Documento di progettazione

MyContactManager

ALESSANDRO PETTI
ALESSANDRO SABIA
ALFONSO VILLANI
GIOVANNI LAMBERTI

Ingegneria del Software
2024/2025

Indice

1. Descrizione del progetto

Il progetto consiste nel realizzare una **rubrica** in grado di contenere dei **contatti** a cui vengono associate le seguenti informazioni:

- nome
- cognome
- numero di telefono
- indirizzo e-mail
- note
- una foto

Il numero massimo di indirizzi e-mail e numeri di telefono che possono essere associati ad un contatto varia da zero a tre.
Affinché possa essere salvato in rubrica, è necessario che per ciascun contatto venga indicato almeno **un nome** o **un cognome**.
Durante l'inserimento del numero di telefono e dell'indirizzo email, il **formato deve essere verificato** tramite un metodo apposito, che consente l'inserimento dei dati in memoria solo dopo la validazione.
Il numero di telefono può essere preceduto da un **prefisso**, scelto tra quelli disponibili, oppure può non essere preceduto da alcun prefisso.

La rubrica dovrà essere persistente per l'utente che l'utilizza, è quindi necessario che vengano implementate delle operazioni di salvataggio e caricamento mediante file binari per permetterne il riuso. Saranno inoltre offerte delle funzionalità di importazione ed esportazione della rubrica da file CSV.

È nostra intenzione fornire agli utenti la possibilità di selezionare più elementi simultaneamente. A tal fine sarà implementata la funzionalità di **selezione multipla.** Attivata cliccando l'apposito tasto, dà modo all'utente di selezionare i contatti desiderati utilizzando la checkbox che compare di fianco a ciascun contatto per effettuare l'operazione di rimozione di ciascuno di essi in un numero ridotto di click.

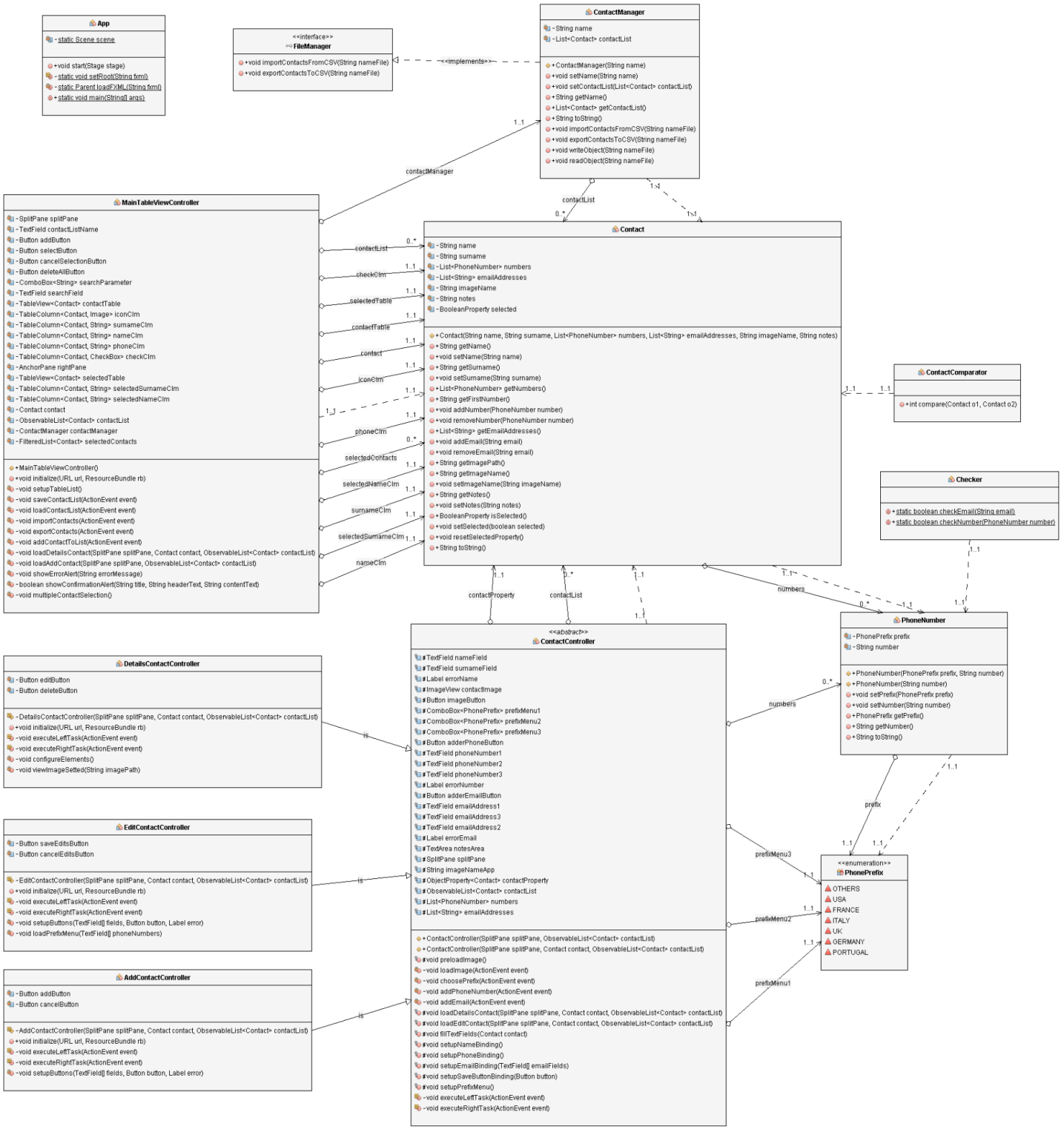
L'interfaccia principale è organizzata mediante uno **SplitPane.**
Sulla sinistra si presenta una **TableView** contenente i contatti della rubrica. L'utente avrà rapido accesso all'operazione di ricerca sui contatti utilizzando vari criteri, selezionabili tramite un menu dedicato. I criteri disponibili includono:

