# ­­­­­Interpretazione del testo

L’obiettivo di questo documento è quello di fornire un quadro generale della interpretazione e delle scelte progettuali che sono state prese per il progetto e a tal proposito saranno chiariti i seguenti aspetti:

1. Si assume che i dati gestiti da ISecureFileContainer (ISF) siano degli oggetti che possono essere rappresentati su un file nella memoria secondaria.
2. Si impone ai clienti di ISF di utilizzare solo tipi generici E che estendono la classe SecureFile.

In questo modo le classi che estendono SecureFile ereditano le seguenti proprietà:

* 1. Sono scrivibili/leggibili su/da file (SecureFile implementa Serializzable).
  2. Sono fornite di un path, che indica la posizione del file nella memoria secondaria, nel quale possono essere scritte/lette.
  3. Due oggetti di tipo o sottotipo SecureFile sono uguali solamente se il loro path è uguale.

È importante notare che se due oggetti “o1” e “o2”, che estendono SecureFile, sono uguali secondo equals(), significa che hanno in comune il campo filePath, ma tutti gli altri attributi eventualmente presenti in generale sono diversi.

Questo vincolo assicura di trattare oggetti che possono essere scritti/letti su/da file della memoria secondaria e permette di non inserire due oggetti uguali (con lo stesso path) all’interno di ISF. Gli oggetti gestiti sono perciò univocamente identificabili all’interno di ISF e questo porta diversi benefici, rispetto ad un approccio che non prevede questo vincolo, come ad esempio:

|  |  |
| --- | --- |
| **<E extends SecureFile>** | **<E>** |
| public E get(String Owner, String passw, E file) | |
| Viene recuperato l’unico oggetto “o” all’interno di ISF tale che o.path = file.path | A causa del metodo copy possono esistere più istanze di file all’interno di ISF, perciò si porrebbe il problema di determinare quale istanza restituire. |
| public void shareR(String Owner, String passw, String Other, E file) | |
| Viene condiviso con Other l’unico oggetto “o” all’interno di ISF tale che o.path = file.path | Questo approccio non permette di condividere in modo selettivo gli oggetti presenti, infatti una volta condiviso un oggetto file di ISF, automaticamente tutte le altre istanze uguali a file presenti in ISF sarebbero condivise con Other. |
| public void copy(String Owner, String passw, E file, String newFilePath) | |
| Viene introdotto un nuovo oggetto, fra i dati posseduti da Owner, identico a file. Fa eccezione il path che sarà uguale a newFilePath. La copia è a tutti gli effetti un nuovo dato inserito, per cui di default nessuno, oltre al proprietario, avrà l’accesso alla copia. | La copia eredita tutti i permessi di accesso definiti sul dato originale (in quanto non distinguibii). |

Seguendo questo approccio, quando si parla di permessi in lettura e scrittura, in realtà è opportuno fare due distinzioni. Permessi relativi al file referenziato (tramite attributo path) da un oggetto e permessi sull’oggetto stesso.

Nello specifico si è ritenuto opportuno gestire i permessi in nel modo seguente:

1. Utente “u” ha accesso in **lettura** al dato “d” in ISF:­ ­
   1. get aggiorna “d” con il contenuto del file referenziato da d.path e restituisce una deep copy di “d” in modo da non permettere all’utente di modificare il dato all’interno del container (la deep copy viene effettuata con il metodo della serializzazione visto che gli oggetti del container implementano già Serializzable).
   2. “u” può richiedere che il dato “d”, all’interno del container, venga aggiornato recuperando i dati dal file referenziato (tramite attributo path) utilizzando il metodo readFileFromDisk.
   3. “u” non può richiedere che il dato “d” all’interno del container venga scritto sul suo file referenziato (tramite attributo path) utilizzando il metodo writeFileOnDisk.
2. Utente “u” ha accesso in **scrittura** al dato “d” in ISF:
   1. get aggiorna “d” con il contenuto del file referenziato da d.path e restituisce direttamente “d” in modo che possa essere modificato da “u”.
   2. “u” può richiedere che il dato “d”, all’interno del container, venga aggiornato recuperando i dati dal file referenziato (tramite attributo path) utilizzando il metodo readFileFromDisk.
   3. “u” può richiedere che il dato “d” all’interno del container venga scritto sul suo file referenziato (tramite attributo path) utilizzando il metodo writeFileOnDisk. È da notare che le eventuali modifiche effettuate sull’oggetto ottenuto dopo una get non sono consolidate sul file referenziato fino a quando non viene richiamato il metodo writeFileOnDisk.

Questo approccio presenta un inconveniente­: per poter implementare il metodo copy è stata ritenuta necessaria la modifica del metodo copy nell’interfaccia da implementare aggiungendo il parametro newFilePath. Questa scelta è stata valutata conveniente per consentire la gestione della copia di oggetti rispettando il vincolo dell’univocità dell’identificativo (path).

# Implementazioni

Di seguito sono riportate le strutture dati sulle quali si basa il funzionamento delle due implementazioni dell’interfaccia ISecureFileContainer:

|  |  |
| --- | --- |
| ListSecureDataContainer | MapSecureDataContainer |
| private User admin; //admin del container private Set<User> users; //Utenti presenti nel container private List<E> dataSet; //dati presenti nel container private List<User> owners; //proprietari dei dati presenti nel container private List<Map<User,AccessLevel>> accesses; //Livello di accesso ad ogni dato presente nel container assegnato a ogni utente nel container | private User admin; //admin del container private Set<User> users; //Utenti presenti nel container private Set<E> dataSet; //Dati presenti nel container private Map<E,User> owners; //Proprietario associato a ciascun dato presente nel container private Map<E,Map<User,AccessLevel>> accesses; //Livello di accesso ad ogni dato presente nel container assegnato a ogni utente nel container |

# Testing

È stato utilizzato il noto framework di test JUnit per effettuare il test delle due implementazioni. Sono stati definiti test per tutte le classi di rilievo presenti all’interno del progetto, ma in questa sezione verranno affrontati solamente i test definiti per verificare il corretto funzionamento delle due implementazioni di ISecureFileContainer. È stato progettato un test per ciascun metodo dell’interfaccia SecureFileContainer ed ognuno di essi ha lo scopo primario di stressare il funzionamento del relativo metodo.

Prima della esecuzione di ogni test viene inizializzata una cartella: SecureFileContainer\_test sul desktop dell’utente corrente ed al suo interno, se il test lo prevede, potranno essere inseriti dei file durante l’esecuzione del test. Ciascun test viene eseguito due volte, la prima volta viene eseguito sull’implementazione MapSecureDataContainer, mentre la seconda su ListSecureDataContainer.