
Politecnico di Torino

Politecnico di Torino

Corso di Laurea in Ingegneria Gestionale L8
A.a. 2021/2022
Sessione di Laurea Settembre 2022

# Sviluppo di un software per la gestione di una 24-hour Readathon

**Relatore:**

Prof. Fulvio Corno

**Candidato:**

Medoro Carla Maria (271882)

# Sommario

# Capitolo 1: Proposta di progetto

## 1.1 Studente proponente

S271882 Medoro Carla Maria

## 1.2 Titolo della proposta

Sviluppo di un software per la gestione di una 24-hour Readathon

## 1.3 Descrizione del problema proposto

Una 24-hour Readathon è una "gara" che i lettori ingaggiano con se stessi. Nata nel 2007, tutt'oggi viene ripetuta due volte l'anno, e l'obiettivo principale di chiunque vi partecipi è quello di impiegare quanto più tempo possibile delle 24 ore nella lettura, al fine di alleggerire la propria TBR (To Be Read list). I lettori possono decidere di partecipare a questa gara, seguendo, se stabilite, alcune regole, oppure di creare la propria Readathon in autonomia.

Tramite questa applicazione si desidera aiutare il lettore a scegliere quali libri leggere, avendo come fine quello di massimizzare il numero di libri letti nel tempo dedicato e cercando di minimizzare il tempo "avanzante", durante il quale non è possibile iniziare un nuovo libro per la Readathon perché non verrebbe completato in tempo. L'utente-lettore dovrà specificare, per esempio, il numero di pagine che è in grado di leggere mediamente in un'ora, il numero di ore in cui non sarà impegnato nella lettura, i generi che preferisce, il numero minimo e massimo di pagine che i libri devono avere. Qualora la soluzione proposta dall'applicazione sia di proprio gradimento, questa verrà accettata e i libri verranno inseriti nell'elenco dei libri letti, così che in iterazioni successive non vengano più presi in considerazione.

Immaginando che l'utente debba eventualmente comprare questi libri sarà possibile specificare un ulteriore vincolo, in modo che la soluzione venga elaborata sulla base di un proprio budget.

## 1.4 Descrizione della rilevanza gestionale del problema

Il problema è rilevante da un punto di vista gestionale, in quanto si tratta di un problema di ottimizzazione.

L'applicazione aiuta l'utente a ottimizzare il tempo a disposizione per effettuare questa Readathon grazie alla proposta di una soluzione personalizzata, ovvero il massimo numero di libri che potranno essere letti, e cosa più importante completati, impiegando così al massimo il tempo a disposizione e rispettando una serie di parametri specificati in input dall'utente.

Inoltre, l'applicazione sarà in grado di fornire una soluzione al problema sulla base di un ulteriore vincolo di budgeting, nel caso in cui l'utente abbia intenzione di acquistare i libri da leggere.

## 1.5 Descrizione del data-set per la valutazione

Per lo sviluppo dell'applicazione verrà utilizzato un dataset preso dal sito Kaggle: https://www.kaggle.com/datasets/canggih/goodreads-choice-awards-20112020?select=data-total.csv.

Questo dataset si basa su dati ottenuti a loro volta da Amazon e Goodreads, una piattaforma dove i lettori possono segnare i libri letti, quelli in lettura e quelli che vorrebbero leggere, scrivere recensioni e interagire con la comunità. In particolare, i libri sono stati prelevati a partire dalla lista di libri candidati ai Goodreads Choice Awards, fra il 2011 (prima edizione di questi "premi") e il 2020 (penultima edizione), e per ognuno di essi sono stati estratti una serie di dati e associati ad altrettanti dati dalle pagine Amazon dei libri.

Questo dataset è stato opportunamente ripulito e adattato, in quanto presenta righe errate e dati mancanti, e infine è stato convertito da file .csv a file .sql, per poterlo installare su HeidiSQL.

## 1.6 Descrizione preliminare degli algoritmi coinvolti

La funzionalità principale dell'applicazione sarà quella di utilizzare un algoritmo ricorsivo per fornire al lettore una lista di libri che potranno essere letti e completati nell'arco di tempo che l'utente ha a disposizione, sulla base di vincoli impostabili in input. In particolare, fra i vincoli più importanti per l'elaborazione corretta di una soluzione ci sono il numero di pagine che l'utente è in grado di leggere mediamente in un'ora, le ore da non considerare per la challenge (l'utente potrebbe avere degli impegni o volersi dedicare ad altro, e non esclusivamente alla lettura per 24 ore), una serie di generi a cui questi libri devono appartenere e il numero minimo e massimo di pagine che il lettore vuole che questi libri abbiano. Potranno poi essere introdotti altri parametri come il numero medio di rating minimo ed eventualmente anche una lista di autori, qualora l'utente voglia fare una maratona di libri dello stesso autore o voglia leggere quelli dei suoi autori preferiti, anche per essere sicuro che gli piacciano e per avere una maggiore probabilità di riuscire a concluderne la lettura.

Sarà eventualmente possibile implementare lo stesso algoritmo ricorsivo aggiungendo come obiettivo quello di generare una soluzione al problema che soddisfi un ulteriore vincolo, il budget, specificato in input dall'utente, immaginando che egli abbia la necessità di acquistare i libri e che questi non siano già presenti nella propria libreria.

## 1.7 Descrizione preliminare delle funzionalità previste per l'applicazione

L'interfaccia grafica si comporrà di un'unica sezione.

Essa si comporrà di una finestra in cui l'utente potrà inserire tutti i vincoli che riterrà necessari affinché l'applicazione restituisca una lista di libri validi. Avrà anche la possibilità di "accettare" la soluzione proposta, inserendo così questi libri nella lista dei propri libri letti, in modo tale che a una successiva iterazione dell'algoritmo questi non siano più presi in considerazione. Nella stessa finestra, infine, l'utente potrà specificare un budget e farsi generare una soluzione che contenga una serie di libri che, oltre a rispettare tutti gli altri vincoli, non superino questo budget.

# Capitolo 2: Descrizione dettagliata del problema

## 2.1 Cos'è una Readathon

Uno dei problemi principali per qualsiasi avido lettore, ed eventualmente anche collezionista di libri, è il desiderio di leggere tanti, troppi libri, ma non avere mai abbastanza tempo per farlo, così che la lista di libri da leggere (in gergo TBR – To Be Read List) si allunga a dismisura e non c'è nulla che si possa fare per contenere la situazione.
Nel 2007, Dewey, la proprietaria del blog "The Hidden Side of a Leaf", ha proposto una possibile soluzione a questo problema: una 24-hour Readathon, una sorta di gara a cui chiunque può partecipare e il cui obiettivo è proprio quello di alleggerire il più possibile la propria TBR, dedicando un'intera giornata, ovvero ben 24 ore, alla lettura. L'evento ha avuto così tanto successo che è stato ripetuto più volte nel corso degli anni, diffondendosi ampiamente nella comunità dei lettori.
Non ci sono delle vere e proprie regole per una Readathon: l'unica regola è divertirsi, leggere quanto si vuole e, perché no, diminuire di qualche unità la pila di libri da leggere.
Di solito le Readathon vengono organizzare da blogger, che possono decidere di aggiungere dei premi in base alle interazioni social degli altri lettori nell'arco dell'evento, ma non è raro che i lettori in autonomia ne creino una, dandosi obiettivi a loro misura, così come non è raro che si approfitti di queste giornate per cercare di terminare un libro in particolare che non si è mai riusciti a terminare, senza per forza avere l'obiettivo di leggerne il più possibile.
Una 24-hour Readathon è dunque una sfida con se stessi, e l'obiettivo finale può essere diverso per ciascun lettore.

## 2.2 Readathon in esame

Immaginiamo di avere un avido lettore che ha accumulato un discreto numero di libri da leggere: il modo più semplice per smaltire la sua TBR è organizzare una semplice Readathon. Non ci sono regole particolari, se non le "voglie" del lettore in questione, e l'unico obiettivo è quello di leggere quanti più libri possibili, occupando la maggior parte delle 24 ore dell'evento nella lettura.

Per organizzare una Readathon personalizzata, il lettore può tenere in considerazione una serie di voci che lo aiuteranno a scegliere l'insieme di libri da leggere, e che eventualmente possano aumentare la probabilità che i libri scelti vengano effettivamente completati nella lettura, e non abbandonati.

Innanzi tutto, bisogna tenere conto del tempo che il lettore non potrà dedicare alla lettura nell'arco delle 24 ore, per via di altri impegni o per la necessità di fare una pausa.

In secondo luogo, è fondamentale indicare la velocità di lettura dell'utente, ovvero il numero di pagine che in media egli riesce a leggere nell'arco di un'ora: sulla base di questo valore, infatti, verrà calcolata una stima del tempo di completamento di un libro e in questo modo si riuscirà a capire se esso può essere completato nel tempo rimanente e quanto tempo avanza per eventualmente completare altre letture. Per semplicità, l'applicazione permetterà di inserire un valore compreso fra 30 pagine all'ora e 70 pagine all'ora, in quanto mediamente le persone tendono a leggere in 60 minuti un numero di pagine ivi compreso.

L'utente dovrà inoltre specificare un massimo di tre generi a cui vuole appartengano i libri ed il numero minimo e massimo di pagine che i libri devono avere (potrebbe non voler leggere libri troppo lunghi, oppure troppo corti).

Per restringere ulteriormente l'insieme di libri a partire dai quali l'applicazione applicherà l'algoritmo per trovare la lista di libri da leggere nel tempo indicato, l'utente potrà inserire un minimo valore di rating medio, affinché si possa avere una maggiore probabilità che il libro possa piacergli, se complessivamente è stato apprezzato anche da altri (seppur questo non costituisca alcuna certezza).

