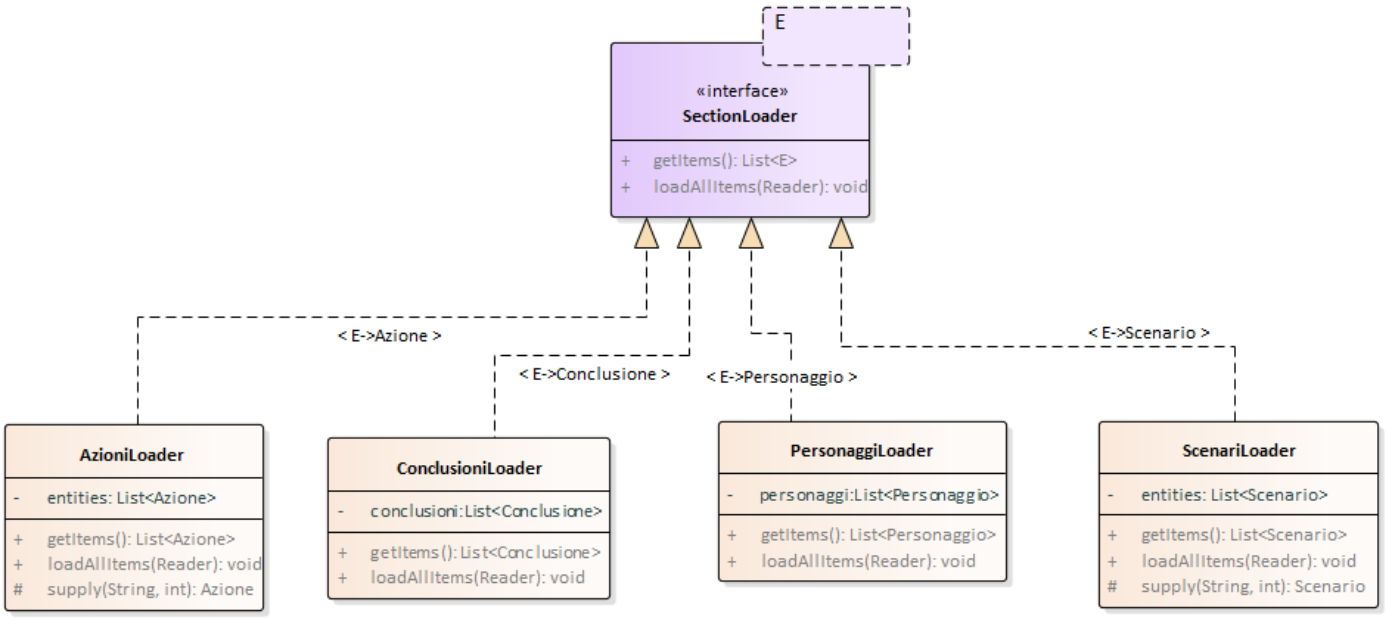
- Il file di testo Azioni.txt contiene l'elenco delle azioni (una per riga), descritte tramite semplici frasi: ogni riga contiene la frase, seguita – nella forma "***#N***" – dall'indice numerico di durezza (dove N è un intero da 1 a 5)

```
andò a salvarla con la spada #2
lanciò un incantesimo per carbonizzare gli avversari #4
...
```

- Il file di testo Conclusioni.txt contiene l'elenco delle frasi di chiusura (una per riga)

```
Vissero così infine tutti felici e contenti
Dopo una tale impresa, il cattivo fu sconfitto, i buoni si sposarono e vissero felici per molti anni
...
```

La struttura di questa parte dell'applicazione è illustrata nel diagramma UML sotto riportato.



SEMANTICA:

a) L'interfaccia tipizzata ***SectionLoader*<E>** specifica il reader della generica sezione della favola (destinato quindi a leggere secondo i casi *personaggi, scenari, azioni, conclusioni*):

- il metodo `getItems` restituisce una **List<E>** (con **E** che sarà nei vari casi rispettivamente ***Personaggio***, ***Scenario***, ***Azione*** o ***Conclusione***, come indicato con apposite frecce nei commenti sull'UML stesso)
- il metodo `loadAllItems` si occupa della lettura del file lanciando le eccezioni ***IOException*** o ***BadFileFormatException***: la prima viene lanciata in caso si verifichino errori di IO, la seconda nel caso in cui il formato del file non sia quello che ci si aspetta; in questo secondo caso, la stringa passata all'eccezione deve segnalare in modo preciso l'errore riscontrato nel file (es. "indice non numerico", "descrizione vuota", "mancanza tipologia in personaggio: XXXXXXXX:").

b) La classe ***ConclusioneLoader***, che implementa ***SectionLoader<Conclusione>***, è fornita nello start kit

c) Le tre classi ***PersonaggiLoader***, ***ScenariLoader*** e ***AzioniLoader***, sono invece da realizzare e precisamente:

- ***PersonaggiLoader*** implementa ***SectionLoader<Personaggio>***
- ***ScenariLoader*** e ***AzioniLoader***, quasi identiche data l'identica struttura dei file, implementano rispettivamente ***SectionLoader<Scenario>*** e ***SectionLoader<Azione>.***
  Si suggerisce (come indicato nel diagramma UML) di fattorizzare in un metodo ausiliario `supply` la creazione dello specifico oggetto ***Scenario***/***Azione***, così da rendere il più possibile invariante il codice dei due loader.

