DESIGN

PROJECT REPORT

Gruppo 6

Avallone Emanuele
Barbato Emanuele
Barberio Gregorio

1 Diagramma delle classi

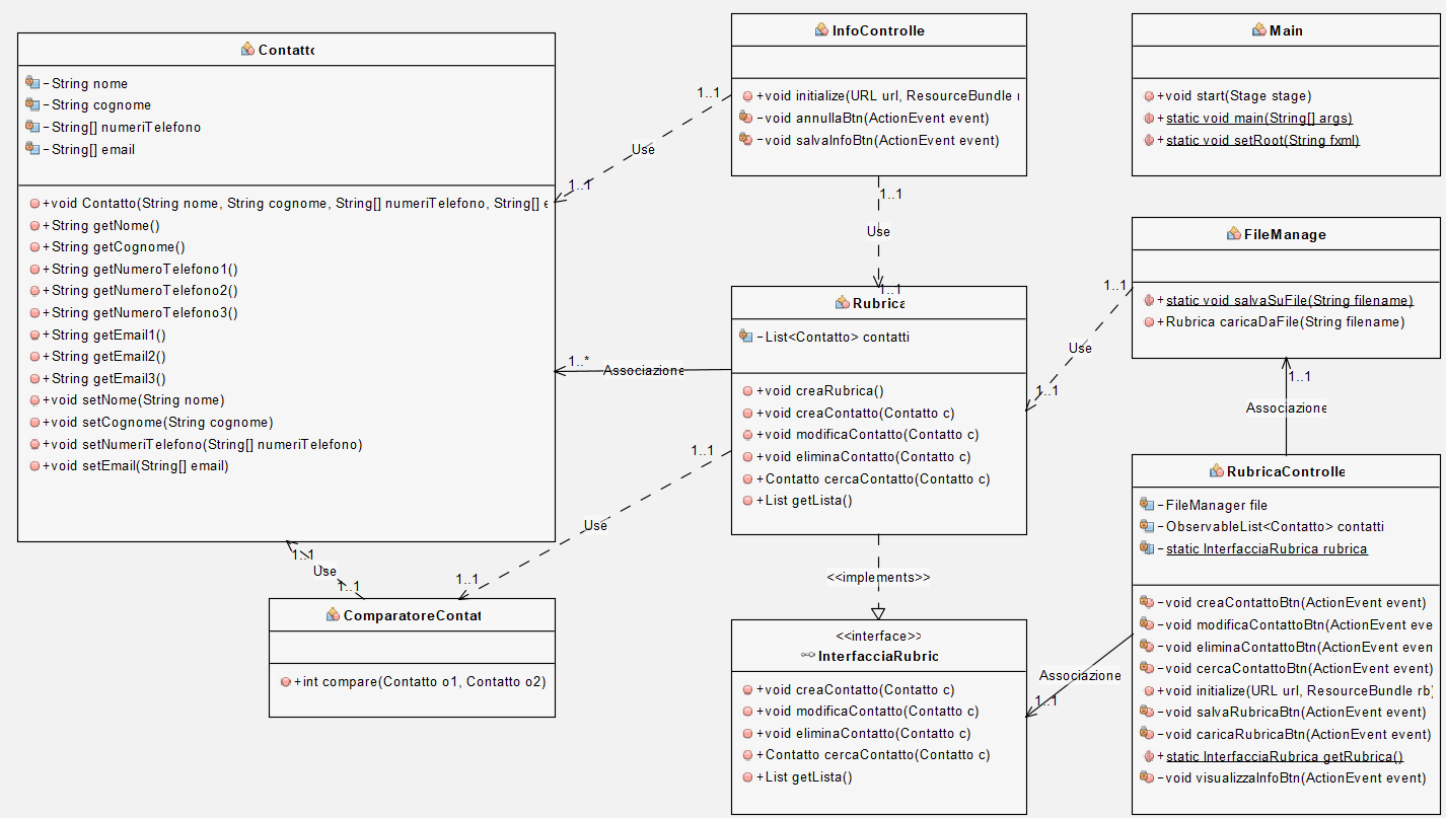

Figure 1: Diagramma delle classi

1.1 Chiarimenti sul diagramma delle classi

Per la progettazione è stato seguito un approccio alla scomposizione di tipo ObjectOriented. Il sistema è stato diviso in classi, ciascuna delle quali rappresenta un'entità del dominio del problema.
Le scelte progettuali sono state effettuate in accordo ai principi di buona progettazione orientata agli oggetti (**S.O.L.I.D.**) ed in modo da garantire al sistema un certo numero di Attributi di Qualità(**Q.A.**). Cooerentemente con quanto detto poc'anzi, la scelta di inserire, tra i moduli in cui è decomposto il sistema, l'interfaccia "*InterfacciaRubrica*" è finalizzata a fornire maggiore **Modularità**, **Riusabilità** e **Manutenibilità** al sistema stesso.

La **Modularità** è garantita dal fatto che, con l'aggiunta di un'interfaccia, si possono modificare in qualunque modo le classi (o la classe) che la implementano e tali modifiche non avranno alcun effetto sugli altri componenti del sistema. Ciò è realizzabile poichè il contratto pubblico dei metodi definiti dall'interfaccia rimarrà invariato.
Per quanto riguarda la **Riusabilità**, possiamo affermare che è garantita. L'interfaccia potrebbe essere implementata in qualsiasi altra applicazione senza alcun problema, gli unici vincoli da dover rispettare sono i contratti pubblici dei metodi dell'interfaccia stessa.
E' garantita anche una buona **Manutenibilità**. E' possibile correggere o modificare facilmente una delle classi che implementa l'interfaccia senza dover necessariamente modificare anche le altre classi che la implementano.

Oltre a tali attributi di qualità possiamo notare che la scelta di definire un'interfaccia nel sistema garantisce il rispetto dell'**Open-Closed Principle**, poiché la classe "Rubrica" risulta chiusa alla modifica (modifiche dei metodi della classe non hanno impatto sui client della stessa) ed aperta all'estensione (la classe può essere modificata ed adattata a scenari differenti da quello attuale).

