Università degli Studi di Salerno

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Progetto

di

Ingegneria Del Software

Design

Autori:

Carmine Terracciano 0612708084
Ferdinando Paparo 0612708395
Gabriele Pannuto 0612703845
Pasquale Piccolo 0612708338

Anno Accademico 2024/2025

Sommario

Diagramma Delle Classi

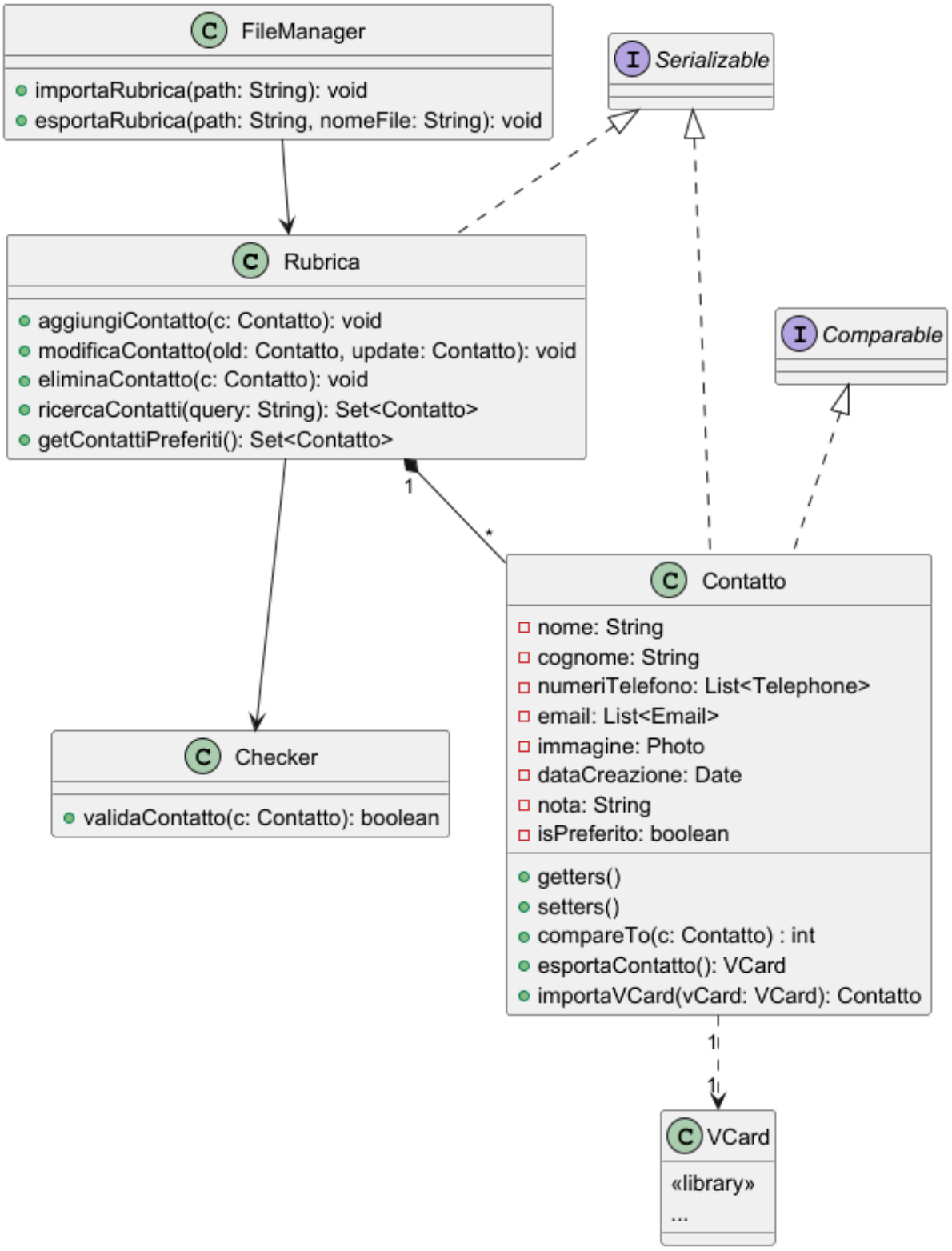
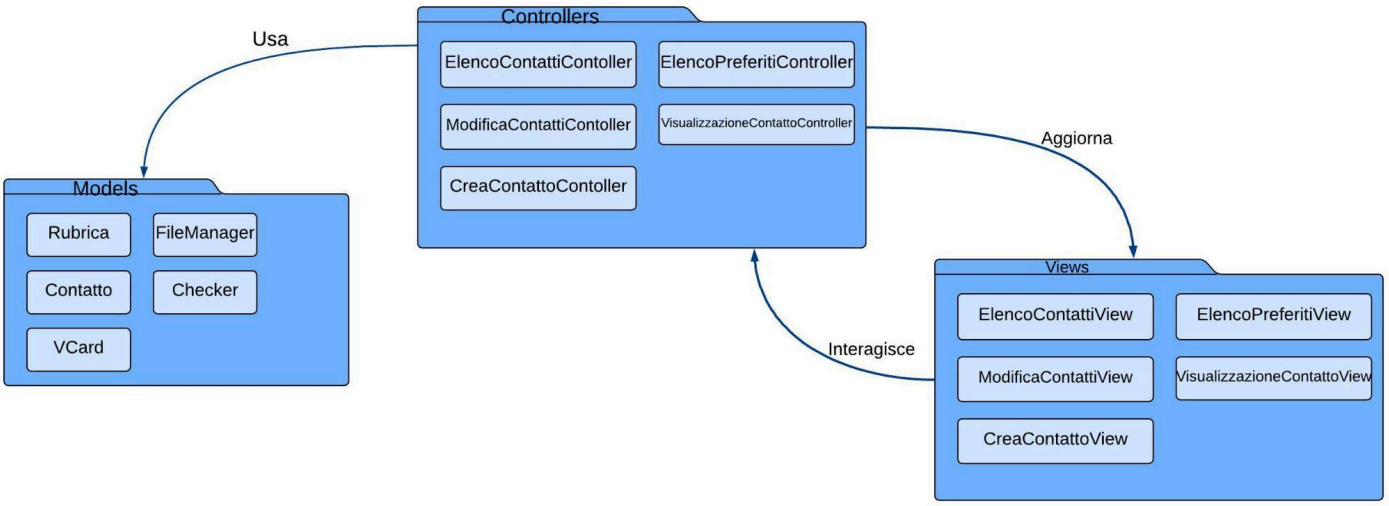


Diagramma Dei Package



Descrizione

Il diagramma rappresenta un'architettura basata sul pattern MVC (Model-View-Controller) per una rubrica. Le frecce indicano le dipendenze tra i componenti.

