# Documento di Design della Rubrica Telefonica

## 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

### Principi SOLID applicati

- **Single Responsibility:**
  - **Contatto:** gestisce solo i dati di un contatto
  - **Rubrica:** gestisce solo la collezione di contatti
  - **NumeroTel/Email:** validazione e formattazione
- **Open/Closed:**
  - Facile aggiungere nuovi tipi di contatti o informazioni estendendo le classi base
- **Interface Segregation:**
  - Interfacce minimali per ogni componente
  - Separazione tra logica di business e presentazione

### Coesione e Accoppiamento

#### Alta Coesione

- Ogni classe ha responsabilità ben definite
- **Contatto**: gestisce solo i dati relativi a un singolo contatto
- **Rubrica**: gestisce solo operazioni sulla collezione

#### Basso Accoppiamento

- **Controller** comunica con il **Model** tramite interfacce
- **View** completamente separata dalla logica di business
- Utilizzo di eventi per la comunicazione **View**-**Controller**

## Decisioni di Design

### Validazione

- Validazione dei numeri di telefono e email a livello di classe specifica
- Controlli di business logic nel controller

### Persistenza

- Serializzazione CSV per salvare/caricare la rubrica

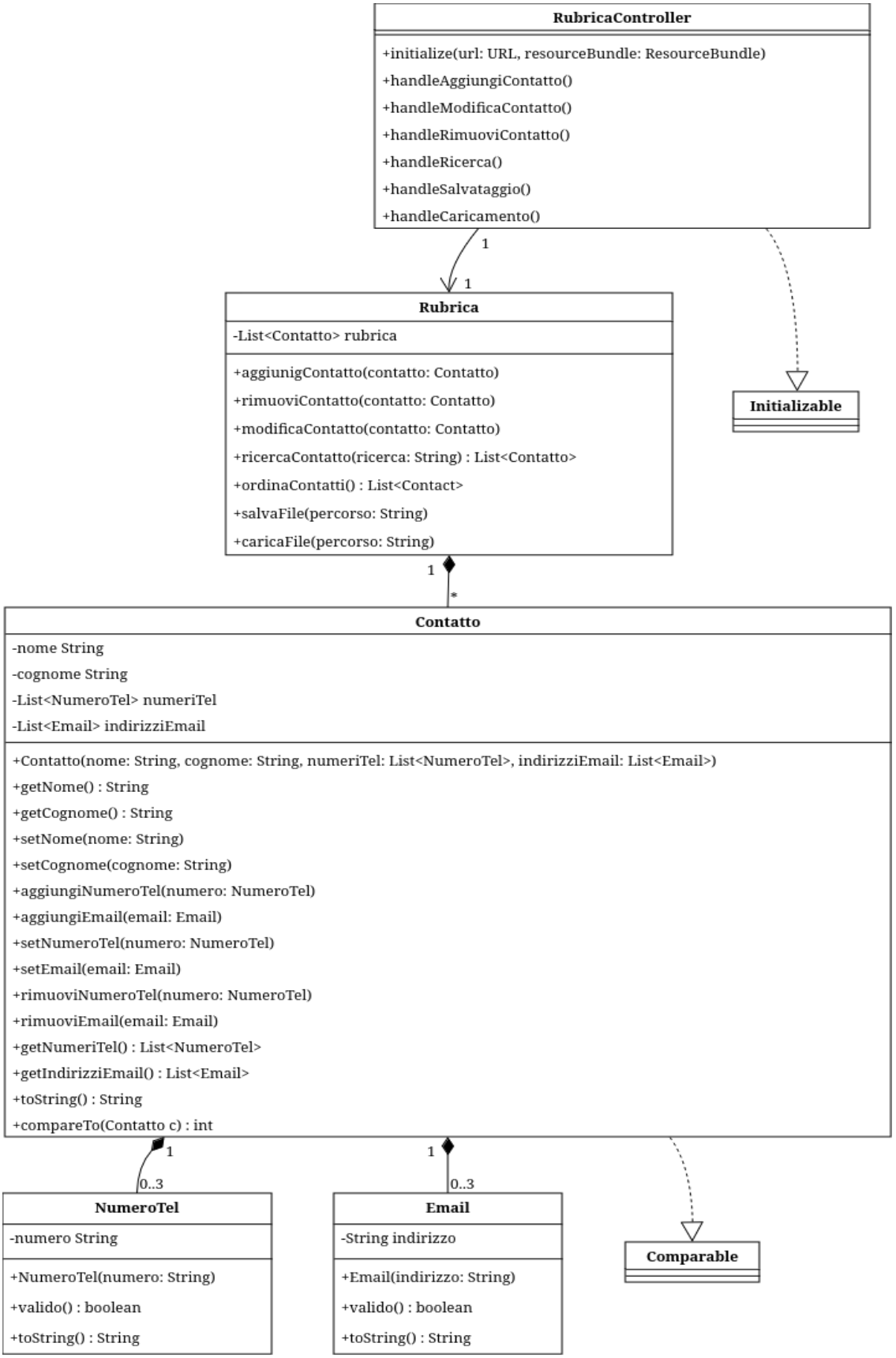
### Ricerca e Ordinamento

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

## Gestione Errori

- Utilizzo di eccezioni custom per gestire errori di business
- Validazione input utente nel controller

# Diagramma delle classi



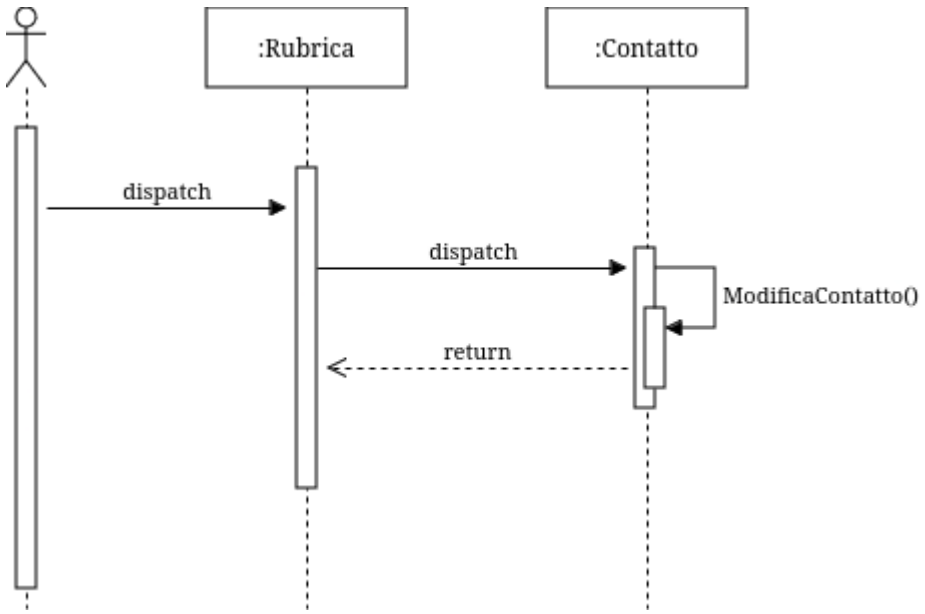
- Tra **RubricaController** e **Rubrica** (freccia semplice con 1:1):
    - Il controller ha una relazione di associazione con una singola **Rubrica**;
    - Le cardinalità 1:1 indicano che ogni controller è associato ad una sola **Rubrica** e viceversa;
    - Il controller usa la **Rubrica** per gestire le operazioni sui contatti.
- Tra **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/ordinamento dei contatti)

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:

