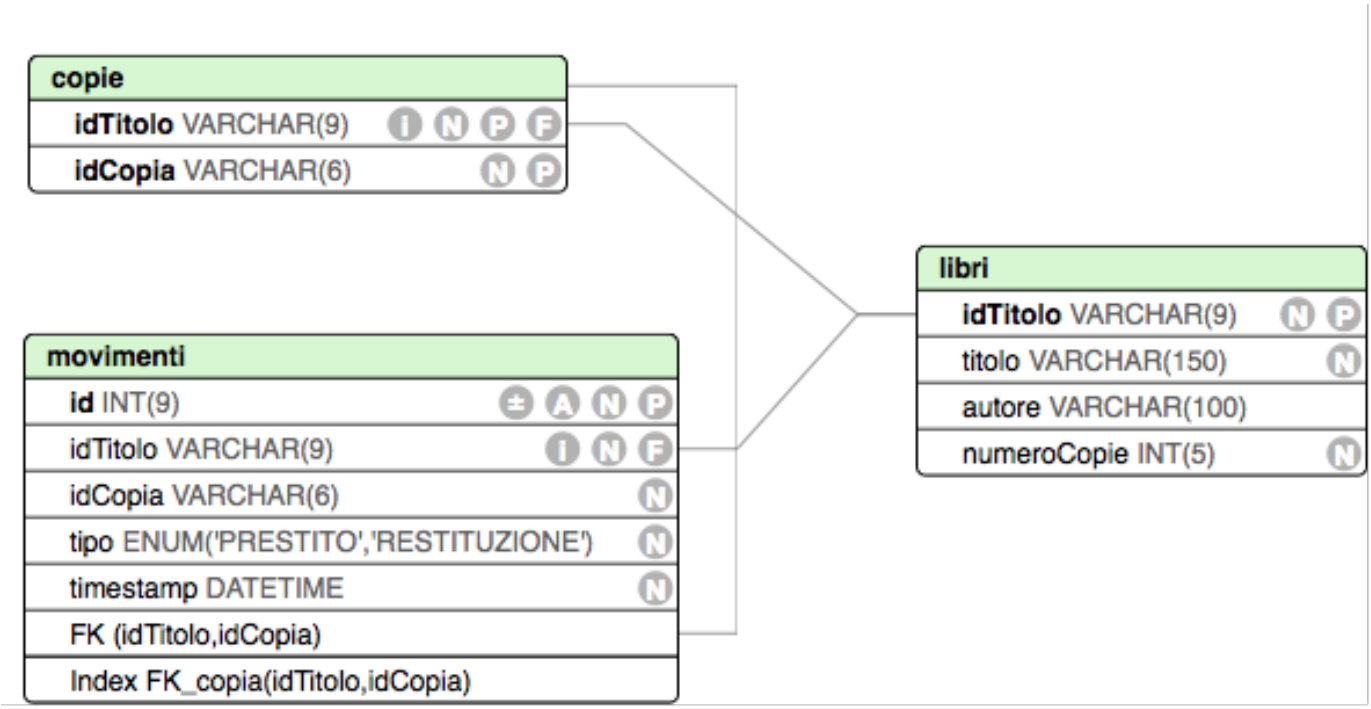
Simulazione dell'andamento di prestiti e restituzioni della Biblioteca di Ingegneria del Politecnico di Torino

