Relazione HotelManage

Albonetti Simone
Chiara Tozzi

26 aprile 2021

Indice

Capitolo 1

Analisi

Il software commissionato dal gestore dell'Hotel Morelli, consiste in un gestionale, HotelManage, che ha il compito di rendere più agevole la gestione delle camere e delle prenotazioni da parte dei dipendenti. Infatti tale software permetterà di capire quali camere sono attualmente disponibili, o quando lo saranno in futuro. In questo modo l'operatore avrà una visione complessiva della struttura e potrà scegliere quali camere associare alle varie prenotazioni pervenute.

1.1 Requisiti

Requisiti funzionali

- Il software dovrà fornire all'utente informazioni riguardanti la disponibilità delle camere in base alle richieste dei clienti dell'hotel.
- Dovrà fornire la possibilità di aggiungere/rimuovere prenotazioni.
- Dovrà tenere un registro dei clienti che conterrà le persone che in passato hanno già pernottato presso l'hotel, in modo da rendere più rapida la compilazione della prenotazione.
- Le informazioni sulle camere dovranno aggiornarsi selezionando una data diversa (da quella odierna) nella schermata principale del software.

Requisiti non funzionali

- Il software avrà un interfaccia grafica che dovrà essere il più "user-friendly" possibile per permettere ai dipendenti (con conoscenze informatiche eterogenee) di poter utilizzare il gestionale in modo efficace.

- Cliccando su un tasto che rappresenta la stanza, questo dovrà mostrare le informazioni inerenti tale stanza

1.2 Analisi e modello del dominio

HotelManage si occuperà di mostrare ai dipendenti dell'hotel quali camere sono disponibili e quali no, in tempo reale. In questo modo si potranno inserire nel software le prenotazioni, composte da: periodo di pernottamento, cliente e stanza. In caso di indisponibilità di camere per una certa prenotazione saranno i dipendenti stessi a concordare con il cliente una data futura disponibile di pernottamento, oppure l'assegnazione di una tipologia di camera diversa(es: una camera doppia invece che singola). Non è previsto un controllo automatico sull'inserimento di più prenotazioni nello stesso periodo per la medesima stanza. Questo controllo non è stato implementato nel monte ore previsto, perciò sarà compito del personale dell'hotel evitare di inserire più prenotazioni nello stesso periodo. HotelManage si occuperà di rendere più agevole la gestione dei clienti. Sarà infatti possibile registrare nuovi clienti o cercare l'eventuale presenza di un cliente già registrato, così che il dipendente possa riconoscere più velocemente se esso ha già pernottato altre volte. Sara quindi compito del dipendente decidere di applicare sconti al cliente già registrato. Nella prima versione del software fornita non si prevede la possibilità di scegliere clienti già registrati nell'inserimento delle nuove prenotazioni, ma dovranno essere inseriti manualmente dal dipendente, in quanto non è possibile realizzarlo nel monte ore previsto. Tale feature sarà oggetto di futuri lavori.