Infine, l'utente-lettore potrà indicare un massimo di 3 autori "privilegiati", cosa principalmente utile qualora si volesse effettuare una maratona di titoli del proprio autore preferito. L'applicazione verificherà innanzitutto la corrispondenza fra i generi di cui si sono occupati gli autori selezionati e quelli indicati dall'utente: se essa è presente, verranno presi in considerazione solo quei libri di quell'autore che appartengono al genere corrispondente, altrimenti, poiché gli autori indicati hanno priorità sui generi, saranno mantenuti tutti i titoli, anche se non corrispondono a nessuno dei generi selezionati dall'utente.

Specificando questi parametri, l'applicazione sarà in grado di fornire all'utente un insieme di libri che gli consentiranno idealmente di terminare con successo la propria Readathon.

Naturalmente, l'applicazione non fornisce una soluzione efficacie al 100%, perché ci sono alcuni fattori casuali che per semplicità noi non prenderemo in considerazione, come il fatto che la velocità di lettura possa variare rispetto a quella media in base allo stato psicologico del lettore e al fatto che il libro in lettura gli stia piacendo o meno, oppure in base al genere o ancora in base al font utilizzato, però può sicuramente costituire un aiuto prezioso per un lettore indeciso.

# Capitolo 3: Descrizione del data-set

## 3.1 Descrizione

Il data-set utilizzato per questa applicazione è stato preso dal sito Kaggle: https://www.kaggle.com/datasets/canggih/goodreads-choice-awards-20112020?select=data-total.csv.

Il data-set originale comprendeva un totale di circa 4000 righe (libri) e 40 colonne ed era stato creato usando dati estratti da Goodreads, un sito in cui i lettori possono segnare i libri letti, quelli che vorrebbero leggere e quelli attualmente in lettura, in cui possono scrivere recensioni e interagire con altri utenti, e da Amazon. In particolare, i libri all'interno del database sono stati presi a partire dai libri nominati per i Goodreads Choice Awards, una sorta di concorso organizzato da Goodreads in cui i lettori possono scegliere quale libro, fra quelli candidati per una certa categoria, ritengono sia migliore. I dati fanno riferimento agli anni compresi fra il 2011 e il 2020.

Il data-set in questione è stato ripulito di colonne inutili per i nostri fini, di modo da alleggerirne il peso, e di righe errate, oltre che duplicate; è stato necessario modificare la colonna del prezzo dei libri in quanto basata su rupie indonesiane, quindi è stata fatta una conversione sulla base dell'attuale valore del cambio (1 rupia indonesiana = 0.0000067 euro) e si è proceduto anche a integrare i prezzi mancanti prendendoli da Amazon, poiché la colonna comprendeva circa 1250 valori su quasi 4000. Infine, poiché molte celle del file .csv non contenevano alcun valore, si è deciso di riempirle con NA, se i campi contenevano stringhe, o 0.0, nel caso di campi numerici.

## 3.2 Diagramma ER del data-set

Di seguito è proposto il diagramma ER rappresentante l'unica tabella di cui è composto il data-set:

| books | |
|---|---|
| book_id | int |
| **title** | **varchar** |
| **author** | **varchar** |
| description | varchar |
| bookformat | varchar |
| pages | int |
| rating_value0 | double |
| rating_count0 | double |
| genre_0 | varchar |
| genre_1 | varchar |
| genre_2 | varchar |
| genre_3 | varchar |
| genre_4 | varchar |
| genre_5 | varchar |
| genre_0_weight | double |
| genre_1_weight | double |
| genre_2_weight | double |
| genre_3_weight | double |
| genre_4_weight | double |
| genre_5_weight | double |
| price_EUR | double |

La tabella books contiene tutte le informazioni sui libri a partire dai quali verrà proposta una lista di libri da leggere e completare per la 24-hour Readathon:

- Book_id: numero progressivo, identificativo unico del libro
- Title: titolo del libro
- Author: autore del libro
- Description: trama del libro

- Bookformat: formato del libro, può essere hardcover (copertina rigida), Kindle Edition o ebook, paperback o mass market paperback (copertina flessibile), audiobook, comics, spiralbound, board book, audio CD, trade paperback
- Pages: numero di pagine del libro
- Rating_value0: media delle stelle che i lettori hanno dato al libro (da 0 a 5)
- Rating_count0: numero di lettori che hanno lasciato una valutazione
- Genre_0: genere principale del libro (i libri possono appartenere a più generi diversi, alcuni più specifici e altri più generici). I generi su Goodreads di solito vengono determinati sulla base del numero di lettori che hanno inserito quel libro in uno "scaffale" della loro libreria virtuale.
- Genre_1: altro genere del libro
- Genre_2: altro genere del libro
- Genre_3: altro genere del libro
- Genre_4: altro genere del libro
- Genre_5: altro genere del libro
- Genre_0_weight: percentuale di persone che hanno inserito il libro nello scaffale corrispondente al genere 0.
- Genre_1_weight: percentuale di persone che hanno inserito il libro nello scaffale corrispondente al genere 1.
- Genre_2_weight: percentuale di persone che hanno inserito il libro nello scaffale corrispondente al genere 2.
- Genre_3_weight: percentuale di persone che hanno inserito il libro nello scaffale corrispondente al genere 3.
- Genre_4_weight: percentuale di persone che hanno inserito il libro nello scaffale corrispondente al genere 4.
- Genre_5_weight: percentuale di persone che hanno inserito il libro nello scaffale corrispondente al genere 5.
- Price_EUR: prezzo per il libro nel formato specificato (valori presi da Amazon, o in alternativa da Abebooks qualora il libro fosse ormai fuori produzione e non più venduto)

# Capitolo 4: Descrizione delle strutture dati e degli algoritmi utilizzati

## 4.1 Strutture dati

L'applicazione è stata realizzata in linguaggio Java, e implementa il pattern MVC (Model View Controller) e il pattern DAO (Data Access Object).

Per realizzare l'interfaccia grafica in JavaFX è stato utilizzato il software SceneBuilder.

Il progetto si compone di tre package, corrispondenti all'interfaccia utente, al modello e all'interfaccia utente:

- Package it.polito.tdp.h24Readathon: qui sono presenti tre classi, il Main, da cui viene eseguita l'applicazione, l'Entry Point e l'FXMLController, in cui sono definiti i metodi necessari per garantire l'interazione corretta con l'utente.
- Package it.polito.tdp.h24Readathon.db: in questo package è compresa la classe ConnectDB, che implementa la connessione al database tramite il Connection Pooling, e la classe BooksDAO, che contiene i metodi necessari ad ottenere informazioni di varia natura dal data-set stesso, come quelli per ottenere tutti i generi e tutti gli autori, oppure ancora i metodi per selezionare i libri necessari per la parte algoritmica dell'applicazione.
- Package it.polito.tdp.h24Readathon.model: in questo package sono presenti due classi rappresentanti gli oggetti con cui abbiamo a che fare nel programma, ovvero i libri (classe Book) e una classe che permetta di associare un autore a una serie di generi di cui si è occupato (classe AuthorGenres); infine, il package contiene la classe Model in cui sono implementati tutti i metodi che hanno come obiettivo quello di effettuare una serie di operazioni sui dati estratti dal database e di implementare l'algoritmo ricorsivo che permette all'applicazione di generare una listi di libri che il lettore può leggere nell'arco del tempo a disposizione.

## 4.2 Principali algoritmi

Come già anticipato, l'obiettivo dell'applicazione è quello di proporre al lettore una lista di libri che egli sia in grado di completare nel tempo che può dedicare alla lettura, sulla base di una serie di parametri che sono stati indicati in input.

Il primo passaggio consiste nell'estrarre dal data-set i libri che verranno poi utilizzati nella ricorsione.

In primo luogo, si estraggono tutti quei titoli che rispettano i vincoli in termini di numero di pagine minimo e massimo, rating medio e autori.

```
public List<Book> getAllBooksAuthor(int minPages, int maxPages, List<String> authors, double ratings){
    this.allBooksForReadathon = new ArrayList<>();
    for(String a : authors) {
        this.allBooksForReadathon.addAll(this.dao.getBooksByAuthorAndRatings(a, minPages, maxPages, ratings));
    }
    return this.allBooksForReadathon;
}
```

In secondo luogo, si estraggono i titoli che rispettano i vincoli in termini di numero di pagine minimo e massimo, rating medio e generi.

```
public List<Book> getMoreBooks(int minPages, int maxPages, List<String> genres, double ratings){
    this.moreBooks = new ArrayList<>();
    for(String g : genres) {
        this.moreBooks.addAll(this.dao.getBooksByGenreAndRating(g, minPages, maxPages, ratings));
    }
    return this.moreBooks;
}
```

