

Relazione progetto Home-Manager

Marco Mancini

Federico Marinelli

01/03/2015

1 Analisi

- 1.1 Requisiti**
- 1.2 Problema**

2 Design

- 2.1 Architettura**
 - 2.1.1 Model**
 - 2.1.2 View**
 - 2.1.3 MVC**
- 2.2 Design dettagliato**
 - 2.2.1 Divisione package**
 - 2.2.2 Modello dettagliato**
 - 2.2.3 Pattern utilizzati**

3 Sviluppo

- 3.1 Testing automatizzato**
- 3.2 Divisione dei compiti e metodologia di lavoro**
- 3.3 Note di sviluppo**

4 Commenti finali

- 4.1 Conclusioni e lavori futuri**
- 4.2 Difficoltà incontrate e commenti per i docenti**

A Guida utente

Capitolo 1

Analisi

1.1 Requisiti

L'applicazione ha come intento quello di dare all'utente la possibilità di gestire, controllare e aggiornare le proprie spese e/o ricavi all'interno di una abitazione e personali. Inoltre un altro scopo è quello di fornire statistiche e promemoria dei dati inseriti.

Le funzionalità che l'applicazione dovrà gestire sono le seguenti:

- Gestione multiutente
- Gestione multiabitazione
- Divisione di abitazioni (in affitto, di proprietà)
- Aggiunta spese/rientri (di diversi tipi)
- Visualizzazione e modifica spese/rientri (in diverse modalità, totali, per mese, per settimana)
- Promemoria
- Calendario con date evidenziate
- Statistiche

1.2 Problema

HomeManager dovrà essere in grado di gestire più utenti e, per ogni utente, riuscire a gestire più abitazioni. Inoltre, ogni abitazione, dovrà tener conto di tutte le spese e i guadagni (con le relative informazioni) presenti. Analizzando spese e rientri possibili ci siamo accorti che sono molto simili tra di loro, effettivamente si differenziano solo per il tipo e per il nome ma, per il resto (importo, data, pagato o meno) sono uguali. Inoltre l'applicazione deve garantire la persistenza dei dati salvando i dati alla chiusura del programma e caricandoli all'apertura (il percorso predefinito è la cartella utente).

Figura 1.1: Ogni utente (interfaccia IUser) può gestire un portafoglio (IWallet) e più abitazioni (IHabitation) che, a loro volta, possono gestire un portafoglio.

Infine ogni portafoglio è composto spese e guadagni, che sono raggruppati sotto un'unica interfaccia IEarningAndExpense (essendo simili come detto prima, abbiamo deciso di creare un'unica interfaccia, dalle quali poi verranno fatte due implementazioni, Earning e Expense).

Capitolo 2

Design

2.1 Architettura

L'applicazione è stata sviluppata seguendo il pattern architetturale MVC (Model - View - Controller), per garantire la separazione tra le parti del progetto (il modello, la vista e i controller). L'utilizzo di questo pattern consente una facile manutenzione dell'applicativo e la migliore organizzazione possibile per una applicazione GUI.

2.1.1 Model

Figura 2.1: In questo UML viene rappresentato un po' più in profondo (rispetto alla fig. 1.1) il modello. Abbiamo una interfaccia IModel che è utilizzata per la gestione multiutente, la sua implementazione conterrà tutte le operazioni necessarie per interagire con gli utenti. Poi abbiamo l'utente (rappresentato dall'interfaccia IUser), che, oltre ai soliti parametri (nome, cognome, ecc), sarà composto da N abitazioni e da un Wallet. Anche l'abitazione avrà un Wallet a disposizione. Il wallet implementerà un'interfaccia ITransitionsWallet) che contiene le operazioni per aggiungere o eliminare una transizione da esso. Inoltre il wallet conterrà sia spese (expense) che guadagni (earning) rappresentate sotto un'unica interfaccia IEarningAndExpense.

2.1.2 View

Siccome le viste sono parecchie, abbiamo pensato di strutturarle attraverso una vista principale (Mainframe) che implementa un'interfaccia (IMainFrame), la quale ha un pannello superiore contenente un pulsante di logout che permette all'utente di uscire e tornare alla prima vista (il pannello di login), naturalmente questo bottone sarà abilitato o meno a seconda della vista in cui ci si trova, inoltre c'è un pannello centrale dinamico che cambia a seconda della scelta dell'utente (attraverso il metodo setCenterPanel dell'interfaccia IMainFrame). Inoltre ogni pannello estende una classe astratta AbstractMainPanel, la quale serve per dare un'immagine di sfondo uguale a tutti i pannelli (per qualche motivo però l'immagine non v'è).

