Relazione SecureDataContainer

Giovanni Solimeno

27 novembre 2018

1 Scelte progettuali

In generale, si deciso di creare una classe contenitrice `Element<E>` per incapsulare il tipo generico, in modo da rendere pi facili le operazioni sui permessi, e una classe `User`, rappresentante il singolo utente.

1.1 `Element<E>`

La classe `Element<E>` contiene tre campi privati, con relativi metodi setter/getter:

- Il campo `owner` una stringa contenente il proprietario del dato. Non accessibile direttamente, ma possibile controllare se un particolare utente prioprietario del dato tramite il metodo `ownedBy(who)`.
- Il campo `allowed` una lista di stringhe, contenente gli utenti autorizzati ad accedere al dato (escluso il proprietario). Non possibile accedere direttamente alla lista, ma possibile indicare che un utente deve essere autorizzato/disautorizzato tramite i rispettivi metodi `allowUser(other)` (che lancia `UserAlreadyAllowedException` se l'utente gi autorizzato) e `denyUser(other)` (che lancia `UserNotAllowedException` in caso l'utente non sia presente tra gli utenti autorizzati).
- Il campo `el` una riferimento ad una istanza del tipo generico E. possibile accedervi tramite il metodo `getEl()`, che restituisce un riferimento `el`, mentre non possibile cambiarne il valore.

Inoltre, possibile controllare se un utente pu accedere a un dato tramite il metodo `canBeAccessedBy(other)`, che restituisce true se e solo se other il proprietario oppure presente nella lista degli utenti autorizzati.
La classe sovrascrive il metodo `Object.equals(other)`, in modo da ritornare true se e solo se `other.owner.equals(this.owner)` true e `other.getEl().equals(this.getEl())` restituisce true.
Viene generata l'eccezione unchecked `NullPointerException` se `other` nullo (la scelta di chiamare `equals` su `other.owner` e su `other.getEl()` stata fatta in modo da lanciare in automatico l'eccezione se `other null`).

1.2 `User`

La classe `User` contiene due campi privati:

- Il campo `userName` contiene il nome utente dell'utente, ed possibile accedervi tramite il metodo `getUserName()`. Non possibile in alcun modo modificarne il valore.
- Il campo `userPass` contiene la password dell'utente, ed possibile soltanto modificarla, tramite il metodo `setUserPass( newPass )`, mentre non possibile accedervi in alcun modo.

Inoltre, la classe implementa come meccanismo di login la sovrascrittura del metodo `Object.equals(other)`, che restituisce true se e solo se `other.getUserName().equals( this.getUserName )` e `other.userPass.equals (this.userPass)` (si deciso di effettuare il confronto tramite i metodi/campi di other per lo stesso motivo di `Element<E>.equals()`), e implementa l'interfaccia `Comparable<T>`, in modo da ordinare gli utenti in base al nome (propriet che viene usata nella classe `TreeMapSecureDataContainer`).

2 Scelte specifiche

Si scelto di non imporre vincoli sulle classi/interfacce del tipo generico di `SecureDataContainer`, in modo da rendere l'utilizzo dell'interfaccia facile e non creare problemi di compatibilit, e di non criptare gli elementi salvati.

2.1 ListSecureDataContainer

Si deciso di fornire le due implementazioni usando due metodologie diverse: La prima (`ListSecureDataContainer`) si appoggia su due liste non ordinate, la prima contenente gli utenti registrati, mentre la seconda contenente i dati degli utenti.

2.2 TreeMapSecureDataContainer

La seconda (`TreeMapSecureDataContainer`) si basa su una TreeMap (ordinata tramite `User.compareTo()`), che associa ad ogni utente una lista con gli elementi da lui posseduti. Nonostante ci, la ricerca/rimozione/condivisione di un elemento non si avvale delle propriet di un albero, in quanto bisogna scorrere ogni associazione utente/lista elementi, alla ricerca degli elementi condivisi (oltre a quelli posseduti, se non ci fosse bisogno di cercare tali elementi basterebbe controllare nella lista associata all'utente che ha richiesto l'operazione).