Programmazione II – Primo Progetto – SecureDataContainer

Scelte progettuali

Nella realizzazione del progetto sono state prese le decisioni per i seguenti metodi dell'interfaccia:

- **createUser:** non è consentita la creazione di un utente con un username già creato.
- **Put**: è possibile inserire un dato già presente solamente se la lista di condivisioni iniziale (del dato E) è maggiore di 0.
- **copy**: consente la duplicazione di un dato solamente se quest'ultimo è stato condiviso con almeno 1 utente. La copia verrà effettuata utilizzando la *shallow copy* del dato di tipo E, ed azzerando la sua lista di condivisioni.
- **getSize**: ottiene la cardinalità solamente dell'insieme degli oggetti creati da un utente.
- **Share**: non è possibile condividere un dato con se stessi.
- **remove**: rimuove la prima occorrenza dell'elemento cercato.
  L'eliminazione consiste in:
  1. Cercare l'occorrenza del object E
  2. Accedere alla lista delle condivisioni di tale oggetto e andare nel profilo di ogni utente presente in questo insieme.
  3. Eliminare nella lista degli oggetti condivisi il dato E
  4. Una volta eliminate tutte le occorrenze presenti nelle condivisioni, cancella il dato creato dall'utente
- **getIterator**: ottiene l'iteratore della lista degli oggetti creati dall'utente e viene concatenato con quella della lista dei condivisi.
- **Get**: ritorna la shallow copy della prima occorrenza trovata.

Il programma, se effettua solamente inserzioni (quindi solo delle put), non ammetterà duplicati dello stesso dato di tipo E. Nel momento in cui si effettueranno delle *copy* e delle *share*, sarà possibile avere copie dello stesso oggetto, differendo esclusivamente per lista degli utenti con la quale è condiviso.

Ogni ricerca, inserzione ed eliminazione verranno effettuate sulla prima occorrenza trovata. In questo modo non sarà mai possibile avere un dato duplicato che abbia lista di condivisioni uguale. In quanto si opererà esclusivamente sulla prima occorrenza dell'oggetto trovato. Le eventuali copie verranno messe in coda alla lista, rese accessibili solamente alla rimozione delle precedenti occorrenze.

Classi create

Sono state utilizzate 2 classi per sviluppare l'implementazione dell'interfaccia "SecureDataContainer": UserProfile e UserData.

UserProfile: creata per descrivere un utente come un profilo formato da oggetti condivisi, creati e dalle proprie credenziali.

1) **Variabili globali:**
   a) **Username** di tipo String
   b) **Password** di tipo String
   c) **Owned** vettore di oggetti di tipo UserData
   d) **sharedToMe** vettore di oggetti di tipo E
2) **Metodi:**
   a) **public void AddCredential(String usern, String pw)**
   b) **public boolean AddUserData(E data)**
   c) **public boolean Auth(String usern, String pw)**
   d) **public void removeSharedToMeData(E data)**

e) **public UserData<E> RemoveUserData(E data)**
f) **public boolean sameNameAs(String Other)**
g) **public UserData<E> AddSharedElement(String other, E data)**
h) **public void AddSharedToMe(E data)**
i) **public int getSizeOwned()**
j) **public UserData<E> getUserDataFromOwned(E data)**
k) **public E getUserDataFromShaaredToMe(E data)**
l) **public Vector<UserData<E>> getOwned()**
m) **public Vector<E> getSharedToMe()**

UserData: creata per descrivere uno specifico dato con chi è condiviso

1) **Variabili globali:**
   a) **Data** di tipo E
   b) **sharedList** vettore di String
2) **Metodi:**
   a) **public E getData()**
   b) **public Vector<String> getSharedList()**
   c) **public UserData(E data)**
   d) **public E share(String other, E data)**
   e) **public boolean equals(Object object)**

Implementazioni

Per l'implementazione *MySecureDataContainer* come struttura di supporto è stato scelto il *vector*. In questo cosa la realizzazione è semplice ma non si ha un'ottimizzazione perfetta del codice, poiché ogni volta che bisogna cercare un profilo bisogna scorrere la lista fino a che non si trova l'oggetto cercato.

Per l'implementazione *MyHashSecureDataContainer* come struttura di supporto è stato scelto l'HashTable,. In questo caso si ha il beneficio di avere un accesso ai profili immediato, a discapito di un utilizzo maggiore della memoria, poiché viene creata una tabella hash.

Concludendo l'utilizzo della struttura HashTable rende più efficiente la lettura dei profili e non bisogna scrivere il codice per controllare se ci sono duplicati con lo stesso nome, in quanto basta utilizzare il metodo "**putIfAbsent"** dell'HashTable al momento della creazione di un nuovo utente.