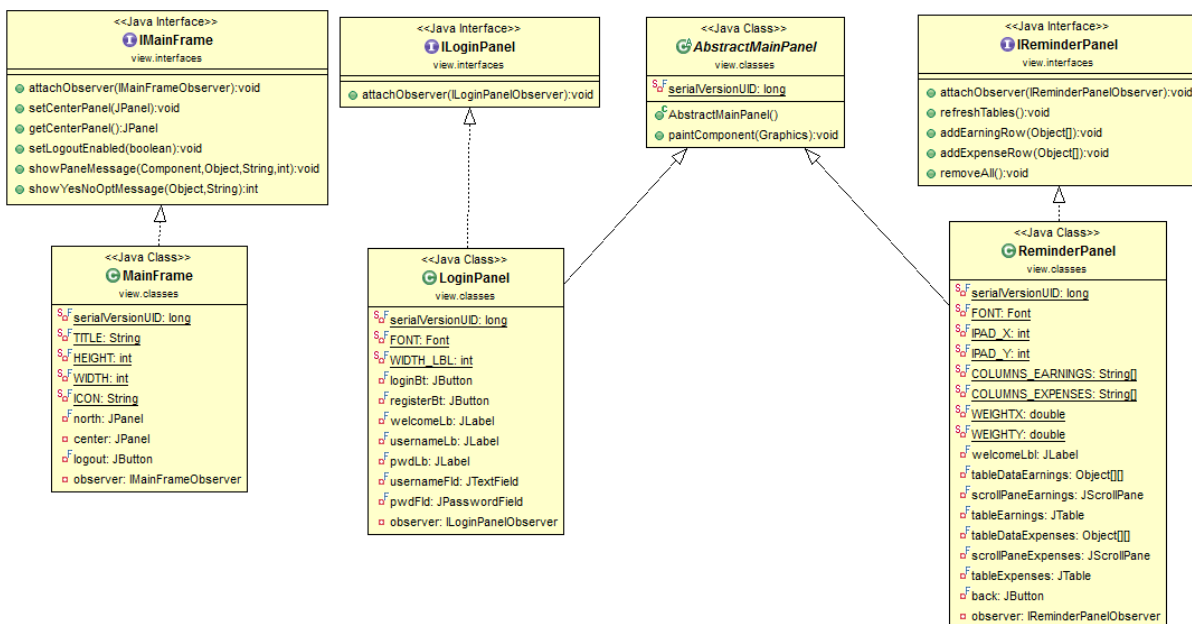


Figura 2.2: Come si può notare dall'UML, l'interfaccia IMainFrame oltre ai metodi per settare e restituire il pannello centrale, ha dei metodi per visualizzare messaggi, questo perché se il controller chiamasse direttamente i metodi presenti all'interno di JOptionPane per mostrare finestre con messaggi, la divisione MVC non sarebbe fatta correttamente. In questo UML sono mostrati, il pannello di login e quello per i promemoria, come si può vedere, entrambi implementano una propria interfaccia ed entrambi estendono un pannello astratto.

2.1.3 MVC

La realizzazione MVC e' stata realizzata nel seguente modo:

- Una classe Model che implementa l'interfaccia IModel, la quale contiene tutti i dati e le operazioni necessarie allo svolgimento delle funzionalità dell'applicazione. Essa implementa anche l'interfaccia Serializable, che ci permette di salvare la classe su file.
- Ogni pannello implementerà una propria interfaccia e conterrà una nested interface la quale rappresenta l'interfaccia per l'osservatore di quel pannello. Ogni interfaccia di un pannello avrà un metodo addObserver che prenderà in ingresso un osservatore (la nested interface sopra citata) e la assegnerà all'osservatore del pannello.
- I controller non saranno altro che le implementazioni delle nested interface presenti all'interno di ogni pannello. Tutte le interfacce dei controller estendono un'interfaccia IObserver, che contiene il metodo back(), che non fa altro che tornare al pannello precedente.

Esempio MVC per la registrazione:

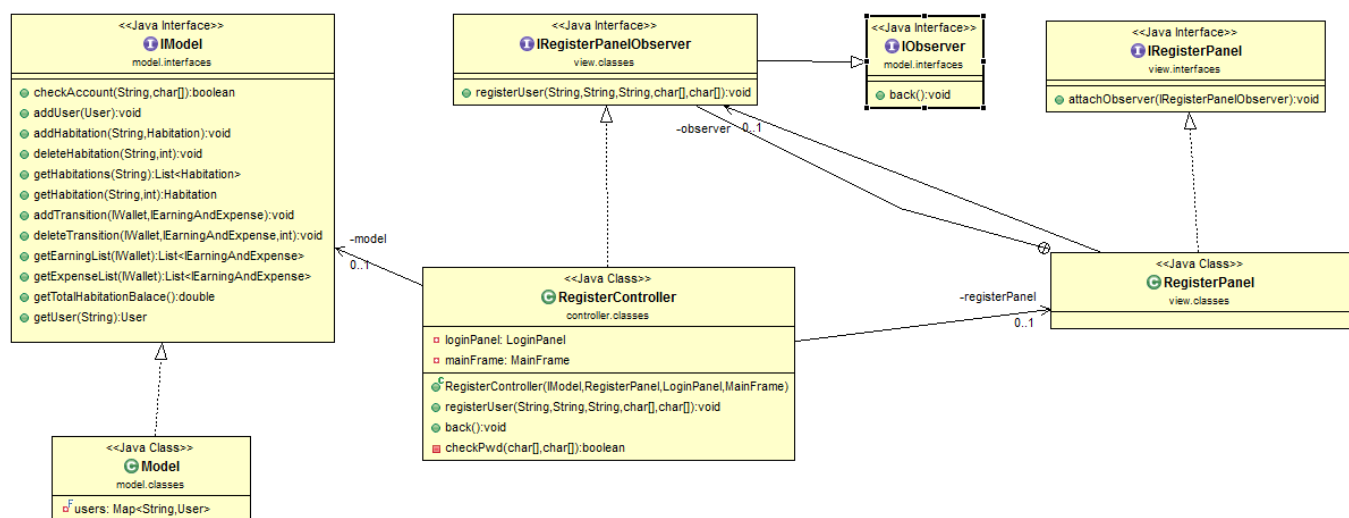


Figura 2.3: Questo UML mostra come è strutturato l'MVC e in particolare mostra l'operazione di registrazione. Il pannello RegisterPanel implementa l'interfaccia IRegisterPanel che contiene il metodo addObserver(IRegisterPanelObserver). L'interfaccia che accetta questo metodo non è altro che l'interfaccia che il controller (RegisterController) dovrà implementare. Questa interfaccia contiene un metodo registerUser che si occupa di registrare l'utente. La classe RegisterController è composta da un IModel e da un RegisterPanel e non dovrà fare altro che richiamare il corretto metodo (addUser), presente nel modello, e passargli i dati presi dal pannello RegisterPanel. Sarà poi il metodo addUser a salvare correttamente i dati all'interno del modello o a lanciare eccezioni (come per esempio l'eccezione riguardante l'esistenza dell'utente), che il controller

intercetterà e, tramite metodi presenti all'interno del mainframe, farà visualizzare all'utente.

2.2 Design Dettagliato

2.2.1 Divisione Package

Per quanto riguarda la divisione in package, abbiamo deciso di effettuarla in questo modo:

- controller.classes : Contiene tutte le implementazioni dei controller relativi ai pannelli (ricordiamo che ogni pannello contiene un'interfaccia per il relativo observer che, in questo caso, è il controller stesso).
- exceptions: Contiene tutte le eccezioni aggiunte da noi, come per esempio la ExistsUserException, che viene lanciata se si vuole registrare un utente il cui username è già presente.
- main: Contiene solo ed unicamente la classe main, la quale carica i dati (se presenti, altrimenti crea un file vuoto con salvata la classe Model), e lancia l'applicazione, mostrando il pannello di login.
- model.interfaces: Contiene tutte le interfacce delle classi relative al modello
- model.classes: Contiene tutte le implementazioni delle interfacce del package model.interfaces, cioè tutte le classi del Modello.
- test: contiene tutti i test.
- view.interfaces: Contiene tutte le interfacce delle classi relative alla vista.
- view.classes: Contiene tutte le implementazioni delle interfacce del package view.interfaces, in parole povere tutte le viste.

