Documento di Design della Rubrica Telefonica

Gruppo 16

Ciasullo Livia, Ciniello Lorenzo, Dong Luisa, Goffredo Vincenzo

Scelte Architetturali

Pattern MVC

Il sistema utilizza il pattern Model-View-Controller per separare :

- **Model (Rubrica, Contatto):** gestione dei dati
- **View (FXML):** interfaccia utente .
- **Controller:** logica applicativa e gestione eventi

Oltre al modello Object-oriented che è il più adatto al linguaggio di programmazione scelto per il progetto, ovvero Java.

Principi SOLID applicati

Single Responsibility:

- **Contatto:** gestisce solo i dati di un contatto
- **Rubrica:** gestisce solo la collezione di contatti
- **NumeroTel/Email:** formattazione di numeri di telefono e indirizzi e-mail (separati quindi da Contatto)
- **Main:** gestisce l'avvio del programma e la view

Ogni classe quindi ha i propri compiti e le proprie responsabilità.

Open/Closed:

- Facile aggiungere nuovi tipi di contatti o informazioni estendendo le classi base

Decisioni di Design

Persistenza

- Serializzazione CSV per salvare/caricare la rubrica su file esterni all'applicazione

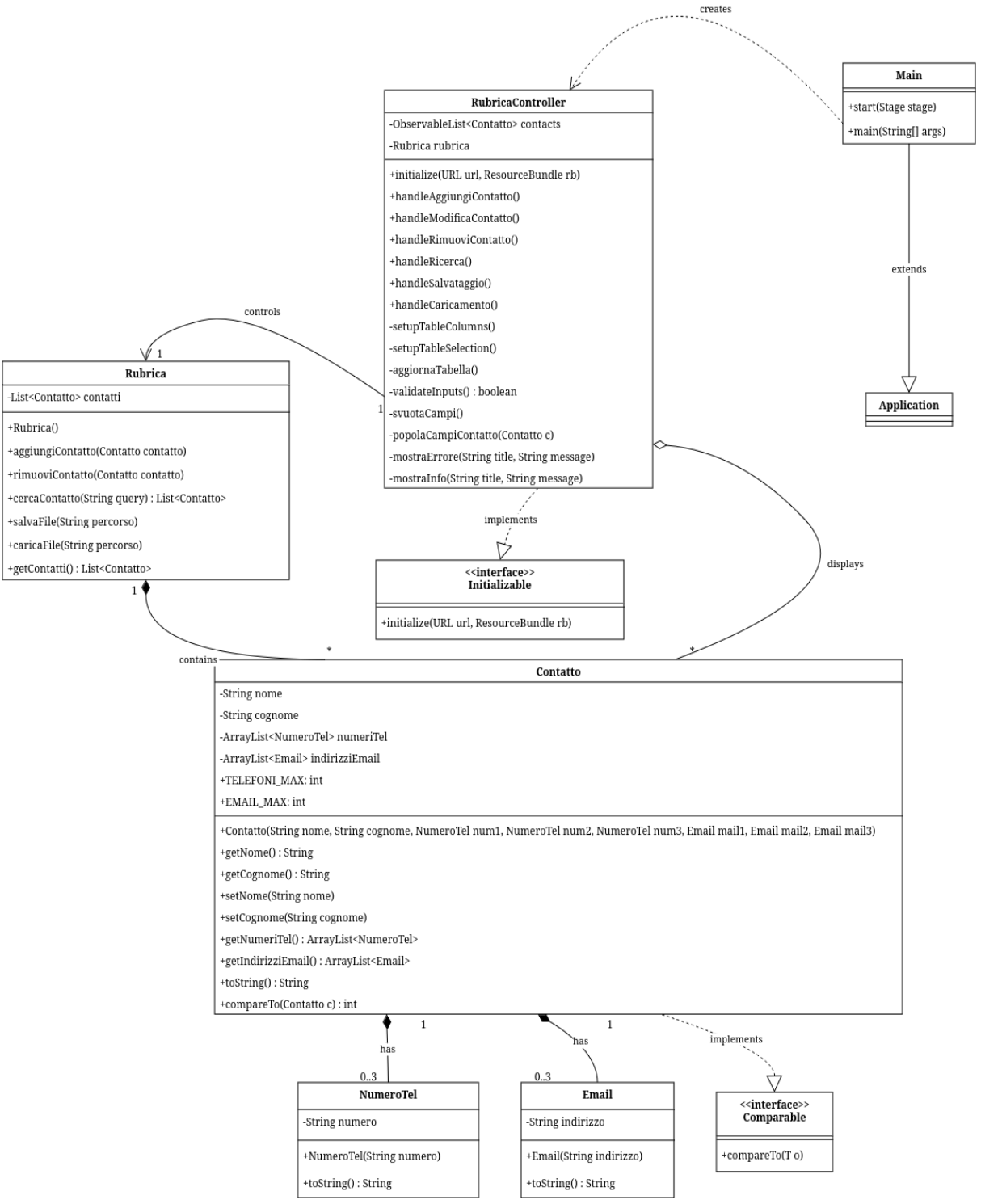
Ricerca e Ordinamento

- Implementazione di **Comparable** per ordinamento naturale
- Ricerca case-insensitive su nome/cognome

Gestione Errori

- Validazione input utente nel controller
- Comportamenti errati in caso di scrittura o di lettura del file csv

Diagramma delle classi



Dipendenze tra classi

Tra **RubricaController** e **Rubrica** (freccia semplice con 1:1):

- Il controller ha una relazione di associazione con una singola **Rubrica**;
- Le cardinalità 1:1 indicano che ogni controller è associato ad una sola **Rubrica** e viceversa;
- Il controller usa la **Rubrica** per gestire le operazioni sui contatti.

Tra **RubricaController** e **Contatto** (freccia con rombo vuoto):

- Il controller si serve della classe **Contatto,** tramite una lista osservabile, per aggiornare le informazioni sulla view

Tra **Rubrica** e **Contatto** (freccia con rombo pieno e 1:*):

- Il rombo pieno indica una composizione (relazione forte)
- La **Rubrica** è composta da una lista di Contatti
- La cardinalità 1:* indica che una **Rubrica** può contenere molti **Contatti**

Tra **Contatto** e **NumeroTel**/**Email** (frecce con rombo pieno e 1:0..3):

- Anche tra **Contatto** e le classi **NumeroTel** e **Email** ci sono composizioni
- Un **Contatto** può avere da 0 a 3 numeri di telefono (cardinalità 0..3)
- Un **Contatto** può avere da 0 a 3 email (cardinalità 0..3)

Le frecce tratteggiate verso le interfacce:

- **RubricaController** implementa l'interfaccia **Initializable**
- **Contatto** implementa l'interfaccia **Comparable** per permettere il confronto dei contatti

La sezione del progetto che si occupa della GUI, non è rappresentata nel diagramma ma è implementata attraverso un file FXML **RubricaView**.

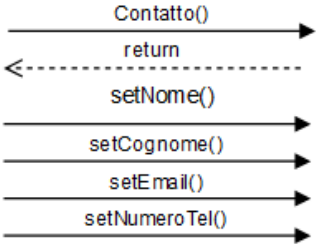
La classe **Main** estende **Application** per poter avere accesso ai metodi di JavaFX che inizializzano l'interfaccia grafica e si legano alle view FXML.