Il Package **Model** contiene gli elementi che rappresentano i dati e la logica di gestione della rubrica.

- È utilizzato direttamente dal package **Controller**, che lo sfrutta per operazioni come l'aggiornamento dei dati o la validazione.

Il package **Controller** funge da intermediario tra la logica di business (**Model**) e l'interfaccia utente (**View**). Ogni classe **Controller** è specializzata in un'area funzionale dell'applicazione.

- **Interagisce con il Model:** Usa le classi del package **Model** (es. Rubrica) per eseguire le operazioni richieste.
- **Aggiorna la View:** Passa i dati processati alla **View** per farli visualizzare all'utente.

Il package **View** rappresenta l'interfaccia utente dell'applicazione. Ogni classe **View** è responsabile di un aspetto specifico della visualizzazione.

- **Interagisce con il Controller:** Riceve dati dal **Controller** e visualizza i risultati delle operazioni eseguite.
- **Non comunica direttamente con il Model:** Qualsiasi richiesta di modifica o aggiornamento passa attraverso il **Controller**.

Flussi principali del diagramma

1. Interazione tra Utente e **View**:
    - L'utente interagisce con un’interfaccia grafica della **View** (ad esempio, `ElencoContattiView` o `RicercaContattoView`) per compiere un'azione, come visualizzare un elenco o cercare un contatto.
2. Comunicazione tra **View** e **Controller**:
    - La **View** invia la richiesta ricevuta dall'utente a un **Controller** appropriato. Ad esempio, un click su "Modifica" potrebbe coinvolgere il `ModificaContattoController`.
3. Interazione tra **Controller** e **Model**:
    - Il **Controller** utilizza il **Model** per eseguire operazioni richieste, come aggiungere, modificare o cercare contatti. Ad esempio, il `ModificaContattoController` potrebbe chiedere alla Rubrica di aggiornare i dati di un contatto.
4. Aggiornamento della **View** da parte del **Controller**:
    - Una volta completata l'operazione, il **Controller** aggiorna la **View** con i risultati (ad esempio, mostrando la lista aggiornata dei contatti o un messaggio di errore).

Coesione e Accoppiamento

Coesione

La coesione rappresenta una caratteristica fondamentale di un modulo software, misurando quanto le parti che sono incluse nello stesso modulo sono legate tra di loro. Si tratta di una caratteristica *intra-modulo*.
Un'elevata coesione significa che le parti di un modulo software sono fortemente connesse tra loro e lavorano insieme per svolgere una determinata funzione.

- **Model**: Il package Model è altamente coeso, poiché tutte le sue classi (Rubrica, FileManager, Contatto, Checker, VCard) si occupano esclusivamente della logica di gestione dei dati.
- **Controller**: Il package Controller ha un'alta coesione, dato che ciascun componente al suo interno è specializzato in un'area specifica dell'applicazione (es. ModificaContattoController gestisce esclusivamente le operazioni di modifica dei contatti).
- **View**: I componenti della View sono anch'essi ben coesi, ognuno dedicato a un aspetto specifico dell'interfaccia utente (es. ElencoContattiView, RicercaContattoView). Questi ricevono i dati dal Controller e li mostrano all'utente, senza gestire logiche aggiuntive.