- Nome
- Cognome
- Nome e cognome
- Numero di telefono
- Email

Per cercare un contatto, è sufficiente selezionare il criterio desiderato e digitare il termine nella barra di ricerca. Alla destra dello split pane si alterneranno invece le interfacce per le operazioni di aggiunta del contatto, visualizzazione del dettagli e modifica.

Individuate le funzionalità principali e chiarito il contesto operativo, si dispone ora di una base solida da cui partire per modellare il problema e prendere le decisioni progettuali, che verranno illustrate nel paragrafo successivo insieme al diagramma delle classi.

2. Diagramma delle classi



3. Scelte progettuali

Come raffigurato nella sezione precedente, è stata dedicata un interfaccia per specificare funzionalità, delle classi per modellare le entità numero di telefono e un tipo enumerativo per il prefisso telefonico.
Queste classi sono state progettate per garantire una separazione chiara delle responsabilità e per facilitare l'espandibilità del sistema.

Per il singolo contatto

La classe *Contact* contiene le seguenti proprietà:

- **String** per il nome, cognome e le note relative al contatto.
- **List<PhoneNumber>** e **List<String>** per una gestione dinamica dei numeri di telefono e degli indirizzi email, permettendo l'associazione di più numeri e email per ogni contatto.
- **BooleanProperty** per indicare se un contatto è selezionato, fondamentale per il corretto funzionamento della funzionalità di selezione multipla.

Inoltre, implementa i metodi per ottenere e modificare le informazioni relative al contatto, garantendo la possibilità di aggiornare i dati in modo sicuro.

Per la rubrica

La classe *ContactManager* contiene le seguenti proprietà:

- **String** per dare un nome alla rubrica.
- **List<Contact>** per gestire i contatti dinamicamente. Questa scelta, sebbene non la più performante in termini di ricerca e accesso, è sufficiente per il numero di contatti previsto, dove le differenze rispetto a strutture dati più efficienti, come le mappe, non sono rilevanti. Inoltre si evita l'uso di strutture dati come Set per non perdere la possibilità di avere contatti omonimi.