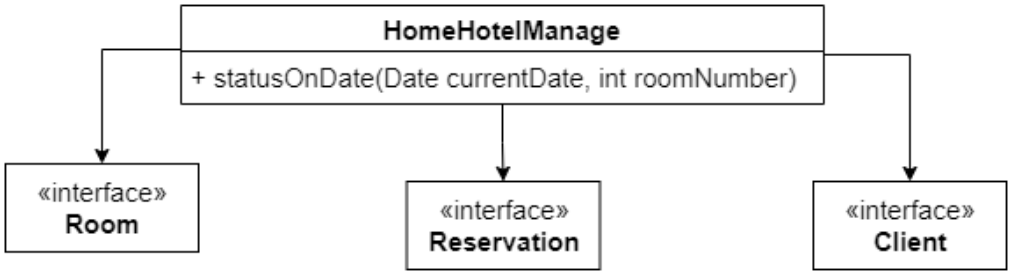


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

L'architettura di HotelManage segue il pattern MVC. Si è quindi scelto di utilizzare "model-view-controller" che consente di tenere una suddivisione chiara e pulita delle diverse parti dell'applicazione. HotelManage è quindi composto dall'interfaccia principale Home attraverso la quale verrà consentita la comunicazione con le diverse interfacce clienti e prenotazioni. Queste view secondarie consentiranno quindi la comunicazione, per la memorizzazione delle informazioni, con i relativi controller. Queste interfacce utente permetteranno