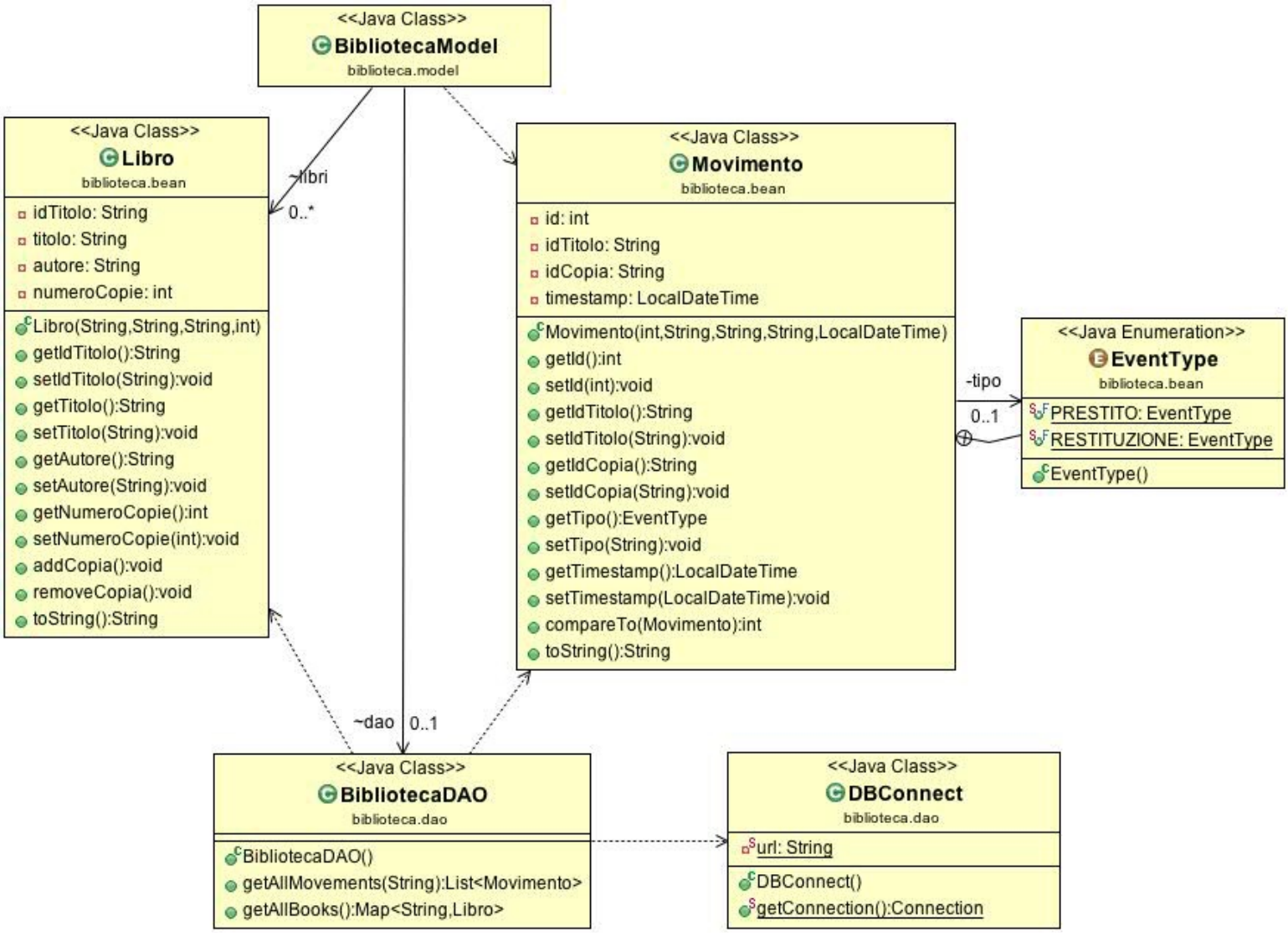
185810 PETROCCHI FRANCESCA

La mia tesi di Laurea Triennale in Ingegneria Gestionale si incentra sulla realizzazione di un'applicazione software in linguaggio Java che possa essere di supporto per la gestione degli acquisti della Biblioteca di Ingegneria del Politecnico di Torino.
Da un'analisi approfondita dello storico prestiti dell'anno 2015, emerge come alcuni volumi siano molto richiesti mentre altri vengano ceduti in prestito solo saltuariamente, presa coscienza di questo divario è possibile provvedere a un riordino consapevole dei nuovi volumi per l'anno accademico successivo.
L'andamento di prestiti e restituzioni è molto variabile e soggetto a sostanziali differenze al mutare di alcune condizioni che vanno, per quanto possibile, tenute presente nella stesura di un piano di riordino. A seconda del periodo dell'anno preso in considerazione, le richieste di alcuni titoli possono aumentare o diminuire notevolmente; in particolar modo le scelte degli utenti di una biblioteca universitaria sono fortemente soggette ai corsi in atto o all'avvicinarsi della sessione d'esami. La puntualità degli utenti nei tempi di restituzione è un altro fattore estremamente variabile e non quantificabile in senso deterministico, può però essere utile valutare quali siano gli effetti globali di un ingente ritardo nella consegna di un volume: a seconda che il titolo sia più o meno richiesto e in base al numero di copie di cui si dispone le conseguenze possono propagarsi sulle richieste di prestito successive in modo più o meno ingente.
Un'applicazione software che consenta di simulare gli ipotetici scenari e valutare complessivamente gli effetti di nuovi acquisti è di grande supporto per l'elaborazione di una corretta politica di gestione che miri a rendere proficui gli investimenti economici devoluti. In questo caso specifico un buon piano di riordino dovrebbe vedere ben bilanciati gli investimenti devoluti e la percentuale di richieste di prestito che la biblioteca riesce a soddisfare.
Usando algoritmi di simulazione, si può costruire un modello del sistema sotto esame e osservare come esso vari qualora vengano modificati alcuni parametri al fine di analizzare tutti gli scenari che si possono prefigurare a seguito di una particolare scelta aziendale. Un simulatore permette di carpire un'immagine globale degli effetti che anche solo un piccolo cambiamento può portare sul sistema aziendale nel suo complesso e di soppesare quindi la convenienza di una determinata scelta aziendale.