A questo punto, bisogna andare a verificare la compatibilità dei generi degli autori indicati dall'utente con i generi, sempre da questi indicati: il metodo filtra i libri dei singoli autori andando a mantenerne tutti i libri in caso di incompatibilità con tutti i generi selezionati, altrimenti manterrà soltanto quelli del genere corrispondente a uno di quelli scelti dall'utente. I generi compatibili sono determinati da un metodo a parte. Viene, inoltre, filtrata anche la lista dei libri che tengono soltanto in considerazione i generi, e non gli autori, togliendo da questa i libri che sono in comune con la lista dei titoli filtrati per autore.

```
public List<String> getCompatibility(String author, List<String> genres) {
    List<String> compatibleGenres = new ArrayList<>();
    AuthorGenres ag = null;
    ag = this.getAuthorGenres(author);
    if(ag != null) {
        for(String gen : ag.getGenres()) {
            for(int i = 0; i < genres.size(); i++) {
                if(gen.equals(genres.get(i))) {
                    compatibleGenres.add(gen);
                }
            }
        }
    }
    return compatibleGenres;
}
```

```
public List<Book> filterBooks(List<Book> allBooks, List<String> authors, List<String> genres, List<Book> more){
    //qui devo filtrare i libri: poichè la ricerca viene fatta dando priorità agli autori, se l'autore in osservazione
    //scrive un genere fra quelli selezionati, prendo in considerazione solo i libri di quel genere, altrimenti vorrà dire
    //che vi è un'incompatibilità fra autore e generi e il programma mantiene tutti i libri di quell'autore
    filteredAll = new ArrayList<>();
    filteredMore = new ArrayList<>(more);
    for(String a : authors) {
        List<String> compatible = new ArrayList<>(this.getCompatibility(a, genres));
        if(compatible.size() == 0) {
            for(Book b : allBooks) {
                if(b.getAuthor().equals(a)) {
                    filteredAll.add(b);
                }
            }
        }else {
            for(String gc : compatible) {
                for(Book b : allBooks) {
                    if(b.getAuthor().equals(a) && b.getGenre0().equals(gc)) {
                        filteredAll.add(b);
                    }
                }
            }
        }
    }
    //devo poi togliere eventualmente i libri che ci sono in comune
    for(Book b : filteredAll) {
        if(filteredMore.contains(b)) {
            filteredMore.remove(b);
        }
    }
    return filteredAll;
}
```

Una volta ricavate le due liste su cui lavorerà l'algoritmo ricorsivo, si può dare avvio alla ricorsione: nel metodo vengono inizializzate tutte le strutture dati necessarie per memorizzare i risultati dell'algoritmo e si distingue il caso in cui sia stato indicato un budget da rispettare o meno. Il metodo restituisce la lista dei libri proposta dalla ricorsione.

```java
public List<Book> findReadathon(int speed, double pause, List<Book> filteredAll, List<Book> filteredMore, Double budget) {
    this.bestReadathon = new ArrayList<>();
    List<Book> partial = new ArrayList<>();
    if(budget != null)
        this.bestPrice = budget;
    this.bestTime = 0.0;
    this.minTot = 0.0;
    availableTime = 24*60-(pause*60);
    long start = System.nanoTime();
    if(budget == null) {
        findBestReadathon(speed, availableTime, partial, filteredAll, filteredMore);
    }else {
        findBestBudgeted(speed, availableTime, budget, partial, filteredAll, filteredMore);
    }
    long fin = System.nanoTime();
    System.out.println(fin-start);

    System.out.println("Libri da leggere: "+this.bestReadathon.size()+"\n");
    for(Book b : this.bestReadathon) {
        System.out.println(b.toString()+"\n");
    }
    this.minTot = availableTime-this.bestTime;
    System.out.println("Tempo impiegato su "+availableTime+" minuti: "+(availableTime-this.bestTime)+"\n");
    System.out.println("Budget impiegato su "+this.bestPrice+" euro: "+this.getTotalPrice(this.bestReadathon));
    return this.bestReadathon;
}
```

Per avere un codice più pulito, si è deciso di implementare due metodi ricorsivi, uno che venga utilizzato qualora l'utente non specifichi un budget da rispettare e un altro che invece tenga conto proprio del budget.

Poiché l'obiettivo principale è quello di determinare la lista di libri più numerosa che permetta di massimizzare il tempo impiegato nella lettura, o, se si vuole, minimizzare il tempo avanzante in cui non è possibile incominciare e completare una nuova lettura, la soluzione si ricava nel momento in cui il tempo rimanente non è più sufficiente a completare nemmeno il libro più corto fra quelli rimasti, oppure quando il budget risulta insufficiente rispetto al libro meno costoso fra i rimanenti.

Se ho terminato di inserire all'interno della soluzione temporanea tutti i libri degli autori selezionati, e qualora avanzasse ancora del tempo per poter inserire almeno il libro più corto fra l'insieme di quelli selezionati in base al genere, si procede con la sostituzione dei libri filtrati e si manda avanti la ricorsione, questa volta considerando i libri filtrati in base al genere.

Per determinare quale libro posso aggiungere alla soluzione parziale, bisogna verificare che il tempo impiegato per completarne la lettura rientri nel tempo rimanente complessivo, ed eventualmente che anche il costo del libro non ecceda il budget restante.

```java
private void findBestReadathon(int speed, double availableTime, List<Book> partial, List<Book> filAll, List<Book> filMore) {
    //caso terminale: non c'è sufficiente tempo per completare nemmeno il libro più piccolo ancora a disposizione.
    // Sovrascrivo la bestSolution solo se partial ha maggior numero di libri e il minor tempo libero
    if(partial.size() > 0 && partial.size() >= this.bestReadathon.size()) {
        if(!this.remainingBooks(filAll, partial) && filMore.size() > 0) {
            if(!this.isTimeEnough(availableTime, this.getMinPages(filMore, partial), speed)) {
                //se non c'è più tempo per ulteriori libri: finito
                if( this.bestTime == 0.0 || (this.bestTime > 0.0 && availableTime < this.bestTime) ) {
                    this.bestReadathon = new ArrayList<>(partial);
                    this.bestTime = availableTime;
                    return;
                }
            }else {
                //Se c'è ancora tempo, vado avanti con la ricorsione
                filAll = new ArrayList<>(this.getRemainingBooks(filMore, partial));
                filMore = new ArrayList<>(filMore);
                this.findBestReadathon(speed, availableTime, partial, filAll, filMore);
            }
        }else { // non ci sono più libri in generale: se non c'è più tempo ---> fine
            if(!this.isTimeEnough(availableTime, this.getMinPages(filMore, partial), speed)) {
                //se non c'è più tempo per ulteriori libri: finito
                if( this.bestTime == 0.0 || (this.bestTime > 0.0 && availableTime < this.bestTime) ) {
                    this.bestReadathon = new ArrayList<>(partial);
                    this.bestTime = availableTime;
                    return;
                }
            }
        }
        //se non ho ancora finito con i libri degli autori e in più non c'è più tempo: finisco la ricorsione
        if(this.remainingBooks(filAll, partial) && !this.isTimeEnough(availableTime, this.getMinPages(filAll, partial), speed)) {
            if( this.bestTime == 0.0 || (this.bestTime > 0.0 && availableTime < this.bestTime) ) {
                this.bestReadathon = new ArrayList<>(partial);
                this.bestTime = availableTime;
                return;
            }
        }
    }

    //caso normale: accetto un libro solo se non è già presente e può essere completato nel tempo rimanente
    for(Book b : this.getRemainingBooks(filAll, partial)) {
        double ttc = this.getTimeCompleted(b.getPages(), speed);
        if(ttc <= availableTime && !partial.contains(b)) {
            partial.add(b);
            this.findBestReadathon(speed, availableTime-ttc, partial, filAll, filMore);
            partial.remove(b);
        }
    }
}
```

```java
private void findBestBudgeted(int speed, double availableTime, Double availableBudget, List<Book> partial, List<Book> filAll,
        List<Book> filMore) {
    if(partial.size() > 0 && partial.size() >= this.bestReadathon.size()) {
        if(!this.isBudgetEnough(availableBudget, this.getMinPrice(filAll, partial)) &&
                !this.remainingBooks(filAll, partial) && filMore.size() > 0) {
            if(!this.isTimeEnough(availableTime, this.getMinPages(filMore, partial), speed)) {
                if( this.bestTime == 0.0 || (this.bestTime > 0.0 && availableTime < this.bestTime
                        && this.getTotalPrice(partial) <= this.bestPrice) ) {
                    this.bestReadathon = new ArrayList<>(partial);
                    this.bestTime = availableTime;
                    return;
                }
            }else { //Se c'è ancora tempo, vado avanti con la ricorsione
                filAll = new ArrayList<>(this.getRemainingBooks(filMore, partial));
                filMore = new ArrayList<>(filMore);
                this.findBestBudgeted(speed, availableTime, availableBudget, partial, filAll, filMore);
            }
        }else if(!this.isBudgetEnough(availableBudget, this.getMinPrice(filAll, partial)) &&
                this.remainingBooks(filAll, partial) && filMore.size() == 0) {
            if(!this.isTimeEnough(availableTime, this.getMinPages(filMore, partial), speed)) {
                if( this.bestTime == 0.0 || (this.bestTime > 0.0 && availableTime < this.bestTime
                        && this.getTotalPrice(partial) <= this.bestPrice) ) {
                    this.bestReadathon = new ArrayList<>(partial);
                    this.bestTime = availableTime;
                    return;
                }
            }
        }
        if(this.remainingBooks(filAll, partial) && (!this.isBudgetEnough(availableBudget, this.getMinPrice(filAll, partial)) ||
                !this.isTimeEnough(availableTime, this.getMinPages(filAll, partial), speed))) {
            if( this.bestTime == 0.0 || (this.bestTime > 0.0 && availableTime < this.bestTime &&
                    this.getTotalPrice(partial) <= this.bestPrice) ) {
                this.bestReadathon = new ArrayList<>(partial);
                this.bestTime = availableTime;
                return;
            }
        }
    }
```

```
    for(Book b : this.getRemainingBooks(filAll, partial)) {
        double ttc = this.getTimeCompleted(b.getPages(), speed);
        double price = b.getPriceEUR();
        if(ttc <= availableTime && price <= availableBudget  && !partial.contains(b)) {
            partial.add(b);
            this.findBestBudgeted(speed, availableTime-ttc, availableBudget-price, partial, filAll, filMore);
            partial.remove(b);
        }
    }
}
```

All'interno dei metodi findBestReadathon e findBestBudgeted, vengono richiamati una serie di metodi che hanno l'obiettivo di eseguire controlli atti a capire se si è già in "possesso" di una potenziale soluzione oppure se, di contro, è ancora possibile inserire altri libri all'interno della soluzione parziale.

