ESAME DI FONDAMENTI DI INFORMATICA T-2 del 27/06/2011
Proff. E. Denti – G. Zannoni
Tempo a disposizione: 4 ore MAX

NB: il candidato troverà nell'archivio ZIP scaricato da Esamix anche il software "Start Kit"

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)

È richiesto di realizzare un'applicazione per la gestione del bilancio familiare, intesa come pratico strumento per registrare le spese e le entrate, onde poter produrre un bilancio mensile opportunamente articolato.

DESCRIZIONE DEL DOMINIO. Un ***bilancio familiare*** è definito come insieme di ***conti***, di diverso tipo (si veda oltre). Ogni conto è caratterizzato da un *nome* e costituito da un elenco (inizialmente vuoto) di ***movimenti*****,** ciascuno dei quali trasferisce un certo ammontare di denaro, in una certa data, con una certa *causale*.

I conti sono fondamentalmente di tre tipi: *conto corrente, carta di credito, liquidi*; poiché in famiglia si possono avere più carte di credito, più conti correnti e più portafogli con liquidi (tipicamente, uno per ogni persona del gruppo familiare), un bilancio familiare può contenere un numero arbitrario di conti di vario tipo.

Ogni movimento ha un proprio codice, dettaglia uno spostamento di denaro da un conto a un altro e perciò coinvolge sempre due conti: uno da cui preleva denaro, uno su cui accredita denaro. Per rappresentare le entrate dall'esterno e le spese verso l'esterno della famiglia, nei movimenti può figurare l'entità virtuale ***esterno*****:** chiaramente, le entrate risulteranno in movimenti da *esterno* verso un conto del bilancio, mentre le uscite (spese) saranno rappresentate come movimenti da qualche conto del bilancio familiare verso *esterno.*

Alcuni tipi di conto possono prevedere dei *tetti di spesa:* ad esempio, il conto di tipo *carta di credito* potrà prevedere un *plafond* (tetto di spesa massimo) mensile, mentre il conto di tipo *conto corrente* potrà prevedere un *massimo scoperto* (ossia un massimo saldo negativo autorizzato).

ESEMPI: l'accredito dello stipendio sul conto corrente *ccLuigi* è rappresentato in tale conto come movimento da *esterno* al conto stesso; una spesa con carta di credito *VisaLuigi* è rappresentata come movimento da tale conto a *esterno*; il rimborso mensile delle spese con tale carta di credito come movimento da *ccLuigi* a *VisaLuigi* per un importo pari al totale delle spese con carta effettuate nel mese (tasse incluse); un prelievo dal conto corrente come movimento dal conto *ccLuigi* al conto *portafoglioLuigi;* una spesa in contanti come movimento da *portafoglioLuigi* a *esterno*; un regalo in denaro a un altro membro della famiglia – ad esempio, la figlia Sara – come movimento da *portafoglioLuigi* a *borsaSara;* ecc.

A tal fine, l'applicazione deve mostrare all'utente una GUI con cui inserire i vari movimenti fra i conti: all'inizio della sessione si ricarica dal file binario Bilancio.dat lo stato salvato in precedenza, che verrà poi ri-salvato a fine sessione, premendo l'apposito pulsante SALVA. L'insieme dei conti dello specifico bilancio familiare è prefissato e immodificabile e viene preventivamente letto da un file di configurazione StrutturaBilancio.txt (descritto oltre).

La GUI mostra costantemente il saldo attuale dei vari conti nonché il totale delle entrate e delle spese.