2.2.2 Modello dettagliato

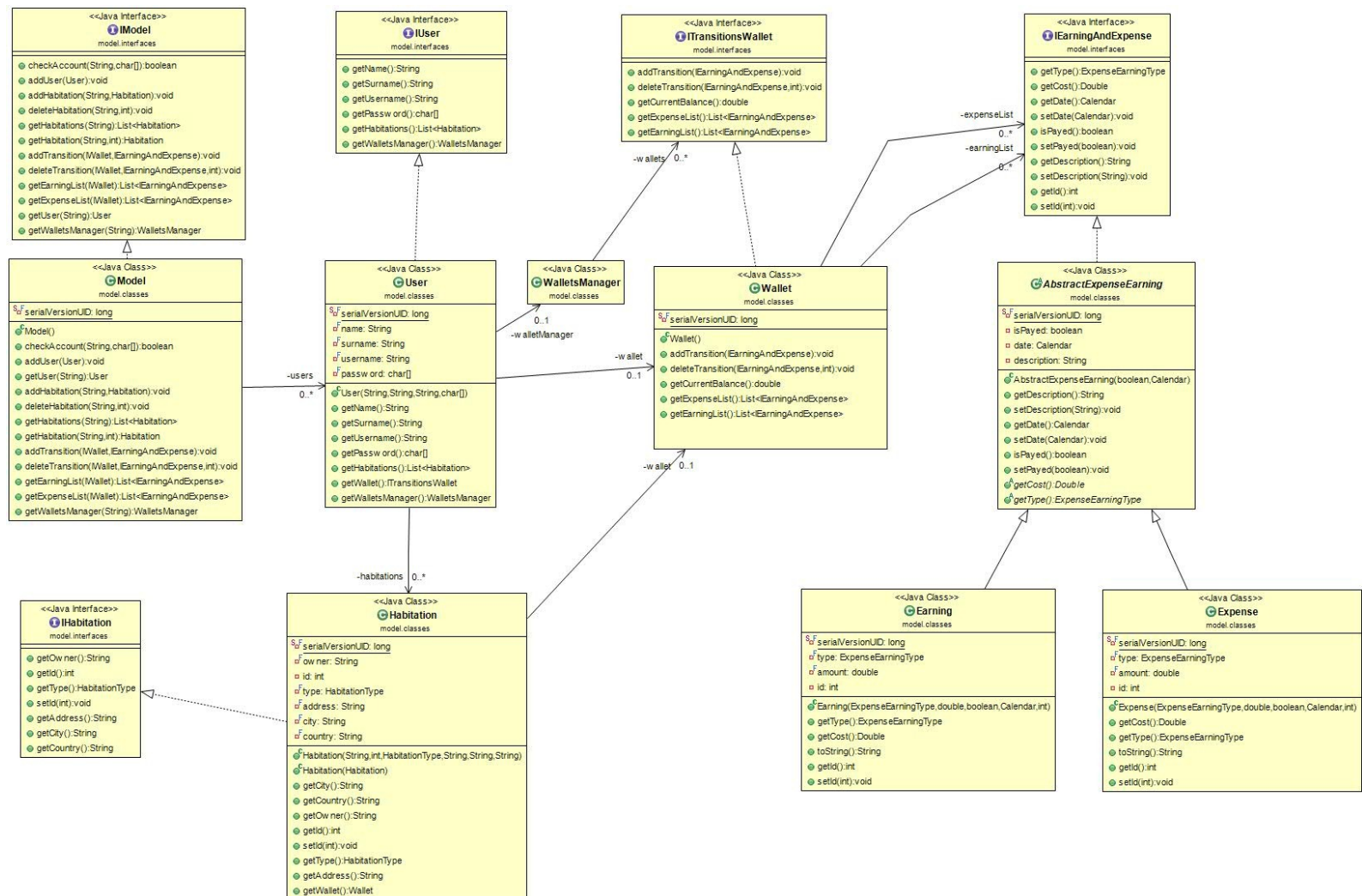


Figura 2.4: Questa figura mostra il modello dettagliato dell'applicazione

Descrizione modello dettagliato (tutte le citazioni dello schema UML riguardano la figura 2.4):

- **Model:** Implementa l'interfaccia IModel, questa classe contiene tutte le operazioni necessarie all'applicazione per eseguire le proprie funzionalità, come per esempio il metodo addHabitation, che prende in ingresso l'username dell'utente e una abitazione, e aggiunge quest'ultima alla lista di abitazioni dell'utente. Dallo schema UML si può notare che questa classe è composta da N utenti, infatti essa contiene una mappa che associa ad ogni user il proprio username.
- **User:** Implementa l'interfaccia IUser, questa classe modella un generico utente che può effettuare il login dell'applicazione e usufruire dei servizi di quest'ultima. Come si può notare dallo schema UML, ogni utente può possedere più abitazioni, possiede un proprio Wallet e anche un WalletManager.

- Habitation: Implementa l'interfaccia IHabitation, essa modella una generica abitazione (che, oltre ai generici campi, ha anche un tipo, modellato dalla Enum HabitationType). Ogni abitazione ha un proprio Wallet per gestire le spese e i rientri all'interno di essa.
- Wallet: Implementa l'interfaccia ITransitionWallet. Questa classe contiene due liste, una di spese(Expense) e una di guadagni(Earning), e contiene metodi per la gestione di queste due liste, come per esempio l'aggiunta di una spesa/guadagno o l'eliminazione.
- Earning/Expense : Estendono entrambe la classe astratta AbstractExpenseEarning la quale, a sua volta, implementa l'interfaccia IEarningExpense. Rappresentano un generico guadagno/spesa. Il tipo di guadagno/spesa è gestito tramite delle enum (EarningType,EarningUserType / ExpenseType,ExpenseUserType).
- WalletsManager: Questa classe è utilizzata per la gestione di tutti i Wallet che un utente possiede (il proprio e uno per ogni abitazione). Contiene tutti i metodi necessari a ricavare statistiche generali per utente.