Innanzitutto, è stato creato un metodo per determinare il tempo di completamento di un libro a partire dal numero di pagine e dalla velocità di lettura dell'utente.

```
public double getTimeCompleted(int pages, int speed) {
    double pg = (double) pages;
    double rSpeed = 60.0/(double)speed;
    double completed = pg*rSpeed;
    return completed;
}
```

Successivamente sono stati implementati due metodi che permettono di, rispettivamente, individuare i libri rimanenti per la ricorsione e altri controlli, andando a eliminare dalla lista di libri complessiva quelli già inseriti nella soluzione parziale, e determinare se ci sono libri rimanenti. Quest'ultimo metodo è utile per verificare se ho finito di inserire nella soluzione parziale tutti i libri filtrati per autore, in modo tale da eventualmente far proseguire la ricorsione usando quelli filtrati per genere, qualora ci sia ancora spazio per altri libri.

```
public Set<Book> getRemainingBooks(Collection<Book> all, Collection<Book> partial){
    Set<Book> remaining = new HashSet<>(all);
    for(Book b : partial) {
        if(remaining.contains(b)){
            remaining.remove(b);
        }
    }
    return remaining;
}

public boolean remainingBooks(Collection<Book> all, Collection<Book> partial){
    Set<Book> remaining = new HashSet<>(this.getRemainingBooks(all, partial));
    if(remaining.size() != 0) {
        return true;
    }else {
        return false;
    }
}
```

Il metodo getMinPages() consente di individuare quale sia il numero di pagine più piccolo fra i libri rimanenti.

```
public Integer getMinPages(Collection<Book> all, Collection<Book> partial) {
    Set<Book> remaining = new HashSet<>(this.getRemainingBooks(all, partial));
    Integer min = null;
    for(Book b : remaining) {
        if(min == null) {
            min = b.getPages();
        }else if(b.getPages() < min) {
            min = b.getPages();
        }
    }
    return min;
}
```

Invece, il metodo isTimeEnough() serve per capire se il tempo rimanente per la Readathon è sufficiente a coprire quantomeno il libro con il numero più piccolo di pagine oppure no. Se non ci sono più libri, perché sono stati tutti inseriti nella soluzione parziale, il metodo ritorna comunque false, perché vuol dire che il tempo rimanente non può essere coperto.

```
public boolean isTimeEnough(double availableTime, Integer minPg, int speed) {
    if(minPg == null) {
        return false; //non ci sono più libri rimanenti ---> fine
    }
    double ttc = this.getTimeCompleted(minPg, speed);
    if( ttc <= availableTime) {
        return true;
    }else {
        return false;
    }
}
```

Nel caso in cui l'utente abbia inserito anche un budget, sono stati implementati dei metodi analoghi.

È presente un metodo per calcolare il costo totale dei libri presenti nella soluzione parziale o in quella definitiva.

```
public double getTotalPrice(Collection<Book> partial) {
    double tot = 0.0;
    for(Book b : partial) {
        tot += b.getPriceEUR();
    }
    return tot;
}
```

Un altro metodo consente di individuare quale sia il libro con il prezzo più basso, fra i rimanenti.

```
public Double getMinPrice(Collection<Book> all, Collection<Book> partial) {
    Set<Book> remaining = new HashSet<>(this.getRemainingBooks(all, partial));
    Double min = null;
    for(Book b : remaining) {
        if(min == null) {
            min = b.getPriceEUR();
        }else if(b.getPriceEUR() < min) {
            min = b.getPriceEUR();
        }
    }
    return min;
}
```

Un altro permette di capire se c'è ancora del budget disponibile per coprire almeno il libro meno costoso fra i rimanenti, così da far andare avanti la ricorsione o terminarla perché evidentemente si è in presenza di una potenziale soluzione. Anche in questo caso, se il prezzo risultasse nullo in quanto non ci sono ulteriori libri a disposizione, il metodo ritorna false perché non c'è alcun titolo in grado di coprire il budget rimanente.

```
public boolean isBudgetEnough(double availableBudget, Double price) {
    if(price == null) {
        return false;
    }
    if(price <= availableBudget) {
        return true;
    }else {
        return false;
    }
}
```

# Capitolo 5: Diagramma delle classi delle principali parti dell'applicazione

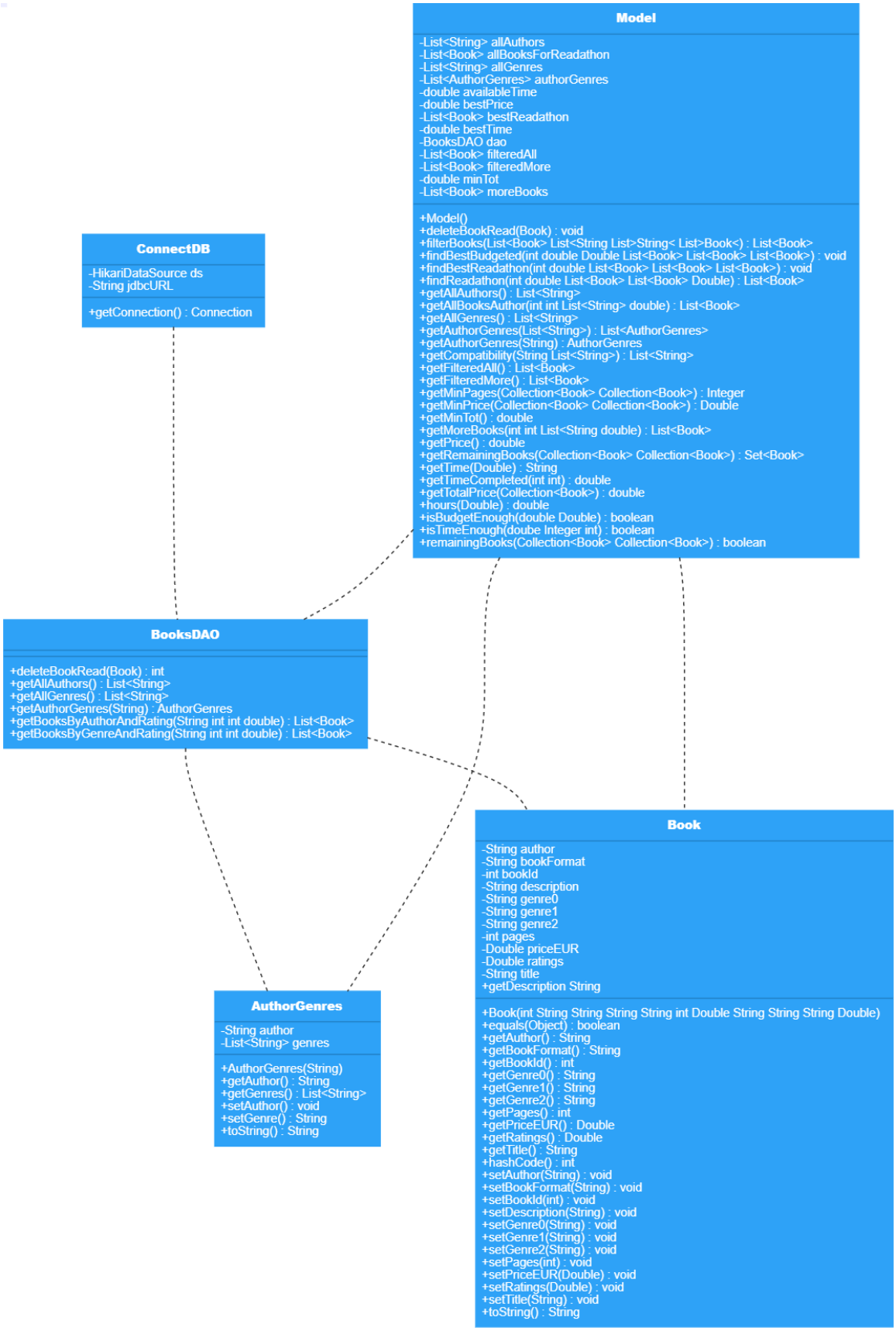
## 5.1 Classi dell'applicazione

Le classi di cui si compone l'applicazione sono:

- Main: punto di partenza dell'applicazione
- EntryPoint: classe che collega il Model al Controller
- FXMLController: classe che implementa i metodi necessari alla gestione dell'interfaccia grafica dell'applicazione
- Model: classe che implementa i metodi necessari per gestire la parte algoritmica del programma
- Book: classe che rappresenta l'oggetto libro e che consente di memorizzare i dati del database
- AuthorGenres: classe utile per memorizzare autore e la lista di generi di cui si è occupato
- ConnectDB: classe necessaria per la connessione al database
- BooksDAO: classe contenente i metodi necessari per recuperare i dati dal database e passarli al Model perché possano essere elaborati.