Fornisce metodi per aggiungere, rimuovere i contatti ed implementa i metodi dell'interfaccia **FileManager** per esportare e importare contatti da file CSV, oltre a supportare la serializzazione per il salvataggio e il recupero dei dati.

Per i numeri di telefono

La classe PhoneNumber per rappresentare un formato che deve avere il numero telefonico mediante attributi:

- **Prefix** per indicare un prefisso in base
- **String** sufficiente per rappresentare il numero.

Include metodi per ottenere e modificare sia il prefisso che il numero. La sua rappresentazione testuale “prefisso numero” sarà poi utilizzata per l’esportazione e la relativa importazione dei dati.

Per i prefissi

Il tipo enumerativo **PhonePrefix** viene utilizzato per definire un insieme limitato di prefissi internazionali. Ogni prefisso è associato a un controllo specifico, poiché i numeri possono seguire formati o schemi differenti a seconda del prefisso. Questo approccio garantisce una gestione più precisa e coerente dei numeri di telefono a livello internazionale. Tuttavia, non è previsto il supporto per tutti i prefissi esistenti; la struttura è progettata per consentire una facile estensione in futuro.

Per il controllo

La classe Checker contiene metodi di validazione per numeri di telefono e indirizzi email. I suoi metodi vengono invocati prima di inserire nuovi numeri e email nelle rispettive liste, garantendo che i dati siano sempre corretti e formattati adeguatamente. È stata preferita una classe rispetto a un'interfaccia, in quanto l'uso di un'interfaccia avrebbe comportato la necessità di implementare una classe dedicata per gli indirizzi email, che si ritiene superflua per le esigenze del progetto.

Per l’interfaccia grafica

L’architettura prevista si ispira al modello MVC per tenere ben separati la parte logica da quella di presentazione.
Ogni View è infatti ben distinta, lasciando al relativo Controller la gestione della logica e degli eventi. Si prevede l’uso di molteplici View e molteplici Controller per modulare maggiormente il codice, favorendo la riusabilità e la coesione.

Questo approccio mira a raggiungere buoni risultati di scalabilità e testabilità del codice.

4. Coesione

Classi	Livello di coesione	Descrizione
ContactManager	Funzionale	Il modulo offre funzionalità coordinate con l'obiettivo specifico di gestire l'elenco dei contatti.
PhoneNumber	Funzionale	Il modulo garantisce una corretta rappresentazione del numero di telefono.
PhonePrefix	Funzionale	Il modulo viene utilizzato per definire un insieme limitato di prefissi internazionali.
Contact	Funzionale	Tutti i campi sono strettamente correlati al concetto di contatto e i metodi che comprende sono le operazioni elementari per la modifica e la lettura di un contatto.
FileManager	Funzionale	L'interfaccia definisce l'insieme di metodi da implementare per l'operazione di scrittura e lettura dei dati usando file CSV.
ContactComparator	Funzionale	La classe si occupa del confronto tra contatti per definire il criterio di ordinamento.
Checker	Funzionale	I metodi che definisce servono alla validazione di formati dei campi di un contact

ContactController *(AddContactController, DetailsContactController, EdiContactController)*	Comunicazionale	La classe implementa le funzioni che determinano l'interazione tra l'utente e l'interfaccia grafica.
MainTableViewContr oller	Comunicazionale	La classe è ampia perché si occupa di gestire l'interfaccia principale e gestire tutte le possibili interazioni con essa. Fa chiamate a tante funzionalità come importazione ed esportazione di ContactManager, ma tutte le funzionalità ruotano attorno alla gestione della tabella principale e alla rubrica che viene presentata in essa..

