PasswordCollector, un'applicazione in Java per la gestione locale delle proprie password

Lorenzo LEONI

Università degli studi di Bergamo, Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

30 aprile 2023

1 Introduzione

Spesse volte ci si trova nella situazione di dover scrivere le proprie password su un foglio cartaceo perché malfidenti nei confronti dei servizi di archiviazione online, i quali si appoggiano a server che frequentemente sono vittima di hacker desiderosi di violare la nostra privacy. PasswordCollector cerca di ovviare a questa problematica fornendo un servizio di **gestione**, di **memorizzazione** e di **crittografia** delle password completamente locale. Nello specifico:

- l'archiviazione dei dati personali avviene tramite documenti JSON, un formato di interscambio che facilita la portabilità non solo dei contenuti informativi, ma anche dell'applicazione, di conseguenza;
- la cifratura dei dati sensibili avviene mediante una versione modificata del cifrario di Cesare. Esso si basa su uno spostamento causale in tabella ASCII delle lettere costituenti la password da mascherare;
- l'interazione e la gestione di PasswordCollector avviene tramite un'interfaccia a riga di comando.

Da sottolineare, infine, la possibilità di registrare più account in modo tale che la stessa istanza dell'applicazione possa essere utilizzata da più utenti senza che l'uno possa accedere ai contenuti dell'altro.

2 Funzionalità

PasswordCollector prevede 2 tipologie di account, le quali si distinguono per le funzionalità accessibili e per i privilegi.

- **user**: oltre al poter eliminare il proprio account e modificarne le informazioni quali, per esempio, indirizzo di posta elettronica e numero di telefono, un utente *standard* ha soltanto la possibilità di inserire e visualizzare le proprie password, mentre un utente *premium* può anche modificarle ed eliminarle. Questa differenziazione è stata inserita in modo tale da permettere un successivo inserimento di un sistema di monetizzazione;
- **administrator**: l'amministratore è unico, può disinstallare l'applicazione e visualizzare la lista degli utenti registrati. Inoltre, ha la possibilità di generare e gestire le *premium key*, ossia un codice da associare a un indirizzo di posta elettronica. In fase di registrazione il nuovo utente può inserire una *premium key*: se il codice esiste ed è associato all'indirizzo email fornito, allora le funzionalità premium vengono abilitate, altrimenti viene creato un account *standard*.

Tutti i dati sensibili vengono crittografati e memorizzati in file JSON appartenenti a una cartella nascosta, ossia una directory non accessibile a occhi indiscreti. Solo gli utenti e l'amministratore sono a conoscenza della locazione del file system in cui la cartella *Pw_C0ll3ct0r* si trova.

3 Descrizione dei package e delle classi

Il codice sorgente dell'applicazione è distribuito in 3 package con l'obiettivo di raggruppare funzionalità simili e dipendenti tra di loro e di realizzare l'information hiding:

- package **encryptors** (figura 1);
- package **accounts** (figura 2);
- package **application** (figure 3 e 4).

3.1 Package encryptors

Contiene le classi `CaesarEncryptor` e `ModCaesarEncryptor`, le quali implementano rispettivamente il cifrario di Cesare e la sua versione modificata. La seconda estende la prima, che a sua volta implementa l'interfaccia `EncryptorMethods`. Nel diagramma delle classi non è specificato, ma sia il metodo `encrypts` sia `decrypts` ricevono (implicitamente) in ingresso un array di stringhe tramite l'operatore `varargs`.

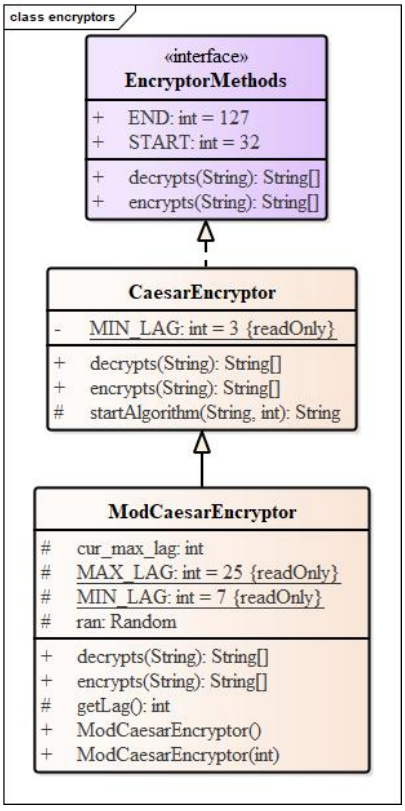


Figura 1: diagramma UML delle classi relativo al package *encryptors*.

