Relazione Primo Progetto Intermedio Programmazione II [Corso A]
Raffaele Apetino - N.M. 549220

Istruzioni per l'utilizzo del programma

Spostarsi nella directory contenti i file con estensione **.java*, compilare sulla propria macchina i file sorgenti ed eseguire il comando *java Main*.

ArrayList o HashMap?

Prima di esporre quale delle due strutture dati è preferibile utilizzare, è giusto fare alcune osservazioni: in entrambe le implementazioni ho deciso di definire una classe accessoria, "User", che è presente in entrambe le implementazioni ma con diverse caratteristiche.

SecureDataContainerImpl

Nella prima implementazione ho deciso di appoggiarmi a una struttura dati nativa di Java, cioè ArrayList (un array di dimensione modificabile implementato sull'interfaccia List). La classe ausiliaria "User" contiene una stringa per l'Username, una seconda stringa per la Password e un ArrayList "Database" che contiene i dati (generici di tipo E). In *SecureDataContainerImpl* ho definito un ArrayList contenente dati di tipo User, tuttavia non troviamo nessun nesso tra indice dell'array e "User" associato.

SecureDataContainerImplv2

Nella seconda implementazione invece mi sono appoggiato a una struttura dati di tipo HashMap, la quale è una tabella hash implementata sull'interfaccia di Java Map. In questo caso "Userv2" contiene una stringa per la Password e un Vector di dati. La stringa per l'Username invece viene utilizzata come chiave per l'indirizzamento nell'HashMap e come valore associato alla chiave il tipo di dato "Userv2".

Who's better?

Si può fare un semplice calcolo della complessità per stabilire quale delle due implementazioni sia migliore. Per una operazione semplice come la ricerca di un utente, con una struttura di tipo ArrayList, è necessario al caso peggiore scorrere tutta la dimensione dell'array con una complessità risultante di **O(n)**, a cui va aggiunto il tempo di accesso alla struttura "User" per la comparazione delle stringhe.
Nel caso della HashMap la ricerca di un utente con il metodo *containsKey* avviene in tempo **O(1)**, quindi molto più performante. La correttezza del fatto che le stringhe siano corrispondenti è data dalla funzione di hashing per l'indirizzamento che funziona in modo opportuno. Poiché, applicando la funzione di hashing a due stringhe uguali, il risultato è lo stesso.

Metodi Share e CheckPass

Era possibile implementare il metodo share, che richiedeva di condividere il dato con un altro utente della collezione, in due modi: con una **Deep Copy** oppure una **Shallow Copy**. Con la Deep Copy avremmo ottenuto una copia esatta del file, ma con una referenza diversa. Con una Shallow Copy otteniamo una copia del file esatta, ma entrambi puntano alla stessa referenza. Le chiamate in java vengono definite come "*call by value dove il valore è reference*". Implementando il metodo share con una Shallow Copy ed in seguito alla modifica di una delle copie dei file, vengono modificate anche tutte le altre copie che fanno riferimento alla stessa referenza. Mentre nel caso di una remove, il file verrà rimosso solamente nella collezione di chi ha deciso di rimuoverlo, ma non in altre collezioni dove è stato condiviso il file. Ho preferito usare una Shallow Copy anche a causa del tipo generico del dato e degli svariati problemi del metodo clone, non potendo istanziare un oggetto con tipo generico E.

E' presente un metodo CheckPass che serve a controllare che la password contenga almeno un numero, questo per rendere la sicurezza un po' più consistente in caso di attacco (brute force) sulla collezione.

Batterie di test

Nel Main sono contenuti tutti i test, dove testo situazioni difficili e che tutte le eccezioni vengano lanciate nel modo corretto. Come in ogni collezione di dati (come può essere dropbox), all'atto del controllo di username e password viene segnalato se l'username o la password sono errati. Per questo ho deciso di lanciare un'eccezione da me definita come "*LoginFailException*" quando la password o l'utente sono errati.
Se durante la createUser la password non contiene almeno un numero viene lanciata un'eccezione "*PasswordNotSecureException*".
La "*NoElemFoundException*" è necessaria nel caso in cui, dopo l'autenticazione, non si trovi "data" che ci è stato passato come parametro.

Conclusioni

Grazie al tipo generico e alla struttura ad oggetti di Java è possibile inserire nella collezione qualsiasi tipo di oggetto, anche tipi/classi definiti dall'utente. La sicurezza si potrebbe migliorare togliendo le password in chiaro ed eseguendo la crittografia dei dati.