Relazione primo progetto Programmazione 2

Serafino Gabriele – 564411 – Corso B

Introduzione

Il progetto richiede la realizzazione di un software per la gestione di un Data Storage, che permette la creazione di utenti (caratterizzati da un id ed una password), e l'inserimento e gestione di dati di tipo generico.
Il software effettua controlli sull'accessibilità dei dati, accertandosi che chi ha richiesto una determinata operazione su un dato abbia i permessi per farlo, gestendo le relazioni tra utenti e dati (Proprietà ed Accesso).

Interpretazione del testo

Seguendo la mia interpretazione del testo del progetto ho effettuato le seguenti scelte riguardo l'idea di collezione:

- Ogni utente ha la sua collezione di dati di cui è proprietario. In questa collezione non sono ammessi duplicati, più utenti però possono avere nella propria collezione copie dello stesso dato (una copia per utente al massimo).
- Il comando ***put(u.id, u.password, d)*** non inserisce ***d*** se quest'ultimo è già presente nella collezione di dati dell'utente ***u***.
- Il comando ***share(u.id, u.password, s.id, d)*** non crea una copia del dato, ma semplicemente aggiorna lo stato della collezione segnalando che il dato ***d*** è stato condiviso con l'utente ***s***, a questo punto ***s*** ha accesso a ***d***.
- L'accesso ad un dato permette all'utente di effettuare i seguenti comandi su quel determinato dato :
  - ***get()***
  - ***copy()***
- Il possesso di un dato permette all'utente di effettuare i seguenti comandi sul quel determinato dato:
  - ***get()***
  - ***share()***
  - ***remove()***
- Il comando ***copy(u.id, u.password, d)*** copia il dato ***d*** (a cui l'utente ***u*** deve avere accesso) nella collezione dell'utente ***u*** (se non era già presente).

Classi ausiliarie

Per la gestione delle informazioni relative ai diversi utenti ed ai dati ho creato rispettivamente le classi ***User*** e ***SecureData<E>***.

- *User*
  La classe ***User*** istanzia un oggetto di tipo utente, contenente i campi ***id(String)*** e ***password(String)***, i metodi *setter* e *getter* ed i metodi ***equals()*** e ***hashcode()***, utilizzati per la gestione delle chiavi nell'implementazione con ***HashTable***.
- *SecureData<E>*
  La classe ***SecureData*** istanzia un oggetto di tipo dato, con il campo ***data(E)***, con l'effettivo dato da conservare nella collezione, il campo ***owner(User)*** che riferisce all'utente proprietario del dato ed il campo ***shared(Vector<User>)*** che riferisce invece all'insieme degli utenti con cui il dato è stato condiviso.

Eccezioni specifiche

Per gestire i comportamenti "scorretti" dell'applicazione ho creato le seguenti eccezioni:

- *InvalidLoginException*
  Viene lanciata quando l'id utente specificato non è presente (perché non è stato creato) o quando la password specificata non corrisponde con quella relativa all'id utente presente nella collezione.
- *PermissionDeniedException*
  Viene lanciata quando un utente effettua il comando ***get()*** o ***copy()*** su un dato a cui non ha accesso (perché non gli è stato condiviso o perché non ne è il proprietario) o se effettua una ***share()*** o ***remove()*** di un dato del quale non è il proprietario.
- *UnavailableUsernameException*
  Viene lanciata in fase di creazione di un utente (***createUser()***) se è già presente un altro utente con lo stesso id.

Implementazioni

Per quanto riguarda le due implementazioni richieste ho deciso di utilizzare come strutture dati ***Vector*** ed ***HashTable***

- *Vector*
  In questa implementazione la collezione è rappresentata da due ***Vector***: ***users(Vector<User>)*** contiene l'insieme degli utenti registrati, ***data(Vector<SecureData<E>>)*** contiene l'insieme di tutti i dati inseriti nella collezione, ognuno con riferimento all'utente proprietario e agli utenti con cui è stato condiviso. Questa scelta di implementazione pecca sul fronte performance, in quanto la ricerca di un dato richiede lo scorrimento del vettore data, che contiene tutti i dati della collezione (potenzialmente un numero molto elevato).
- *HashTable*
  L'implementazione con ***HashTable*** utilizza una tabella hash nella quale le chiavi sono rappresentate da istanze della classe ***User***, ed i valori sono di tipo ***Vector<SecureData<E>>***, e contengono l'insieme dei dati con proprietario l'utente che ne è la chiave. In questo caso il campo ***owner(User)*** degli elementi ***SecureData<E>*** non viene utilizzato, in quanto sarebbe un'informazione duplicata.
  L'implementazione con ***HashTable*** migliora le performance rispetto a quella con ***Vector***, in quanto viene usata la chiave ***User*** per accedere in tempo costante alla lista di dati del determinato utente (di dimensione molto minore rispetto al totale dei dati contenuti nell'intera collezione).

Casi d'uso

I test relativi alle funzionalità del software sono stati effettuati tramite le classi ***VectorMain*** e ***HashTableMain***, rispettivamente per testare le implementazioni di ***Vector*** ed ***HashTable***.

Entrambe le classi effettuano 6 casi di test per verificare i seguenti aspetti:

- Normali funzionalità in assenza di comportamenti anomali
- Operazioni da parte di un utente non valido (nome o password errati)
- Tentativo di inserire più volte lo stesso dato nella collezione di un utente
- Richiesta di un dato al quale l'utente non ha accesso
- Rimozione di un dato non presente nella collezione
- Tentativo di creazione di più utenti con lo stesso id