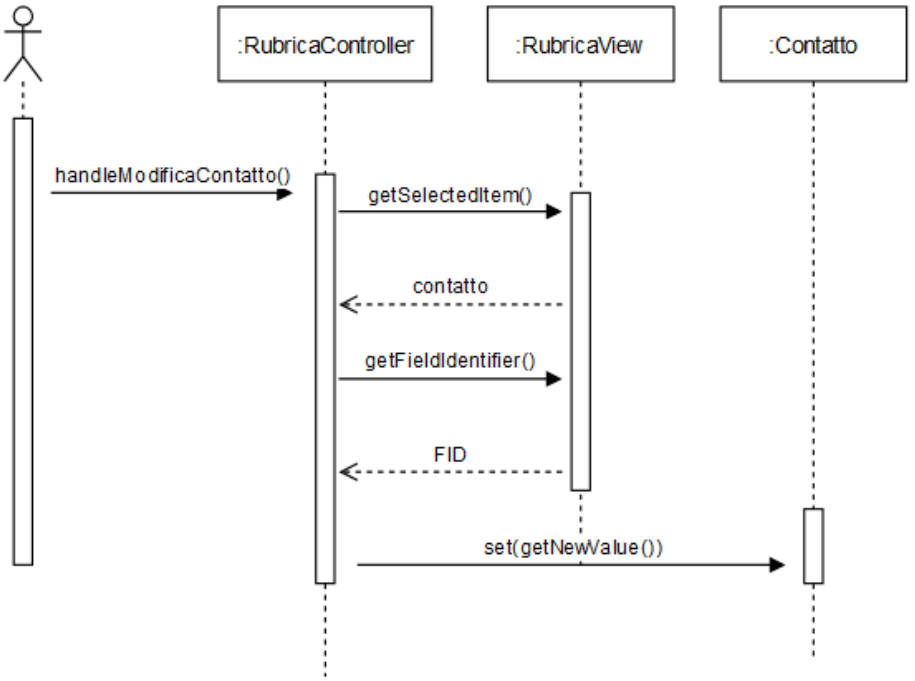
La sintassi è quella classica dei diagrammi di classe con UML, con gli attributi e i metodi di ogni classe, con i loro modificatori di accesso:

- + indica che quell'attributo o metodo è public
- - indica che l'attributo o il metodo è private
- Sono specificati i tipi di ritorno per i metodi

