Doctor's Appointment

Giuseppe Radaelli

14 marzo 2015

Sommario

DsA (*Doctor's Appointment*) è un software open source scritto in Java con licenza *GPL with linking exception*.

Quest'applicazione è stata realizzata solo ed esclusivamente come attività didattica per il progetto del corso di Programmazione ad Oggetti a.a. 2014/2015 della facoltà di Ingegneria e Scienze Informatiche di Cesena.

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il software che si intende sviluppare mira a smaltire ed organizzare il lavoro del medico aziendale all'interno dell'azienda stessa.

Il programma potrà essere utilizzato in due modalità: da utente/paziente o da dottore.

I primi potranno solo accedere all'interfaccia che gli consentirà di scegliere il giorno e l'orario desiderato per la loro visita (se disponibile), e dopo se volessero anche di cancellarla, tramite codice ricevuto al momento della prenotazione.

Il medico invece, conoscendo la password, avrà l'accesso alla sua interfaccia, dalla quale potrà vedere la lista delle prenotazioni, cambiare la password e scegliere gli orari in cui ricevere visite ed impostare il numero massimo di pazienti per giorno (ovviamente le modifiche saranno valide solo dopo il salvataggio e non saranno retroattive, quindi per esempio, se il medico decidesse di non voler più visitare alle 12:00, chi fino a quel momento avesse prenotato a quell'orario non verrà cancellato dall'elenco).

1.2 Problema

Siccome ogni PC della LAN aziendale ha diritto di accedere al file della prenotazioni per scriverci, la home del software presenta una casella di testo nella quale va inserito il percorso della directory in cui è contenuto il file (decisa dall'azienda ovviamente).

La difficoltà primaria sarà quella di riuscire a trovar euna strategia che consenta di non perdere dati e di garantire che ogni prenotazione rimanga in memoria, infatti, avere un solo file comune a tutti gli host in LAN vuol

dire correre il rischio che venga aperto da più utenti in contemporanea, e cioè perdere dei dati.

Per esempio, se due utenti aprono lo stesso file da due postazioni differenti per modificarne il contenuto, solo le modifiche di chi lo chiuderà per ultimo andranno salvate.

Capitolo 2

Design

2.1 Architettura

L'applicazione è basata sul pattern architetturale Model-View-Controller.

Nella Figura 2.1 viene mostrato lo scheletro di questo software, nel quale si è cercato di separare al meglio i tre protagonisti di quest'architettura.

2.2 Design dettagliato

Il problema della concorrenza al file dati è stato risolto creando il file temporaneo *lock.tmp*, il quale, se presente nella directory in cui c'è il file condiviso, blocca l'accesso ad un'altra applicazione, consentendo solo ad un utente/dottore per volta d apportare modifiche.

Quando quest'ultime verranno salvate (o comunque si decide di chiudere l'applicazione), lock.tmp verrà cancellato, quindi il file dei dati tornerà disponibile, consentendo a qualcun altro di lavorarci.

Volendo scendere più nel dettaglio del software, esso è costituito da tre package uno per ogni elemento del pattern MVC.

- *it.unibo.mysoftware.view*:

  mostrato in Figura 2.2, nella quale si evince come in *AbstractViewImpl* il metodo *public abstract void attachController(final Controller c)* sia un Templete Method, il quale è lasciato implementato alle sottoclassi; inoltre per evitare duplicazioni di codice si è usato il pattern Abstract Factory.

- *it.unibo.mysoftware.controller*:

mostrato in Figura 2.3, dove viene riusato il pattern Abstract Factory per evitare di riscrivere il codice.

- *it.unibo.mysoftware.model*:

  Figura 2.4, in cui si mostrano solo le entità fondamentali, al suo interno si fa uso del pattern Singleton; questo package contiene anche la main class *Application* ed anche un altro package contenente le eccezioni lanciate dal model: *it.unibo.mysoftware.model.exceptions*; inoltre per creare il codice di cancellazione per ogni appuntamento viene usato l'*hash code* della classe *Person*, calcolato sui suoi campi, si è deciso di adottare questo metodo per due ragioni, primo per non salvare ulteriori dati sul file e secondo perchè così facendo ogni utente può mantenere il suo stesso codice (sempre che non cambi i dati inseriti).