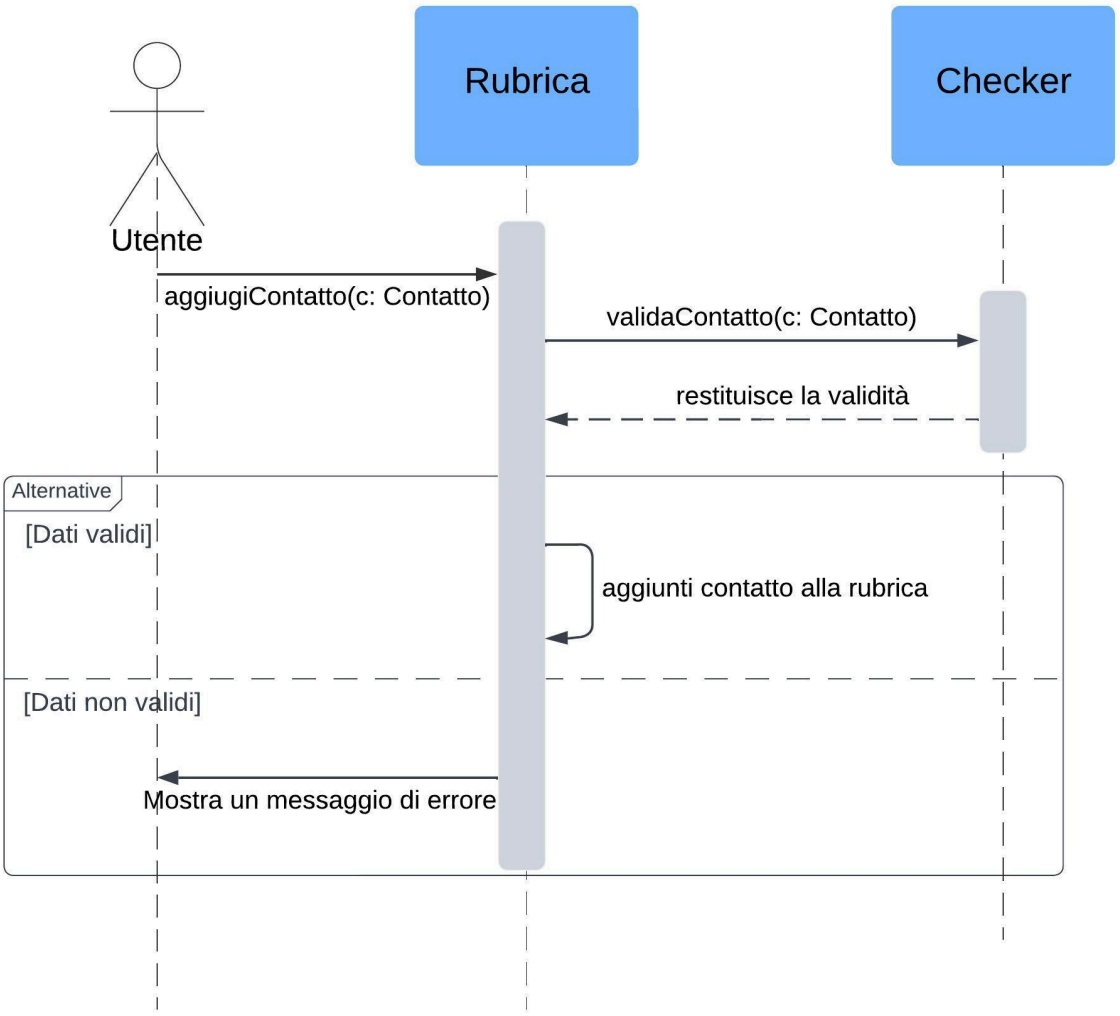
Accoppiamento

L'accoppiamento misura il grado di interdipendenza tra moduli software diversi. Un basso accoppiamento rappresenta un obiettivo architetturale importante da raggiungere poiché consente una maggiore flessibilità del sistema, facilitando gli interventi di manutenzione, modificabilità e potenziale scalabilità del software.

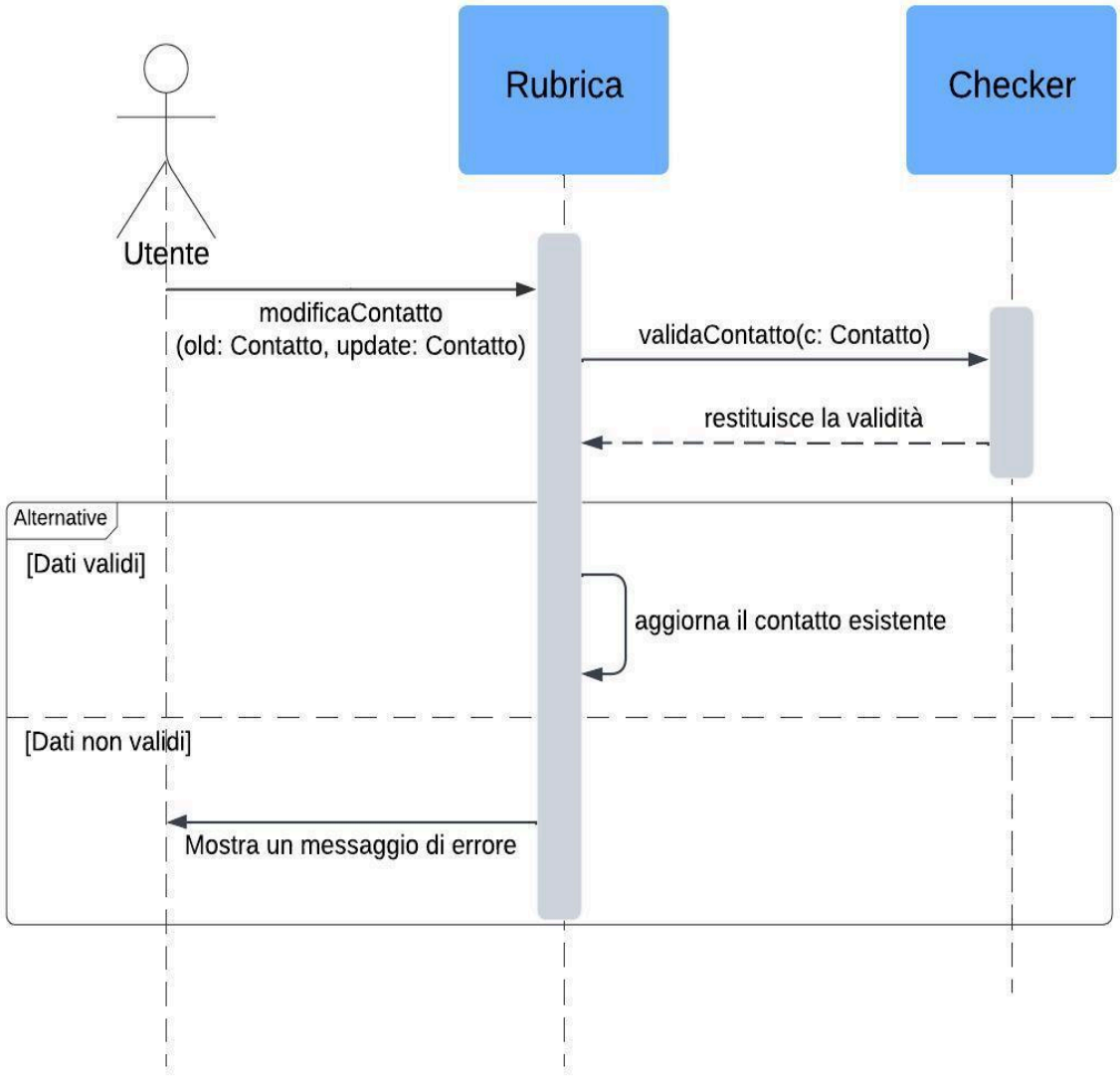
- Seguendo il pattern MVC (Model-View-Controller) siamo in grado di garantire un basso livello di accoppiamento grazie alla separazione delle responsabilità tra i diversi moduli, già descritti precedentemente.