Per l'analisi ho preso in considerazione la lista dei movimenti di prestiti e restituzioni registrati dalla biblioteca per tutto l'anno 2015 e ho realizzato il database *biblioteca* costituito da tre tabelle che raccolgono lo storico dell'anno appena trascorso, l'elenco dei libri allora disponibili e delle rispettive copie.
La tabella *movimenti* contiene la lista delle 48560 transazioni effettuate nel corso dell'anno, ciascuna identificata dalla chiave primaria *id*. Gli attributi *idTitolo* e *idCopia* individuano rispettivamente il titolo e la specifica copia del libro in oggetto, *timestamp* riporta la data e l'ora in cui è stato salvato il record.
La tabella *libri* contiene codice identificativo, titolo, autore e numero di copie dei 5697 libri presenti nel database ed è referenziata dalla tabella *prestiti* attraverso la chiave esterna *idTitolo*. Ogni libro è univocamente identificato dalla chiave primaria *idTitolo* e l'attributo *autore* è l'unico per cui è ammesso valore nullo.
Le corrispondenze tra gli id dei libri e gli identificativi delle singole copie degli stessi sono memorizzate nella tabella *copie* che riporta le 8784 coppie *idTitolo* - *idCopia* e referenzia la tabella *libri* per mezzo della chiave esterna *idTitolo*.



L'applicazione JavaFX segue il pattern architetturale Model-View-Controller, l'interfaccia utente è definita mediante file XML usando il tool grafico Scene Builder. L'FXML Document *Biblioteca.fxml* contiene la descrizione dell'interfaccia grafica e la classe *BibliotecaController* intercetta e gestisce gli eventi innescati dall'utente.
La classe *BibliotecaModel* del package *biblioteca.model* viene inizializzata nel *Main* e fornisce i metodi per accedere ai dati ottenuti dal DAO (classe *BibliotecaDAO* del package *biblioteca.db*) che a sua volta gestisce l'interazione con il database.
All'avvio dell'applicazione, viene invocato nel *Main* il metodo *getAllBooks()* che riceve, dall'omonimo metodo della classe *BibliotecaDAO*, l'intero contenuto della tabella *libri* ottenuto con una query di tipo *SELECT* al database.
La classe *Libro* del package *biblioteca.bean* consente di creare un nuovo Libro passando come parametri al costruttore i valori di tutti gli attributi, accessibili grazie ai metodi Getter&Setter; i libri sono memorizzati nel Model in una TreeMap e identificati dalla chiave univoca *idTitolo*. Il *bean* contiene anche la classe *Movimento* grazie a cui è possibile creare un oggetto con gli stessi attributi della tabella *movimenti;* l'attributo *tipo* è vincolato ad assumere unicamente i valori *PRESTITO* o *RESTITUZIONE*, istanze dell'enum *EventType*.