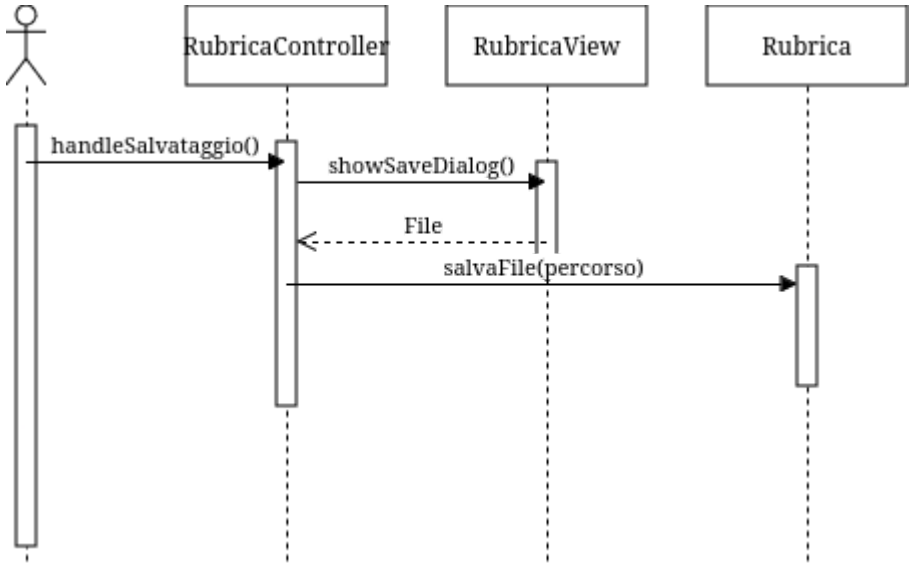
- + indica che quell'attributo o metodo è public
- - indica che l'attributo o il metodo è private
- Per ogni metodo sono i tipi di ritorno

# Diagrammi delle sequenze

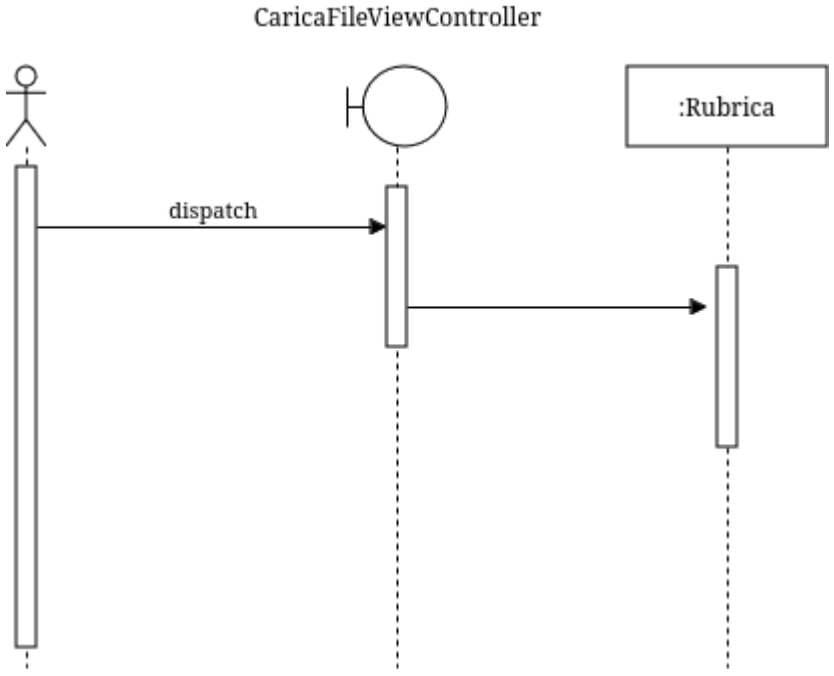
## Sequenza di modifica del contatto



## Sequenza di salvataggio della rubrica su file esterno




## Sequenza di caricamento della rubrica da file esterno



## Sequenza di aggiunta di un contatto