Altra scelta progettuale dettata dai principi di buona progettazione è stata la definizione delle classi "*ComparatoreContatti*" e "*FileManager*". Queste due classi sono state progettate per garantire il rispetto del **Single Responsibility Principle**. In questo modo, ogni modulo in cui il sistema è stato decomposto si occupa di un singolo compito ben definito, favorendo una maggiore chiarezza e manutenibilità del codice.

Il sistema rispetta anche altri principi fondamentali per una corretta progettazione, come il **K.I.S.S.**(Keep It Simple Stupid) ed il **D.R.Y.** (Don't Repeat Yourself). In particolare si noti come l'aggiunta della classe *"ComparatoreContatti"* sia stata finalizzata proprio ad evitare inutili ripetizioni nel codice, che ne avrebbero per'altro diminuito la leggibilità.

2 Valutazione del livello di coesione:

Classe	**Livello di coesione**	**Descrizione**
Contatto	Funzionale	La classe implementa le operazioni necessarie alla realizzazione della gestione della struttura dati che rappresenta un contatto, in particolare i metodi getter e setter
Rubrica	Funzionale	La classe presenta tutti i metodi necessari per la creazione e la gestione di una lista di contatti. Implementa tutti i metodi di *InterfacciaRubrica*
RubricaController	Comunicazionale	La classe implementa metodi che lavorano sugli stessi dati di input, al fine di gestire le operazioni sulla rubrica.
InfoController	Funzionale	La classe contiene i metodi necessari per il salvataggio o l'annullamento delle operazioni di creazione e modifica di un contatto dalla rubrica.
FileManager	Funzionale	La classe presenta i metodi per salvare o caricare la rubrica su/da un file.
ComparatoreContatti	Funzionale	La classe ridefinisce il metodo dell'interfaccia *Comparator* che consente di comparare i contatti in accordo con il criterio di ordinamento richiesto.
Main	Funzionale	La classe contiene i metodi che consentono l'esecuzione dell'applicazione.