Diagrammi Di Sequenza

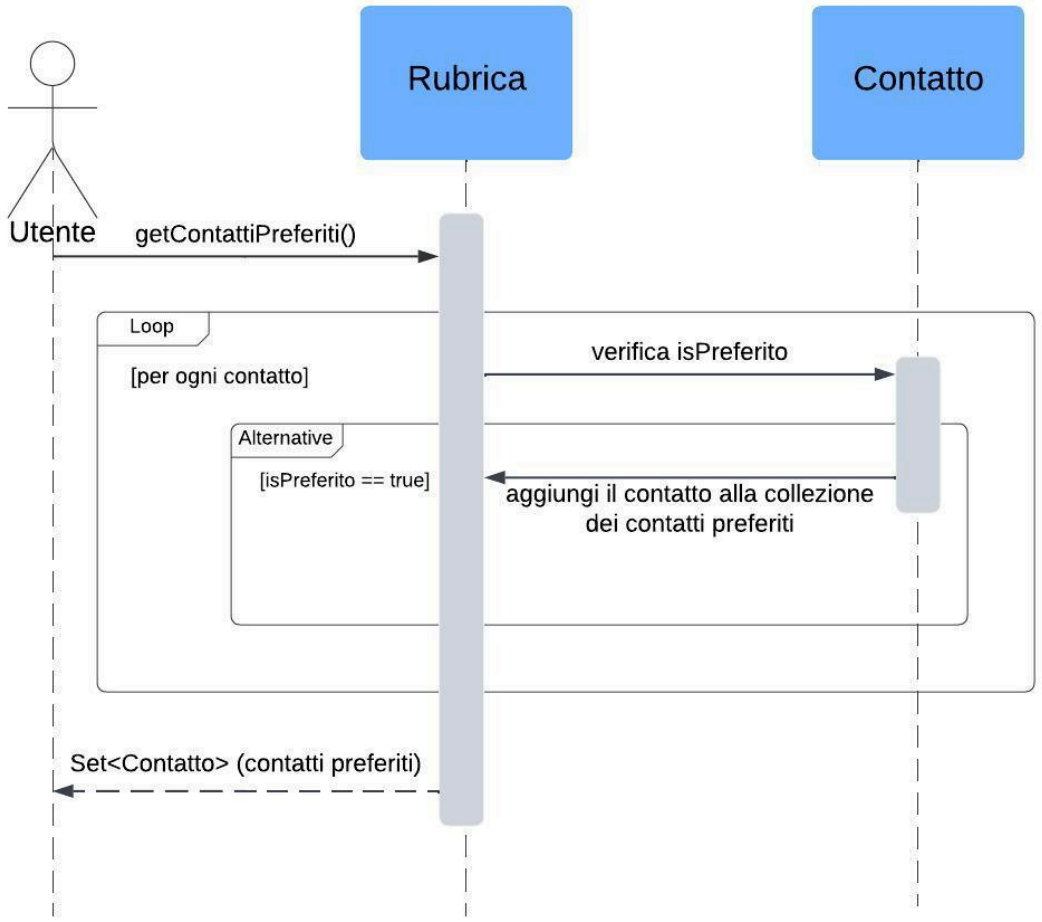
Aggiunta di un Nuovo Contatto alla Rubrica



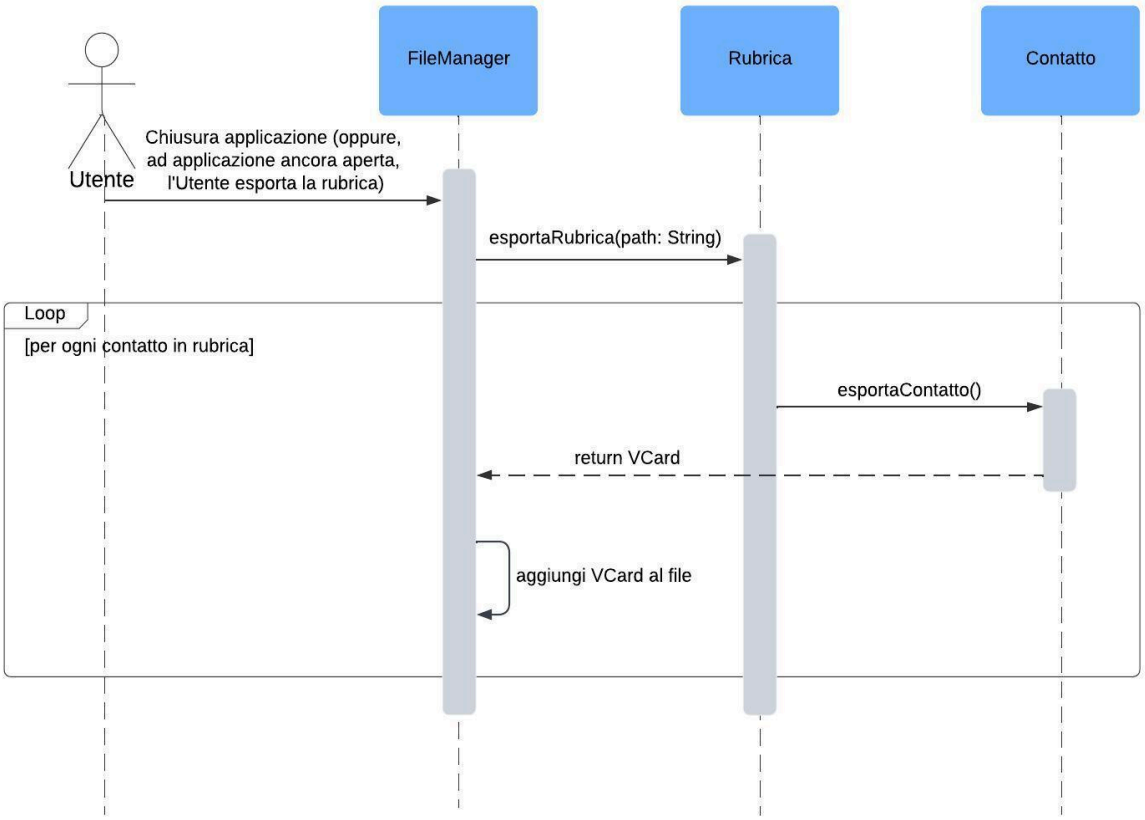
Modifica dei Dati di un Contatto nella Rubrica