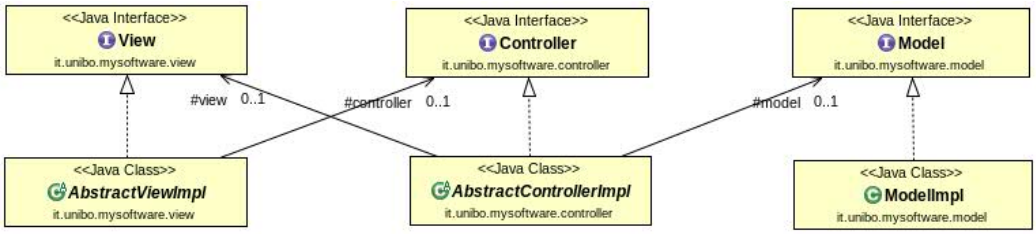

Figura 2.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

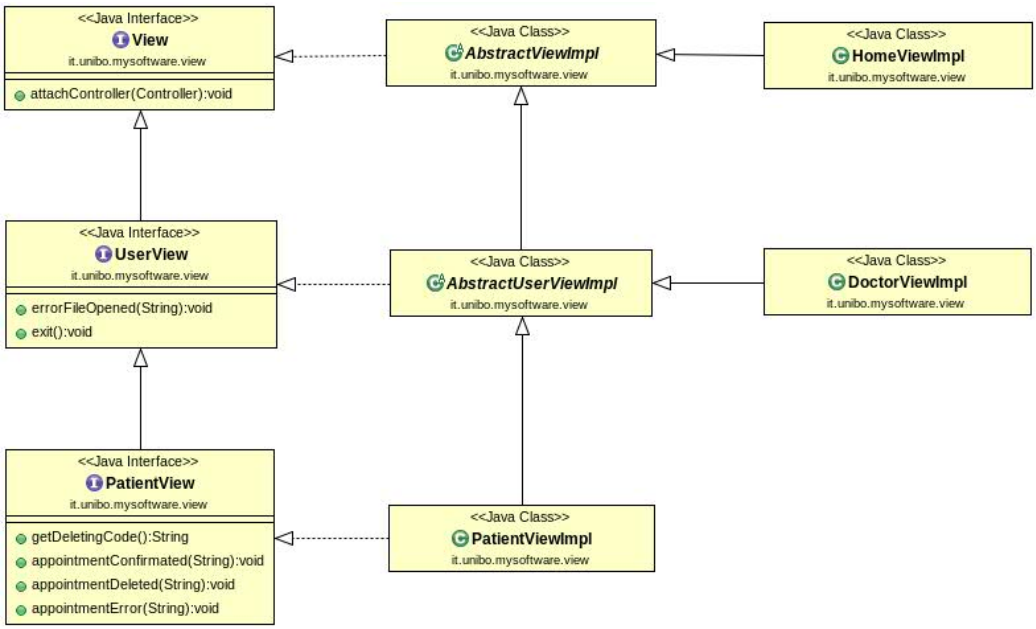

Figura 2.2: Schema UML delle entità principali nel package *it.unibo.mysoftware.view*