Diagrammi delle sequenze

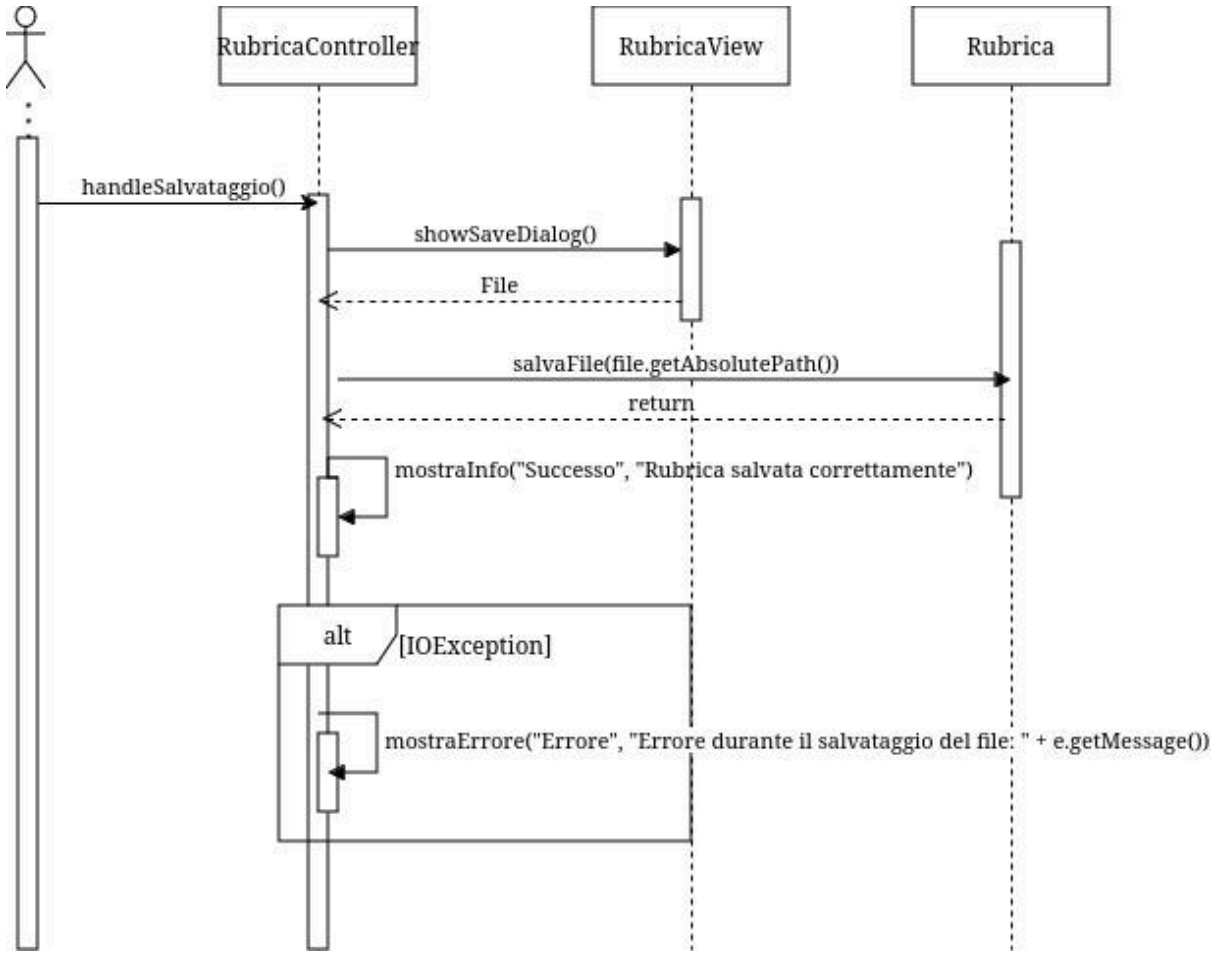
Sequenza di modifica del contatto



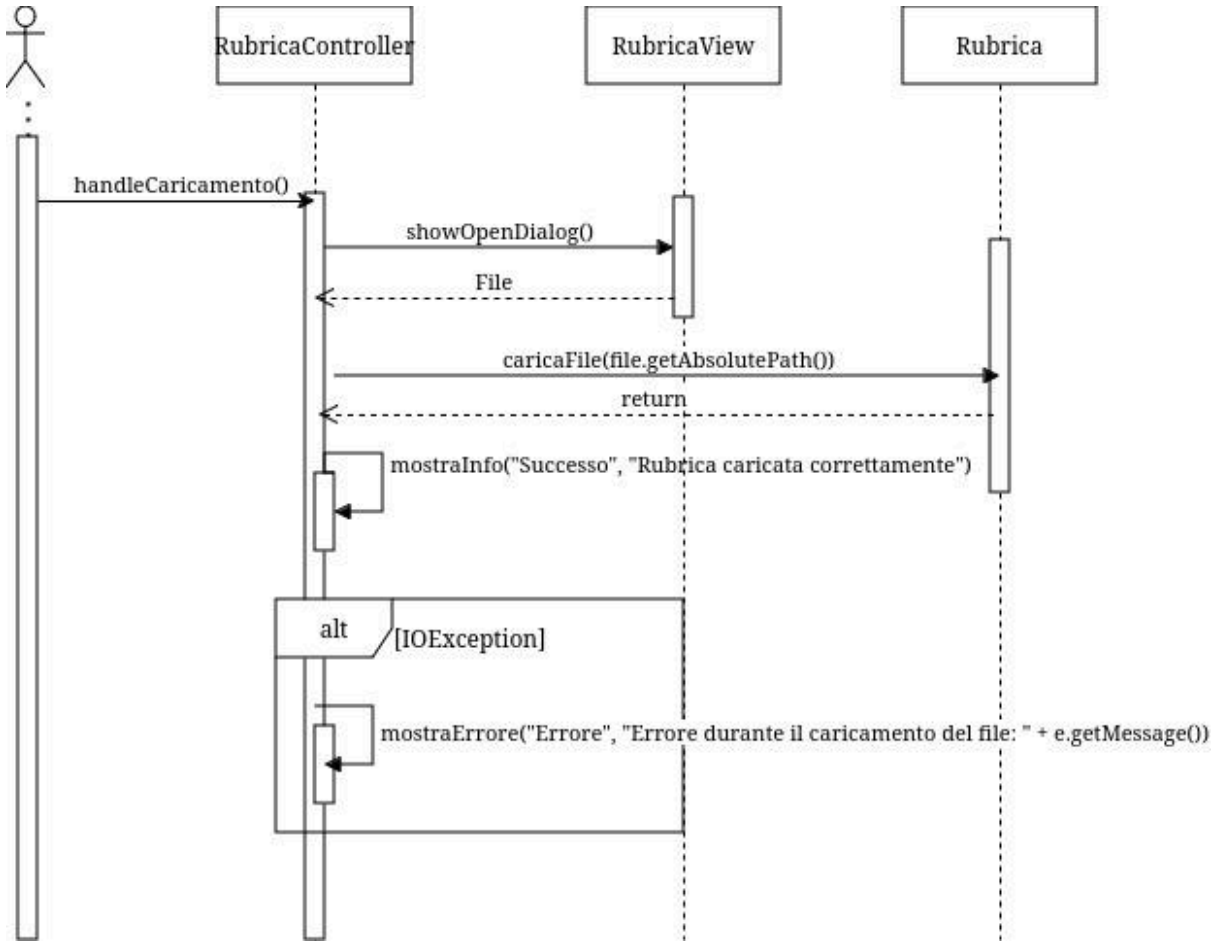