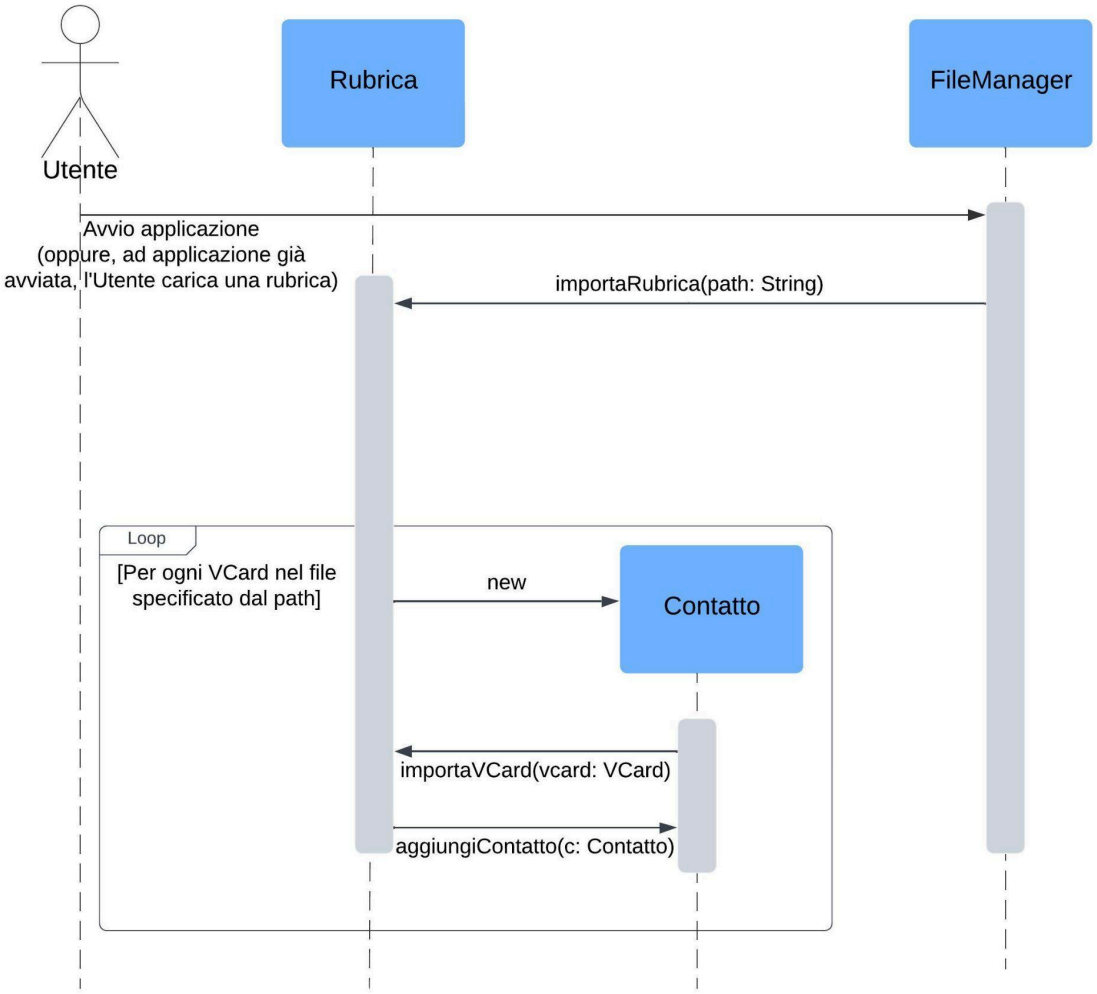
Recupero dei Contatti Preferiti dalla Rubrica