## 5.2 Diagramma delle classi principali

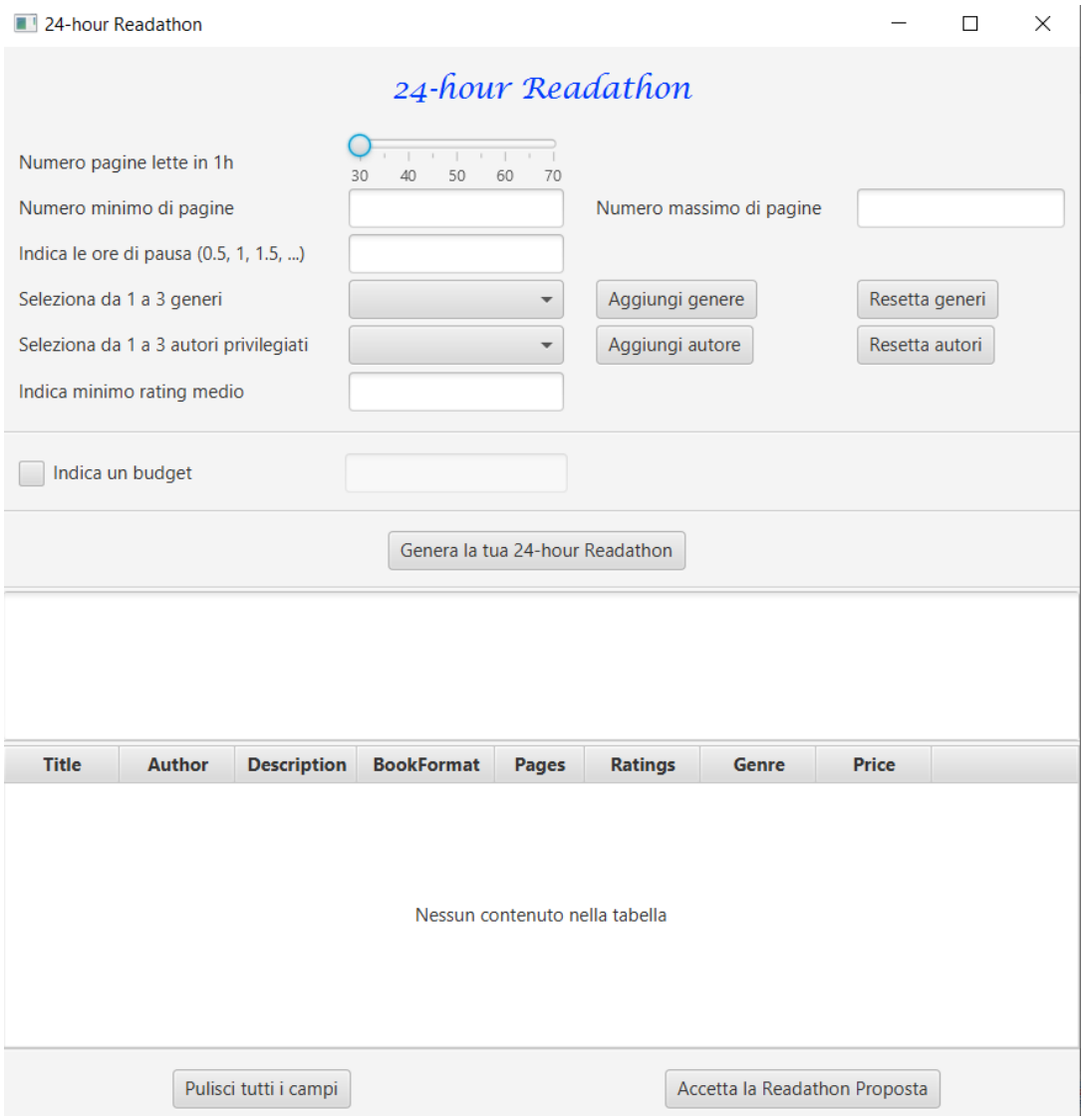
Di seguito un diagramma delle principali classi:


Model
-List<String> allAuthors
-List<Book> allBooksForReadathon
-List<String> allGenres
-List<AuthorGenres> authorGenres
-double availableTime
-double bestPrice
-List<Book> bestReadathon
-double bestTime
-BooksDAO dao
-List<Book> filteredAll
-List<Book> filteredMore
-double minTot
-List<Book> moreBooks
+Model()
+deleteBookRead(Book) : void
+filterBooks(List<Book> List<String List>String< List>Book<) : List<Book>
+findBestBudgeted(int double Double List<Book> List<Book> List<Book>) : void
+findBestReadathon(int double List<Book> List<Book> List<Book>) : void
+findReadathon(int double List<Book> List<Book> Double) : List<Book>
+getAllAuthors() : List<String>
+getAllBooksAuthor(int int List<String> double) : List<Book>
+getAllGenres() : List<String>
+getAuthorGenres(List<String>) : List<AuthorGenres>
+getAuthorGenres(String) : AuthorGenres
+getCompatibility(String List<String>) : List<String>
+getFilteredAll() : List<Book>
+getFilteredMore() : List<Book>
+getMinPages(Collection<Book> Collection<Book>) : Integer
+getMinPrice(Collection<Book> Collection<Book>) : Double
+getMinTot() : double
+getMoreBooks(int int List<String double) : List<Book>
+getPrice() : double
+getRemainingBooks(Collection<Book> Collection<Book>) : Set<Book>
+getTime(Double) : String
+getTimeCompleted(int int) : double
+getTotalPrice(Collection<Book>) : double
+hours(Double) : double
+isBudgetEnough(double Double) : boolean
+isTimeEnough(doube Integer int) : boolean
+remainingBooks(Collection<Book> Collection<Book>) : boolean
ConnectDB
-HikariDataSource ds
-String jdbcURL
+getConnection() : Connection
BooksDAO
+deleteBookRead(Book) : int
+getAllAuthors() : List<String>
+getAllGenres() : List<String>
+getAuthorGenres(String) : AuthorGenres
+getBooksByAuthorAndRating(String int int double) : List<Book>
+getBooksByGenreAndRating(String int int double) : List<Book>
Book
-String author
-String bookFormat
-int bookId
-String description
-String genre0
-String genre1
-String genre2
-int pages
-Double priceEUR
-Double ratings
-String title
+getDescription String
+Book(int String String String int Double String String String Double)
+equals(Object) : boolean
+getAuthor() : String
+getBookFormat() : String
+getBookId() : int
+getGenre0() : String
+getGenre1() : String
+getGenre2() : String
+getPages() : int
+getPriceEUR() : Double
+getRatings() : Double
+getTitle() : String
+hashCode() : int
+setAuthor(String) : void
+setBookFormat(String) : void
+setBookId(int) : void
+setDescription(String) : void
+setGenre0(String) : void
+setGenre1(String) : void
+setGenre2(String) : void
+setPages(int) : void
+setPriceEUR(Double) : void
+setRatings(Double) : void
+setTitle(String) : void
+toString() : String
AuthorGenres
-String author
-List<String> genres
+AuthorGenres(String)
+getAuthor() : String
+getGenres() : List<String>
+setAuthor() : void
+setGenre() : String
+toString() : String

# Capitolo 6: Videate dell'applicazione

## 6.1 Funzionamento dell'applicazione

L'applicazione si compone di un'unica finestra in cui l'utente può inserire i vincoli che vuole la lista di libri proposta rispetti.




L'utente può specificare la propria velocità di lettura tramite uno slider, che permette di indicare un valore compreso fra 30 e 70 pagine all'ora, con una granularità di 5 unità.

È poi possibile inserire manualmente due numeri corrispondenti rispettivamente al numero minimo e il numero massimo di pagine che i libri da considerare devono avere.

Poiché il lettore ragionevolmente non impegna tutte e 24 le ore nella lettura, l'applicazione gli permette di specificare la quantità di "pausa" da tenere in

considerazione: qui si richiede di inserire un numero decimale che rappresenti il numero di ore, o una frazione di esse, di pausa, che poi l'applicazione convertirà in minuti per le elaborazioni successive.

Altri due vincoli da specificare sono quelli riguardanti i generi e gli autori: in entrambi i casi è stata utilizzata una ComboBox in cui sono caricati i rispettivi dati presenti nel database. Per inserire un genere o un autore, bisogna selezionarlo dalla ComboBox e successivamente cliccare sul bottone "Aggiungi genere/autore", per salvare la selezione: quando sono stati selezionate tre opzioni, la ComboBox e il Button saranno disabilitati in automatico, per impedire all'utente di inserire più opzioni del previsto. I Button "Resetta generi/autori" servono per cancellare gli attuali valori memorizzati per le due ComboBox e permettono all'utente di ripartire da capo con la selezione delle opzioni; inoltre, se l'utente cambiasse idea dopo aver già confermato tre valori, il pulsante cancella la selezione e abilita nuovamente gli elementi disabilitati.

Ultimo parametro obbligatorio da inserire è il minimo rating medio che l'utente vuole che i libri proposti abbiano.

Opzionalmente, cliccando sulla CheckBox corrispondente, il lettore può anche inserire un budget da rispettare come ulteriore vincolo che la soluzione proposta dall'applicazione deve tenere in considerazione.

Per generare la Readathon, basta cliccare sull'apposito pulsante.

Al di sotto di questo, sono presenti una TextArea, in cui vengono riportati eventuali messaggi di errore oppure risultati della ricorsione, e una TableView, in cui vengono presentati i libri ricavati dall'esecuzione dell'algoritmo.

Infine, l'applicazione prevede un pulsante che ripulisce tutti i campi e un altro che serve per accettare la soluzione proposta: nel momento in cui questa viene accettata, perché il lettore la ritiene di proprio gradimento, l'applicazione procede con la cancellazione di questi libri dal database in modo tale che iterazioni successive non ripresentino gli stessi libri, e inoltre vengono aggiornati i valori presenti nelle ComboBox dei generi e degli Autori, cosicché l'utente non rischi di selezionare valori che non hanno più corrispondenza nel database. L'idea alla base di quest'ultimo pulsante è che nel momento in cui si accetta la soluzione, è come se quei libri venissero letti, quindi vanno

tolti dal data-set perché questo rappresenta l'insieme dei titoli che l'utente non ha ancora letto.

## 6.2 Video dimostrativo

È possibile visionare un video dimostrativo del funzionamento dell'applicazione al seguente indirizzo: https://youtu.be/fVZBbvGGsYk.