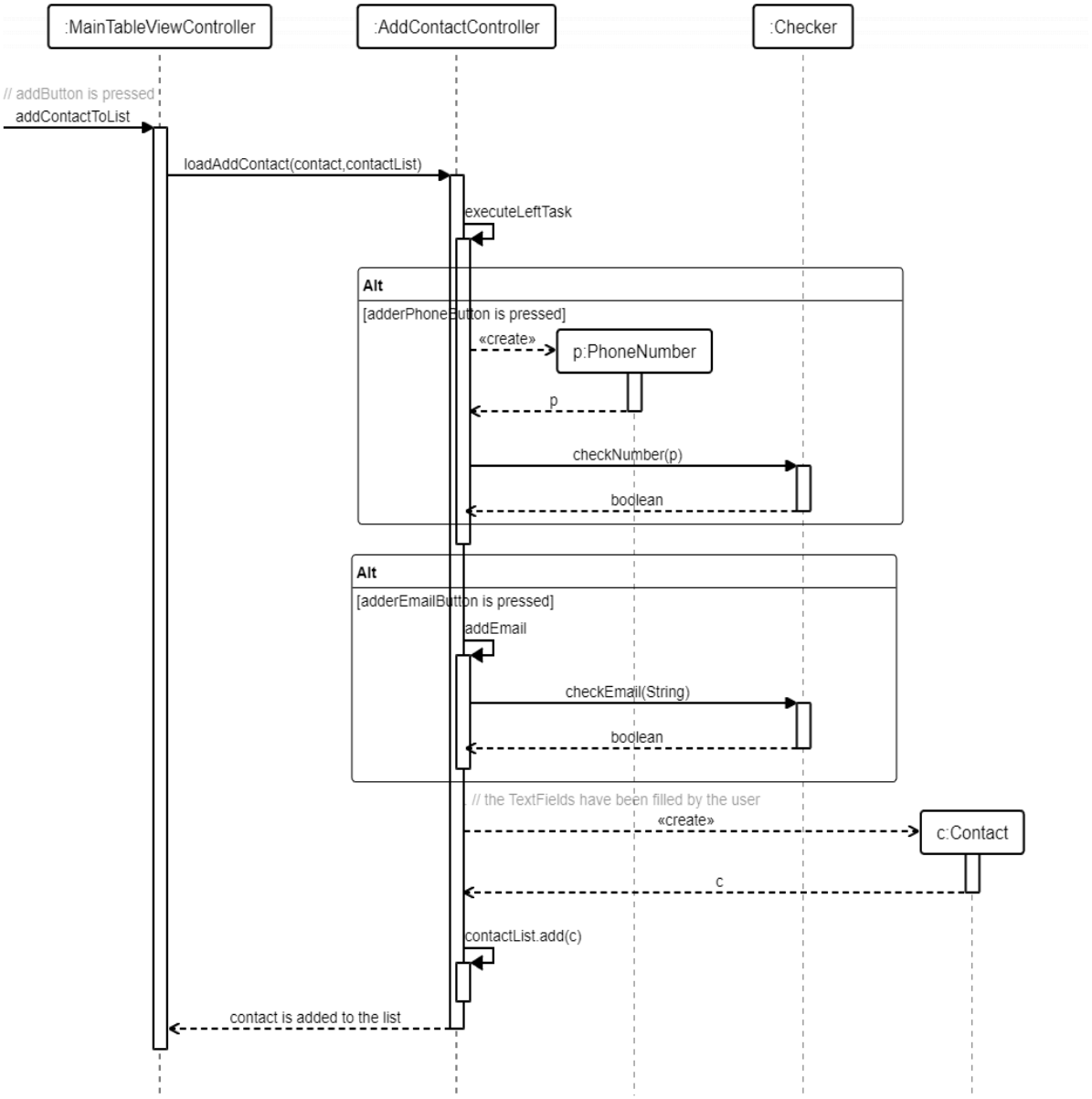
5. Accoppiamento

Classi	Livello di Accoppiamento	Descrizione
PhonePrefix - PhoneNumber	Per dati	L'interazione diretta tra i moduli è volta alla definizione di un attributo prefisso all'interno della classe PhoneNumber.
PhoneNumber - Contact	Per dati	La classe PhoneNumber passa alla classe Contact solo le informazioni necessarie al suo funzionamento; che sono rispettivamente il numero e il prefisso telefonico del contatto.
Contact - ContactManager	Per dati	ContactManager gestisce una lista di Contact
ContactController- Checker	Per dati	Le sottoclassi di ContactController verificano i dati del contatto tramite Checker passandogli ciascun PhoneNumber.
Checker - PhoneNumber	Per dati	La classe Checker lavora su un'istanza di PhoneNumber per la validazione dell'informazione.