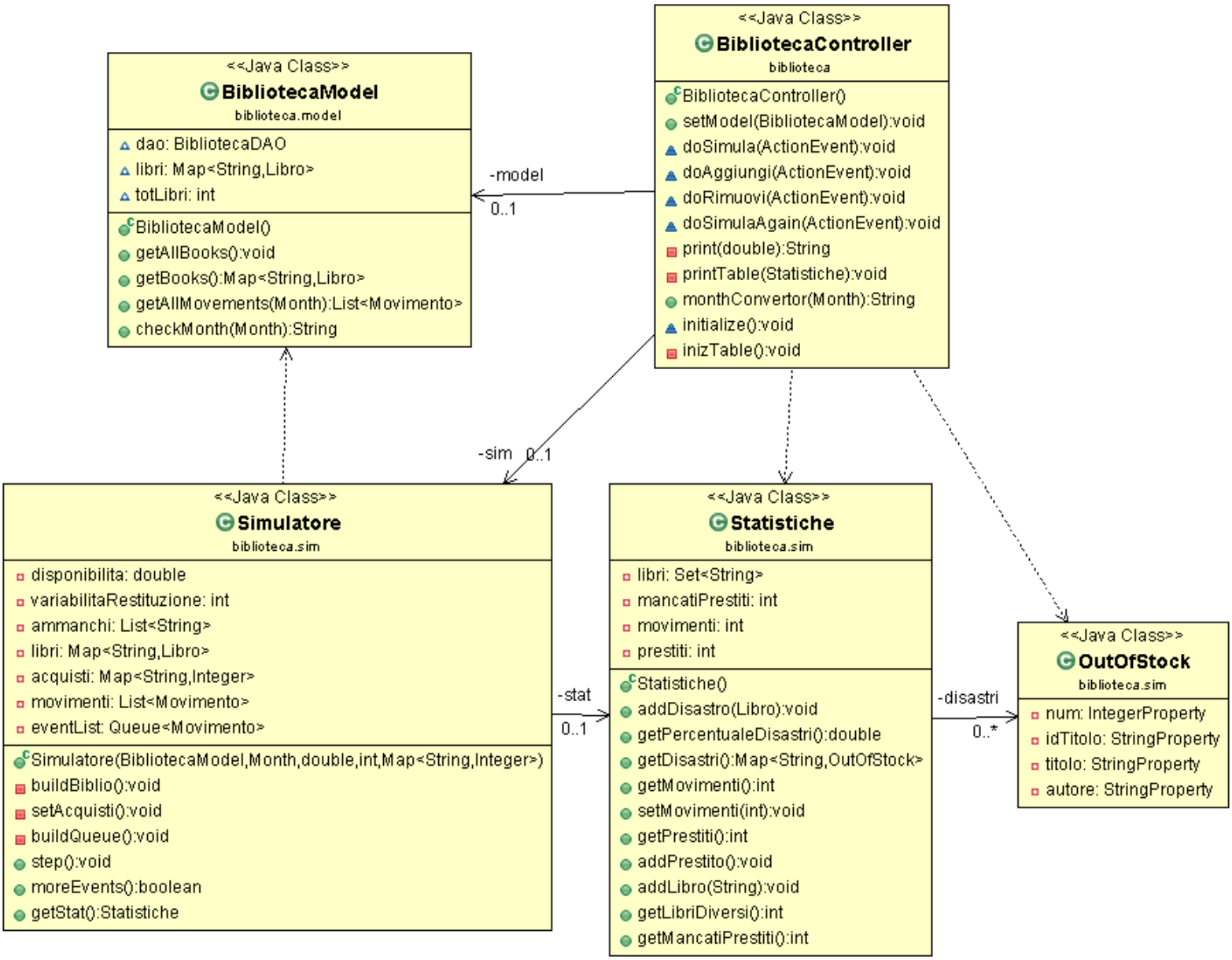


L’interfaccia grafica consente all’utente di scegliere i parametri di interesse, avviare la simulazione e eventualmente ripetere più volte l’analisi simulando nuovi acquisti. Dato l’elevato numero di transazioni presenti per tutto l’anno 2015, l’intervallo di tempo di simulazione è limitato a un singolo mese selezionabile dalla ComboBox *boxMese*. Questa scelta consente di concentrare l’attenzione sui mesi più critici e di soppesare il divario tra le richieste di alcuni titoli in periodi dell’anno diversi.
La disponibilità iniziale della biblioteca si imposta usando lo *sliderCapienza* e permette di analizzare la risposta del sistema, in termini di efficienza, qualora una determinata percentuale di libri sia già evasa in prestito. Se la maggior parte dei volumi sono fruibili le performance sono sempre mediamente buone, ma al diminuire della disponibilità i risultati sono sempre peggiori; eseguendo la simulazione più volte variando questo parametro è possibile valutare quale sia il limite oltre il quale la biblioteca non è più in grado di offrire il servizio previsto.
Infine, selezionando dalla ComboBox *boxRestituzione* un numero maggiore di 0, viene addizionato all’informazione temporale dei movimenti di restituzione un numero casuale di giorni compresi tra 0 e il valore scelto, così da aggiungere una variabilità stocastica che rispecchi la variabilità realmente insista nelle tempistiche di reso. In questo modo si otterrà un risultato sempre potenzialmente diverso pur ripetendo la simulazione più volte lasciando invariati gli altri parametri.
A simulazione terminata, vengono abilitati anche altri comandi per aggiungere o rimuovere copie di alcuni libri e ripetere l’operazione così da valutare i potenziali vantaggi che si otterrebbero da nuovi acquisti.

Al click del bottone *Avvia Simulazione* dell'interfaccia grafica, viene invocato il metodo *doSimula()* della classe *BibliotecaController* che, se tutti i parametri sono stati correttamente impostati, istanzia un nuovo oggetto *Simulatore*. La classe *Simulatore* del package *biblioteca.sim* è il core del simulatore: usa i parametri ricevuti in input dal Controller e definisce nuove strutture dati per processare la lista di eventi e fornire in output i risultati ottenuti.