di effettuare in modo semplice e intuitivo le varie operazioni di inserimento/controllo. I controller avranno quindi il compito di interagire con le classi(clienti - prenotazioni - stanze) e con la classe file(incaricata di trascrivere su file tutti i dati delle prenotazioni/clienti). Per quanto riguarda le stanze non è prevista un interfaccia utente in quanto non è necessario effettuare modifiche ad esse nello specifico. Sarà comunque presente nella Home la possibilità di visualizzarle e comprendere se esse sono disponibile in una specifica data.

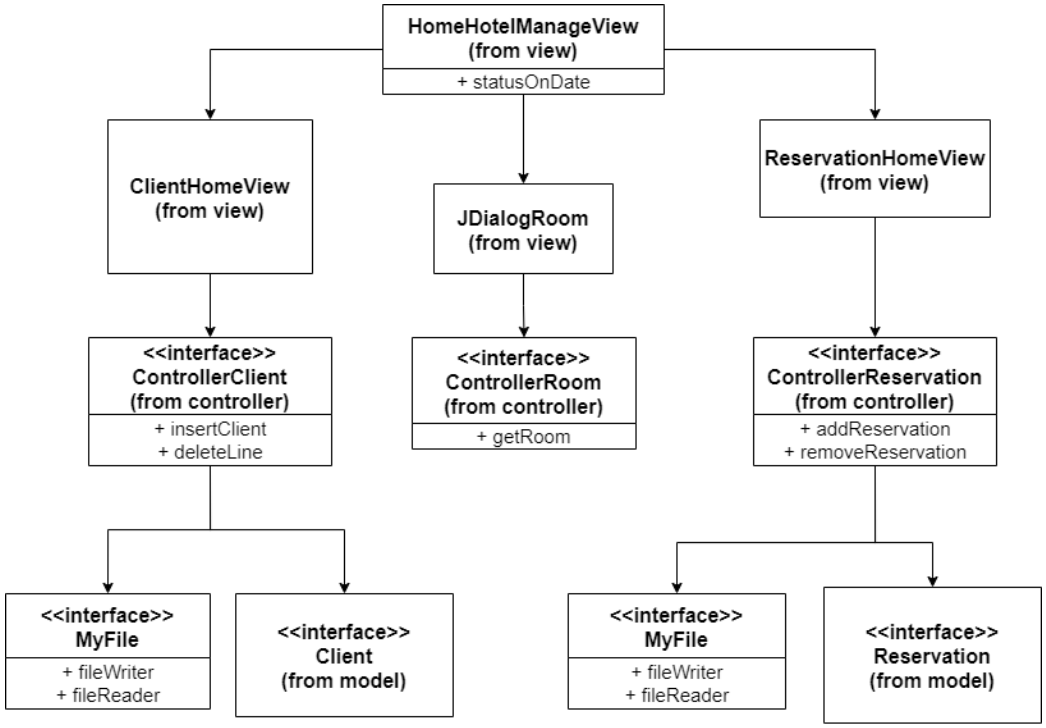
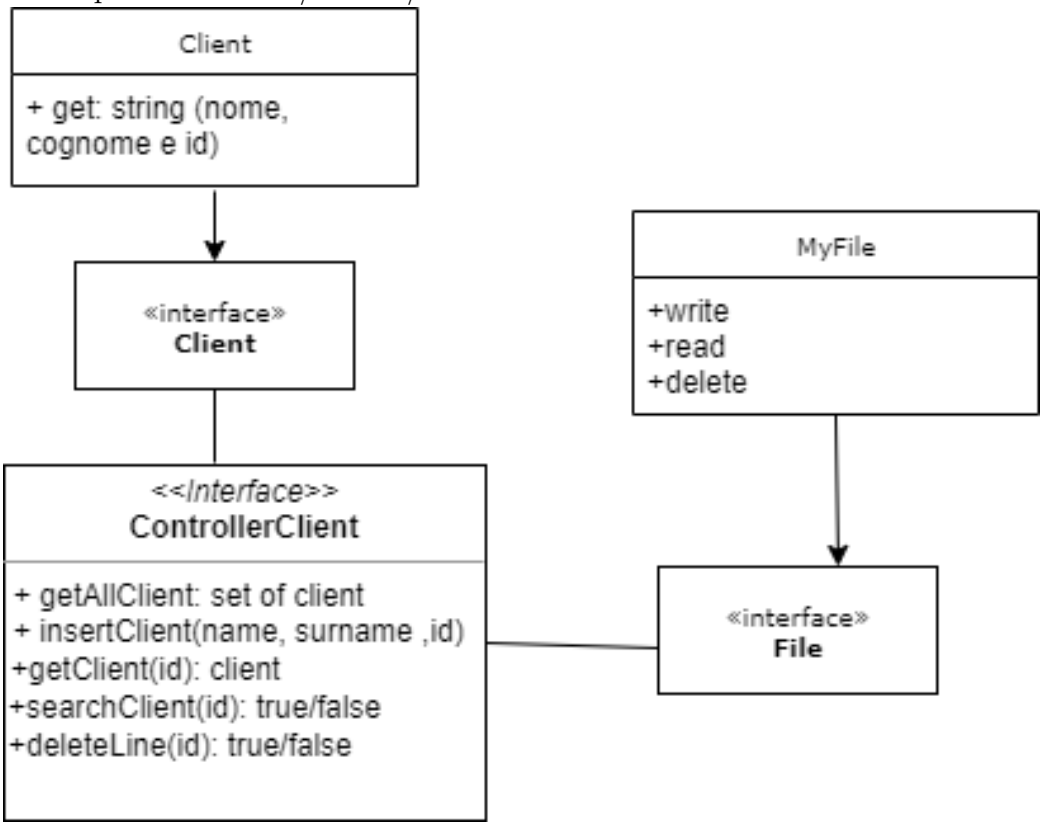