Parte 1 (punti: 17)

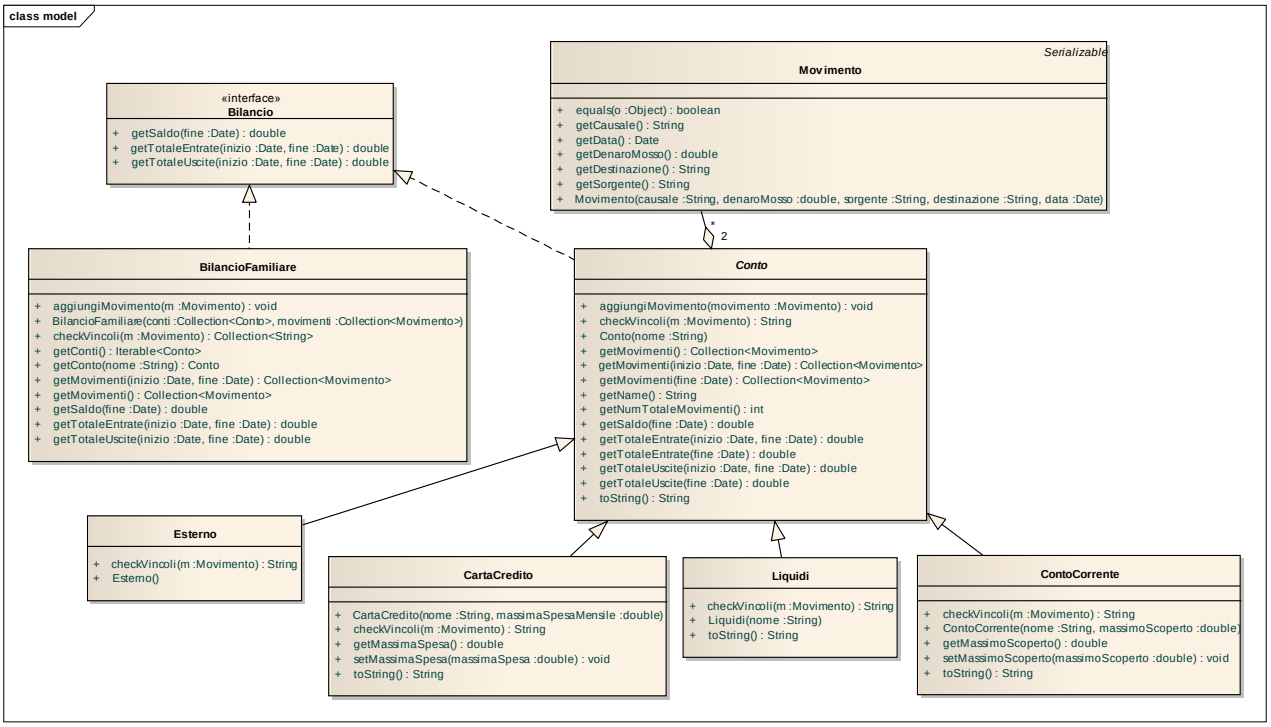
Dati (namespace money.model) *(punti: 12)*

Il modello dei dati relativo all'esame deve essere organizzato secondo il diagramma UML in figura.

SEMANTICA:

a) la classe astratta ***Conto*** (fornita nello Start Kit) rappresenta il generico conto, caratterizzato da un nome univoco e un insieme di movimenti con i relativi metodi accessor; essa incapsula tutta la logica di gestione spiegata nella "*Descrizione del Dominio"* e ha come unico metodo astratto *checkVincoli(Movimento)*, che delega alle sottoclassi la verifica dei vincoli di uno specifico tipo di movimento, nel modo specificato più oltre. I vari metodi *getSaldo, getMovimenti, getTotaleUscite* e *getTotaleEntrate* fanno tutti riferimento a un periodo specificato (data iniziale, data finale) e considerano tutti i tipi di movimento (sia quelli provenienti/diretti dall'esterno sia quelli intra-conti); per comodità, ne sono fornite anche versioni con un solo argomento (la data finale), col significato di "fino a" quella data. L'insieme restituito da *getMovimenti* è già ordinato per data.

b) la classe ***Movimento*** (fornita nello Start Kit) rappresenta il generico movimento, con le relative proprietà e i relativi metodi accessor; è serializzabile e comparabile, in base a un criterio di confronto per data.