Per garantire la persistenza dei dati , tutte le classi sopracitate implementano l'interfaccia Serializable.

Approfondimento gestione delle spese e dei rientri:

Per una migliore gestione e riusabilità del codice è stata creata un'interfaccia IWallet implementata sia dalla classe Habitation sia dalla classe User. IWallet ha semplicemente il metodo getWallet che mi restituisce il "portafoglio" di chi implementa tale classe. Grazie a questa implementazione posso accedere al "portafoglio" di una classe che implementa IWallet, attraverso i metodi getEarningList e getExpenseList della classe Model.

2.2.3 Pattern Utilizzati

Pattern Observer

Descrizione applicazione pattern observer:

Come già detto ogni pannello implementa una propria interfaccia, ogni interfaccia possiede un metodo (attachObserver) che prende come parametro l'interfaccia dell'observer e non fa altro che assegnarla al campo observer presente dentro ogni pannello. Abbiamo deciso di inserire l'interfaccia di ogni observer all'interno di ogni pannello (come nested interface) e la relativa implementazione non sarà altro che il controller (presente all'interno del package controller.classes). Saranno poi i vari listeners ad assegnare le varie operazioni degli observer agli eventi della GUI, oppure saranno i controller ad inviare alla GUI informazioni da gestire.

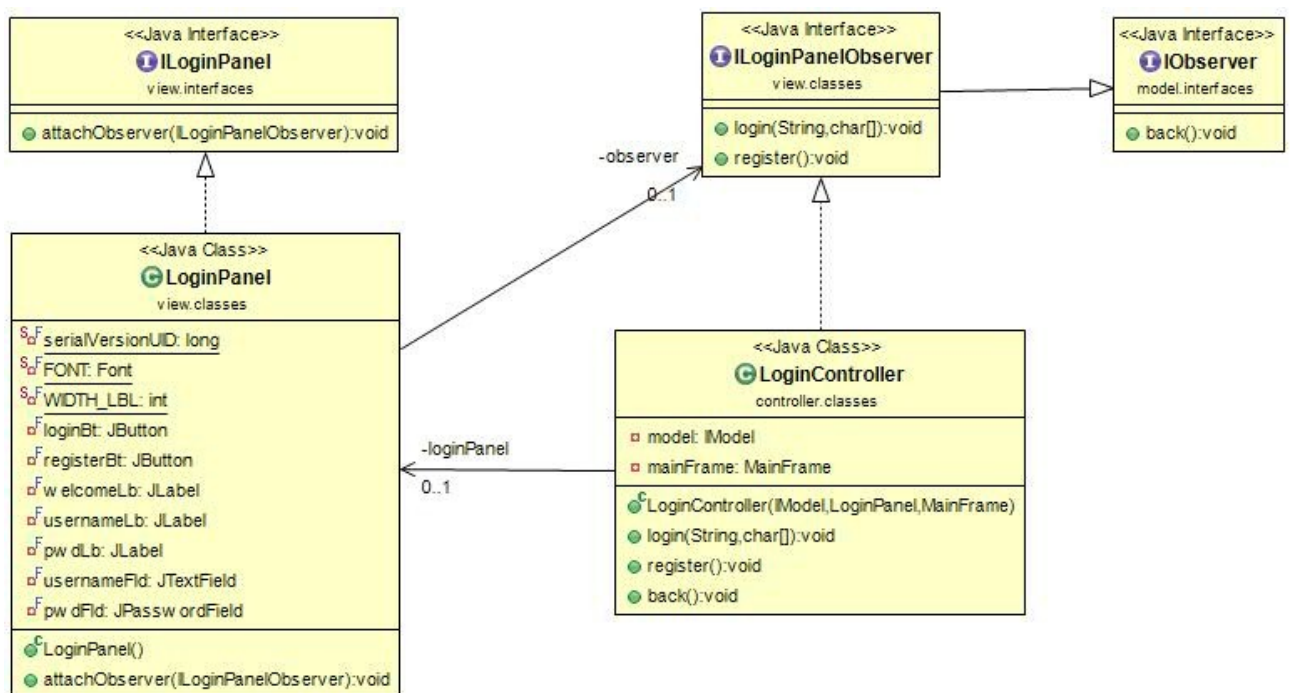


Figura 2.5: Questo UML rappresenta l'applicazione del pattern observer per quanto riguarda il pannello di login. Si può notare che la classe LoginPanel contiene l'interfaccia ILoginPanelObserver che è implementata dalla classe LoginController, la quale gestirà tutti gli eventi associati ai metodi login, register e back (quest'ultimo metodo non è presente nell'interfaccia ILoginPanelObserver, ma è contenuto in IObservable, dal quale estendono tutti gli observer).