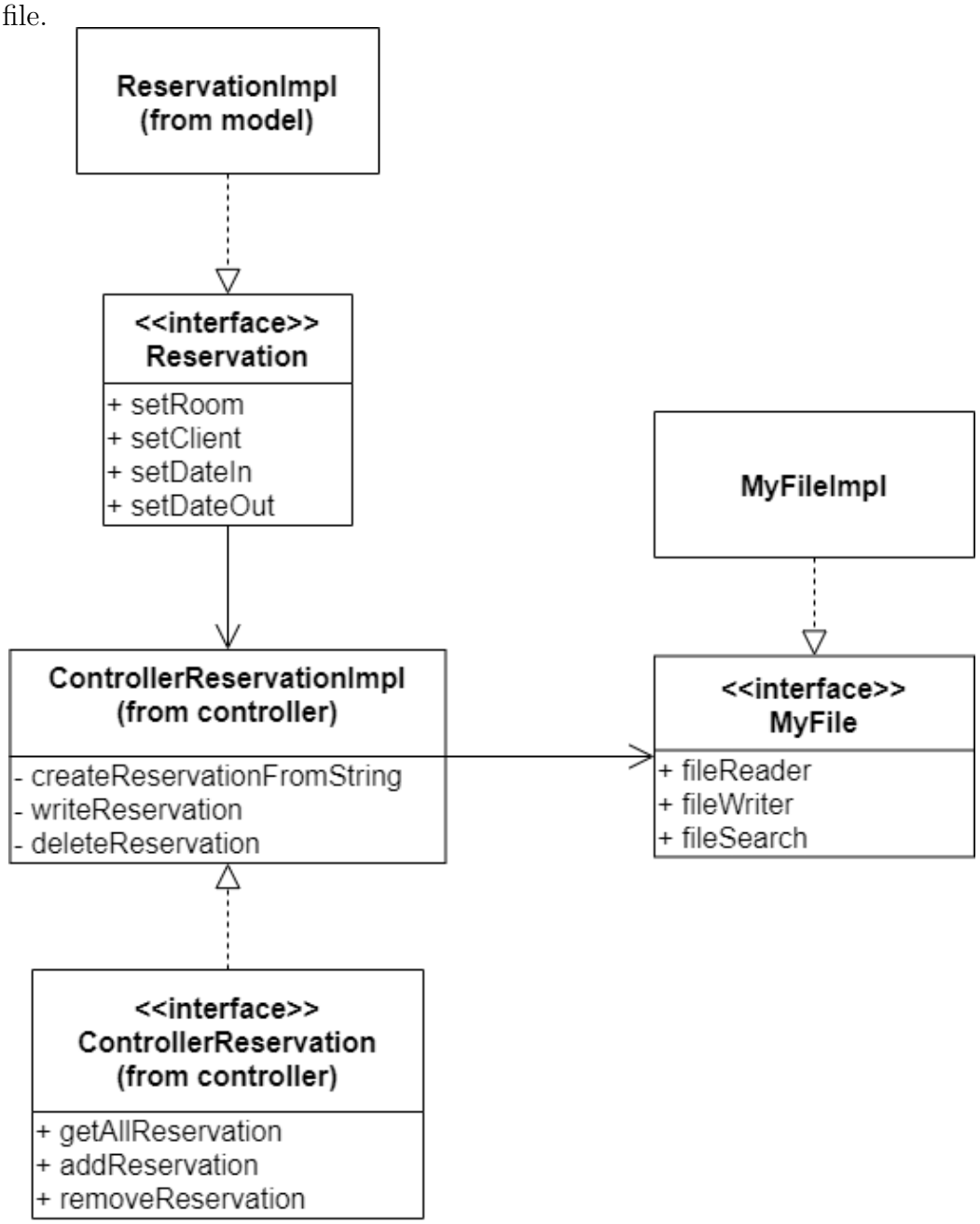
Figura 2.1: Schema UML dell'architettura. HomeHotelManageView è il punto di ingresso, dal quale si accede alle altre entità.

