RELAZIONE PROGETTO PR2

Luca Cataldo

1 Scopo del progetto

1.1 Traccia:

Si richiede di progettare, realizzare e documentare la collezione `SecureDataContainer<E>`. `SecureDataContainer<E>` è un contenitore di oggetti di tipo E. Intuitivamente la collezione si comporta come una specie Data Storage per la memorizzazione e condivisione di dati (rappresentati nella simulazione da oggetti di tipo E). La collezione deve garantire un meccanismo di sicurezza dei dati fornendo un proprio meccanismo di gestione delle identità degli utenti. Inoltre, la collezione deve fornire un meccanismo di controllo degli accessi che permette al proprietario del dato di eseguire una restrizione selettiva dell'accesso ai suoi dati inseriti nella collezione. Alcuni utenti possono essere autorizzati dal proprietario ad accedere ai dati, mentre altri non possono accedervi senza autorizzazione.

2 Scelte progettuali

La principale scelta progettuale per la risoluzione del problema è stata, suddividere le varie tipologie di dato presenti all'interno della traccia. Ho preso in considerazione gli utenti con la loro rispettiva password, e gli ho attribuito una lista di dati, una per ogni utente. Gli utenti all'interno della struttura sono univoci tra di loro, invece i dati posso essere anche ripetuti all'interno della lista. Facendo queste ipotesi iniziali ho deciso di fare le seguenti implementazioni:

(i) Per la prima implementazione ho usato l'interfaccia `List<E>` la quale implementa la classe `ArrayList<E>`. La mia idea è stata: rappresentare gli utenti e la loro rispettiva password con due liste: `List<String> Utente` e `List<String> Password`, e per idetificare i dati che ogni utente contiene all'interno della struttura, ho usato `List<ArrayList<E» dato`. Ho scelto questa struttura per il fatto che molte operazioni presenti all'interno sono eseguite in tempo costante, come per esempio get,set,isEmpty, invece le altre vengono eseguite in tempo lineare. Quindi ho deciso di puntare inizialmente su questa struttura.

(ii) Per la seconda implementazione invece l'idea e ricaduta sulla classe `Vector<E>`, con la quale ho strutturato tutti e tre i tipi di dato che avevo cioè `Vector<String> Utente`, `Vector<String> Password` e `Vector<Vector<E»Dato`. Questa seconda scelta e ricaduta su questa struttura perche dal punto di vista dell'utilizzo in questo caso e risultata comoda,è una struttura che può variare le proprie dimensioni in base alle necessità per consentire l'aggiunta e la rimozione di elementi dopo che il vettore è stato creato.

3 Scelte sull'implementazione dei metodi

(iii) `public void createUser(String Id, String passw);`
Questo metodo crea un utente con la sua rispettiva password e la sua lista di dati all'interno della collezione.

(iv) `public void RemoveUser(String Id, String passw);`
Questo metodo rimuove l'utente e la sua lista di dati dalla collezione.

(v) public int getSize(String Owner, String passw);
Questo metodo restituisce il numero dei dati dell'utente presenti nella collezione.

(vi) `public boolean put(String Owner, String passw, E data);`
Questo metodo ho deciso di implementarlo cosi: essendo un metodo che restituisce un booleano nel mio caso gli faccio restituire sempre true nel caso inserisca un utente nella collezione con successo. In tutti gli altri casi, cioè errori di inserimento o parametri non corretti sollevo eccezioni, e non faccio mai restituire false al metodo.

(vii) `public E get(String Owner, String passw, E data);`
Questo metodo ottiene una copia del valore del dato che è presente nella collezione.

(viii) `public E remove(String Owner, String passw, E data);`
Questo metodo elimina l'utente dalla collezione, eliminando con esso tutti i suoi dati.

(ix) `public void copy(String Owner, String passw, E data);`
Questo metodo mi crea una copia del dato all'interno della lista di colui che richiama questo metodo. La decisione che ho preso su questo metodo è quella che tratta la copia del dato all'interno della struttura. La mia implementazione, implementa una "Shallow Copy" cioè una copia superficiale del dato. Tale copia agisce in questo modo: se l'utente che ha un dato, lo copia, la copia che è stata creata è soggetta a modifiche, cioè se il dato originale viene modificato, tutte queste modifiche si ripercuotono anche sulla copia. Ho usato questo tipo di copia perche non volevo imporre vincoli all'utente cioè l'implementazione di clone() oppure serializable().

(x) `public void share(String Owner, String passw, String Other, E data);`
Questo metodo condivide il dato con un'altro utente. L'implementazione della condivisione l'ho voluta gestire cosi: ho fatto che per ogni dato condiviso, colui che richiedeva l'autorizzazione di accedere ai dati avesse una copia diretta del dato all'interno della sua lista. Questa copia viene fatta tramite "Shallow Copy" sempre per i motivi citati sopra.