3.2 Package accounts

La classe astratta `Account` serve per definire i metodi e i campi comuni a tutte le classi che da essa derivano, ossia `AdminAccount` e `UserAccount`, classe base di `PremiumUserAccount`. Quest'ultima possiede una `KeysCollection`, ovvero un oggetto che permette di accedere e gestire una lista di

`PremiumKey`, e implementa l'interfaccia `PwMethods` poiché l'utente *premium* è l'unico avente il privilegio di modificare ed eliminare le proprie password; esse sono conservate durante l'esecuzione in una lista di `Password`, il cui riferimento è memorizzato in `PwList`, un campo ereditato dalla classe padre `UserAccount`. Da notare, infine, l'enumerazione `AccountTypes`: essa viene utile ogniqualvolta si vuole verificare la tipologia di un oggetto che rappresenta un utente senza ricorrere all'operatore `instanceof`.

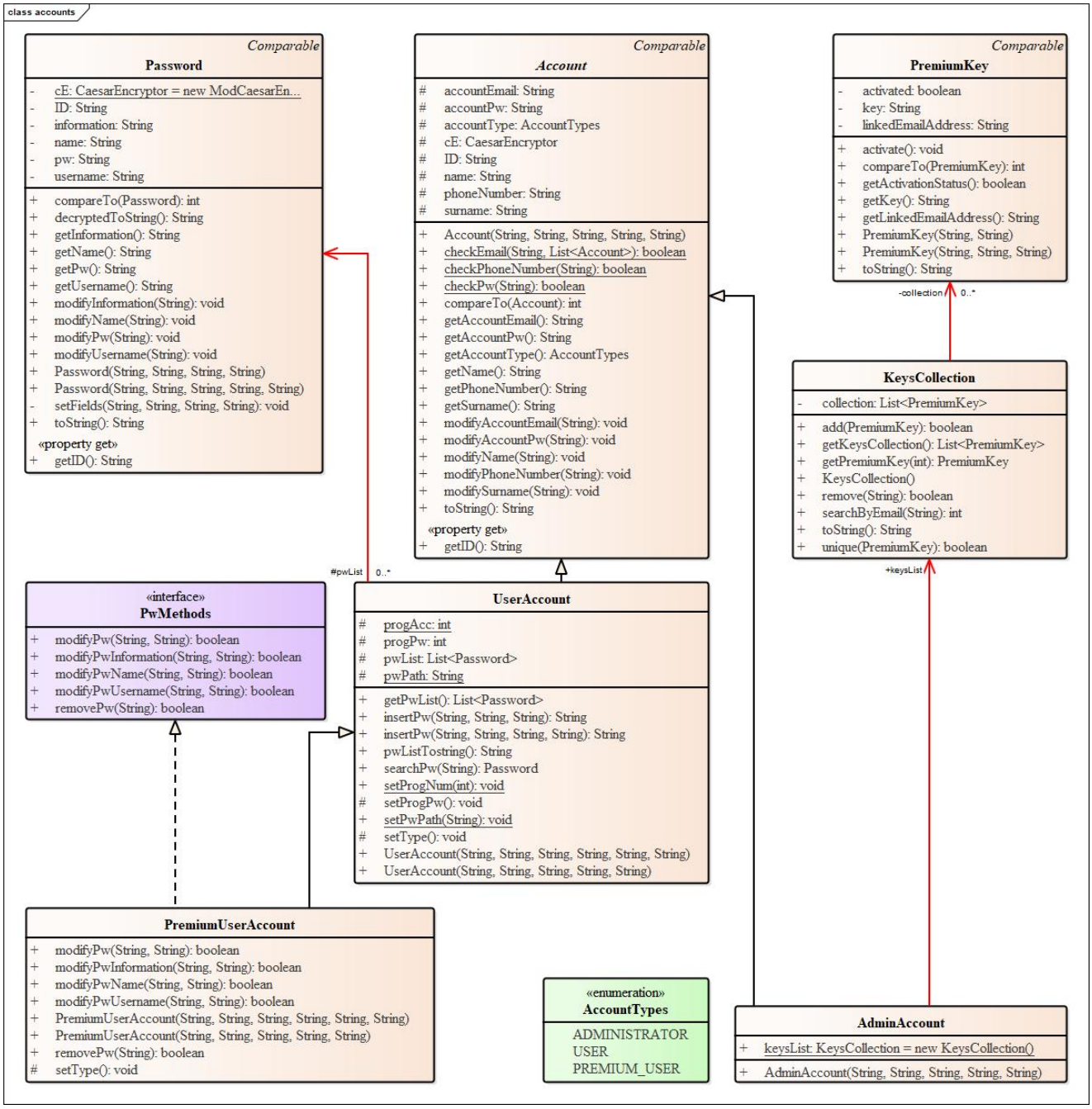


Figura 2: diagramma UML delle classi relativo al package *accounts*.