2.2 Design dettagliato

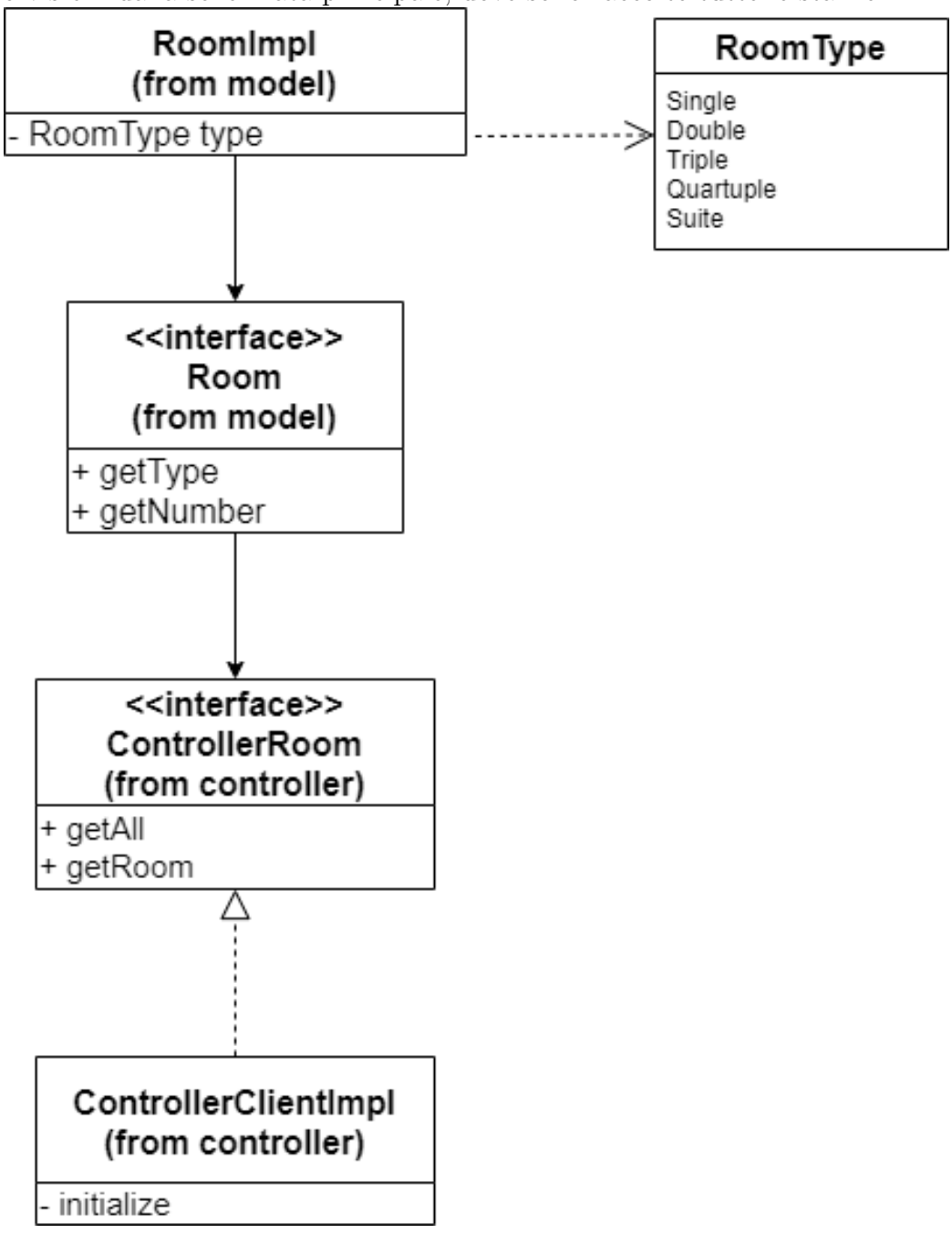
Per la sessione dei clienti e file (riportata nell'immagine sottostante) si utilizza un pattern Strategy per ogni componente e sotto-componente. Il controller del cliente avrà il compito di creare il cliente ma anche di interagire con la classe file per la scrittura/lettura/eliminazione dei dati.



Le prenotazioni (reservation) utilizzano il pattern Strategy. Ogni prenotazione si compone di una stanza, data di arrivo, data di partenza e un cliente a cui è associata la prenotazione. Il controller si occupa di aggiungere, rimuovere e visualizzare le prenotazioni, attraverso letture e scritture su un file. Inoltre ha un campo interno con l'insieme delle prenotazioni (organizzato in set per non avere ripetizioni), in modo da ridurre il numero di letture su file.



Le room modellano l'oggetto stanza, che si compone di un numero di stanza e di una tipologia. La tipologia viene definita attraverso una enum, poichè generalmente non viene cambiata la tipologia di stanza. Questo campo, contiene anche il prezzo e il numero di posti per ciasun tipo. Queste informazioni sono visibili dalla schermata principale, dove sono raccolte tutte le stanze.



Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per quanto riguarda la parte di testing sono stati effettuati dei test manuali durante il corso di tutto lo sviluppo, ad esempio per il funzionamento delle view e delle classi. Alcuni di questi sono poi stati resi automatici con JUnit (clienti, prenotazioni e stanze). Per quanto riguarda i clienti viene testato il corretto funzionamento del controllerCliente - cliente, la ricerca/scrittura/eliminazione. Anche per quanto riguarda le prenotazioni viene testato il funzionamento del controller e delle relative letture e scritture. Generalmente veniva creato prima il test e successivamente veniva fatta l'implementazione di un metodo che doveva soddisfare tale test. Non sono state usate le assertion ma si è preferito avere messaggi nello standard output che indicassero il superamento o meno del test. Per quanto riguarda le stanze, si testa solamente la restituzione della stanza/e richiesta/e dal controller.