NOTE: la set è generica e non specifica per il campo selezionato perchè in base al fliedID ricevuto dalla view si attiva una set diversa

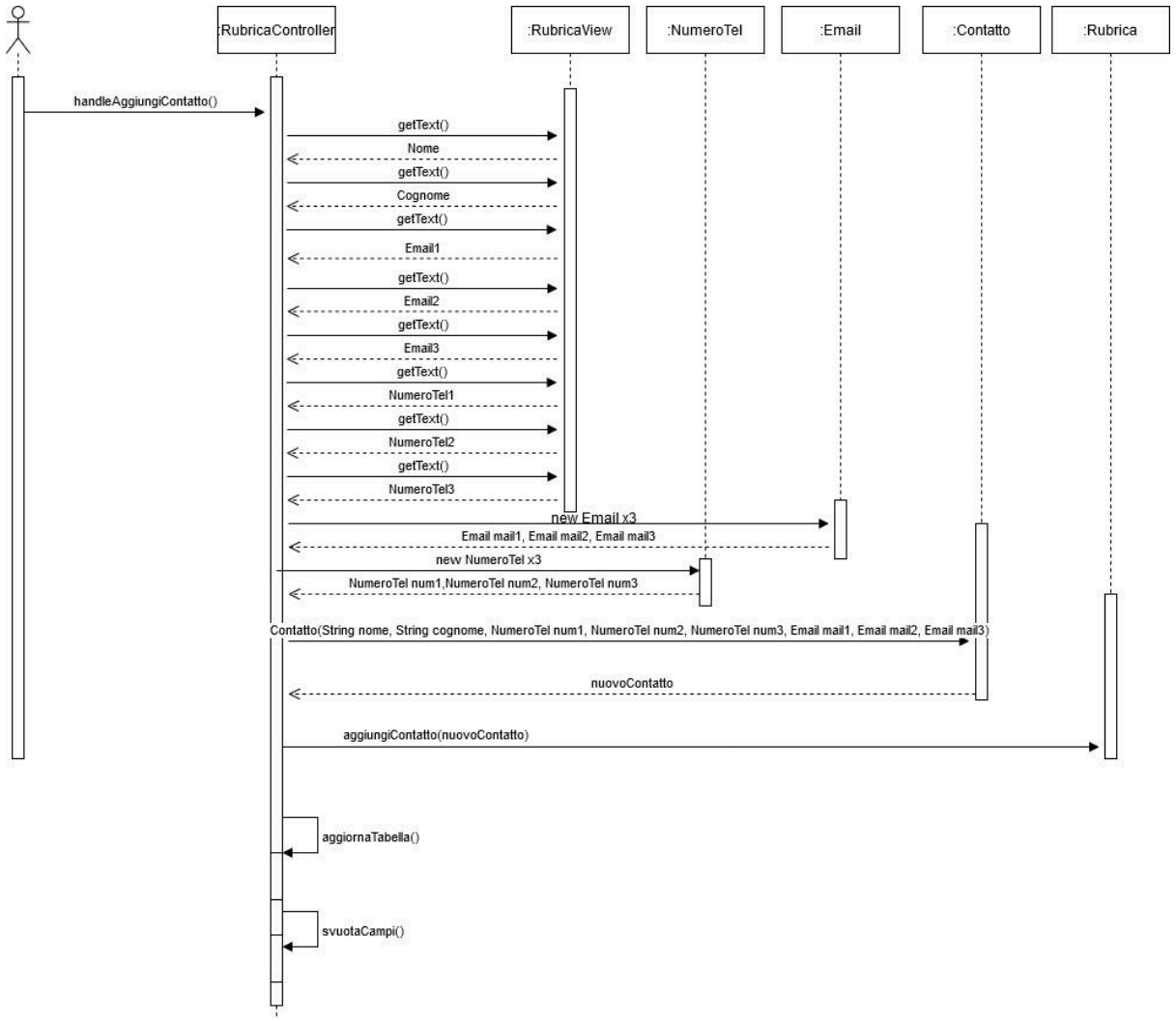
Sequenza di salvataggio della rubrica su file esterno



