Primo Progetto Intermedio Programmazione II
Relazione sull'implementazione scelta.

Una volta analizzata la consegna del progetto, ho deciso di dividere la mia implementazione in due entità distinte:

1) Il *singolo utente* della collezione dati
2) La *collezione dati* nel suo insieme.

Queste due entità le ho concretizzate in due classi, dove specifico anche i loro vincoli concreti.

La prima è la ***UserAccount<E>*** che implementa la ***UserInterface<E>***(Overview e invariante nel file .java o in fondo alla relazione). Questa classe descrive il singolo utente tramite 3 attributi:

- Il ***NomeUtente***, descritto con una variabile *String*.
- La ***Password***, descritta come una variabile *String*.
- La ***CollezioneDati***, rappresentata come un *ArrayList* generico.

Le mie scelte di implementazione riguardo a questa classe sono state le seguenti:
Il NomeUtente rende univoco l'account all'interno del Drive, perciò non può essere inizializzato o settato a *null*.
Analogamente, per rendere sicura la collezione dati dell'utente ci avvaliamo della password, perciò non può essere settata o inizializzata anch'essa a *null*.
Ho scelto di utilizzare un *ArrayList* come collezioneDati per l'utente poiché mi sembrava la collezione più adatta. A seconda del numero di elementi della collezione dati, se pochi con un *HashTable* rischiavo di non riempirla in modo ottimale, causando uno spreco di memoria, mentre se invece erano un numero elevato con una *LinkedList* rendevo lenta il meccanismo di ricerca e di recupero degli stessi. In base a queste mie supposizioni ho ritenuto che un *ArrayList* fosse il compromesso perfetto. L'insieme di elementi che faranno parte della CollezioneDati può contenere duplicati. Riguardo alla condivisione di dati fra due utenti, la mia implementazione mi impone la seguente procedura: La condivisoine avviene tramite una shallow copy dalla collezione dati del mittente all'interno della collezione dati del utente destinatario. Questo significa che se elimino un elemento condiviso verrà eliminato solo dentro la collezione dati selezionata, e rimarranno invariete le altre copie (o l'originale).

La seconda è la ***MyDrive*<E>*** che implementa la ***SecureDataContainer<E>*** (Overview e invariante nel file .java o in fondo alla relazione).
Il vincolo più importante che mi sono posto sull'implementazione di questa classe riguarda la necessità di non avere account duplicati nel mio drive ne avere account con lo stesso nome utente.
Come richiesto dal progetto, ho implementato questa classe con due collezioni dati differenti, mantenendo inalterata l'intefaccia e la classe base ***UserAccount<E>***.
La mia prima implementazione sfrutta un *HashTable<String, UserAccount>*, dove la chiave primaria è il nome utente dell'account. In questo modo è banale distinguere i vari account e si previene la possibilità di duplicato o omonimi.
La seconda implementazione utilizza un *ArrayList<UserAccount>* . Questa implementazione purtroppo richiede più controlli per garantire l'assenza di duplicati e omonimi.
Fra le due implementazioni è sicuramente preferibile la prima per una serie di motivi:

- Bisogna Iterare l'*ArrayList* ogni volta si vuole effettuare un azione su di esso, per assicurare i controlli di sucurezza.
- All'aumentare della dimensione della collezione dati aumenta anche il tempo necessario a svolgere un'azione.
- Tramite l'*HashTable* riusciamo a eliminare i duplicati.
- Il tempo di ricerca della *HashTable* è *O(1)* al caso medio, e non *O(n)* come l'*ArrayList*.

All'interno del mio codice attuo una programmazione difensiva. In ogni metodo che descrivo vado a controllare chei miei attributi e i parametri immessi rispettino l'invariante di rappresentazione. Se non la rispettano sollevo un eccezione adeguata.

SecureDataContainer:

OVERVIEW: Tipo mutabile che rappresenta una collezione di Drive relativi a degli utenti.
Typical Element: D[0..n] di tipo UserAccount.
Rep Invariant: RI(m) = D!=null &&
(forall element : D => (forall x : D-{element} | element!=x)).

Utente:

OVERVIEW: Tipo mutabile che rappresente un utente e i suoi dati privati.
Typical Element: m: < user, password, data > dove user è un valore di tipo String univoco che descrive il nome dell'utente, password descrive la chiave di accesso ai dati e data è il campo generico che descrive i dati.
Rep Invariant: RI(m) = m.user!=null && m.password!=null