Template method

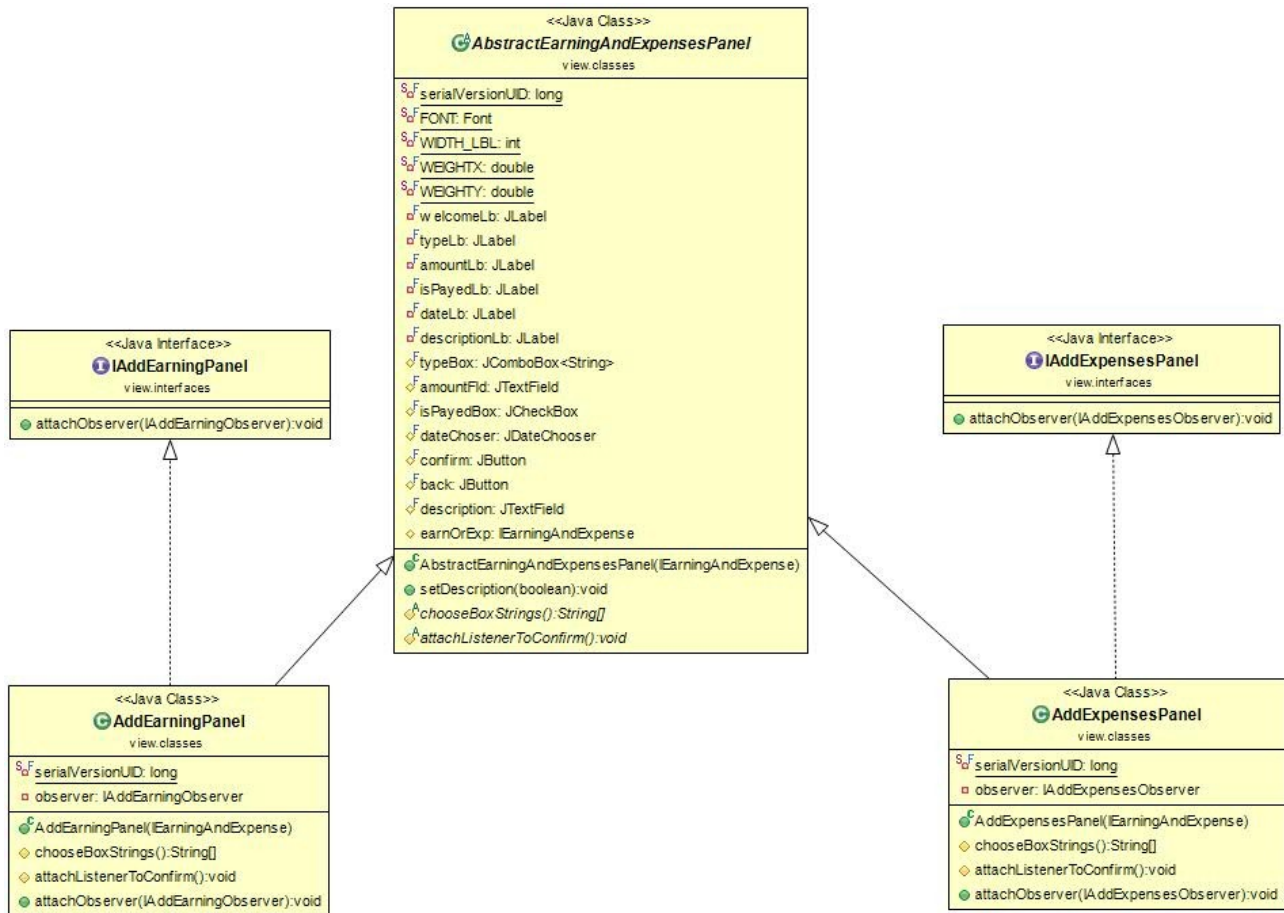


Figura 2.6: In questo UML si può notare l'applicazione del template method. Visto che i due pannelli per inserire spese e rientri sono praticamente uguali, eccetto per i tipi di spesa/rientro da selezionare e per l'observer da collegare, ho deciso di creare una classe astratta che contiene tutti gli elementi della vista da presentare e che possiede anche due metodi astratti. Il primo (chooseBoxStrings) non fa altro che restituire la lista di elementi che andranno a finire dentro il JComboBox typeBox (che appunto mostra il tipo di spesa o rientro). Il secondo (attachListenerToConfirm) aggiunge il Listener (che è diverso se siamo nel pannello per aggiungere spese o in quello per aggiungere i rientri) ai bottoni confirm e back. Il template method sembrerebbe non esserci ma diciamo che è "nascosto" all'interno del costruttore, visto che è proprio lì dentro che viene richiamato il metodo chooseBoxString che popolerà la nostra JComboBox.

Pattern Strategy

```
private List<IEarningAndExpense> showForChoose(final String choose, final int month, final Calendar week) {
    List<IEarningAndExpense> returnList = null;
    DateComparator comparator = new DateComparator();
    if (choose == "Total") {
        returnList = this.list;
    } else if (choose == "Month") {
        Calendar dateCompare = Calendar.getInstance();
        Calendar dateCompare2 = Calendar.getInstance();

        dateCompare.set(Calendar.MONTH, month);
        dateCompare.set(Calendar.DATE, 1);

        dateCompare2.set(Calendar.MONTH, month);
        dateCompare2.set(Calendar.DATE, dateCompare2.getActualMaximum(Calendar.DAY_OF_MONTH));

        returnList = list.stream().filter(e -> comparator.compare(e.getDate(), dateCompare) >= 0 && comparator.compare(e.getDate(), dateCompare2) < 0).collect(Collectors.toList());
    } else if (choose == "Week") {
        final int DAYS_IN_A_WEEK = 7;

        Calendar dateCompare = Calendar.getInstance();
        dateCompare.setTime(week.getTime());

        Calendar dateCompare2 = Calendar.getInstance();
        dateCompare2.setTime(week.getTime());
        dateCompare2.add(Calendar.DATE, DAYS_IN_A_WEEK);

        returnList = list.stream().filter(e -> comparator.compare(e.getDate(), dateCompare) >= 0 && comparator.compare(e.getDate(), dateCompare2) < 0).collect(Collectors.toList());
    }
    return returnList;
}
```

Figura 2.7: Questo screen mostra un metodo presente all'interno del controller ShowEarningAndExpenseAvancedController. Questo metodo è utilizzato per filtrare, in tre modi diversi, i dati in una lista di spese o guadagni. Il metodo accetta tre parametri, il primo (choose) indica il metodo di filtraggio (totale, per mese, settimanale), il secondo indica il mese sul quale si vuole filtrare (naturalmente verrà usato solo nella porzione di codice che tratta il filtraggio mensile, il terzo indica la settimana in base alla quale si vuole filtrare la lista (anche in questo caso verrà utilizzato solo nella porzione di codice che tratta il filtraggio settimanale). Questo metodo, oltre alla lista di ritorno (returnList), contiene anche un comparatore (contenuto nello stesso package) che ho dovuto creare, visto che il comparator di base per i Calendar mi dava dei problemi. Il pattern strategy è applicato per popolare la lista di ritorno (nei casi "Month" e "Week"), infatti al metodo filter di uno stream, va passata la strategia da utilizzare e, in questi casi, non ho fatto altro che confrontare le date di tutti gli elementi della lista e selezionare solo quelli che appartengono al mese (o alla settimana) passato come parametro.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

All'interno del progetto abbiamo svolto due tipi di testing, il primo automatizzato che si occupa di verificare la corretta implementazione e il corretto funzionamento del modello del nostro software, mentre per la view e il controller non abbiamo utilizzato nessun testing automatizzato, ma abbiamo visionato manualmente e visivamente la corretta implementazione della view e del controller.