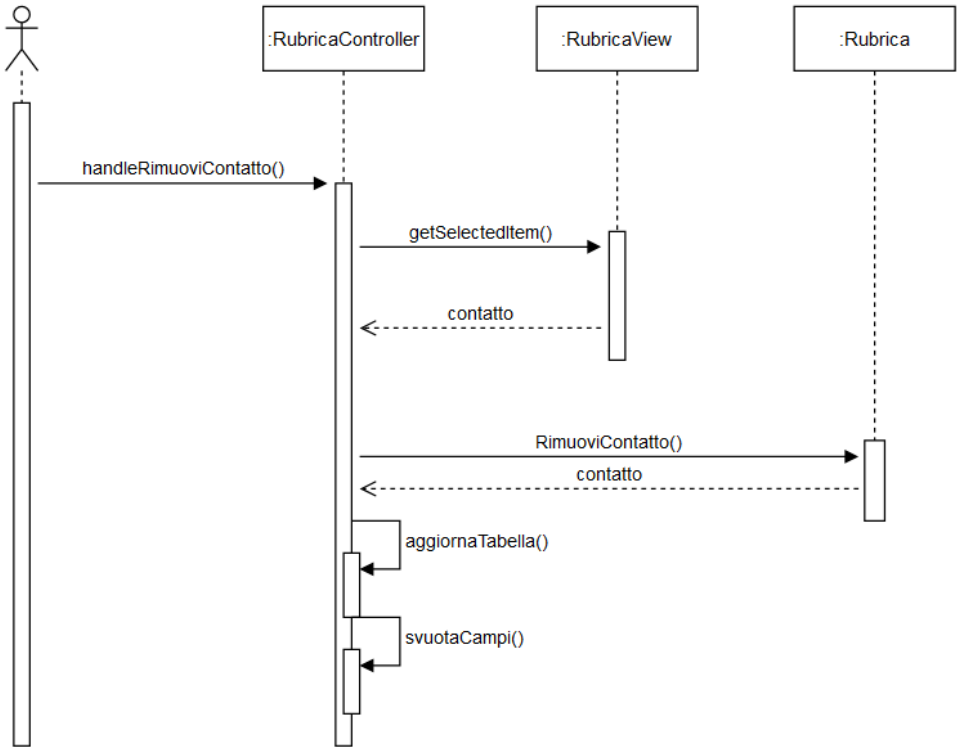
Sequenza di caricamento della rubrica da file esterno