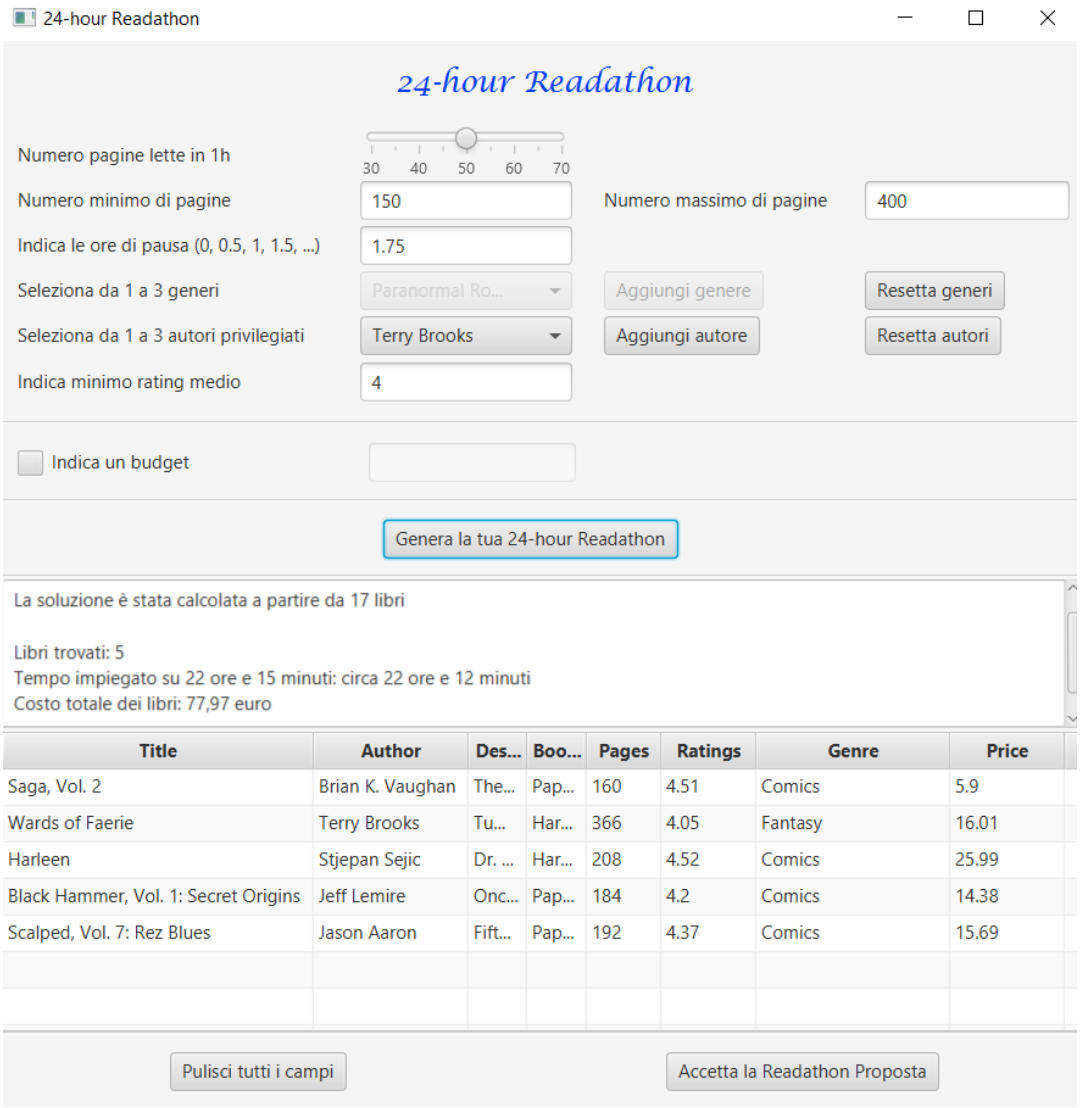
# Capitolo 7: Risultati Sperimentali

## 7.1 Esempio: No Budget vs Budget

In un primo esempio, immaginiamo che l'utente-lettore legga in media 50 pagine all'ora, voglia leggere libri con un numero di pagine compreso fra 150 e 400, che abbiano un rating superiore a 4 e che abbia a disposizione 22 ore e 15 minuti.

I generi scelti sono: Chick-Lit, Comics e Paranormal Romance. Gli autori privilegiati sono: Brian K. Vaughan e Terry Brooks.

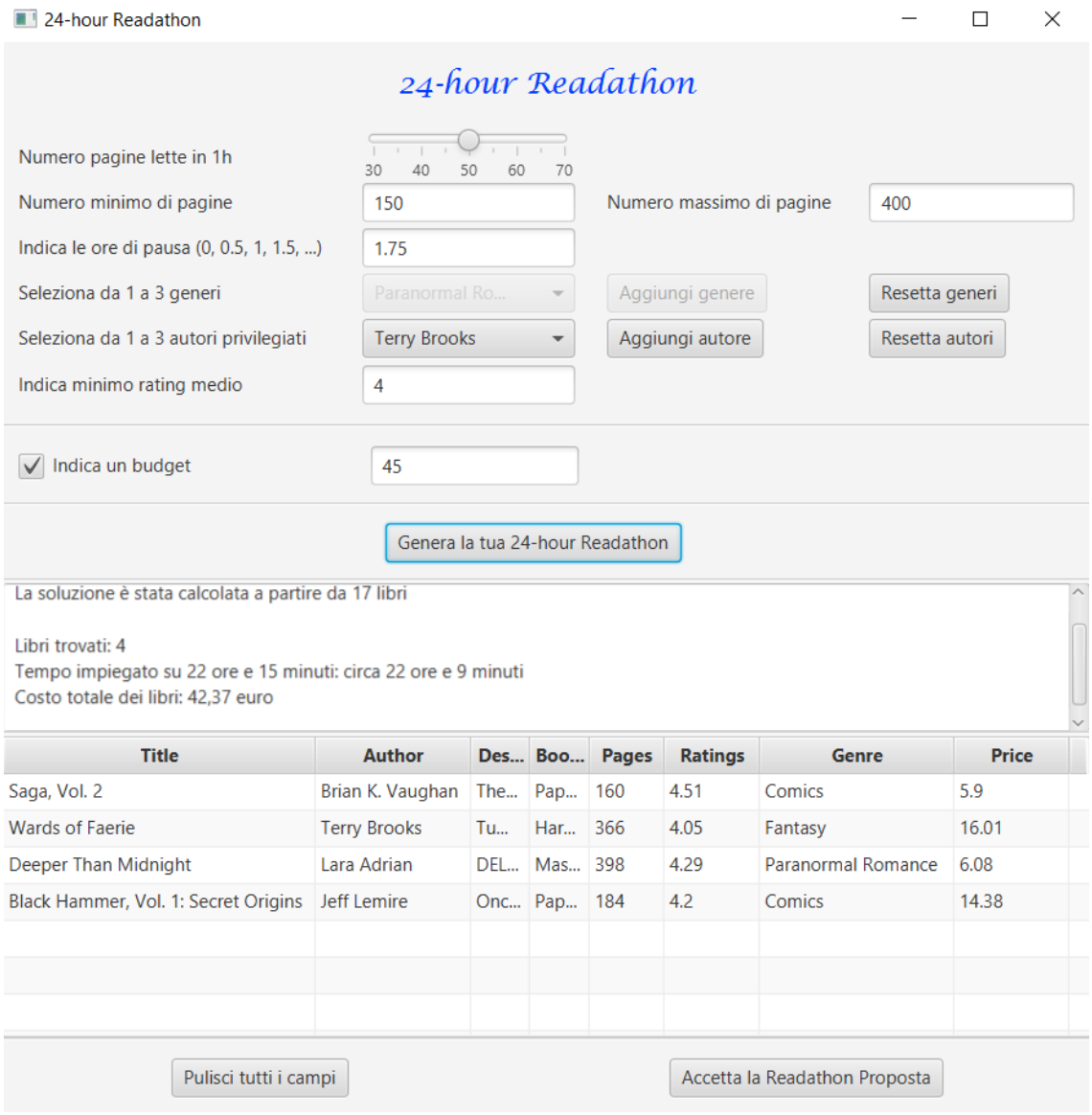
L'utente non inserisce alcun vincolo in termini di budget.




I vincoli inseriti permettono di partire da un insieme di 17 libri, da cui ne vengono ricavati 5 da leggere e completare nel tempo a disposizione.

L’applicazione impiega circa 0.016 secondi per produrre il risultato.

Se l’utente inserisse come ulteriore vincolo il fatto che l’insieme dei libri selezionati non debba costare più di 45 euro, i risultati sarebbero i seguenti:




Per rispettare il vincolo del budget, questa volta la soluzione proposta comprende solo 4 libri, riuscendo comunque a massimizzare il tempo impiegato nella lettura. Rispetto al caso precedente, l’introduzione di un nuovo vincolo, il budget, fa sì che la soluzione trovata sia leggermente meno efficacie della precedente, in quanto aumenta di 3 minuti il tempo inutilizzabile.