Per il test automatizzato abbiamo utilizzato la suite di JUnit. Grazie ad essa abbiamo testato tutti i principali metodi pubblici della classe "model" e abbiamo verificato che la creazione di un utente, di un abitazione o di una spesa avvenisse correttamente. Inoltre abbiamo testato anche la corretta implementazione degli algoritmi e delle strutture dati che ci permettono di agire sulla gestione delle spese e dei rientri sia di una singola abitazione o utente, oppure una stima globale delle spese e entrate.

Grazie all'uso del test automatizzato abbiamo riscontrato diversi errori di implementazione del software all'interno del modello che abbiamo potuto così risolvere.

3.2 Divisione dei compiti e metodologia di lavoro

Il lavoro è stato diviso tra i due componenti (Marco Mancini e Federico Marinelli), nel seguente modo:

- Marco Mancini: GUI e controllers della parte riguardante le spese e i rientri all'interno di una abitazione. (Compreso calendario e promemoria)
- Federico Marinelli: GUI e controllers della parte riguardante le spese e i rientri del singolo utente. (Comprese statistiche)
- Insieme: Divisione package, sviluppo di tutto il modello, applicazione MVC.

Avendo applicato il pattern MVC come descritto nel paragrafo 2.1.3 non è stato difficile poter operare separatamente sul progetto visto che il modello è stato sviluppato insieme. Ogni membro ha la possibilità di crearsi la propria vista, il proprio controller e operare sul modello, senza intaccare il lavoro dell'altro membro.

Come DVCS abbiamo utilizzato Mercurial. Come richiesto dal prof, è bastato creare un repository su bitbucket sul quale entrambi i membri lavorano attraverso il plugin per Eclipse di Mercurial. Attraverso questo plugin è molto semplice effettuare commit, push, pull, merge e tutte le altre operazioni disponibili, in poco tempo.

3.3 Note di sviluppo

Essendo, per entrambi, il primo progetto a cui lavoravamo, inizialmente avevamo la sindrome del foglio bianco, quindi abbiamo deciso di prendere spunto dai buoni progetto degli anni scorsi.

Nota su Calendario:

Nel progetto è presente una funzionalità che mostra un calendario con la possibilità di selezionare una data e mostrare spese/rientri presenti in quella data. La realizzazione della vista di questo pannello è stata presa da internet (<http://www.dreamincode.net/forums/topic/25042-creating-a-calendar-viewer-application/>) Il codice trovato su internet però non dà la possibilità di evidenziare le date, quindi ho modificato il renderer della tabella per fare in modo che le date presenti all'interno di una lista (opportunamente riempita dal controller), vengano evidenziate e che ci sia la possibilità di vedere tutte le spese e i rientri nella data selezionata, attraverso un bottone.

Capitolo 4

Commenti finali

4.1 Conclusioni e lavori futuri

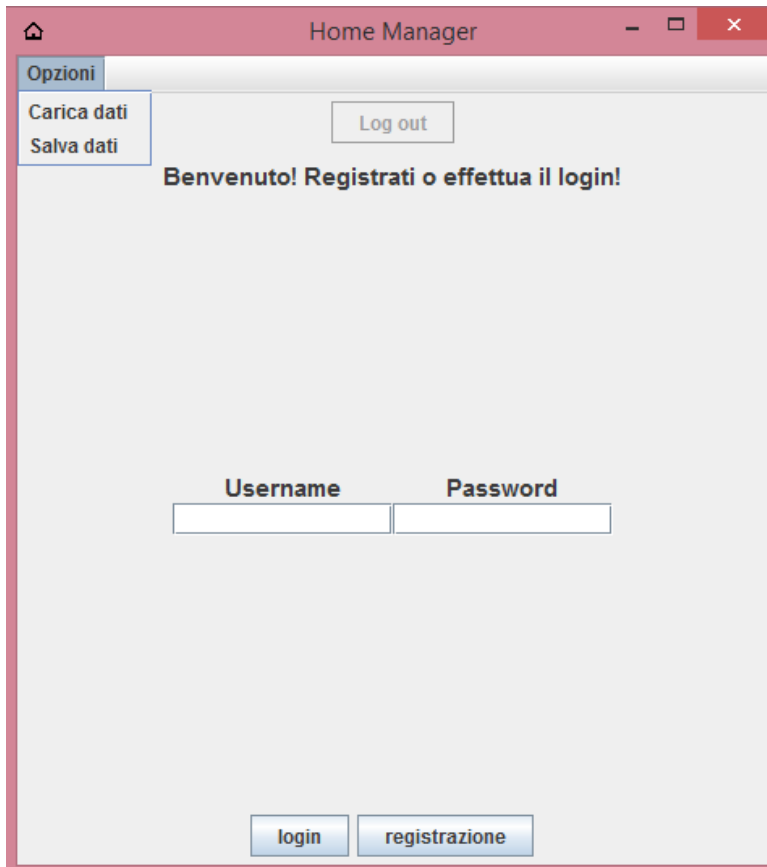
Il progetto ci sembra ben svolto, copre praticamente tutti gli argomenti svolti durante il corso, dall'implementazione di interfacce all'utilizzo di stream e di pattern come lo strategy, l'MVC, observer, all'esecuzione di test. Alcune cose non sono fatte alla perfezione, per esempio il salvataggio dei dati avviene nella cartella principale dell'utente, quando sarebbe più opportuno creare una cartella nascosta per ogni utente registrato nell'applicazione. Un altro problema sta nell'ordinamento per colonne delle tabelle che mostrano spese e ricavi, se si aggiungono un po di spese/ricavi e si prova ad ordinarli per la colonna dell'importo, l'ordinamento non avverrà correttamente, abbiamo provato a debuggare ma non siamo riusciti a risolvere il problema. Inoltre la grafica non è delle migliori (sì, fa abbastanza schifo) ma non riesco a settare uno sfondo, anche se non è difficile visto che tutti i pannelli estendono un pannello principale in cui basta eseguire l'override del metodo paintcomponent. In conclusione possiamo affermare che questo progetto ci sembra svolto correttamente.

Avevamo pensato di estendere questa applicazione anche al marketplace di Android. Inoltre si potrebbe dare la possibilità all'utente di inserire i propri dati PayPal e scaricare tutti i movimenti all'interno dell'applicazione.