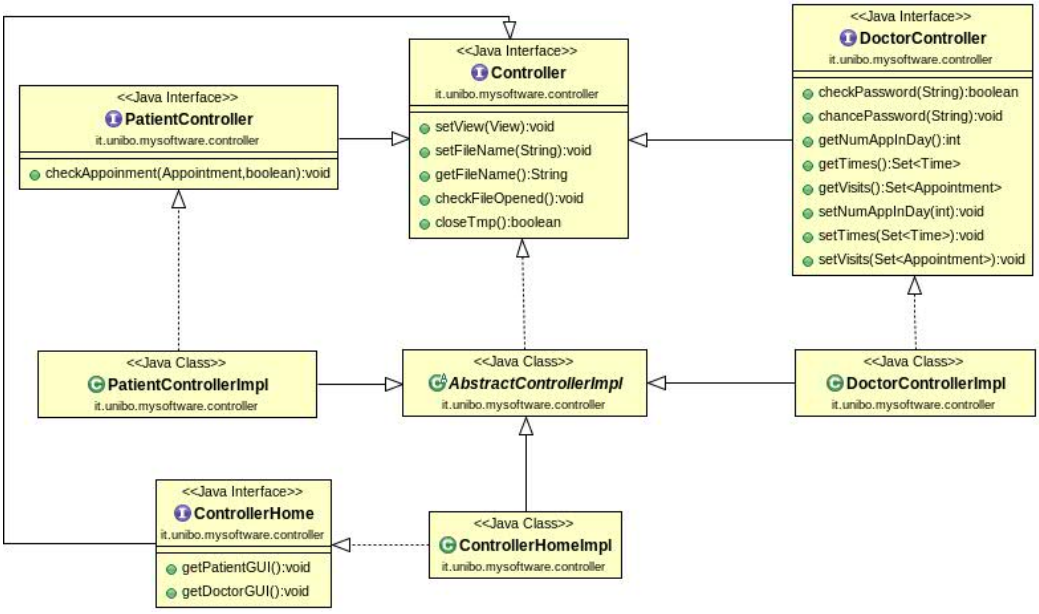

Figura 2.3: Schema UML delle entità principali nel package *it.unibo.mysoftware.controller*

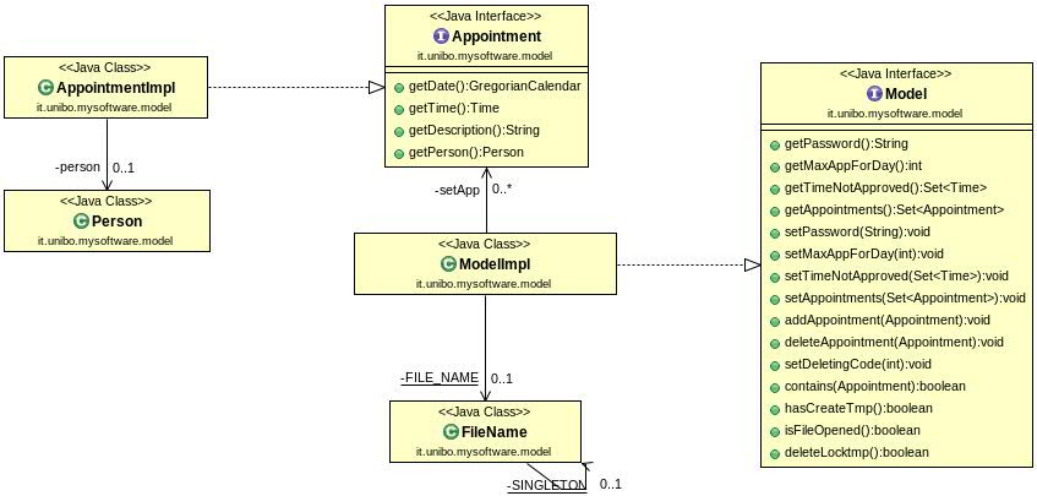

Figura 2.4: Schema UML delle entità principali nel package *it.unibo.mysoftware.model*

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Sulla parte riguardante il modello si è provveduto a realizzare una classe di test automatizzata, tramite la suite di JUnit.

In particolare si è controllato che l'accesso al file condiviso potesse avvenire solo ad un utente per volta, in modo da non perdere dati, si è anche pensato a intercettare tutte le eccezioni possibili, permettendo al programma di terminare sempre in maniera corretta.

3.2 Divisione dei compiti e metodologia di lavoro

Per il versioning dell'applicazione si è fatto uso del DVCS Mercurial e del seguente repository Bitbucket:

https://bitbucket.org/ilRada17/oop2014-dsa.

Capitolo 4

Commenti finali

4.1 Conclusioni e lavori futuri

Questo software, a mio avviso molto semplice da realizzare, è stato creato solo ed esclusivamente dall'autore di questa breve relazione, apparte la classe *CalendarPanel*, di cui si è preso spunto da un progetto gia creato trovato su internet e modificato successivamente.

DsA è realizzata solo come attività didattica per il progetto del corso di Programmazione ad Oggetti a.a. 2014/2015 della facoltà di Ingegneria e Scienze Informatiche di Cesena.

Per questo non si prevedono sviluppi futuri.