Table 1: Valutazione del livello di coesione

3 Valutazione del livello di accoppiamento:

Classi	**Livello di accoppiamento**	**Descrizione**
Rubrica-Contatto	Per dati	La classe Rubrica chiama i metodi pubblici di Contatto, scambiando solamente le informazioni necessarie alla gestione della rubrica
Rubrica-InfoController	Per dati	InfoController accede ai metodi pubblici di Rubrica, scambiando solamente i dati necessari per le operazioni di modifica e creazione
Rubrica-ComparatoreContatti	Per dati	Rubrica chiama il costruttore della classe ComparatoreContatti per istanziare un nuovo comparatore da passare al metodo *sort()*
Rubrica-FileManager	Per dati	FileManager utilizza i metodi pubblici di Rubrica per inserire la lista dei contatti letta dal file selezionato dall'utente. Inoltre istanzia un oggetto Rubrica per l'esecuzione del metodo *salvaSuFile*
RubricaController-FileManager	Per dati	RubricaController accede solo ai metodi pubblic di FileManager, per garantire le operazioni di lettura e scrittura della rubrica dal/sul file
Contatto-InfoController	Per dati	InfoController accede ai metodi public di Conatto per salvare le modifiche effettuate
Contatto-ComparatoreContatti	Per dati	ComparatoreContatti usa i metodi getter pubblici messi a disposizione dalla classe Contatto per ottenere i cognomi ed i nomi da comparare