c) Le quattro classi ***ContoCorrente***, ***Liquidi, Esterno*** e ***CartaCredito*** (fornite nello Start Kit) rappresentano i vari tipi di conto: ciascuna implementa il metodo *checkVincoli(Movimento)* in accordo alle proprie specifiche (rispettivamente: verifica del rispetto del *massimo scoperto* per *ContoCorrente*, verifica della disponibilità di sufficienti liquidi per *Liquidi*, nessuna verifica per *Esterno*, verifica del rispetto del tetto di spesa massimo mensile per *CartaDiCredito*). Per ipotesi, *checkVincoli(Movimento)* restituisce un messaggio d'errore (senza lanciare eccezione) in caso di violazione, mentre restituisce *null* in assenza di problemi; lancia invece una *IllegalArgumentException* se il movimento non riguarda questo conto, ovvero se non prevede questa carta di credito né come sorgente, né come destinazione.

d) la classe ***BilancioFamiliare*** **(da realizzare)** realizza l'interfaccia ***Bilancio*** (fornita) nel caso del bilancio familiare, modellato come un insieme arbitrario di conti (descritti nel file di configurazione) più un conto *Esterno* (che viceversa non viene letto dal file di configurazione, essendo presente e unico per definizione). Il costruttore riceve a tal fine due argomenti – una collezione di conti e una collezione di movimenti – già popolate (per il loro popolamento, si rimanda alla sezione sulla persistenza): per favorire il successivo rapido reperimento dei conti per nome, si suggerisce l'uso di un'opportuna mappa. Il metodo più importante è *inserisciMovimento,* che aggiunge simultaneamente un movimento nei due conti coinvolti, *previa verifica del rispetto dei vincoli*. *Per garantire la consistenza, l'aggiunta di un movimento in un conto dev'essere permessa solo se la data di tale movimento è successiva a quella dell'ultimo movimento già presente nel conto.* La classe espone inoltre adeguati metodi di accesso, tra cui *getTotaleEntrate, getTotaleUscite, getSaldo* (dovuti all'implementazione di Bilancio) e *getMovimenti* che calcolano e restituiscono rispettivamente il totale delle entrate (movimenti dall'esterno verso un conto qualsiasi), delle uscite (movimenti da un conto qualunque verso l'esterno), il saldo fra questi due valori, e l'insieme ordinato di tutti i movimenti di tutti i conti del bilancio, riferiti ad un specifico periodo di tempo (data iniziale, data finale). Il metodo *getMovimenti* compare anche senza parametri – in tal caso devono essere restituiti tutti i movimenti senza alcun filtro.

Nota 1: *getSaldo* prende in ingresso solo la data di fine periodo, intendendo con questo che si desidera il saldo di tutti i movimenti fino alla data specificata; pertanto, nei metodi che prevedono una data iniziale, occorrerà utilizzare una data "certamente precedente" ogni altra, come ad esempio *new Date(0)*

Nota 2: nell'implementazione di *getMovimenti* si tenga presente che ogni movimento compare sempre in due conti (sorgente, destinazione)...

Nota 3: per l'ordinamento dei movimenti nel metodo *getMovimenti*, si consideri l'utilizzo di un opportuno ***Comparator***.