Per svolgere le sue funzioni, il simulatore si serve delle copie di tutti i libri presenti in biblioteca, ottenute con una chiamata al metodo *getBooks()* del Model e registrate nella TreeMap *libri* e di un elenco di tutte le transazioni registrate dalla biblioteca nel mese scelto dall'utente. La LinkedList *movimenti* contiene le transazioni ottenute tramite chiamata al metodo *getAllMovements()* della classe *BibliotecaModel* che a sua volta riceve dall'omonimo metodo della classe *BibliotecaDAO* tutte le tuple della tabella *movimenti* corrispondenti a transazioni registrate nel mese passato come parametro.
Prima di eseguire la simulazione vengono impostate le condizioni iniziali del sistema e costruita la coda di eventi da processare con una chiamata ai metodi *builBiblio()* e *buildQueue()*, inoltre viene istanziato un oggetto della classe Statistiche che si occupa di tenere a registro gli out-of-stock e aggiornare i dati da fornire in output all'utente.

Il metodo *buildBiblio()* ricalcola la disponibilità della biblioteca in base alla percentuale indicata dall'utente e agli eventuali nuovi acquisti. Il Controller fornisce in input al Simulatore la TreeMap *acquisti* che contiene come valori il numero di copie aggiunte dall'utente per ogni libro e come chiavi gli identificativi del titolo. Se la mappa non è vuota, viene invocato il metodo *setAcquisti()* che esegue un ciclo sul keySet() per modificare il numero di copie dei rispettivi libri della biblioteca. Dopo aver aggiornato le disponibilità iniziali, è necessario ricalcolare la giacenza secondo la percentuale scelta dall'utente. A tal scopo viene creata una lista degli identificativi di tutti i libri presenti, eventualmente ripetuti nel caso in cui di uno stesso libro ci siano più copie, e riordinati in modo casuale. E' necessario eliminare dalla lista un certo numero di volumi in modo da avere infine una giacenza proporzionale alla percentuale selezionata dall'utente, la scelta deve essere puramente aleatoria per non deviare il risultato della simulazione. Si esegue con un ciclo avente un numero di iterazioni pari al numero di elementi da rimuovere; il metodo *nextInt()* invocato su un'istanza della classe *java.util.Random* consente di ottenere ad ogni ciclo un numero pseudocasuale intero compreso tra 0 e la dimensione della lista, ad ogni iterazione è rimosso l'elemento che si trova all'indice corrispondente a tale numero.



Il metodo *buildQueue()* costruisce una coda di oggetti di tipo *Prestito* aggiungendo gli elementi della lista *movimenti* alla PriorityQueue *eventList*. Se l'utente ha indicato una variabilità dei tempi di restituzione diversa da zero, per ogni oggetto di tipo *RESTITUZIONE* viene modificato il valore del *timestamp* aggiungendo un numero di giorni variabile tra 0 e al più il numero scelto dall'utente.

