Relazione per il progetto di Programmazione 2

Francesco Bocchi 550145 Corso B

3 dicembre 2018

Indice

1 Introduzione

Il progetto richiede lo sviluppo di una collezione detta SecureDataCointainer capace di immagazzinare oggetti di un determinato tipo generico E e garantire l' accesso e la condivisione dei dati tra gli utenti che sono registrati alla collezione, attraverso un nome utente e una password.

2 Scelte progettuali

2.1 Strutturazione del codice

La specifica del progetto richiede che un utente, prima di accedere ai dati presenti nella collezione, deve necessariamente identificarsi mediante uno username ed una password. Per tale ragione è presente una classe *Utente* che contiene due variabili di istanza chiamate *owner* e *passw*, le quali identificano ogni utente che si registra alla collezione.

È inoltre specificato che ad ogni utente corrispondono una serie di dati, ragion per cui nel progetto sono utilizzate due classi che specificano tale insieme:

- *DataIstance* , la quale contiene il dato generico presente nella collezione dell'utente ed un vettore di stringhe, che rappresenta la lista degli utenti a cui è stato condiviso quello specifico dato.
- *Dato* , la quale contiene un vettore di DataIstance per rappresentare tutti i dati associati all'utente.

2.2 Strutture dati

Per l'implementazione sono state utilizzate due strutture dati tra loro simili, ma che si differenziano sotto l'aspetto dell'efficienza. Le strutture dati in questione sono *HashTable* e *TreeMap*, le quali implementano tutte e due una forma di associazione *<chiave,valore>*, dove il primo campo corrisponde alla classe *Utente*, mentre il secondo alla classe *Dato*, citate sopra.

La scelta di queste due strutture dati è dovuta al fatto che se immaginiamo una mole di carico di utenti registrati pari a qualche decina di migliaia di utenti, utilizzare ad esempio una lista avrebbe abbassato drasticamente l'efficienza della collezione, mentre attraverso la tabella hash, possiamo rapidamente accedere ad uno specifico campo, tramite la funzione di hashing.

Utilizzando quindi una tabella hash, se definiamo α il fattore di carico, ovvero lo stato di riempimento della tabella, allora l'operazione di ricerca avrà un costo al caso pessimo di

$$\frac{1}{1-\alpha}$$

a differenza di una scansione di una lista che può avere al caso pessimo un costo lineare.

Per quanto riguarda la *TreeMap*, il principio è simile alla tabella hash, con la differenza che le chiavi sono ordinate lessicograficamente. Ciò garantisce un notevole incremento delle prestazioni, perchè operazioni come la ricerca, l'inserimento e la rimozione hanno un costo *logaritmico* rispetto al numero delle chiavi.

2.3 Implementazione

I metodi forniti nell'interfaccia *SecureDataContainer* sono stati implementati secondo il seguente criterio:

createUser , controlla che l'utente non sia già registrato (quindi gli utenti sono tutti distinti per quanto riguarda il nome), e in tal caso lo inserisce nella collezione.

getSize , fornisce il numero di dati che sono presenti nella collezione dell'utente, senza tenere in considerazione i dati che gli sono stati condivisi.

put , controlla che il dato non sia già presente nella collezione dei dati associati all'utente, ed in tal caso lo inserisce.

get , controlla prima la presenza del dato nella collezione dell'utente. Se il dato non è presente, viene ricercato nella struttura andando a verificare, per ogni dato uguale a quello richiesto, la presenza dell'utente che ha richiesto il dato nella lista di utenti a cui è stato condiviso quel dato.

remove , rimuove il dato passato come parametro solo se è presente nella collezione propria dell'utente, senza cercarlo fra i dati condivisi.

copy , va a cercare il dato da copiare in tutta la collezione (senza considerare quella dell'utente passato come parametro). Se si ha un match con il dato, viene controllato se nella lista di utenti a cui quel dato è stato condiviso sia presente o meno il nome passato come parametro alla copy. Se è presente,

allora il dato viene copiato nella collezione dell'utente che ha richiesto l'operazione.

share , inserisce nella lista di utenti associati al dato, presente nella collezione del proprietario, il nome dell'utente passato come parametro.

iterator , un iteratore per i dati presenti nella collezione propria dell'utente.

2.4 Classi secondarie e istruzioni per l'esecuzione

Oltre alle classi presentate sopra, nel progetto ne sono presenti altre che codificano le eccezioni utilizzate nel progetto, tra cui *FailedLoginException* che viene lanciata se un utente non supera i controlli di identità, *PermissionDeniedException*, lanciata quando non si hanno i permessi per copiare, condividere o rimuovere il dato, *DataAlreadyInException*, lanciata quando ad esempio si vuole effettuare una copia di un dato già presente nella collezione, *UserUnknownException*, utilizzata quando non troviamo un determinato utente nella collezione ed infine *UserAlreadyInException* relativa alla registrazione di un utente già presente nella collezione.

Sono contenuti poi due main chiamati *MainTreeMap* e *MainHashTable* che se lanciati eseguono dei test con la struttura specificata nel nome. Ogni main all'avvio richiederà di inserire un numero (1 o 2) per scegliere che tipo di test lanciare:

- 1 , verrà eseguito il test1, un test base per verificare la correttezza di tutte le funzionalità richieste dall'interfaccia
- 2 , verrà eseguito il test2, un test più scrupoloso che verifica le funzioni in casi limite, come ad esempio la registrazione di utenti già presenti, condivisioni con utenti non esistenti, copia di dati già presenti nella collezione.