Programmazione II

A.A. 2018/19

Primo Progetto Intermedio

Federico Matteoni 530257

Descrizione del progetto

Il progetto richiedeva la progettazione di un Data Storage per memorizzare dati di tipo generico E accessibili previa identificazione tramite nome utente e password.
Ogni utente possiede una propria collezione di dati, e può accedere alla propria collezione. Un utente che possiede un dato può condividerlo nella collezione di un altro utente. Se un utente non possiede un dato allora non può accedervi.

Scelte progettuali

Nella stesura delle specifiche e del codice ho effettuato le seguenti scelte ed interpretazioni della specifica data:

- Le **password degli utenti registrati sono memorizzate in chiaro per semplicità** di progettazione e di lettura del codice
- Ogni utente possiede **ha accesso ad un determinato dato se e solo se tale dato è presente nella collezione** dell'utente in almeno una copia. Inoltre, **ogni utente può vedere e rimuovere dati solo dalla propria collezione**, mentre **può aggiungere un dato che possiede anche all'interno della collezione di un altro utente**, se ne conosce il nome, tramite il metodo `share`.
- **Quando un dato viene copiato all'interno di una collezione**, **viene eseguita una shallow copy** e non una deep copy. Questa scelta è stata fatta per mantenere la collezione aperta a più tipi di dato possibili e non limitarla a determinate classi.
- La specifica proposta **possiede un metodo astratto di verifica del dato di tipo generico**, che dovrà essere **implementato dal programmatore a seconda delle proprie esigenze** e del tipo di dato.

Implementazione proposta: `HashMapSecureDataContainer`

L'implementazione sfrutta due HashMap per poter memorizzare i dati richiesti.
In una HashMap, `usrPwd`, vengono memorizzate le coppie utente-password e viene sfruttata per verificare l'identità degli utenti quando essi vanno ad utilizzare il data storage.
L'altra HashMap, `usrData`, mappa ad ogni nome utente un'ArrayList di tipi generici che conterrà la collezione dell'utente associato.
Questa implementazione si appoggia ai metodi forniti dalle classi sfruttate per ricerca, aggiunta, rimozione e iterazione. Le implementazioni sono altamente efficienti e ciò rende molto veloce l'utilizzo del data storage così implementato.
La funzione di astrazione associa ogni stato ad una tripla `<nome utente, password, dato>`, mentre l'invariante di rappresentazione assicura che ogni tripla contenga la giusta coppia di nome utente e password e che il dato presente appartenga alla collezione del nome utente.

Implementazione proposta: `MatrixSecureDataContainer`

Questa seconda implementazione si appoggia solo tre array e una matrice.
Due array, `usrs` e `pwd`, memorizzano nome utente e password, con la password dell'utente `usrs[i]` memorizzata in `pwd[i]`.

Un terzo array, `data`, memorizza i dati univoci presenti nel sistema, senza contenere copie. Essendo un array di tipi generici, non è possibile costruirlo senza sapere il tipo di dato. Per mantenere l'implementazione aperta a qualsiasi tipo di dato, è richiesto al programmatore di inizializzare opportunamente l'array `data` nel costruttore della classe che estende `MatrixSecureDataContainer`.
La matrice `usrData` ha una struttura così fatta:
ogni riga corrisponde ad un utente, con la riga `usrData[i][]` che corrisponde all'utente `usrs[i];`
ogni colonna corrisponde ad un dato univoco, con la colonna `usrData[][j]` che corrisponde a `data[j]`.
Ogni elemento `usrData[i][j]` avrà valore 0 se il dato j non appartiene alla collezione dell'utente i, n > 0 se appartiene in n copie.
L'implementazione sfrutta le proprietà della matrice `usrData` e degli array per poter verificare le identità, la possessione dei dati, effettuare i conteggi richiesti ed iterare.
L'implementazione fornisce anche un metodo per poter visualizzare la matrice.
La funzione di astrazione associa ogni stato ad una tripla `<nome utente, password, dato>`, mentre l'invariante di rappresentazione assicura che la password nella tripla sia corretta e che il dato presente appartenga alla collezione dell'utente verificando il contatore opportuno nella matrice.

MainClass e Test forniti
Sono fornite le entrambe le implementazioni su dati di tipo String per poter eseguire i test.
Eseguendo il main verrà richiesta l'implementazione da eseguire e, in seguito, si interagirà con essa tramite le opzioni selezionate da un menu.
I test forniti sfruttano JUnit e vanno a testare entrambe le implementazioni sull'aggiunta di utenti, dati ed eccezioni lanciate.