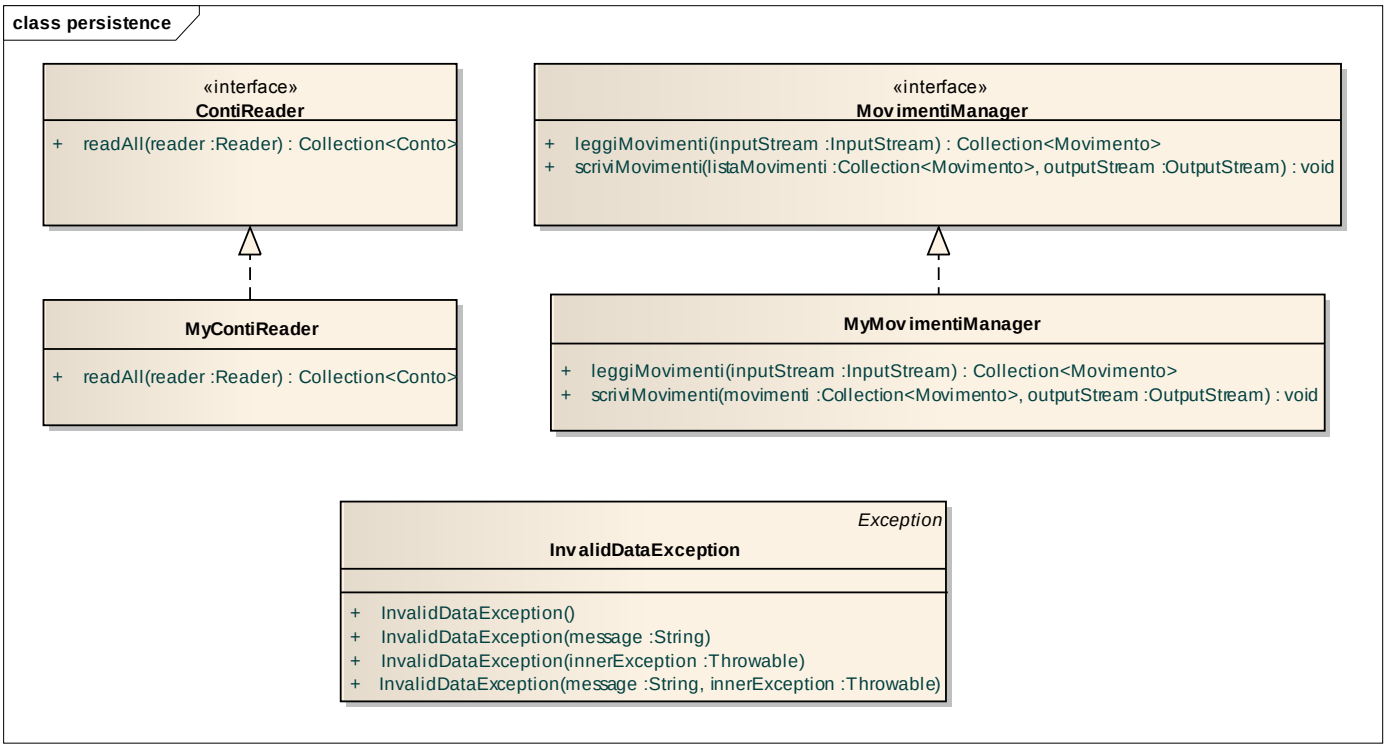
Lo Start Kit contiene anche i test (non commentati) per verificare il funzionamento delle classi da realizzare.

Persistenza (namespace money.persistence) *(punti 5)*

Il file di testo StrutturaBilancio.txt specifica l'elenco dei conti desiderati per il bilancio di una data famiglia: più esattamente, ogni riga specifica un conto, i cui dettagli sono separati uno dall'altro da **spazi o tabulazioni**. Ogni riga ha perciò la seguente struttura:

- la parola chiave `#define`
- Il nome del conto (univoco e privo di spazi)
- Il tipo del conto (`"contocorrente"`, `"cartadicredito"`, `"liquidi"`)
- eventuali parametri accessori e precisamente: il massimo scoperto nel caso del conto corrente, il tetto di spesa nel caso della carta di credito

Nel diagramma UML l'interfaccia ***ContiReader*** (fornita nello Start Kit) modella l'entità in grado di leggere una lista di



istanze di ***Conto.*** La classe ***MyContiReader*** (**da realizzare**) implementa tale interfaccia in modo da leggere conti espressi nel formato di cui sopra, rilanciando ***IOException*** in caso di problemi nella gestione del file, ovvero lanciando una ***InvalidDataException*** (fornita nello Start Kit) se il file non è formattato in modo adeguato.

Il file binario Bilancio.dat ospita l'elenco dei movimenti salvati e ha lo scopo di mantenere lo stato fra una sessione e l'altra: viene letto all'atto della creazione del ***Bilancio*** per ripristinare lo stato precedente e riscritto quando si preme il pulsante SALVA (tipicamente, al termine della sessione di lavoro).

L'interfaccia ***MovimentiManager*** modella entità in grado di leggere/salvare una lista di movimenti, ed è implementata dalla classe ***MyMovimentiManager*** **(da realizzare)**; i metodi rilanciano ***IOException*** in caso di problemi nella gestione del file, e lanciano ***InvalidDataException*** (fornita) in caso di problemi di serializzazione.