Appendice A

Guida utente

Avviando l'applicazione ci si ritroverà davanti da una schermata (fig A.8) che ci dà la possibilità di effettuare il login o la registrazione al programma.



The screenshot shows a window titled "Home Manager" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there is a menu bar with "Opzioni" selected, which has a dropdown menu containing "Carica dati" and "Salva dati". To the right of the menu bar is a "Log out" button. Below the menu bar, the text "Benvenuto! Registrati o effettua il login!" is displayed. In the center, there are two input fields labeled "Username" and "Password". At the bottom of the window, there are two buttons: "login" and "registrazione".

Figura 2.8: Pannello di login e registrazione.

Se si sceglie di registrarsi si verrà reindirizzati ad un pannello che vi chiederà informazioni (nome, cognome, username e password) e una volta inserite, se non è presente un utente con lo stesso username, vi registrerà al sistema. Inoltre nella parte alta della finestra è presente un menù che dà la possibilità di caricare o salvare i dati in una cartella a scelta. Una volta registrati non rimarrà che effettuare il login, la schermata che si presenta una volta effettuato quest'ultimo (fig A.9) dà la possibilità all'utente di scegliere tra tre voci, la prima (Gestione spese personali) servirà per gestire tutte le spese personali dell'utente, la seconda (Gestione spese abitative) servirà per la gestione delle singole abitazioni mentre la terza (Statistiche generali) servirà per mostrare le statistiche generali dell'andamento delle spese e rientri dell'utente.



Figura A.9: Pannello home

Inoltre possiamo notare che, nella parte alta della finestra, viene mostrato il nome dell'utente e il pulsante di logout è abilitato. Selezionando la voce "Gestioni spese abitative" si verrà reindirizzati ad un pannello che ci mostra tutte le abitazioni fin'ora inserite dall'utente (mostrando una panoramica generale dell'abitazione, via, città, saldo). Questo pannello ci dà la possibilità, oltre che di vedere le abitazioni, anche di poterle cancellare, aggiungere o gestire. Se si sceglie di cancellare l'abitazione non si avrà più la possibilità di accedere ai dati di essa, mentre se si sceglie di aggiungere una abitazione verrà mostrato un pannello che chiederà tutte le informazioni dell'abitazione e provvederà ad aggiungerla alla lista. Se invece si sceglie l'opzione "seleziona" sull'abitazione, verrà mostrato un altro pannello (fig. A.10) che in alto mostra le informazioni di essa, mentre al centro contiene tutte le possibili operazioni. Ora verrà data una descrizione di tutte le operazioni possibili:

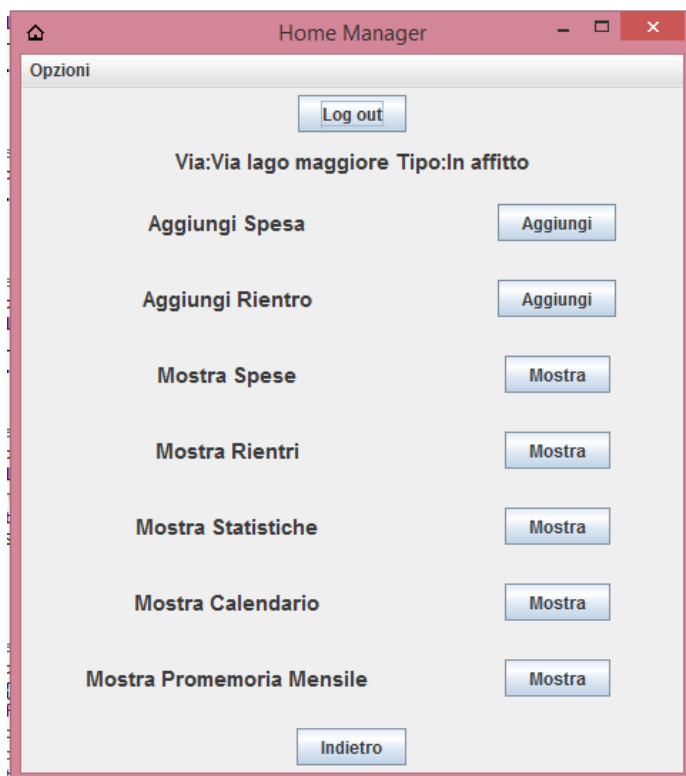


Figura A.10: Pannello gestione abitazione

Aggiungi Spesa/Rientro

Le operazioni di “Aggiungi Spesa” e “Aggiungi Rientro” mostreranno entrambe un semplice pannello che chiederà tutte le informazioni di spesa/rientro (tipo, ammontare in euro, se la spesa/rientro è stata pagata o meno e la data del pagamento) e conterrà un bottone per confermare l’inserimento e un bottone per tornare al pannello precedente (come tutti i pannelli hanno). Le opzioni “Mostra spese” e “Mostra Rientri” presentano all’utente una tabella (fig. A.11) divisa in 5 colonne (spesa/ricavo, importo, tipo, pagato, data), contenente tutte le spese/rientri, inserite. Questa tabella è ordinabile cliccando sulle colonne, ma è anche filtrabile (totale, per mese o per settimana) selezionando l’opzione di filtraggio nella parte bassa del pannello. Inoltre è possibile modificare e cancellare le spese/rientri visualizzati. Per quanto riguarda la modifica il programma non farà altro che aprire lo stesso pannello visualizzato per l’aggiunta di spese/rientri ma con la differenza che, questa volta, i campi avranno già i valori impostati. Per la cancellazione, invece, basterà cliccare sulla spesa/ricavo che si vuole eliminare e premere il pulsante “cancella”.

