FEDERICO BERNACCA, 536683

Per questo progetto ho fornito tre implementazioni differenti, le quali condividono le seguenti scelte:

- ‘remove’ rimuove tutte le occorrenze del dato se vengono rispettati i controlli di identità
- ‘share’ condivide tutte le occorrenze del dato presenti al momento della richiesta se vengono rispettati i controlli di identità
- ‘copy’ copia la prima occorrenza del dato se vengono rispettati i controlli di identità
- ‘get’ restituisce la prima occorrenza del dato se vengono rispettati i controlli di identità
- Le altre operazioni hanno l’ovvio significato

Per le prime due implementazioni ho adottato la seguente strategia implementativa, mantenendo un approccio restrittivo:

Ho definito le classi “Dato<E>” e “User”, le quali vanno considerate come parte integrante della classe che implementa l’interfaccia “SecureDataContainer”.

Nella classe “User” sono presenti due variabili d’istanza:

1. ‘String id’
2. ‘String password’.

Questa classe rappresenta l’utente registrato nel Container proprietario di 0 o più elementi di tipo “Dato<E>” definito sotto.

Nella classe “Dato<E>” sono presenti tre variabili d’istanza:

1. ‘E dato’
2. ‘User owner’
3. ‘ArrayList<User> sharedWith’.

Ogni elemento generico ‘E dato’, quando aggiunto al Container avrà, come proprietario, l’utente ‘User owner’ già registrato nel Container, e non sarà condiviso.

Potrà essere eventualmente condiviso in un successivo momento con altri utenti tramite l’aggiunta di questi in ‘ArrayList<User> sharedWith’.

Per ogni elemento “Dato<E>” del Conatiner, l’utente ‘User owner’, proprietario di ‘E dato’, ha diritto esclusivo sulle operazioni di ‘copy, ‘remove’ e ‘share’, anche se lo ha condiviso con qualche altro utente, mentre l’operazione di ‘get’ può essere eseguita da tutti gli utenti che si trovano in sharedWith() di quel determinato elemento.

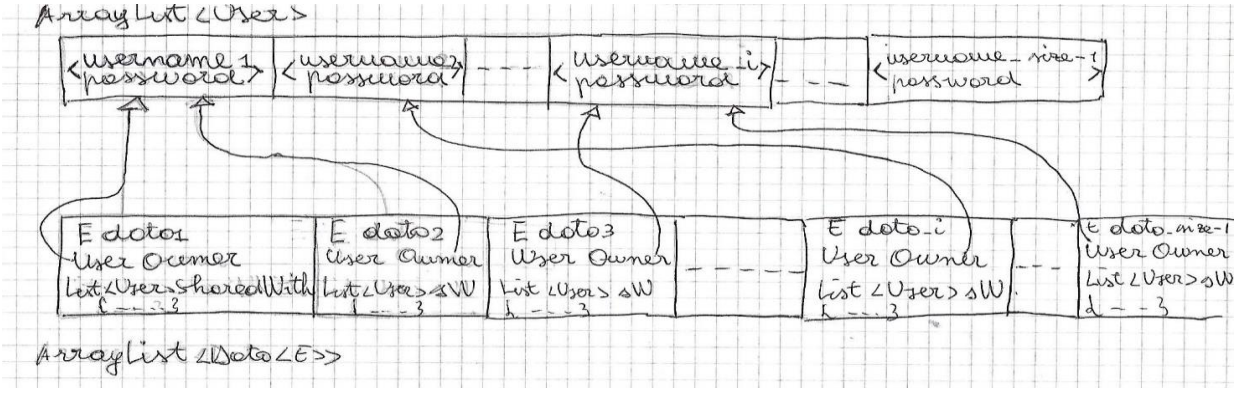
Da notare quindi le operazioni di ‘remove’ e ‘share’:

con ‘E remove(String Owner, String passw, E data)’ vengono rimosse tutte le occorrenze di ‘data’ che hanno come utente proprietario ‘<Owner, passw>’, rimuovendo così l’accesso a ‘data’ da parte di eventuali altri utenti presenti in data.sharedWith().