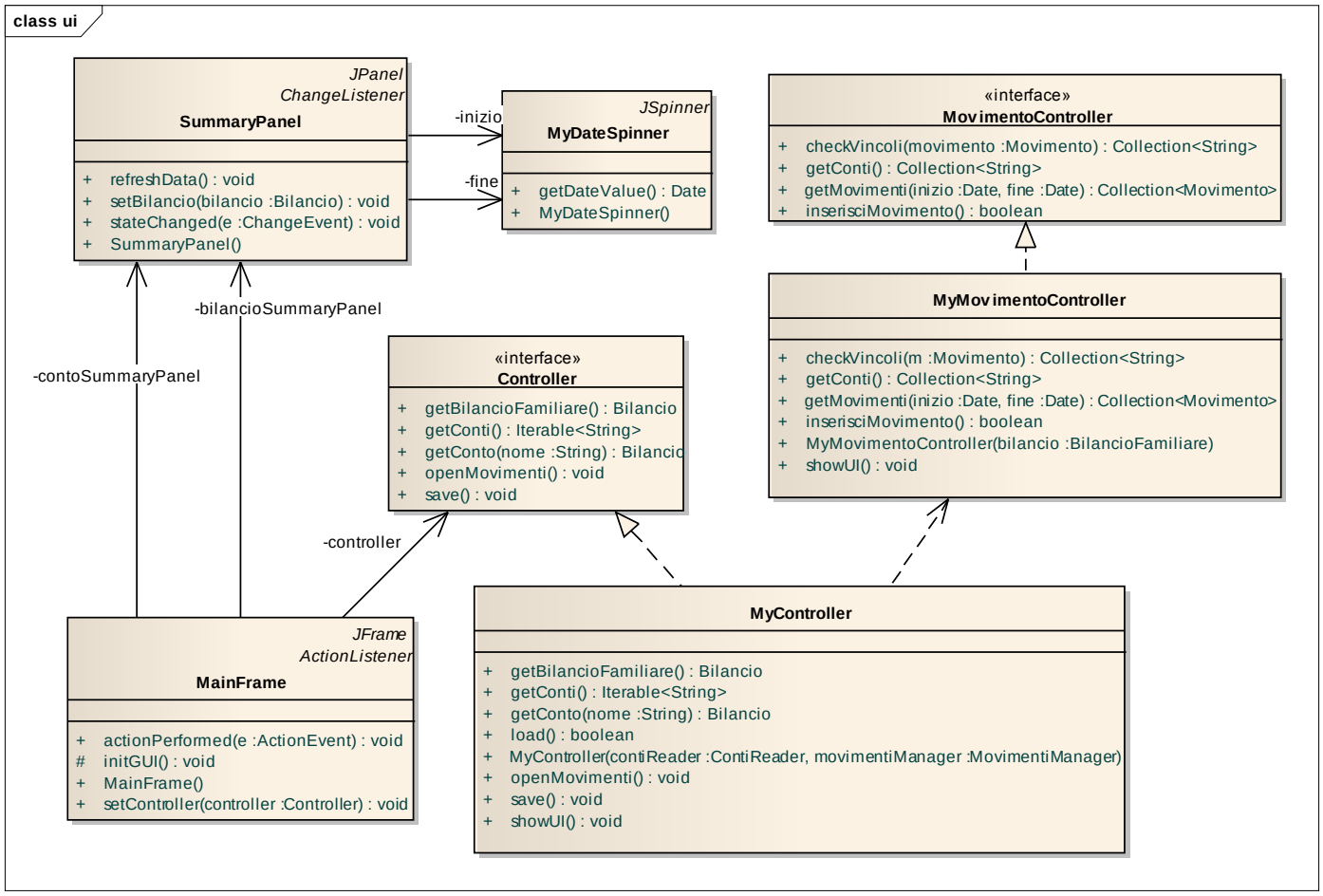
Lo Start Kit contiene anche i test (non commentati) relativi a MyContiReader e MyMovimentiManager.