Esportazione dei Contatti della Rubrica su File

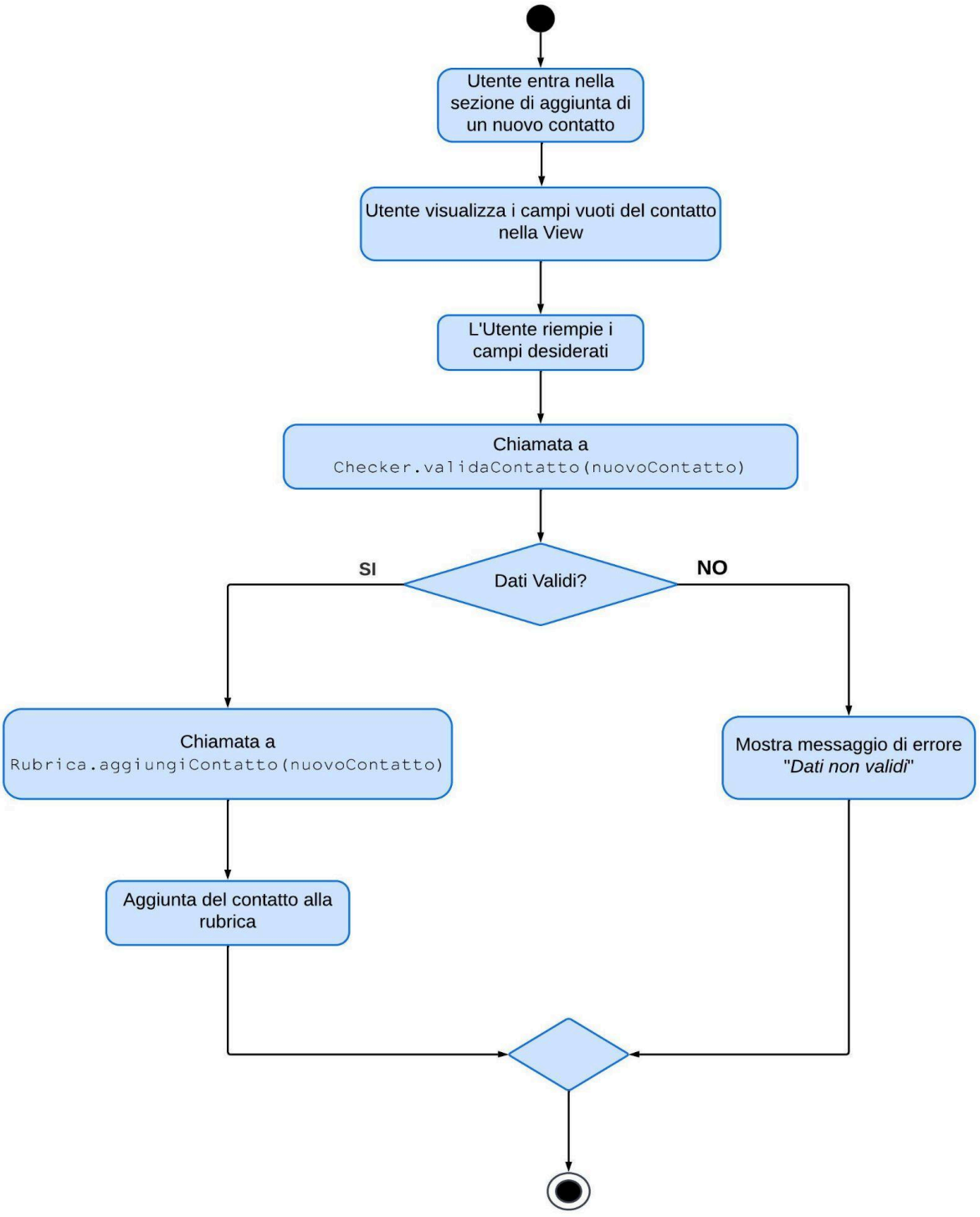


Importazione dei Contatti da un File nella Rubrica