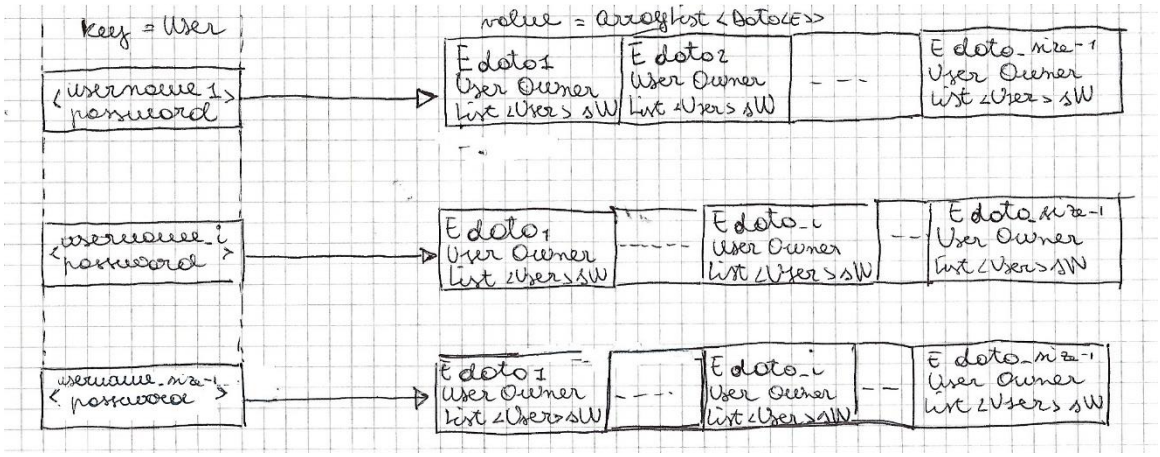
Come sopra, l’operazione ‘share(String Owner, String passw, String Other, E data)’ da accesso all’utente ‘Other’ a tutte le occorrenze di ‘data’ che hanno come utente proprietario ‘<Owner, passw>’, inserendo ‘Other’ in data.sharedWith().

Per l’implementazione di “TwoListsContainer” ho utilizzato due ArrayList.

Un ‘ArrayList<User> users’ atto a contenere, senza duplicati, gli utenti registrati nel Container, l’altro invece ‘ArrayList<Dato<E>> dati’ atto a contenere tutti i dati di tipo E coi rispettivi proprietari e con le rispettive liste di accesso, graficamente:



Per l'implementazione MapContainer ho utilizzato una 'hashMap<key, value> map' dove
la 'key' è un utente della classe "User" e 'value' è un ArrayList<Dato<E>> il quale corrisponde alla lista di elementi di chiave "User", graficamente:



Per l'implementazione "SingleListContainer" ho invece cambiato tipo di strategia: ho mantenuto un approccio meno restrittivo per avere come conseguenza un codice più compatto e più efficiente:
Ho riutilizzato la classe "User" definita sopra e cambiato invece l'implementazione della classe "Dato<E>", non più tre variabili d'istanza ma due:

1. User user
2. ArrayList<E> userData.

Come sopra, queste due classi sono da considerarsi parte integrante per l'implementazione di "SecureDataContainer"
Ad ogni utente 'user' del Container è associata la propria lista di dati, non esistono più vincoli di restrizione sulle varie operazioni: un utente è autorizzato a fare qualsiasi operazione a patto che il dato sia presente nel proprio 'userData', graficamente:

Da notare le operazioni di 'remove' e 'share':
con 'E remove(String Owner, String passw, E data)' vengono rimosse tutte le occorrenze di 'data' presenti in 'userData' del proprietario '<Owner, passw>'.
L'operazione di 'share(String Owner, String passw, String Other, E data)'
aggiunge tutte le occorrenze di 'data' presente in 'userData' del proprietario '<Owner, passw>', nella lista userData di 'Other', diventando così a tutti gli effetti un suo elemento.
Per l'implementazione di "SingleListContainer" ho utilizzato un ArrayList<Dato<E>>.

Per testare le varie implementazioni ho scritto due semplici classi "Foto" e "Canzone", composte da sole due variabili di istanza di tipo 'String'. I file "nomeclasse_manual_test.java" è un test di tipo "manuale" dove il programma chiederà all'utilizzatore quale operazione svolgere, mentre il file "nomeclasse_auto_test.java" esegue operazioni prendendo l'input in automatico dai vari file.txt
Per testare il codice è sufficiente compilare ed eseguire i Test.java delle varie implementazioni.
Per 'SingleListContainer' ho creato anche una semplice interfaccia grafica con l'ide netbeans per testare le varie operazioni, eseguibile compilando ed eseguendo il 'Interface.java'.
Se si volesse testare il codice con altre classi, potrebbe essere necessario ridefinire il metodo 'boolean equals(Object o)'.