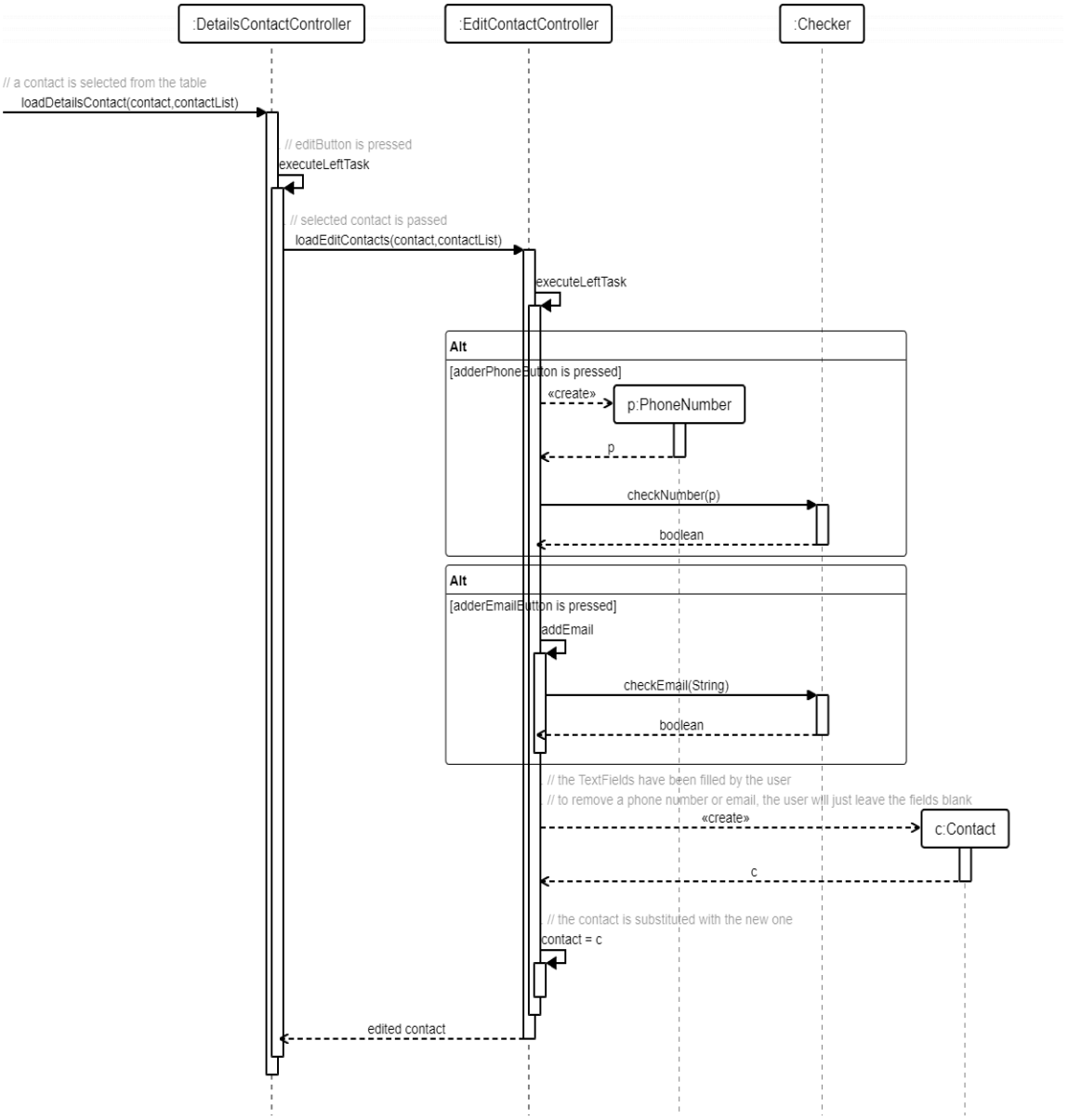
ContactComparator - Contact	Per dati	ContactComparator confronta di dati di due istanze di Contact per il confronto.
ContactManager - FileManager	Per dati	ContactManager implementa i metodi definiti in FIleManager.
ContactController - MainTableViewController	Per dati	Queste classi comunicano passandosi delle istanze di ContactManager, Contact e SplitPane.
ContactController - ContactManager	Per timbro	La classe ContactManager passa una struttura contenente anche informazioni che non sono necessarie all'altro modulo. Difatti avviene il passaggio dell'intera lista contatti.
MainTableViewController - ContactManager	Per timbro	