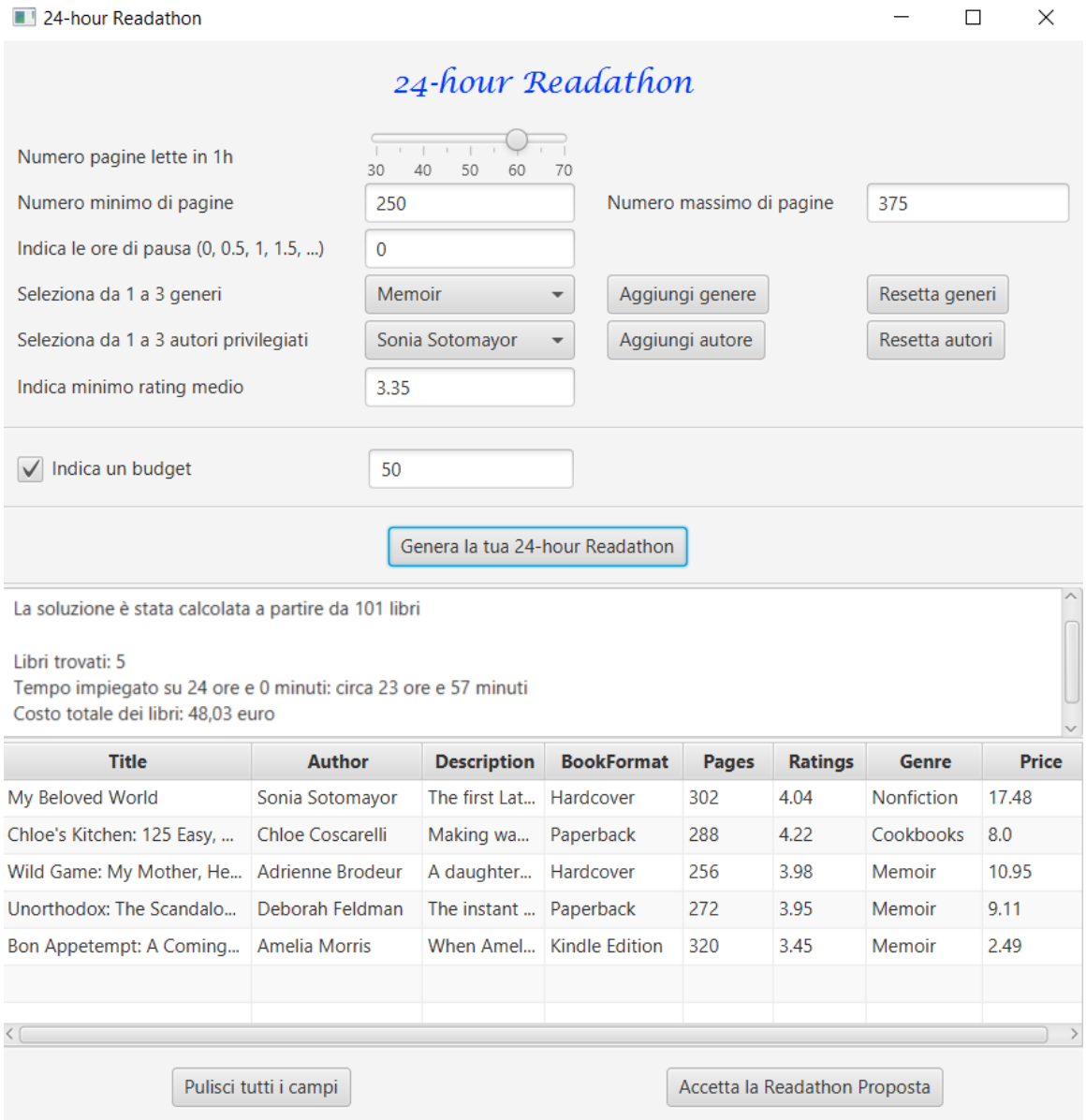
L’applicazione impiega circa 0.001 secondo per proporre una Readathon conforme ai parametri inseriti in input dall’utente.

## 7.2 Secondo Esempio

Immaginiamo, ora, che l'utente in questione sia in grado di leggere circa 60 pagine all'ora, che voglia leggere libri aventi un numero di pagine compreso fra 250 e 375 e aventi un rating superiore a 3.35. L'utente non prevede pause.

I generi selezionati sono Cookbooks e Memoir, mentre gli autori sono Anna Quindlen e Sonya Sotomayor.

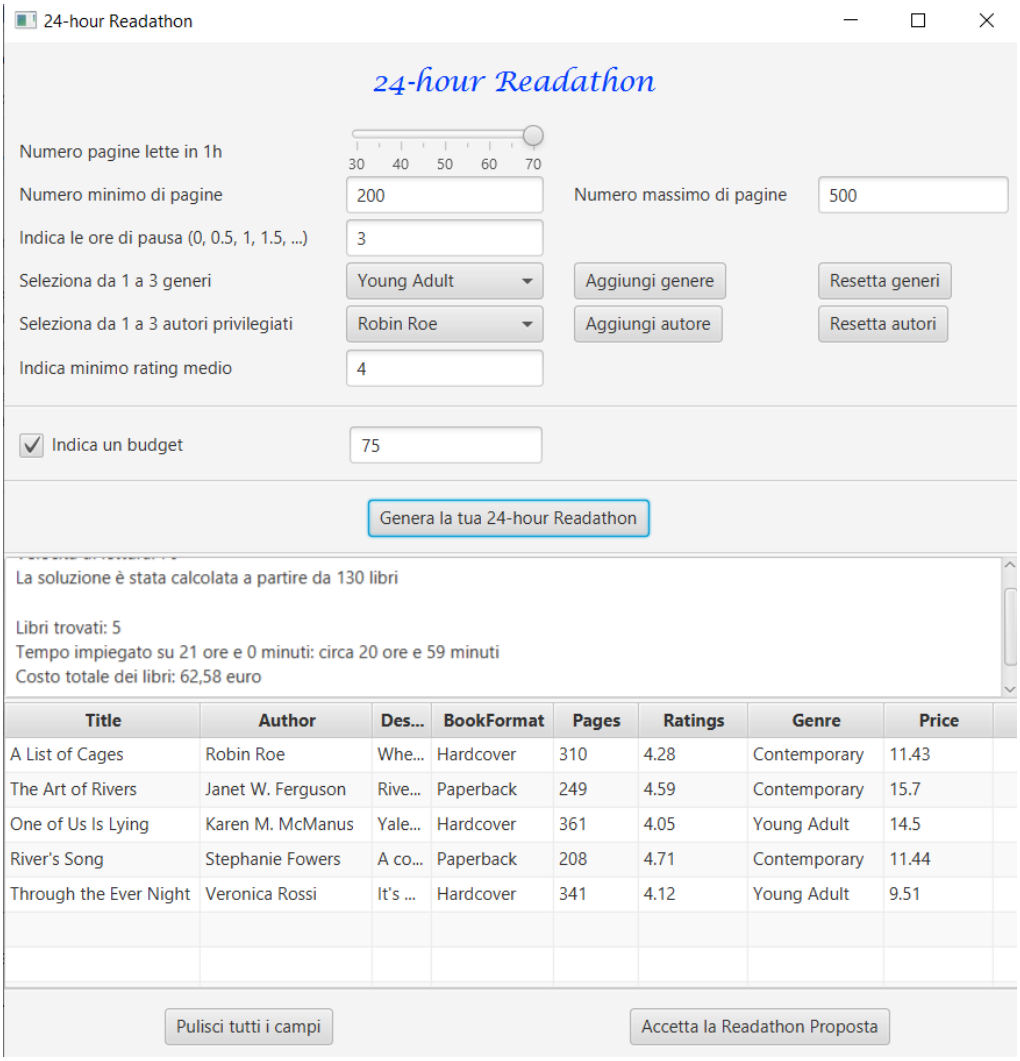
Il lettore specifica un budget di 50 euro.



In questo esempio, l'algoritmo ricorsivo parte da un insieme di 101 libri, per ricavarne uno di 5 titoli che soddisfino i vincoli inseriti dall'utente.
L'unico vincolo, per così dire, non soddisfatto riguarda gli autori. Non è infatti presente alcun libro di Anna Quindlen, sebbene l'autrice si occupi di uno dei generi selezionati: questo è dovuto al fatto che il suo unico libro presente nel database non soddisfa i requisiti in termini di numero di pagine minime.
L'applicazione ha impiegato circa 0.5 secondi per produrre questa proposta di Readathon.

## 7.3 Terzo Esempio

Immaginiamo, ora, che l'utente sia in grado di leggere 70 pagine all'ora e desideri leggere libri aventi un numero di pagine compreso fra 200 e 500, con un rating superiore a 4. Le ore di pausa specificate sono 3.
I generi selezionati sono Young Adult e Contemporary, mentre l'autore privilegiato è Robin Roe.
L'utente specifica che il proprio budget è di 75 euro.