Table 2: Valutazione del livello di accoppiamento

4 Diagrammi di sequenza

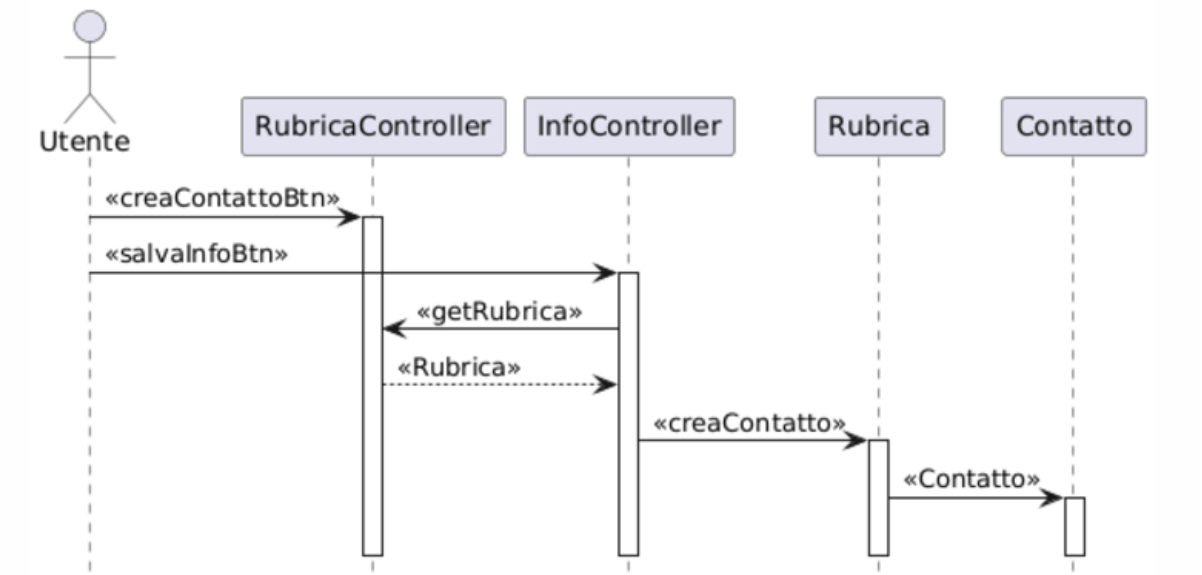

Figure 2: Interazione 1 - Crea contatto

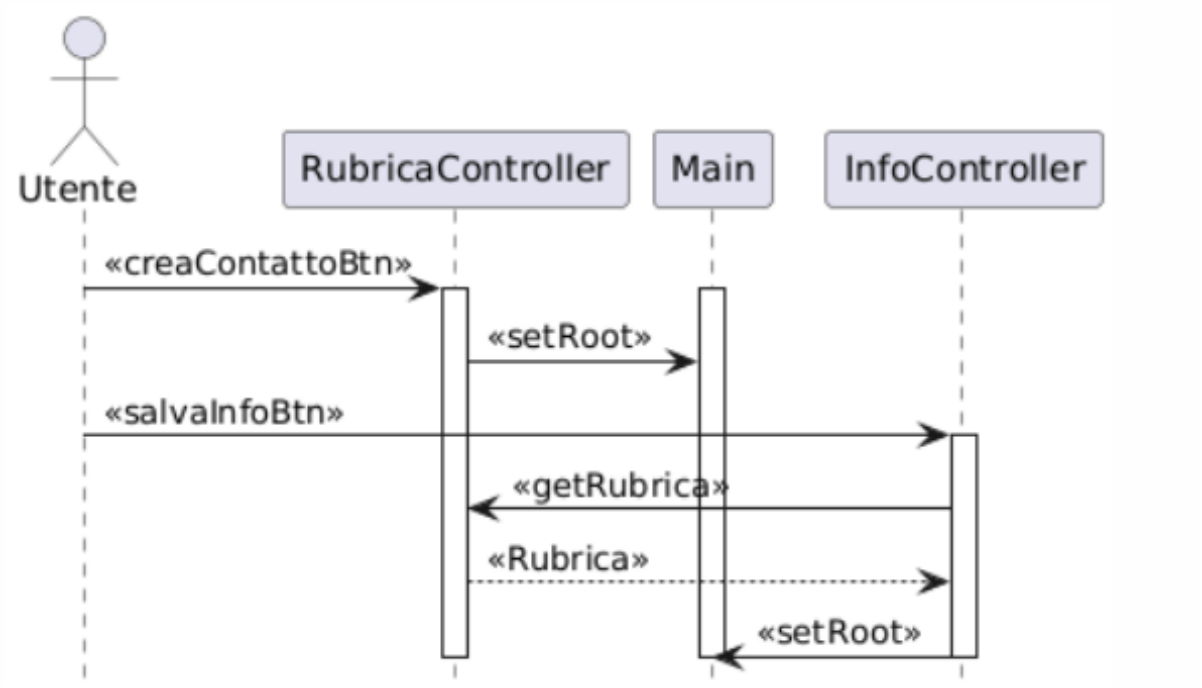

