**Coesione, accoppiamento e principi di buona progettazione:**

**Coesione:**

App: Funzionale.

I metodi della classe App sono richiamati dal metodo start per l’inizializzazione dell’interfaccia grafica.

Contatto: Funzionale.

I metodi della classe Contatto servono solo per la creazione, la lettura e il settaggio degli attributi dei contatti della rubrica.

HomeController: Funzionale.

La classe HomeController lavora solo alla gestione della relativa interfaccia grafica.

SecondaryController: Funzionale.

La classe SecondaryController lavora solo alla gestione della relativa interfaccia grafica.

SuperController: Funzionale.

I metodi della classe SuperController servono solo per l’aggiunta e la rimozione di elementi dalle sue ObservableList, FilteredList e SortedList.

**Accoppiamento:**

App: per dati.

La classe App ha solo un paio di metodi statici che vengono usati per operazioni semplici da altre classi, gli altri metodi non vengono usati da altre classi.

Contatto: Nessuno.

La classe contatto non interagisce direttamente con altre classi.

HomeController: per aree comuni.

La classe HomeController gestisce solo la propria interfaccia grafica, usando solo piccoli metodi statici dalla classe App e modificando all’occorrenza le ObservableList, FilteredList e SortedList (tutte statiche) della classe SuperController.

SecondaryController: per aree comuni.

La classe SecondaryController gestisce solo la propria interfaccia grafica, usando solo piccoli metodi statici dalla classe App e modificando all’occorrenza le ObservableList, FilteredList e SortedList (tutte statiche) della classe SuperController.

SuperController: Nessuno.

La classe SuperController gestisce una ObservableList, una FilteredList e una SortedList (tutte e 3 statiche) e ha metodi solo per l’aggiunta e la rimozione di elementi, senza interazioni dirette con le altre classi.

**Principi di buona progettazione:**

KISS (Keep It Simple, Stupid!): abbiamo implementato classi con metodi chiari che rendono il codice facilmente leggibile anche grazie ai commenti.

DRY (Don’t Repeat Yourself):

YAGNI (You Aren’t Going to Need It): abbiamo implementato le richieste del progetto ed evitato funzionalità che avremmo voluto inserire (ad esempio contatore nella selezione dei contatti) che avrebbe reso il codice più complesso. L’unica eccezione è stata la foto profilo che è possibile aggiungere ad ogni contatto.

Separazione delle preoccupazioni: Ogni metodo gestisce una funzionalità specifica, questo rende il codice più facilmente comprensibile e manutenibile.

Ortogonalità:

Principio della minima sorpresa: Il codice è strutturato in maniera da essere semplice e intuitivo sia per chi lo legge sia per chi usa l’applicazione. Infatti il codice mantiene una coerenza tra nomi, funzionalità e comportamenti attesi.

Evitare l’ottimizzazione precoce:

Regola del boy-scout: il codice delle varie classi ha subito modifiche sostanziali nel corso dello sviluppo, in particolare il metodo initialize di HomeController, in quanto diverse funzionalità che dovevano essere avviate all'avvio dell'interfaccia grafica sono aumentate gradualmente. Nonostante le modifiche il codice non ha perso qualità.

SOLID:

Privilegiare l'associazione rispetto all'ereditarietà: Il programma non presenta alcuna classe estesa: ogni classe è indipendente dalle altre. L'unica dipendenza riguarda l'uso della lista osservabile statica della classe SuperController condivisa tra le varie schermate.

Principio di robustezza: il programma soddisfa pienamente tutte le postcondizioni dei vari requisiti. Il programma non si porta in uno stato inconsistente e tutte le funzionalità vengono svolte secondo le precondizioni richieste.