Parte 2 (punti: 10)

IMPORTANTE: per consentire il test della GUI anche nel caso di *reader* non funzionanti, è fornita la classe **FavoliereMock** che replica **FavoliereApp** utilizzando dati fissi pre-cablati al posto di quelli letti da file (utilizzando quindi un **ControllerMock**).

L'applicazione deve permettere all'utente-genitore di specificare semplicemente età ed indole del bambino.

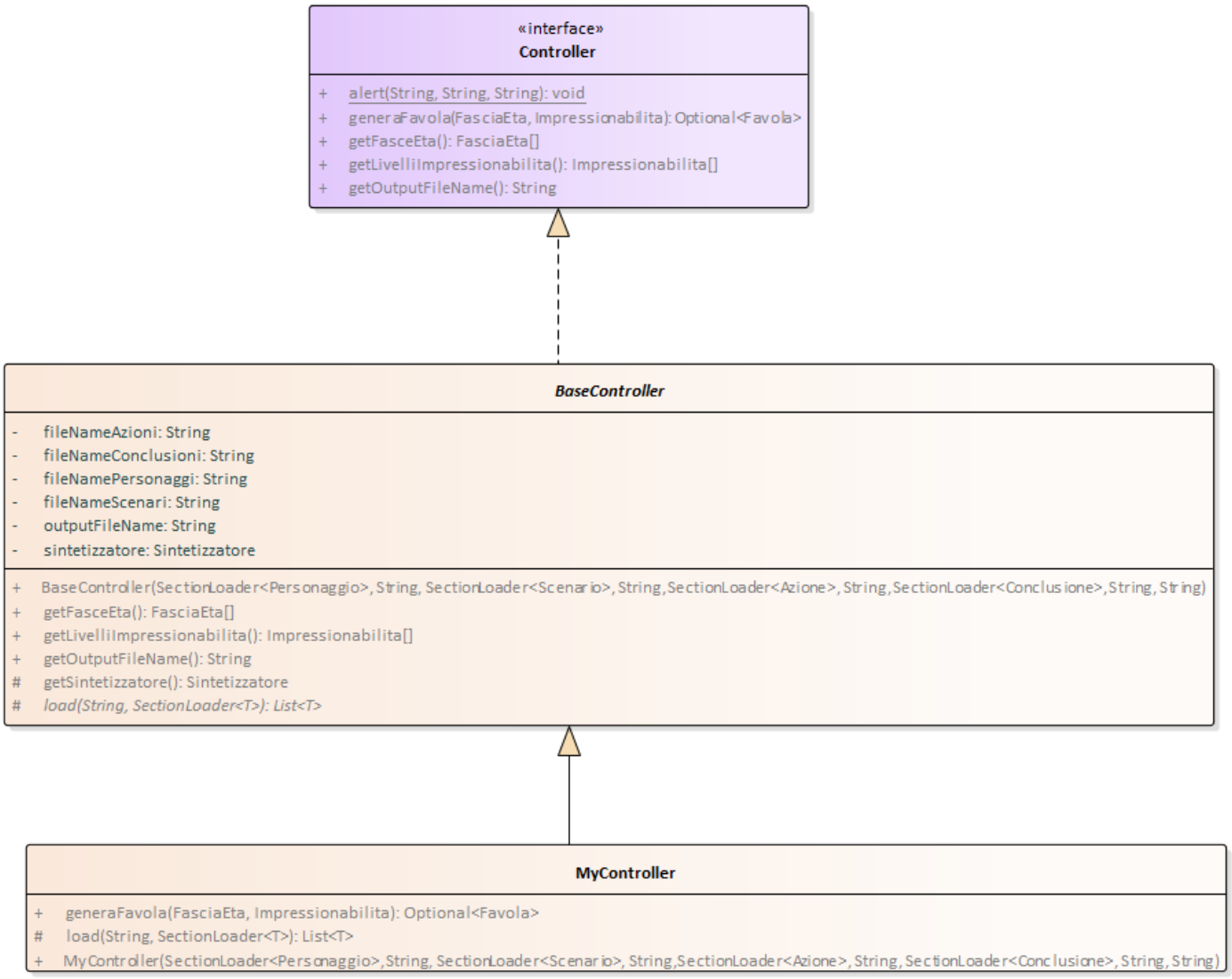
Con riferimento alle figure seguenti: premendo il pulsante ***Genera favola***, l'applicazione deve generare e mostrare sull'area di testo la favola richiesta. Successivamente, ove venga premuto il pulsante ***Salva su file***, la favola sarà trascritta sul file di testo Favola.txt.