Figure 3: Diagramma di dettaglio della creazione

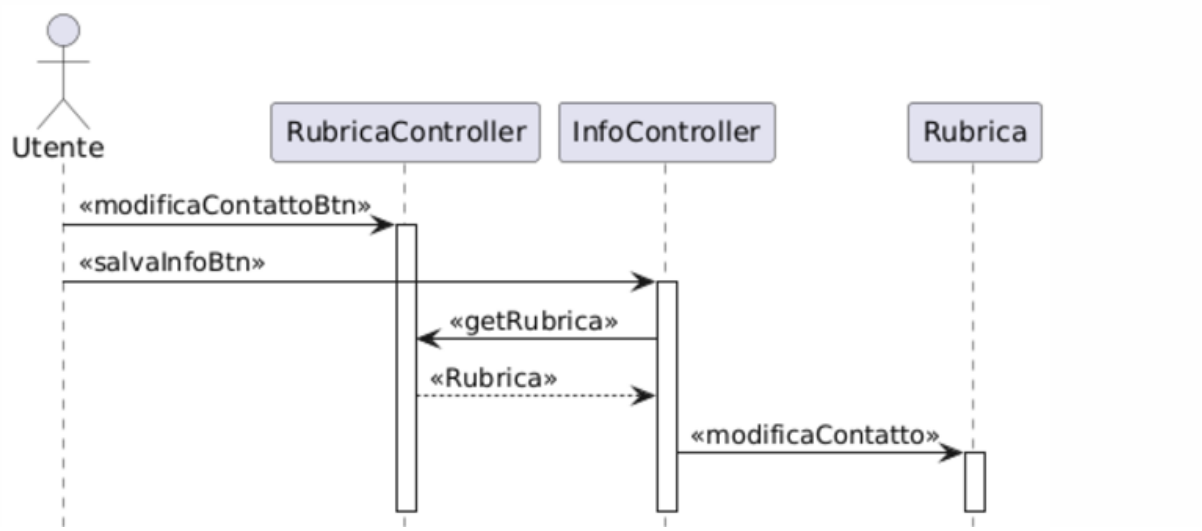

Figure 4: Interazione 2 - Modifica

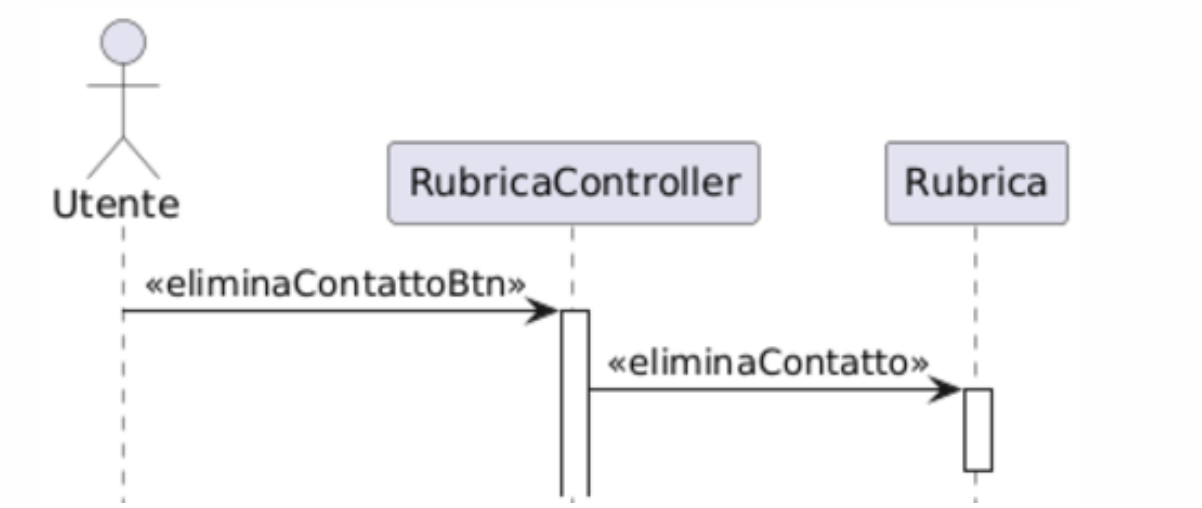

Figure 5: Interazione 3 - Elimina