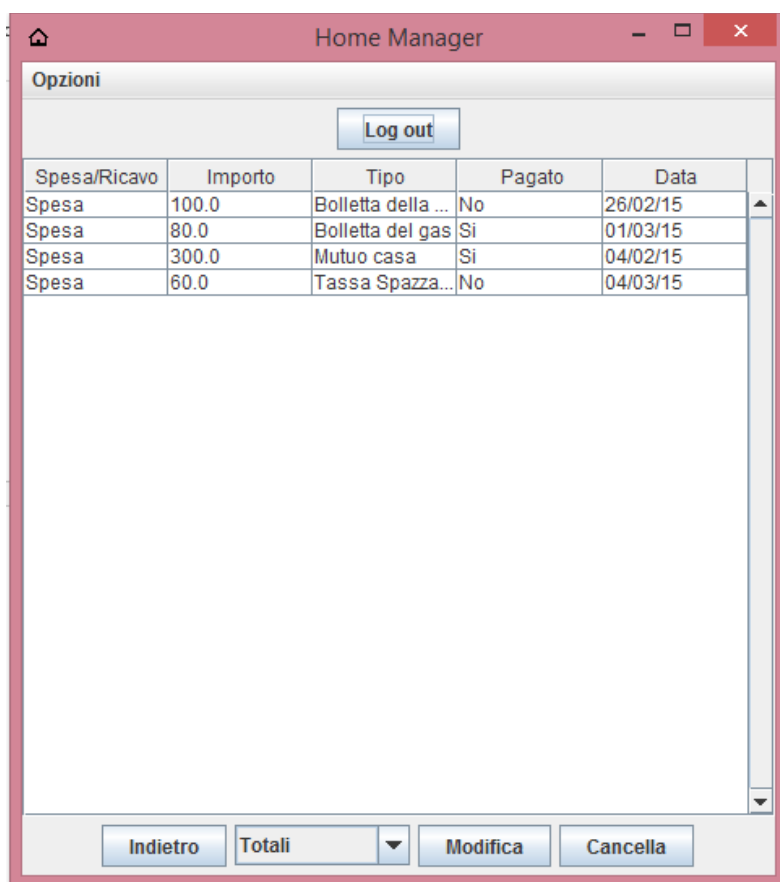


Figura A.10 pannello mostra spese

Mostra statistiche

Cliccando su mostra statistiche verrà presentato un pannello contenente il saldo delle entrate/uscite e il saldo totale. Anche qui si ha la possibilità di visualizzare i vari saldi in tre differenti modi, cioè totali, per mese o per settimana. In aggiunta c'è la possibilità di vedere grafici (a barre 3D e a torta) che mostrano gli andamenti delle entrate e delle uscite (anche qui totali, mensili e settimanali) e le loro percentuali.

Mostra Calendario

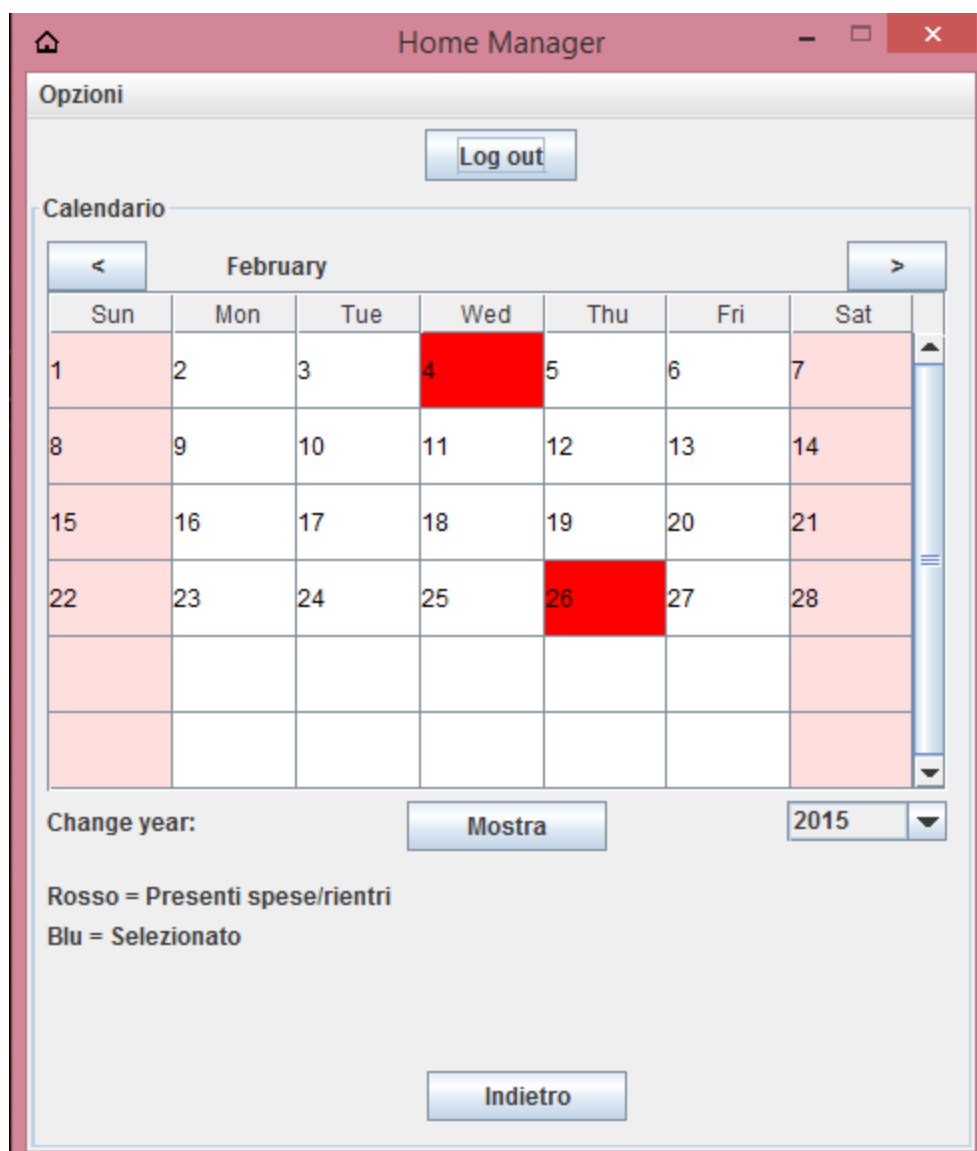


Figura A.11: Pannello che mostra il calendario

Questa opzione ci mostra un calendario(figura A.11) in cui le date evidenziate in rosso sono quelle in cui sono presenti spese/ricavi, cliccando sopra di esse e poi sul pulsante “mostra” sarà possibile visualizzare le spese/rientri in quella data attraverso una tabella uguale a quella in figura A.10.

Mostra promemoria

L'ultima opzione di questo menù ci mostra due tabelline (una per le entrate e una per le uscite) contenenti i promemoria mensili, cioè tutte quelle entrate e uscite che ancora non sono state pagate.