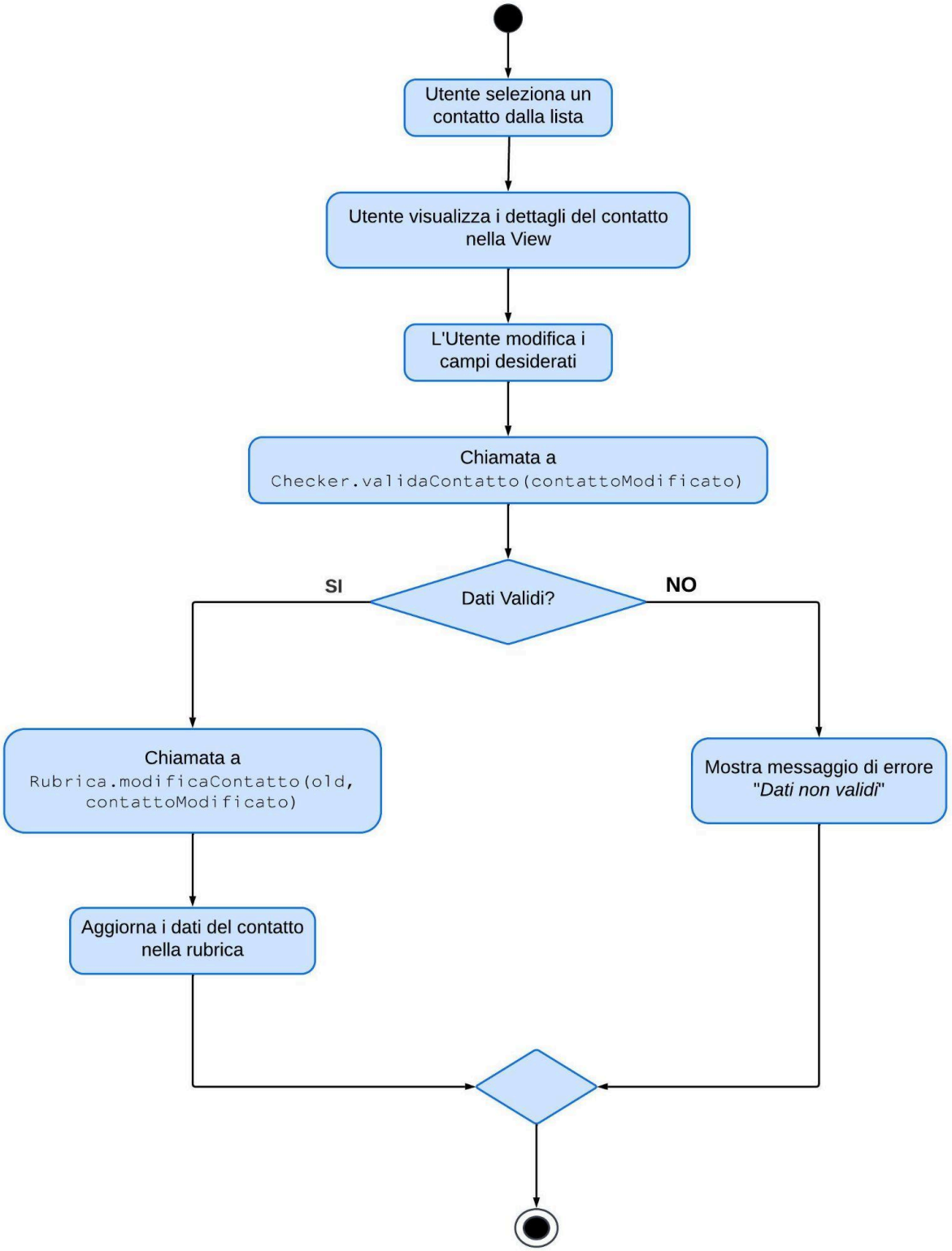


Diagrammi Delle Attività

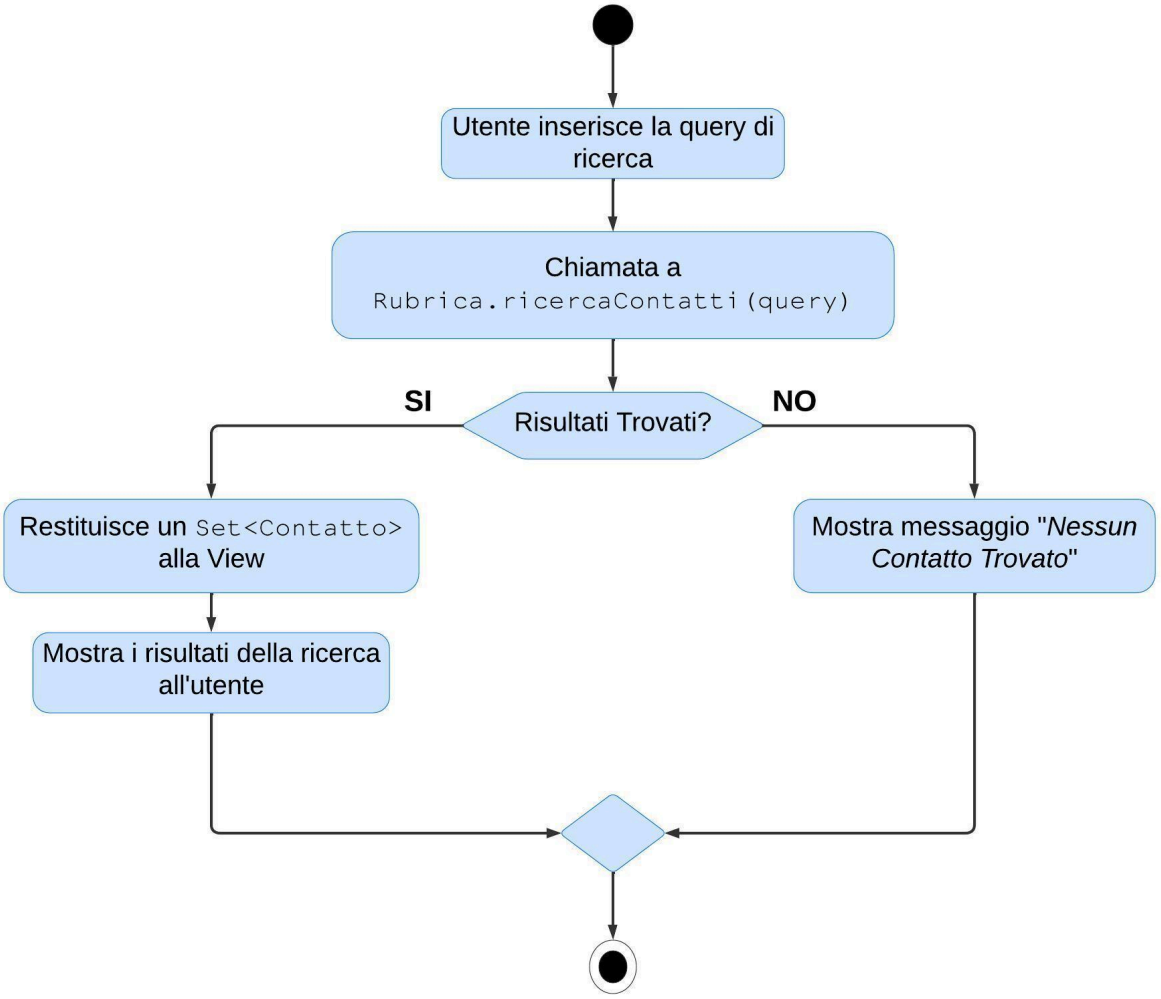
Aggiungi Contatto



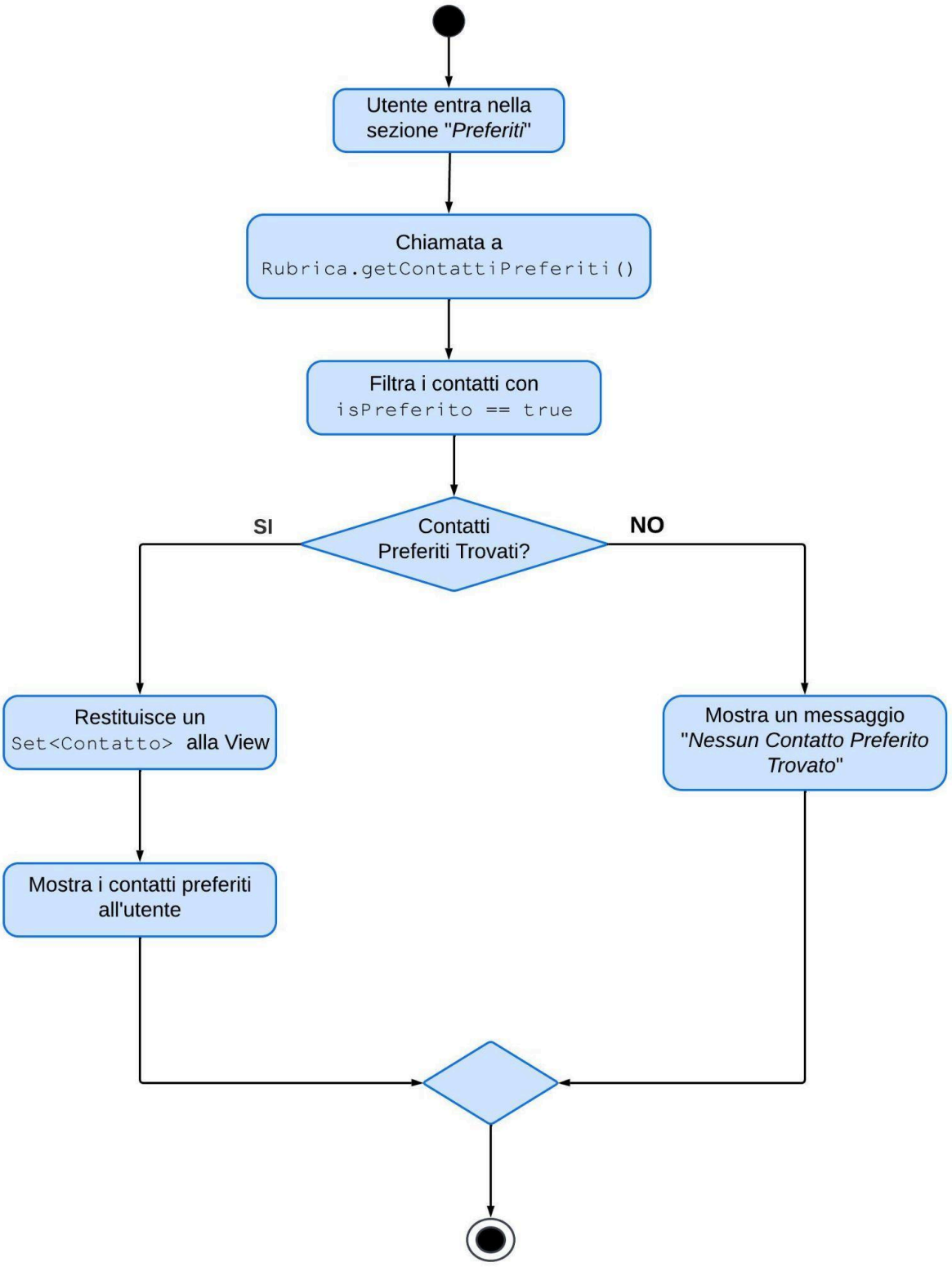