Diagrammi di sequenza

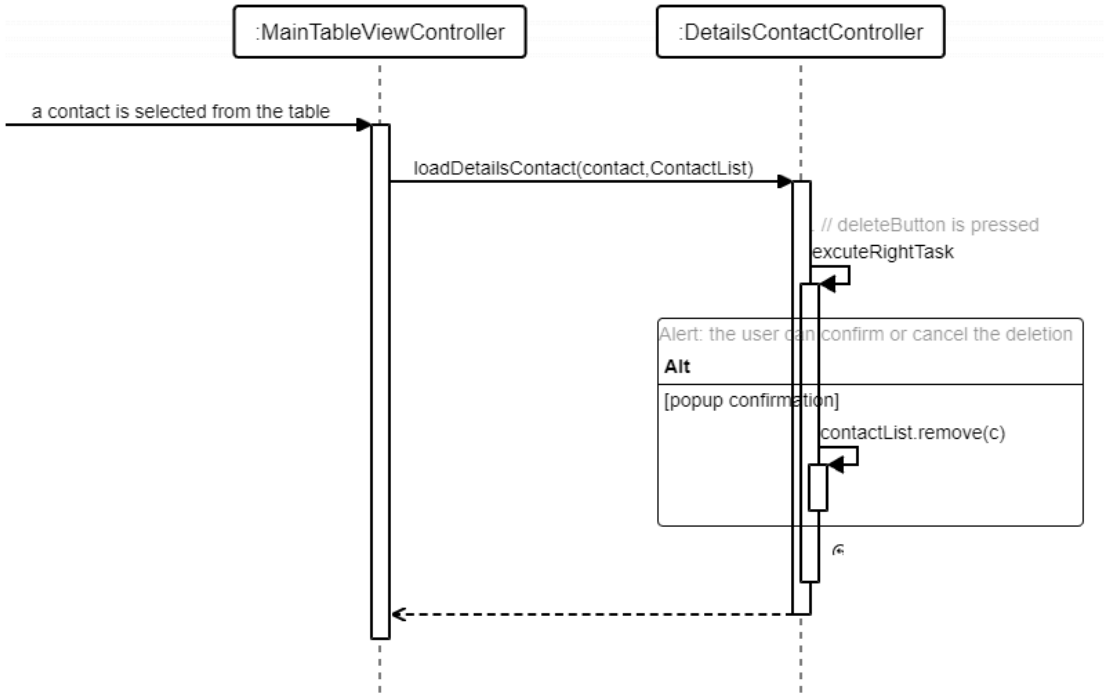
6.1 Aggiungi Contatto



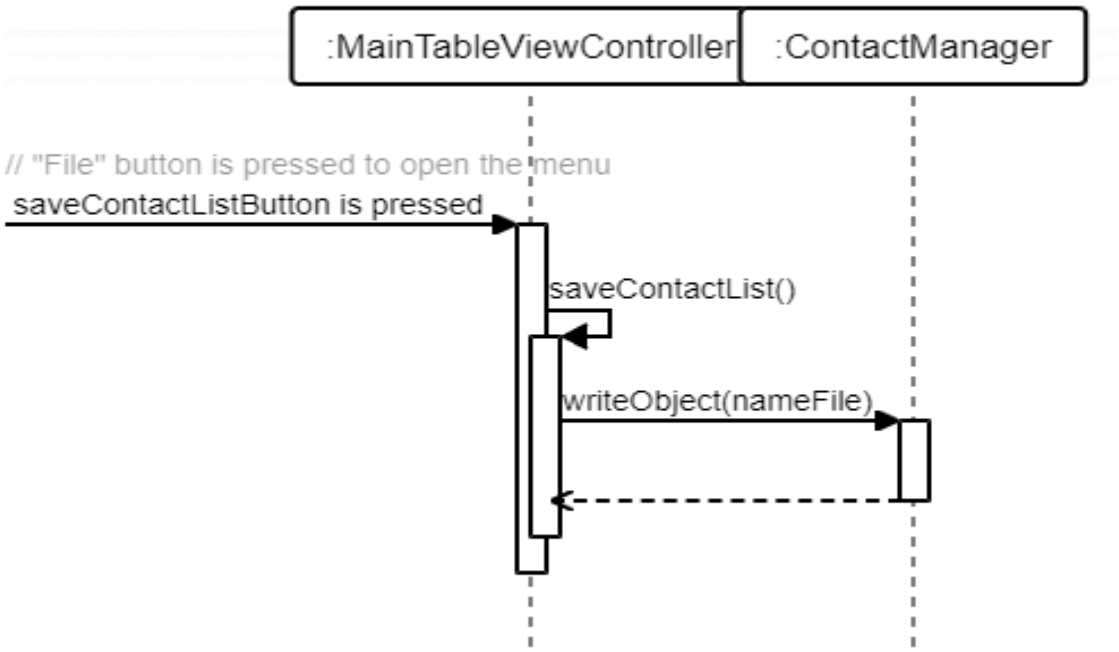
6.2 Modifica Contatto



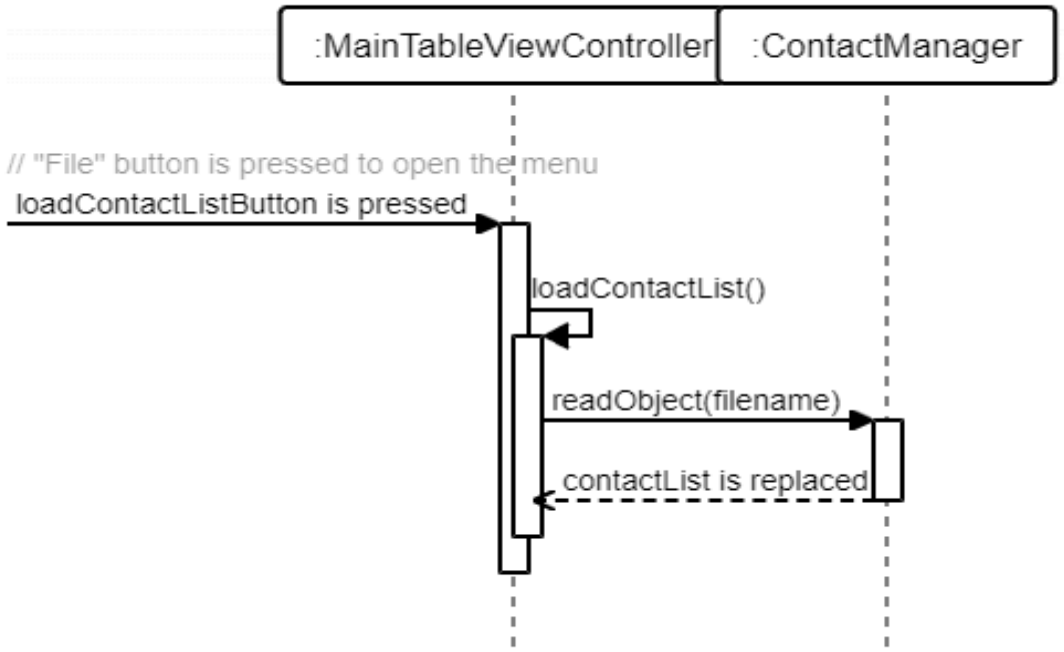
6.3 Rimuovi Contatto



6.4 Salva Rubrica



6.5 Carica Rubrica



7. Verifica dei principi di buona progettazione

L'architettura e l'implementazione del progetto mirano a raggiungere un buon livello per gli attributi di qualità, con particolare attenzione ai principi SOLID e ad altre linee guida note. Questo approccio contribuisce a rendere il codice più chiaro, flessibile, estendibile e facilmente manutenibile.