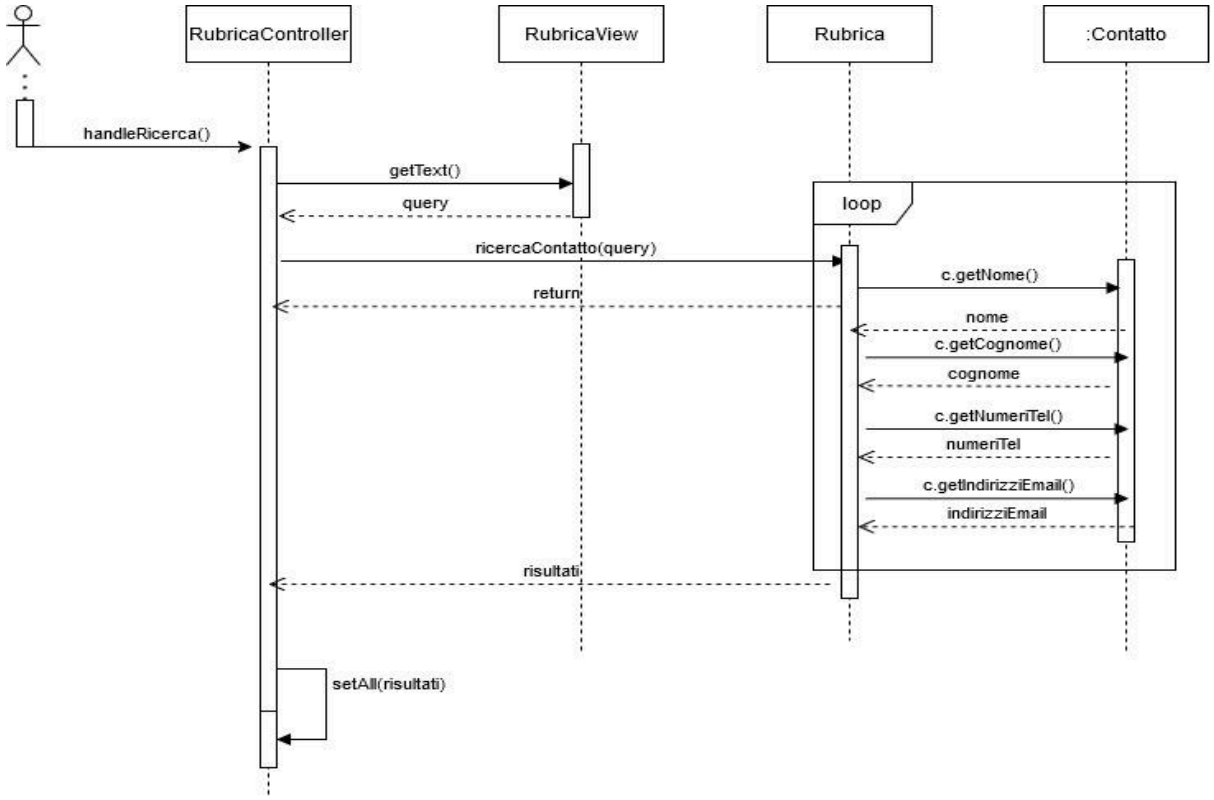
Sequenza di aggiunta di un contatto



Sequenza di rimozione di un contatto



Sequenza di ricerca di un contatto



Coesione e accoppiamento

Coesione

Classe	Coesione	Descrizione
NumeroTel	Funzionale	Contiene solo i metodi fondamentali alla gestione della struttura dati
Email	Funzionale	Contiene solo i metodi fondamentali alla gestione della struttura dati
Contatto	Funzionale	Contiene i metodi che lavorano insieme per realizzare un compito ben definito quale la costruzione del contatto
Rubrica	Comunicazionale e sequenziale	Contiene sia dei metodi che lavorano sugli stessi dati (es aggiungiContatto, rimuoviContatto) sia metodi che utilizzano come input gli output di altri metodi dello stesso modulo (es. cercaContatti)
RubricaController	Comunicazionale	Con i suoi metodi ha il compito di gestire i dati e gli eventi che passano da Rubrica alla view e viceversa

Accoppiamento

Classi	Accoppiamento	Descrizione
Contatto-NumeroTel	Per dati	NumeroTel fornisce a Contatto solo il numero di telefono, che è l'unica informazione necessaria a Contatto per fare le sue operazioni (stampa, inizializzazione)
Contatto-Email	Per dati	Email fornisce a Contatto solo l'indirizzo mail, che è l'unica informazione necessaria a Contatto per fare le sue operazioni (stampa, inizializzazione)
Rubrica-Contatto	Per dati	Contatto fornisce a Rubrica l'intera struttura dati contenente i vari contatti, ma richiama i singoli metodi (set, get ecc) che servono sui singoli contatti in modo che Contatto abbia in input solo il singolo contatto su cui deve operare
Rubrica-RubricaController	Per timbro	Le due classi comunicano tra di loro tramite alcuni metodi per consentire il corretto comportamento dell'applicazione, ma alcune informazioni di Rubrica potrebbero non servire a RubricaController