Modifica Contatto



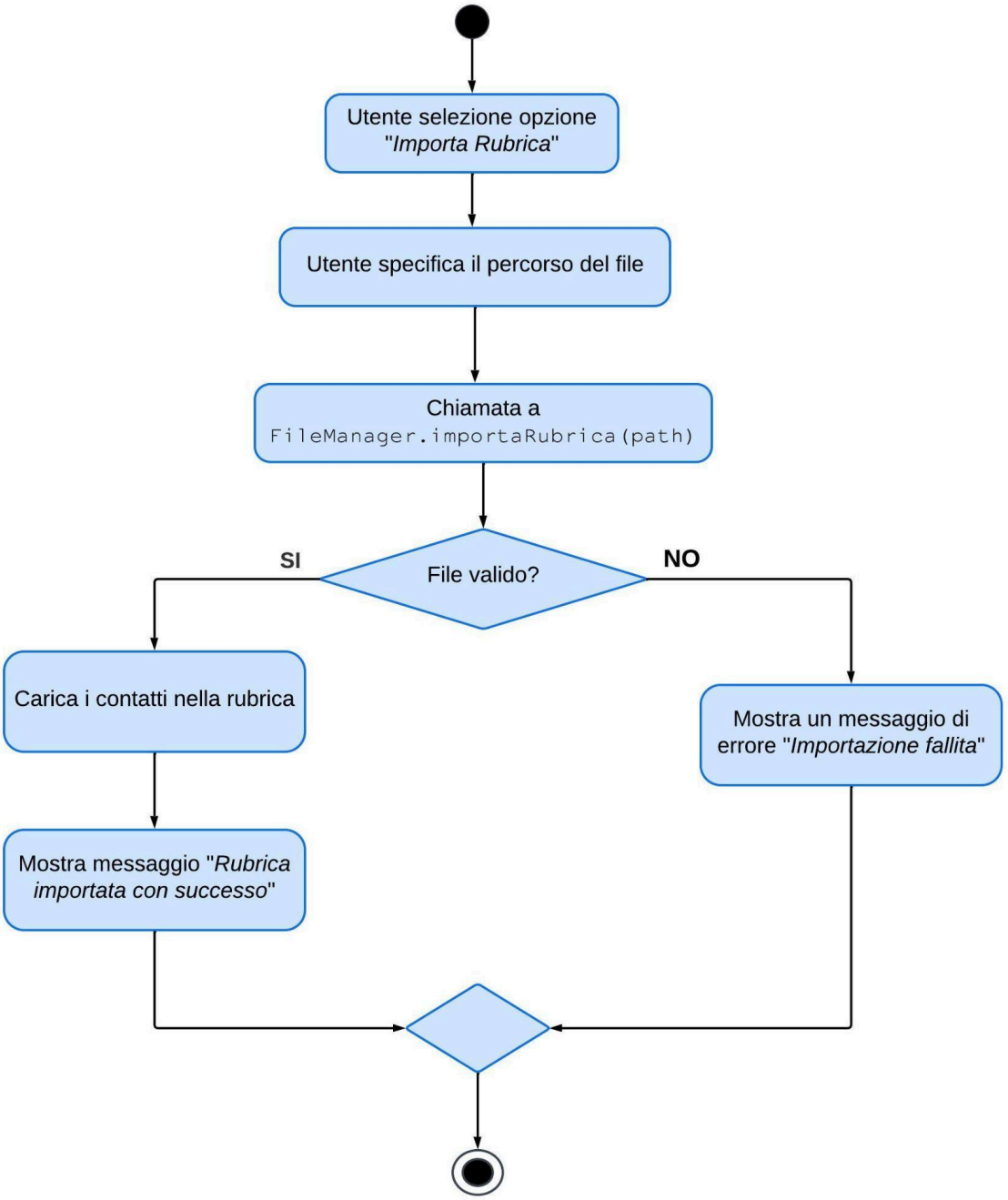
Ricerca Contatti



Get Contatti Preferiti



Importa Rubrica



Esporta Rubrica