Parte 2 (punti: 13)

Controller (namespace money.ui) *(punti 4)*

La classe ***MyController*** (**da realizzare**) implementa l'interfaccia **Controller** (fornita) e fornisce i servizi necessari al funzionamento della finestra principale dell'applicazione; in particolare, il metodo *getBilancioFamiliare* restituisce l'istanza di ***BilancioFamiliare*** (sotto forma dell'interfaccia implementata **Bilancio**), il metodo *getConti* restituisce l'elenco dei **nomi** dei conti disponibili, il metodo *getConto(String)* restituisce un oggetto ***Conto*** dato il nome del conto stesso, il metodo *load* carica i dati da file sfruttando i due oggetti **ContiReader** e **MovimentiManager** passati durante la costruzione di ***MyController***, il metodo *openMovimenti* crea un'istanza di ***MyMovimentoController*** e, mediante

l'invocazione di *showUI*, mostra l'interfaccia utente corrispondente; infine, il metodo



save salva su file l'elenco dei movimenti, mentre il metodo *showUI* visualizza l'interfaccia utente, ovvero crea un'istanza di ***MainFrame*** (v. dopo) e imposta ***MyController*** stesso come controller.

La classe ***MyMovimentoController*** (fornita) implementa l'interfaccia **MovimentoController** (pure fornita) e gestisce la finestra modale per la gestione dei movimenti sfruttando le utilities offerte dall'interfaccia **Bilancio**.

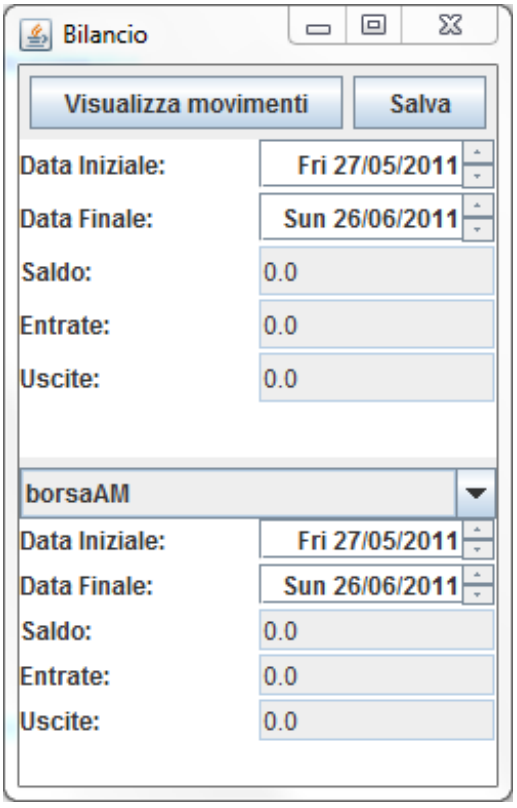


Figura 1

Interfaccia utente (namespace money.ui)

L'interfaccia utente deve essere simile (non necessariamente identica) agli esempi mostrati nelle figure che seguono.

Come già detto, all'avvio dell'applicazione vengono automaticamente ricaricati i dati di bilancio, che sono quindi immediatamente visualizzati. L'interfaccia permette di:

- avere costantemente sott'occhio il saldo attuale dei vari conti nonché il totale delle entrate e delle spese;
- visualizzare la lista **ordinata** dei movimenti compresi tra due specifiche date, nonché inserire un nuovo movimento, corredato di importo, conto di addebito e conto di accredito, data e causale: premendo il pulsante INSERISCI, il movimento viene controllato e, se corretto, immediatamente applicato; in caso di violazione di vincoli il movimento è invece rifiutato, con opportuna segnalazione di errore. Non è invece possibile eliminare o modificare un movimento già inserito.
- salvare la situazione corrente premendo il pulsante SALVA.