Fino a quando la coda degli eventi non sarà vuota, il metodo *step()* rimuove il primo elemento della PriorityQueue *eventList* e lo processa. Nel caso in cui si tratti di un prestito, se è ancora disponibile una copia del volume indicato la transazione è evasa correttamente ed è aggiornata la giacenza del libro, se invece le copie sono esaurite si registra un "out-of-stock" con una chiamata al metodo *addDisastro()* della classe *Statistiche* e l'identificativo del libro è aggiunto alla lista *ammanchi*. Nel caso in cui il

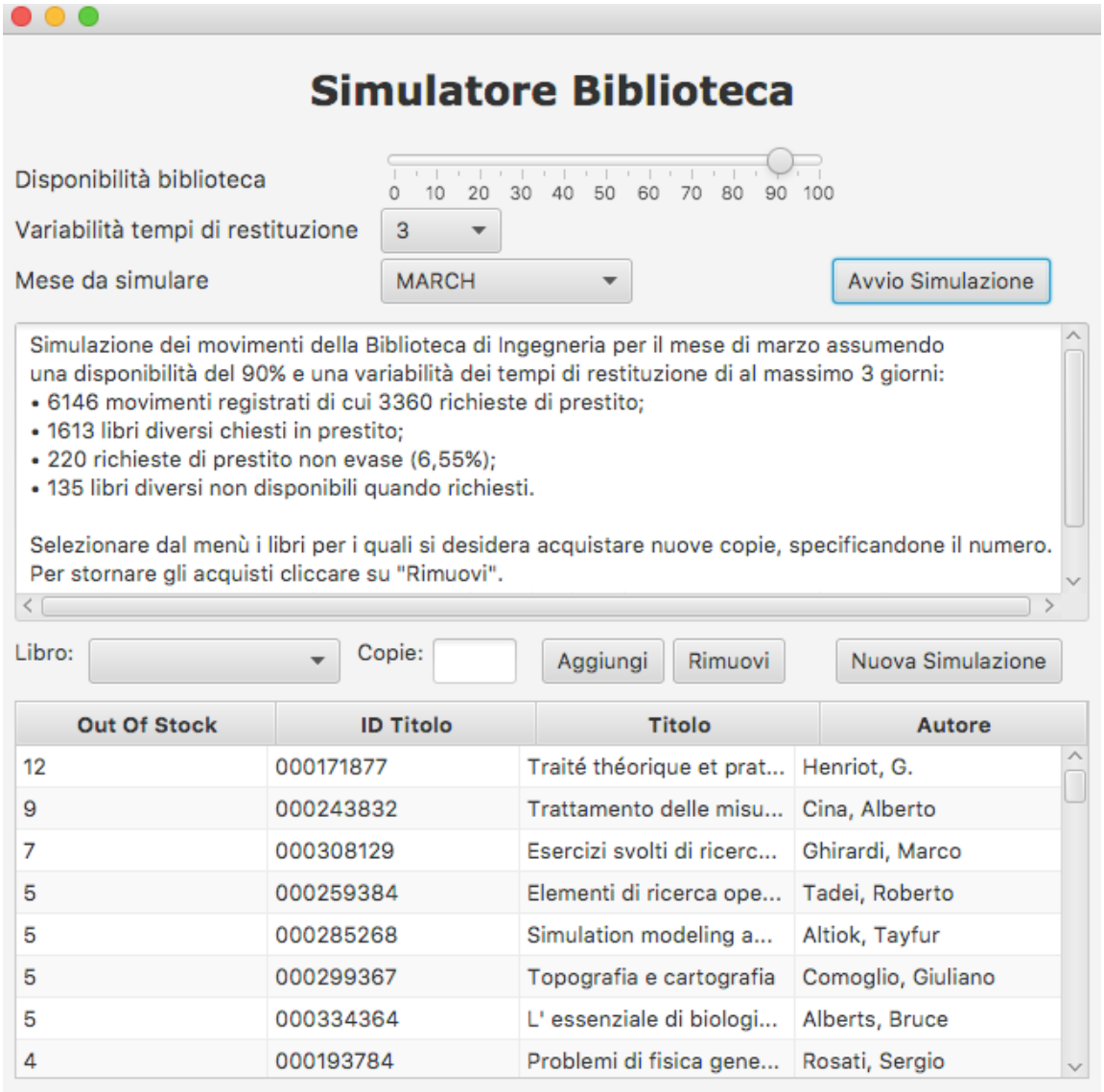
tipo di evento sia una restituzione, si controlla se la lista *ammanchi* contiene l'*idTitolo* del libro interessato: se è così significa che il prestito non è mai avvenuto poiché il volume richiesto non era disponibile e quindi è sufficiente rimuovere l'id da questa lista, altrimenti si registra la corretta riconsegna aggiungendo una copia tra quelle presenti per quel libro.