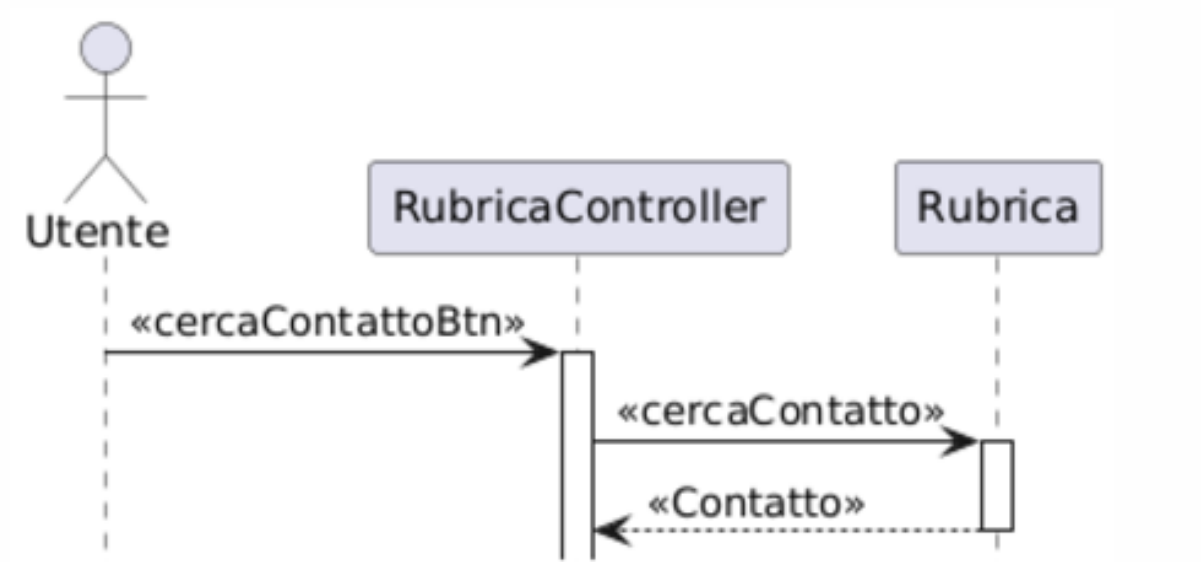

Figure 6: Interazione 4 - Cerca

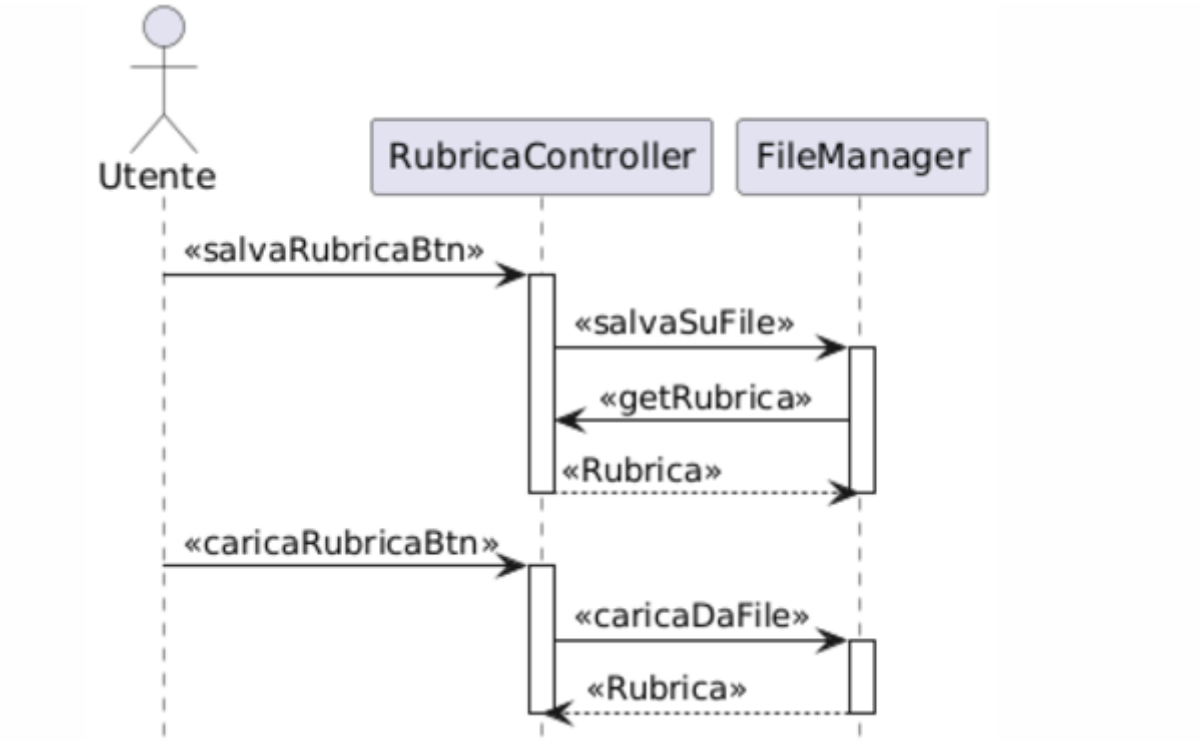

Figure 7: Interazione 5 - Operazioni su file

5 Diagramma dei Package