La Figura 1 mostra la finestra principale dell'applicazione, istanza della classe ***MainFrame*** (**da realizzare**). I due tasti "Visualizza Movimenti" e "Salva" consentono rispettivamente di aprire una nuova finestra contenente la lista dei movimenti (Figura 2) e di salvare lo stato corrente del bilancio. Sotto di due pulsanti, mostrare le caratteristiche di BilancioFamiliare mediante l'uso del componente ***SummaryPanel*** (fornito) (tale componente consente di mostrare i dati di un un'implementazione di Bilancio, date le date iniziale e finale, mostrando il saldo, le entrate e le uscite). Mediante una combo box deve essere possibile selezionare un conto di cui visualizzare le caratteristiche (riutilizzare ***SummaryPanel***).

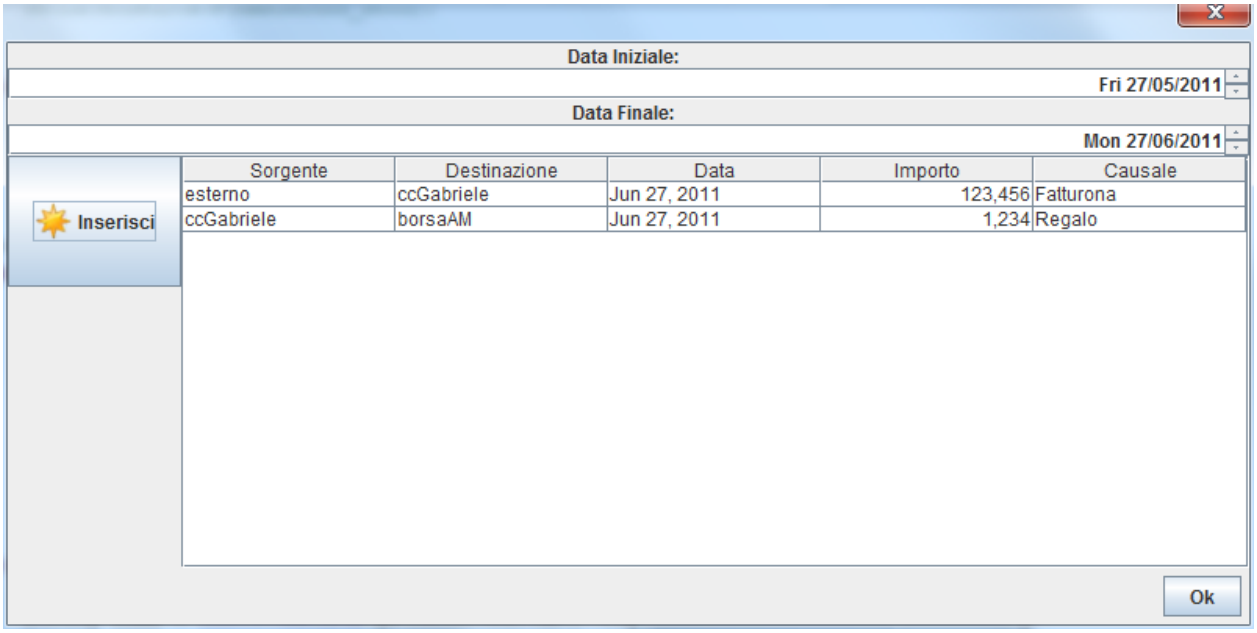


Figura 2

La Fig. 2 mostra la finestra modale dei movimenti, istanza dalla classe ***MyMovimentiDialog*** (fornita), che specializza opportunamente ***FormDialog*** (pure fornita). Di nuovo, due ***MyDataSpinner*** permettono di scegliere la data iniziale e finale del periodo: viene quindi prodotta e mostrata la lista dei movimenti, ordinata per data; la gestione di tale lista è delegata a un componente ***JBindableTable*** (fornito).

Figura 3

La pressione del bottone "Inserisci" causa l'apparizione di una nuova finestra per l'inserimento dei dati del movimento **(Fig. 3)**: se l'inserimento ha successo, la lista movimenti di Fig. 2 viene corrispondentemente aggiornata.

Inoltre, se il nuovo movimento ricade nel periodo di interesse del Bilancio e coinvolge movimenti da/verso esterno, quando si ritorna alla schermata principale occorre aggiornare anche il saldo globale con relative entrate e uscite.