La classe *Statistiche* registra gli out-of-stock verificatisi aggiornando ogni volta la TreeMap *disastri* che contiene un elenco di oggetti di tipo *OutOfStock*. Ogni volta che si registra un primo ammanco per un determinato titolo, il metodo *addDisastro()* crea un nuovo oggetto *OutOfStock* passando al costruttore l'identificativo, il titolo e l'autore del libro. Per tutti gli out-of-stock successivi relativi allo stesso titolo sarà sufficiente aggiornarne la quantità.

A simulazione terminata, compaiono nella TextArea i risultati ottenuti e, nel caso in cui non sia stato possibile soddisfare tutte le richieste di prestito, l'elenco dei libri esauriti è disponibile nella TableView in ordine decrescente secondo il numero di ammanchi registrati per ogni titolo.
E' possibile simulare l'acquisto di nuove copie di questi libri e riprovare la simulazione per vedere se il relativo numero di out-of-stock subisca o meno variazioni. Il bottone *Nuova Simulazione* consente di avviare una nuova simulazione tenendo sempre in considerazione i nuovi acquisti; nel caso in cui l'aggiunta di nuove copie non abbia portato i risultati sperati, si possono sempre stornare gli acquisti selezionando l'id del libro e il numero di copie da eliminare.
Oltre al numero di mancati prestiti e alla percentuale sul totale dei movimenti, è utile anche conoscere il numero di transazioni registrate e il rapporto tra numero di prestiti e numero di restituzioni: conviene simulare l'andamento dei movimenti in periodi dell'anno differenti per valutare quali siano i mesi di maggiore utenza.

Da una prima analisi dei risultati ottenuti si osserva come i nuovi acquisti abbiano spesso un basso impatto sul sistema se si sceglie una percentuale di disponibilità iniziale molto bassa o una variabilità dei ritardi molto alta.
Le performance sono in genere nettamente migliori se c'è una buona puntualità nella restituzione, in caso contrario si rileva spesso inutile comprare nuove copie. Inoltre, avviando la simulazione più volte mantenendo invariati mese e disponibilità, si osservano ingenti cambiamenti se la variabilità dei ritardi è alta. Dato il forte impatto della variabilità dei tempi di reso, potrebbe essere utile introdurre una sanzione sulle restituzioni in ritardo per cercare di arginare il problema.
Mantenendo una bassa variabilità nelle restituzioni e un'alta disponibilità iniziale, gli unici periodi dell'anno potenzialmente critici sono quelli a inizio semestre (in questo caso i mesi di marzo e ottobre) durante i quali sono registrate il maggior numero di transazioni; all'aumentare del numero di richieste di prestiti sono sempre più frequenti gli out-of-stock.

Nei mesi in cui le richieste sono più contenute, riuscendo ad avere una buona giacenza iniziale e una discreta puntualità degli utenti, risultano funzionali nuovi acquisti dei titoli più richiesti, tuttavia l'efficienza del sistema aumenta notevolmente qualora sia garantito un buon livello di rimanenze mentre gli acquisti, se considerati singolarmente, portano miglioramenti minimi che diventano del tutto trascurabili qualora non siano garantite alta disponibilità e bassa variabilità dei ritardi.
Il software non consente di identificare quali siano i titoli che invece non vengono quasi mai richiesti dagli studenti, tuttavia sarebbe molto vantaggioso capire quali siano stati gli acquisti passati potenzialmente errati in modo da quantificare gli investimenti devoluti che non hanno avuto successo e cercare di evitare di ripeterli in futuro.



Video dimostrativo

https://youtu.be/pHLhgo2HYq8