Controller (package favoliere.ui.controller)

a) L'interfaccia ***Controller*** (fornita) specifica il controller dell'applicazione, che dichiara metodi per
   - recuperare l'insieme dei possibili livelli di impressionabilità e delle possibili fasce d'età
   - recuperare il nome del file di uscita su cui stampare (sua proprietà privata)
   - generare una favola coi livelli di impressionabilità e fascia d'età specificati (il risultato è un Optional perché potrebbero non esserci scenari, azioni o personaggi adatti a tutte le combinazioni)

b) La classe ***BaseController*** (fornita) concretizza parzialmente ***Controller*** prevedendo un unico costruttore che riceve in ingresso tutti gli argomenti necessari (v. UML) e implementando tutti i metodi tranne `generaFavola` ; espone inoltre due metodi ausiliari protetti rispettivamente per rendere accessibile dalle sottoclassi il sintetizzatore (metodo accessor `getSintetizzatore`) e per caricare un file tramite il giusto ***SectionLoader*** (metodo `load`)

c) La classe ***MyController*** (anch'essa fornita ☺) estende ***BaseController*** implementando
   a. il metodo `generaFavola`: lo fa delegando di fatto il lavoro al sintetizzatore, ma catturando l'eventuale eccezione ***NoSuchTaleException*** (da questi lanciata) gestendola in modo da restituire l'optional atteso.
   b. Il metodo `load` che si occupa della lettura della sezione interessata gestendo opportunamente eventuali eccezioni: ☺ in particolare, in caso il sistema non riesca a caricare uno dei file (personaggi, scenari, azioni, conclusioni) o vi siano errori di formato, deve far comparire un apposito dialogo col messaggio appropriato (Fig. 4) tramite il metodo statico ***Controller.alert***.



Interfaccia utente (package favoliere.ui) ***(punti 10)***

L'interfaccia utente deve essere simile (non necessariamente identica) all'esempio mostrato di seguito.

Se il caricamento preliminare (realizzato nell'***FavoliereApp*** fornita nello start kit) ha esito positivo, compare la finestra principale dell'applicazione (Fig. 1), costituita da un'istanza di ***MainPane*** (***da realizzare***) che riceve come unico argomento il ***Controller***.

Essa deve mostrare innanzitutto due combobox, precaricate coi valori standard delle fasce d'età e dei gradi di impressionabilità, *avendo cura che sia già selezionato un opportuno valore iniziale*.

Premendo il pulsante ***Genera favola***, l'applicazione genera e mostra sull'area di testo a lato la favola richiesta (Fig. 2): ciò abilita il pulsante ***Salva su file***, in precedenza disabilitato, per l'eventuale scrittura della favola sul file di testo. Nel caso in cui non sia possibile generare una favola con le specifiche richieste, si deve far comparire nell'area di testo il messaggio *"impossibile generare una favola coi vincoli richiesti"*.

Cambiando le specifiche di età e/o di impressionabilità, si possono generare via via nuove favole (Fig. 3).

Come già anticipato, premendo il pulsante ***Salva su file***, la favola precedentemente generata viene stampata sul file di output fornito dal controller. In caso non ci sia alcuna favola da stampare o vi siano errori di scrittura, deve comparire un apposito dialogo col messaggio appropriato (metodo statico ***Controller.alert***).

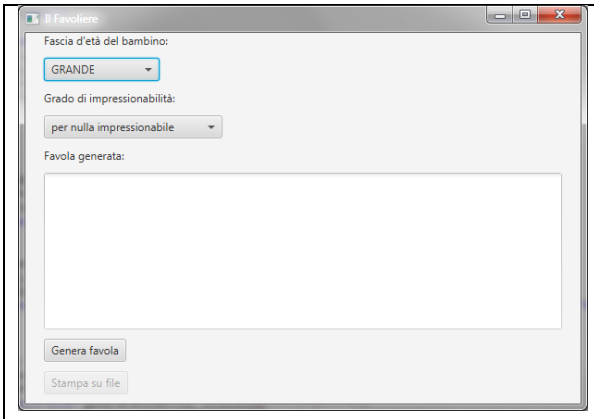


Fig. 1

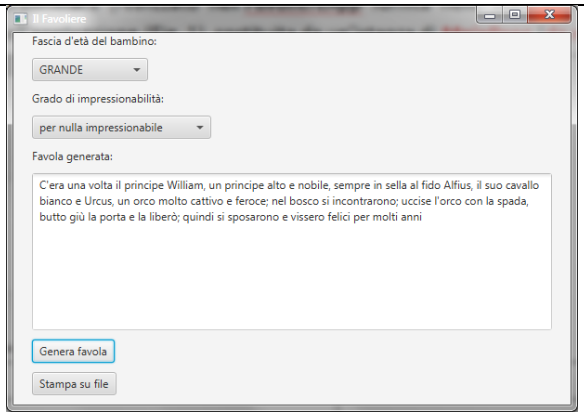


Fig. 2

Fig. 3

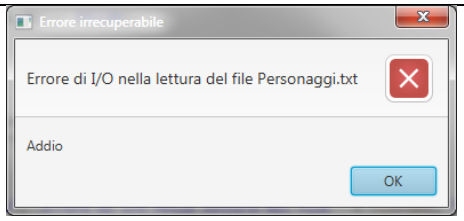


Fig. 4