Progetto Programmazione 2

Parte 1: Java

Autore: Francesco Buti

Data consegna: 7/01/2019

Richieste soddisfatte:

Definire l'implementazione del tipo di SecureDataContainer (in allegato) fornendo due implementazioni che utilizzano differenti strutture di supporto. In entrambi i casi si richiede sia la funzione di astrazione sia l'invariante di rappresentazione.

In seguito, progettare una batteria di test (senza variazioni tra la prima e la seconda implementazione) volta a provare il corretto funzionamento del codice.

Struttura del progetto:

Il codice è stato interamente realizzato mediante l'uso dell'IDE Eclipse, con l'ausilio di Sublime Text 3 per i commenti.

Per semplicità, la cartella src contiene un'unica sottocartella Progetto, contenente le eccezioni, l'interfaccia, le classi e il main (che implementa una batteria di test).

Questo rende necessaria esclusivamente la clausola ***package progetto;*** in ogni classe e permette di accedere facilmente alla modifica del codice. Tale scelta implementativa si rende possibile solo grazie alla dimensione ridotta dell'intero progetto e al limitato numero di classi/interfacce.

Ogni eccezione è opportunamente commentata, ogni metodo classificato con le clausole standard (MODIFIES, EFFECTS, THROWS)

Funzionalità previste:

Il progetto prevede la creazione di due classi capaci di implementare l'interfaccia SecureDataContainer:

- MySecureDataContainer prevede l'utilizzo delle HashMap, in modo da gestire facilmente l'utente, la password e i dati associati al nome dell'utente. Per questa implementazione, si prevede una HashMap per l'associazione utente-password e una HashMap per l'associazione utente-dati utente. Dati utente è un contenitore di informazioni generico realizzato, per comodità, come arraylist. È facilmente sostituibile da un vector o, meno facilmente, da un array
- MySecureDataContainerV2 prevede l'utilizzo di un vector di Utente. La classe utente è stata opportunamente realizzata per includere username, password e i relativi dati. In questo modo si delega ogni metodo operante sui dati alla classe Utente, lasciando alla classe MySecureDataContainerV2 esclusivamente il compito di gestire la collezione di dati e di implementare i metodi previsti dall'interfaccia.
  Questa scelta implementativa rende estremamente facile sostituire il metodo di gestione della collezione (il vector può essere rimosso in favore di un'arraylist o di qualsiasi altra collezione generica di dati).

Classi ausiliarie e metodi aggiuntivi:

- La seconda implementazione, come già specificato, rende necessaria la creazione di una classe Utente volta a gestire i dati di ogni elemento della collezione. Pertanto, Utente implementa un set di metodi volti a operare sui dati interni.

- Le eccezioni sono state create appositamente per la gestione dei casi in cui si rende necessaria la protezione dei dati da eventuali errori dell'utente (defensive programming).
  In particolare, si tratta di eccezioni gestite nel test tramite la clausola Try-Catch, in modo da non interrompere il corretto funzionamento del programma, ma rendere comunque possibile l'individuazione di eventuali malfunzionamenti.

Note particolari:

La batteria di test usata per verificare la classe MySecureDataContainer è stata lasciata invariata per la classe alternativa MySecureDataContainerV2, al fine di dimostrare l'equivalenza delle soluzioni implementative nel soddisfare l'interfaccia.

Ogni test è opportunamente pensato per andare a buon fine, si invita il lettore a verificare il corretto funzionamento delle eccezioni modificando i dati degli utenti e/o username e password corrompendo intenzionalmente le informazioni inserite