7.1 Principio della Singola Responsabilità (SRP)

Ciascuna entità è stata modellata in modo tale che il componente realizzato svolga un singolo compito ben definito.

Il sistema è organizzato in classi con compiti ben definiti. Ad esempio, Contact gestisce esclusivamente le informazioni di un singolo contatto, ContactManager amministra l'insieme dei contatti e PhoneNumber è costituito dalle sole informazioni riguardanti un numero di telefono. Allo tempo stesso i vari controller (AddContactController, DetailsContactController, EditContactController) gestiscono

soltanto la logica di presentazione, il che rende più semplice intervenire sul codice, assicurando manutenibilità e riutilizzo.

7.2 Principio Aperto/Chiuso (OCP)

Il codice è progettato tenendo in considerazione che le classi debbano essere aperte all'estensione ma chiuse alle modifiche interne. In modo tale da non lasciare che la presentazione di nuovi scenari ne richieda la riprogettazione, ma invece permette di adattarsi a possibili nuove funzionalità.
Il principio è particolarmente evidente nella struttura delle classi dove, mediante l'implementazione di classi astratte come ContactController e interfacce come FileManager, è stata sfruttata la proprietà dell'incapsulamento e dell'ereditarietà.

7.3 Principio di Sostituzione di Liskov (LSP)

L'assenza di gerarchie di classi complesse non mette a dura prova il Principio di Sostituzione di Liskov (LSP). Tuttavia l'ereditarietà tra i ContactController e le sue sottoclassi rispettano il principio.

7.4 Principio di Segregazione delle Interfacce (ISP)

La semplicità del progetto non ha richiesto l'introduzione di numerose interfacce, nonostante ciò, la struttura adottata mantiene le responsabilità ben separate, evitando di sovraccaricare le classi con metodi non necessari.
L'unica interfaccia implementata è FileManager dotata di pochi metodi e tutti strettamente correlati fra loro. Questo sposa quanto richiesto dal Principio di Segregazione delle Interfacce.

7.5 Principio di Inversione delle dipendenza

Per la realizzazione del progetto si è tenuto conto del principio di inversione delle dipendenze nel realizzare la classe astratta ContactController e l'interfaccia FileManager.
Questi due moduli sono implementati in modo tale da sfruttare l'ereditarietà di Java, ma al tempo stesso non dipendere dalle classi che le utilizzano così da favorire manutenibilità e testabilità

7.6 Altre Buone Pratiche Applicate

- **DRY (Don't Repeat Yourself):**
  L'implementazione evita duplicazioni ridondanti nella logica di dominio. Difatti le implementazioni di algoritmi di ordinamento delle liste, caricamento di interfacce grafiche e la visualizzazione di avvisi sono implementati una sola volta mediante classi esterne (ContactComparator) e metodi privati della superclasse ContactController (loadDetailsContact, loadEditContact, showAlertError, showConfirmationAlert).
- **Principio della minima Sorpresa:**
  L'uso coerente di convenzioni, come la notazione CamelCase e nomi intuitivi e per classi e metodi, rende il codice più leggibile e prevedibile, facilitando l'orientamento e la comprensione per chiunque vi metta mano. Il codice è stato scritto in lingua inglese per renderlo comprensibile da un più ampio numero di sviluppatori.
- **Separazione dei Compiti (Separation of Concerns):**
  La distinzione tra gestione dei contatti, logica di presentazione (controller e viste) e servizi di supporto (ad es. Checker e FileManager) aumenta la chiarezza del progetto. Ogni parte può essere modificata o estesa senza influenzare il resto del sistema, migliorando manutenibilità e testabilità.