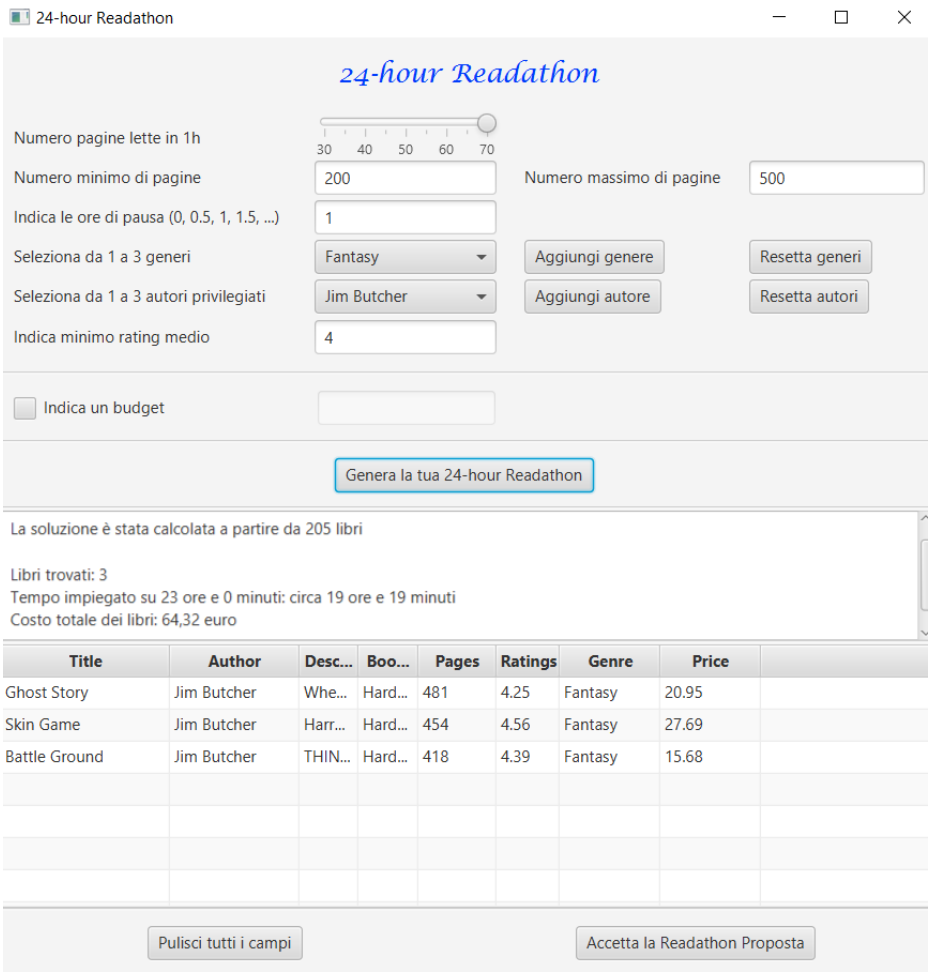

L'insieme dei libri su cui l'applicazione dovrà lavorare per estrarre i libri da leggere durante la Readathon è di 130 titoli: sebbene l'insieme sia decisamente numeroso, l'applicazione è in grado di fornire una soluzione in tempi relativamente brevi, circa 74 secondi.

## 7.4 Quarto Esempio

Per l'ultimo esempio, consideriamo un utente che sia in grado di leggere 70 pagine all'ora e sia interessato a leggere libri con un numero di pagine compreso fra 200 e 500 e con un rating medio superiore a 4. L'utente prevede un'ora di pausa durante le 24 ore della Readathon.

Il genere selezionato è Fantasy e l'autore privilegiato è Jim Butcher.

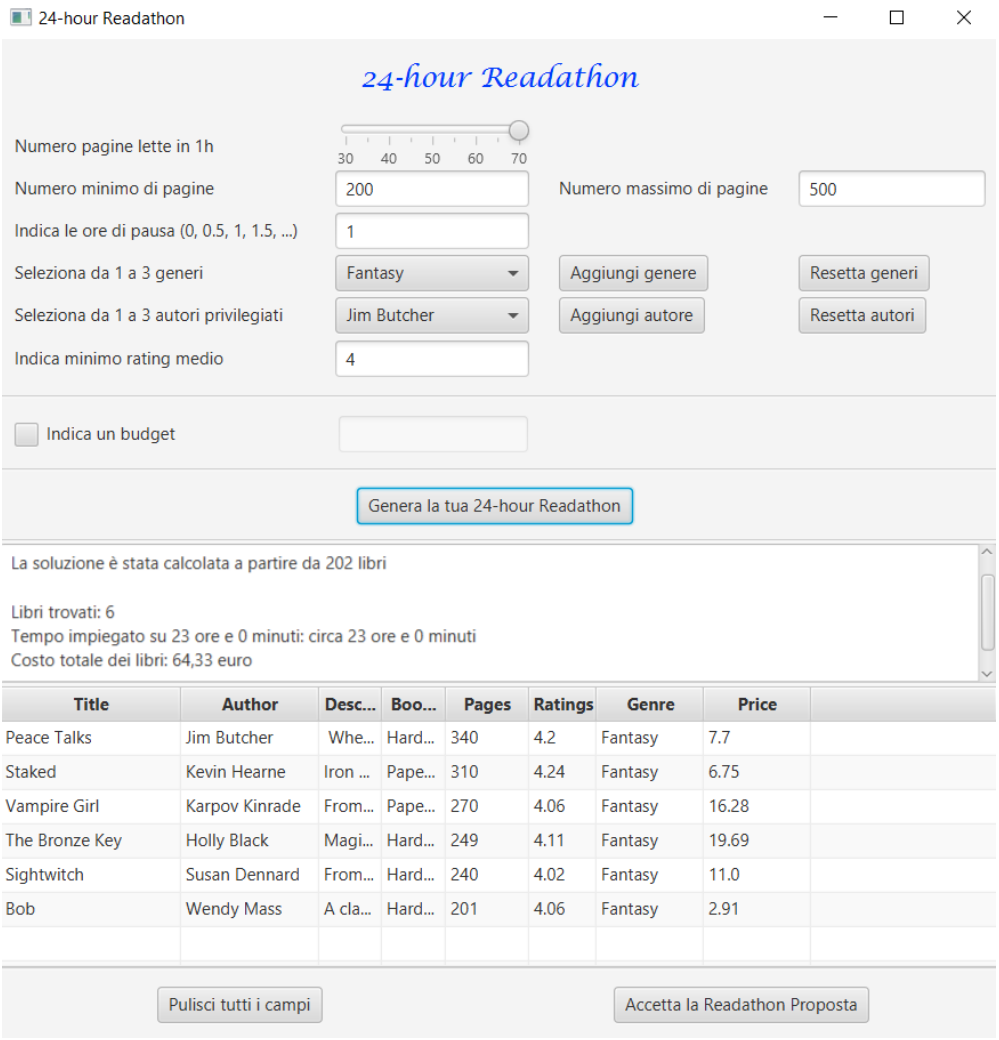
Non viene indicato nessun budget da rispettare.




In questo caso, l'applicazione si trova a dover determinare la migliore Readathon sulla base di 205 titoli: la soluzione ottimale si compone di 3 libri, che sono tutti esclusivamente dell'autore privilegiato (gli autori hanno priorità sui generi selezionati per l'applicazione), e viene computata in 0.0004 secondi circa.

Come si può notare, la soluzione proposta lascia scoperte quasi 4 ore del tempo che il lettore ha a disposizione: questo è dovuto al fatto che l'applicazione, per come è stata costruita, prima cerca di esaurire i libri filtrati sulla base degli autori selezionati, e poi, se c'è ancora tempo a disposizione, va a considerare i libri filtrati sulla base dei generi. In questo caso, l'autore Jim Butcher ha 4 libri che possono essere presi in considerazione per la Readathon, sulla base dei vincoli specificati dal lettore, tuttavia il tempo non è sufficiente a leggerli tutti e 4 (si andrebbe addirittura oltre le 24 ore), perciò la soluzione proposta comprende soltanti i libri dell'autore che possono essere completati in tempo, minimizzando per quanto possibile il tempo rimanente.

Se l'utente decidesse di accettare la Readathon proposta, una iterazione successiva del programma, eseguita con gli stessi parametri, darebbe una soluzione diversa in quanto non andrebbe più a considerare i tre libri sopra indicati.




Come si può notare dalla videata, dopo aver accettato la soluzione precedente, l'applicazione propone una nuova lista di libri contenente l'ultimo libro disponibile (secondo i vincoli) dell'autore privilegiato, e successivamente propone altri cinque libri Fantasy. In questo caso il tempo di computazione è parecchio elevato, superiore ai 5 minuti.

# Capitolo 8: Conclusioni Finali

## 8.1 Valutazione dei risultati

L'applicazione è in grado di fornire una soluzione ottimale al problema, sia qualora l'utente inserisca un budget sia in assenza di questo.

Può costituire un aiuto, uno spunto, per un lettore indeciso o che ha talmente tanti libri che vorrebbe leggere che non sa nemmeno da che parte incominciare per poter smaltire la sua TBR (To Be Read list).

La possibile Readathon viene fornita in tempi relativamente brevi se l'utente ha inserito dei vincoli che riescono a limitare il più possibile l'insieme dei libri su cui può lavorare l'algoritmo ricorsivo implementato, altrimenti l'applicazione potrebbe impiegare anche minuti, perché più è numeroso l'insieme di titoli preso in considerazione e maggiori saranno le combinazioni da valutare per trovare la migliore. Un fattore che influenza la velocità con cui la soluzione viene fornita è anche la velocità di lettura, che determina la numerosità dell'insieme dei libri che possono formare la Readathon.

In alcuni casi, aggiungere il budget come ulteriore vincolo può ridurre il tempo di computazione dell'applicazione.

## 8.2 Limiti dell'applicazione

Il principale limite dell'applicazione è il dataset utilizzato: la tabella books del file 24hour_readathon.sql contiene infatti circa 3600 libri diversi, il che implica che i vincoli inseriti possano potenzialmente determinare insiemi molto numerosi di titoli da passare all'algoritmo ricorsivo, e di conseguenza il tempo di computazione di una soluzione può aumentare considerevolmente (ad esempio, generi come Fantasy e Nonfiction contengono più di 200 libri ciascuno, quindi una soluzione che parta da questi due può impiegare parecchi minuti ad essere computata).

Per lo sviluppo di questo progetto è stato scelto un data-set così ampio al fine di avere una gamma di possibilità fra cui scegliere più grande e bilanciata, possibilmente, nei generi dei libri, tuttavia bisogna tener conto che nella realtà quotidiana quest'applicazione verrebbe tendenzialmente applicata a data-set decisamente più

piccoli. I lettori, infatti, è raro che abbiano così tanti libri da leggere in libreria o nella loro TBR (To Be Read list), di solito il numero di titoli che interessa loro leggere si aggira al massimo intorno a poche centinaia (100 o, in casi estremi, 200), se non anche di meno, pertanto applicare il progetto a un insieme decisamente più ristretto permetterebbe di partire da un numero più contenuto di libri da passare all'algoritmo ricorsivo e ottenere una soluzione in tempi brevi, sotto il minuto.

# Capitolo 9: Licenza