# UtenteClasse

Per quanto riguarda la classe Utente ho lasciato tutto “standard” tranne i metodi trasformatiInStringa e \_\_str\_\_ . Partendo dal metodo \_\_str\_\_ l’ho modificato in modo che non interagisse con altre funzioni, ritornando, uno sotto l’altro, i dati del contatto. Il grosso del lavoro è stato trovare il modo di stampare la rubrica in modo che i dati di ogni utente siano incolonnati in maniera corretta e che non vi sia uno spreco di spazio. Qui entra in gioco la funzione trasformatiInStringa che lavora in sintonia con il metodo \_\_str\_\_ della classe Rubrica.

I metodi di comparazione tra utenti sono stati di facile implementazione (è bastato copiare le slide delle lezioni, facendo opportuni cambiamenti).

# RubricaClasse

Avendo pensato la rubrica in modo che generi un id in maniera automatica quando viene inserito un nuovo utente, è stato necessario implementare il metodo generaId, algoritmo semplice ma necessario.

Il metodo larghezza Colonne l’ho utilizzato per trovare la lista da usare con il sopracitato metodo trasformatiInStringa della classe Utente: in pratica genera una lista in cui al suo interno sono presenti le massime lunghezze, considerando ogni contatto in rubrica (e gli headers), dei singoli campi dei contatti. Algoritmo semplice, si tratta di ripetere la stessa cosa per ogni campo (ecco perché è parecchio verboso).

Il metodo \_\_str\_\_ è la composizione dei metodi descritti in precedenza. Ho dovuto “ingegnarmi” per aggiungere gli headers distanziati correttamente: è bastato aggiungerli, una volta trasformati in stringa, nella lista contenente tutte le stringhe degli utenti per poi stamparli tutti usando la funzione join.

Piccolo commento sul metodo \_\_str\_\_ della Rubrica: rende superfluo il metodo \_\_str\_\_ della classe Utente perché per stampare un singolo utente con le corrette intestature (headers) basterebbe creare una Rubrica con una lista formata da un singolo utente e poi stamparla. Ecco perché con il metodo \_\_str\_\_ di Utente ho fatto si che i dati venissero stampati uno sotto l’altro: semplicemente per differenziare; starà poi al programmatore scegliere il modo più gradevole di visualizzare un singolo utente.

I restanti metodi ricalcano quelli dell’ homework1, niente di particolare se non che con la POO è tutto più naturale, senza quindi scervellarsi sugli indici di liste varie.

# InterfacciaClasse

Ho voluto che tutte le funzioni interattive e di controllo stessero all’interno dell’interfaccia. Effettivamente, alcune di esse, prendendo ad esempio il metodo modificaUtente avrebbero potuto stare all’interno della rispettiva classe ma, per il suddetto motivo, ho scelto diversamente.

I metodi ricercaInRubrica e modificaUtente mi è sembrato logico scriverli in maniera separata, dato che poi vengono richiamati più volte dall’interfaccia. Interfaccia che ho voluto rendere più “smart” rispetto a quella del primo homework, fornendo maggiori opzioni all’ utilizzatore, rendendola, in generale, più “fluida” dando cioè la possibilità all’utente di correggere i propri errori senza dover cominciare ogni volta da capo.

Non ho fatto differenze tra la ricerca tramite id e la ricerca tramite nome e cognome: il programma verifica automaticamente se il primo campo passato in input è un id della rubrica: se non lo trova allora chiede anche di inserire il cognome.

Per quanto riguarda la funzione inoutControllato, era necessario fornire un controllo che impedisse di inserire dati con allinterno il carattere “|”, che interferirebbe con il modo in cui vengono salvati e letti i contatti dal file di testo; il controllo sulla lunghezza, invece, serve a rendere più realistica la rubrica.