# Primo Progetto intermedio: SecureDataContainer

Svolto da Giuseppe Muschetta, matricola 564026, corso A, prof. Ferrari

L'implementazione di DataSecureContainer permette a chiunque di creare un proprio account (fornendo una userID ed una password associata) e di poter inserire e condividere dati.
Come è lecito attendersi, le UserID devono essere uniche, ergo non è possibile che utenti con lo stesso nome possano accedere al contenitore dati.
Gli utenti possono registrarsi tramite il metodo createUser e possono cancellarsi tramite il metodo removeUser.
La cancellazione di un utente implica la rimozione immediata di tutti i dati che ha inserito nel contenitore. Se aveva condiviso dei dati ad altri utenti, questi stessi utenti non avranno più accesso a tali dati.

L'inserimento di un dato da parte di un utente connesso avviene tramite il metodo put, inserendo un dato nella collezione, egli ne diventa automaticamente anche il proprietario, "firstOwner".

Un dato della collezione è incapsulato all'interno di un oggetto Item formato da tre campi:
String owner; (indica chi ha accesso al dato)
String firstOwner; (indica chi ha creato il dato diventandone il "primo" proprietario)
E data; (il dato effettivo su cui si sta operando nel contenitore di storage)
Quando un dato viene inserito per la prima volta da un utente nella collezione, egli ne diventerà il proprietario, il che tradotto significa che il campo owner e firstOwner coincideranno con l'utente stesso.

La piattaforma di Data Storage permette numerose operazioni tra cui la condivisione di dati fra utenti. Se due o più utenti sono registrati all'interno della piattaforma di storage, allora possono facilmente mettere in condivisione i propri dati con altri utenti.
Come già anticipato sopra ogni oggetto Item, oltre al dato vero e proprio, porta con sé due firme: la prima "owner" indica chi ha l'accesso al dato, la seconda invece "firstOwner" indica il proprietario originale del dato. Quando un utente X inserisce un dato nel contenitore e decide di condividerlo con un utente Y, internamente, il sistema crea una copia dell'oggetto impostando Y come "owner", ma preservando X come "firstOwner".
In questo modo X ha autorizzato Y ad accedere al dato.
Se un domani però X dovesse decidere di eliminare il dato dalla sua collezione, anche Y perderà il dato non avendone più accesso.
La cosa da enfatizzare è che Y, se lo desidera, può cancellare il dato che gli è stato condiviso da X, ma solo localmente e solo per lui stesso, non potrà cancellarlo anche a X, proprietario del dato.
Tutto ciò garantisce un filtro di sicurezza aggiuntivo per i proprietari dei dati.

La condivisione viene effettuata tramite il metodo share.
È possibile, tra le altre operazioni, quella di poter prelevare un particolare dato nella collezione dell'utente, tramite il metodo get, di creare una copia del proprio dato e inserirlo nella collezione tramite il metodo copy, di iterare sui dati attraverso un iteratore opportunamente restituito e disponibile all'utente tramite il metodo getIterator. Il metodo getSize restituisce la cardinalità degli oggetti presenti nella collezione dell'utente corrente.

Metodi facoltativi comprendono removeAll che rimuove tutti i dati uguali ad un dato particolare, clear che rimuove tutti i dati dell'utente e se questi ne aveva condivisi alcuni con altri, tutti perderanno il dato, printFirstOwnerData che stampa tutti gli oggetti per cui l'utente corrente è il proprietario originale, tralasciando i dati condivisi a lui da terzi, printSharedFromOthersData che stampa, invece, solo i dati che sono stati condivisi all'utente corrente da altri (cioè questi oggetti avranno come campo owner il nome dell'utente corrente, ma come campo "firstOwner" quello dei proprietari originali), printTotalData che stampa il totale di tutti i dati presenti nella collezione dell'utente corrente, isEmpty e isIn entrambi booleani e autoesplicativi.

Sono state realizzate due implementazioni:
IMPLEMENTAZIONE 1:
Qui si fa uso di una inner class, di nome Item, che può essere vista come una capsula al cui interno vi è l'oggetto data di tipo E in aggiunta alle due etichette, Owner e firstOwner, per le quali si sono date spiegazioni sopra.
La classe esterna, FirstSecureDataContainer<E>, fa uso di una HashMap users per mappare le Users Id e le password corrispondenti, e di un ArrayList, container, contenente tutti gli oggetti inseriti nella piattaforma di storage. Come è lecito attendersi ogni utente può visualizzare solo i suoi dati e accedere solo a quelli sui quali ha i giusti permessi.

IMPLEMENTAZIONE 2:
Qui, invece, si fa uso di due classi innestate.
La prima, di nome Dato, è composta dal dato vero e proprio di tipo generico E, e dall'etichetta firstOwner. Se il first firstOwner del dato è diverso dall'owner (colui che effettua l'accesso) allora il dato gli è stato condiviso da terzi.
La seconda inner class, di nome User, è composta dall'id dell'utente, dalla sua password e dal suo ArrayList contenente oggetti di tipo Dato.
La classe esterna SecondSecureDataContainer<E> fa uso di un ArrayList users al cui interno sono immagazzinati oggetti di tipo User.

Ambedue le implementazioni fanno ampio uso di metodi ausiliari che evitano la ripetizione e duplicazione del codice, e di altri metodi facoltativi descritti sopra.
Eventuali istruzioni per eseguire il codice sono state commentate all'interno dei sorgenti.