3.2 Metodologia di lavoro

Per quanto riguarda la divisione del lavoro, Chiara Tozzi si è occupata di tutto ciò che riguarda la classe cliente e la classe file (controller/model/view per i Clienti, model per i File e view per la Home).
Simone Albonetti si è occupato delle prenotazioni e delle stanze (controller/model/view per le prenotazioni, model/controller per le stanze). Per quanto riguarda la view principale, è stata creata da Tozzi con alcuni successivi interventi di Albonetti, per i pochi componenti non menzionati c'è stata una collaborazione.
Abbiamo utilizzato git per lo sviluppo del software, cercando di non appor-

tare modifiche alle stesse classi contemporaneamente in modo da evitare i merge conflict.

3.3 Note di sviluppo

Eventuali particolarità del metodo di sviluppo di Chiara Tozzi:

- Utilizzo di java.io per Buffer reader e writer
- Uso di WindowBuilder Editor in parte, con successiva revisione di alcune parte di codice

Sono state effettuate parecchie ricerche su vari siti per quanto riguarda la progettazione della classe file, per una maggiore conoscenza e informazione.

Simone Albonetti:

- Utilizzo di Optional nella ReservationView e nel ControllerReservation
- Uso di WindowBuilder Editor in parte, con successiva revisione di alcune parte di codice
- Utilizzo della libreria toedter - JCalendar per l'utilizzo dei DateChooser nelle view

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Chiara Tozzi: Il mio lavoro svolto all'interno del progetto, a mio parere, è ben fatto ma con ampio margine di miglioramento. Dal mio punto di vista può essere migliorato a livello di efficienza e ordine. Riconosco in oltre che a causa dei problemi di organizzazione interna non ho avuto abbastanza tempo per dedicarmi con maggiore attenzione e tranquillità al codice svolto come avrei voluto. Per quanto riguarda la view(a livello dell'aspetto) c'è ampio margine di modifiche e miglioramento, non sono soddisfatta del tutto di quella parte di codice. Per quanto riguarda il mio ruolo credo sia stato fondamentale a pari livello dell'altro componente del gruppo, con un ottima comunicazione.
Simone Albonetti: Il mio lavoro, credo sia ben svolto. Non è ottimo, poichè ci sono alcune parti che sarebbero potute essere sviluppate meglio (es. gestione delle date e controlli su prenotazioni contrastanti), ma a causa dei problemi organizzativi si è cercato di fare il meglio possibile, rispettando la data di consegna prefissata. Il mio ruolo è stato al pari di Tozzi, concordo sul fatto che tra noi c'è stata un'ottima organizzazione e comunicazione, nonostante gli altri problemi organizzativi.

4.2 Difficoltà incontrate e commenti per i docenti

Chiara Tozzi: La principale difficoltà riscontrata è a livello organizzativo, e quindi conseguentemente tempistico, riscontrato con i non più componenti del gruppo. Nessun problema con il corso svolto dai docenti.

Simone Albonetti: Anche secondo me la difficoltà che ci ha penalizzato di più è stata quella organizzativa.

Appendice A

Esercitazioni di laboratorio

A.1 Albonetti Simone

- Laboratorio 05: `https://github.com/Simone-Albonetti/OOP-lab-05.git`
- Laboratorio 06: `https://github.com/Simone-Albonetti/OOP-Lab06.git`
- Laboratorio 07: `https://virtuale.unibo.it/mod/forum/discuss.php?d=62582#p104679`
- Laboratorio 08: `https://virtuale.unibo.it/mod/forum/discuss.php?d=63865#p105150`
- Laboratorio 09: `https://github.com/Simone-Albonetti/OOP-Lab09.git`