Packages

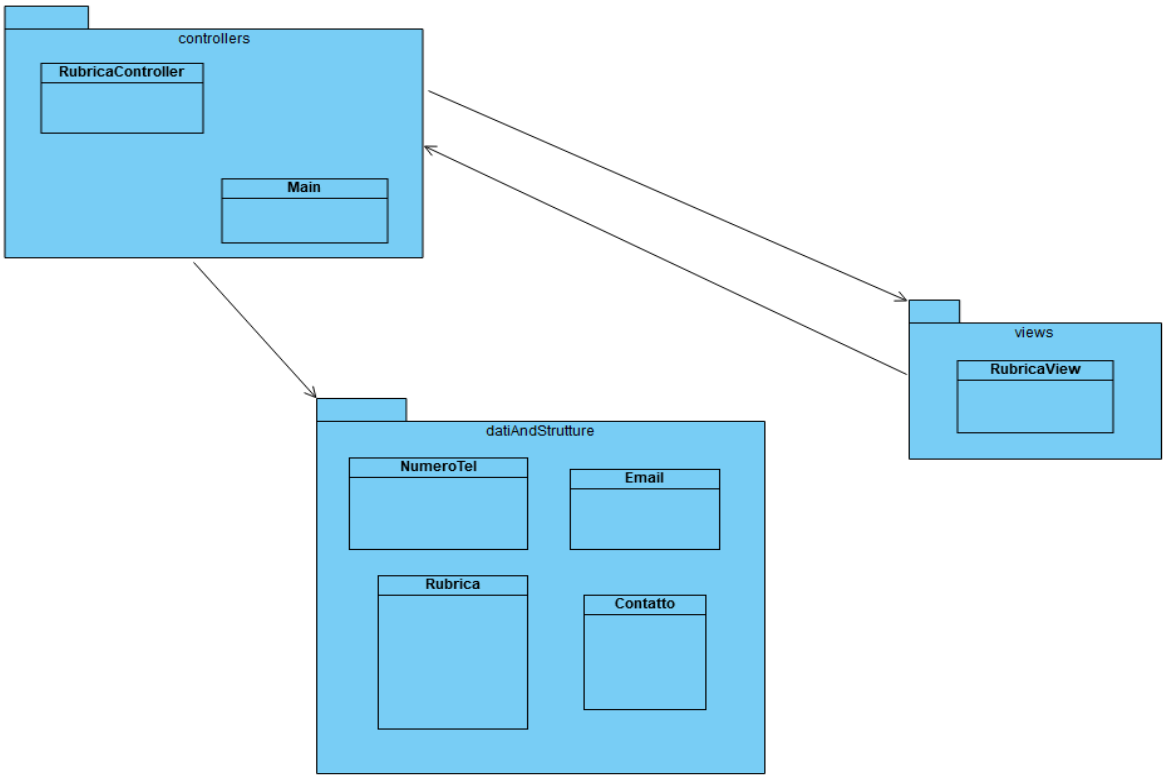


Tabella dei packages

Nome	Descrizione
controllers	Contiene la classe **RubricaController** che gestisce la comunicazione tra la user interface e il programma vero e proprio e il **Main** che è il punto di ingresso dell'applicazione
datiAndStrutture	Contiene le classi responsabili della definizione di un contatto (**Contatto**, **NumeroTel** e **Email**) così come la collezione che li contiene (**Rubrica**)
views	Contiene i file FXML responsabili della costruzione della GUI con cui l'utente interagirà

Il package controllers comunica con views perchè il controller usa delle funzioni della view (per esempio quelle che permettono di ottenere le stringhe inserite nei campi di testo); views dipende da controller perchè è il main che assembla la scene e genera la view, e la view viene aggiornata con i dati forniti dal controller; controllers dipende da datiAndStrutture perché usa i metodi di **Contatto** per istanziare nuovi contatti e ottenere campi specifici di questi ultimi e perchè usa i metodi offerti da **Rubrica** quando crea contatti, li elimina ecc.