3.3 Package application

La classe `Application` è il cuore di PasswordCollector, ossia l'hub dal quale è possibile accedere a tutte le funzionalità. Essa implementa le istanze di `AccountHandler` e `PasswordHandler`, le classi aventi il compito di invocare i metodi rispettivamente di gestione dell'account (compresi il login/logout e la creazione di un nuovo profilo) e delle sue password. In `Application` è contenuto anche un oggetto appartenente alla classe `AppData`, i cui campi contengono i path delle cartelle contenute in *Pw_ C0ll3ct0r* e alcune informazioni riguardanti l'installazione. Invece, le classi `Installer`

e `Initializer` contengono i metodi statici che vengono eseguiti rispettivamente per installare l'applicazione e per inizializzare le strutture dati `accountList` e `data` di `Application` a partire dai documenti JSON nell'istante in cui PasswordCollector viene avviata. Non possono essere avviate più applicazioni, quindi l'unica istanza di `Application` avviabile è contenuta nella classe `Singleton`. Infine, `Util` implementa una serie di funzioni statiche per la lettura e per la scrittura da file.

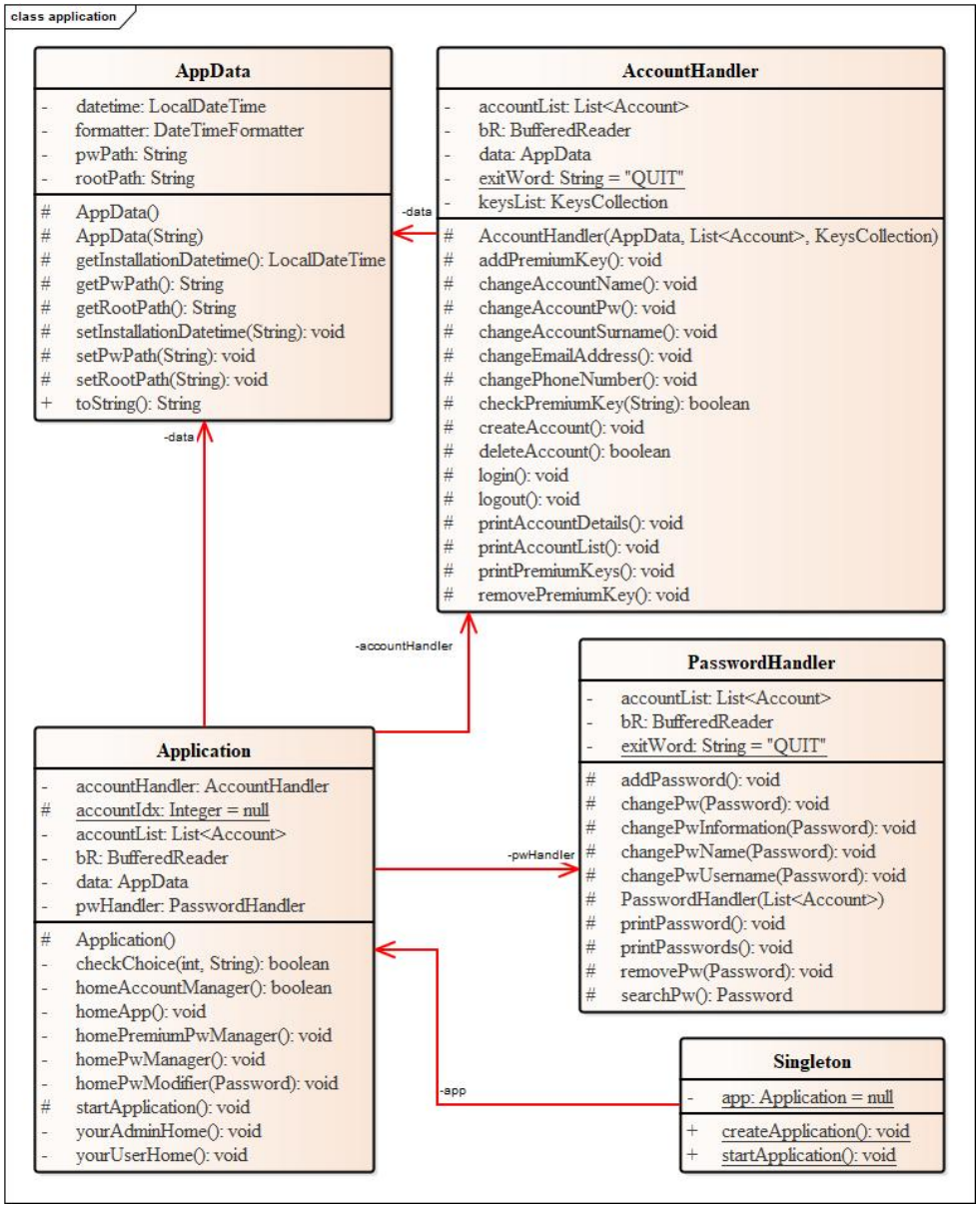


Figura 3: diagramma UML delle classi relativo al package *application*, parte 1.

4 Documenti JSON

Di seguito sono riportate le tracce dei documenti JSON all'interno dei quali vengono memorizzati in maniera persistente i dati di PasswordCollector:

- **app_data.json**: contiene i percorsi e la data di installazione dell'applicazione. Da sottolineare che JSON affianca al backslash anche lo slash normale in modo tale che il path possa essere interpretato da tutti i parser;

```
[
 {
  "installationDetails":{
```

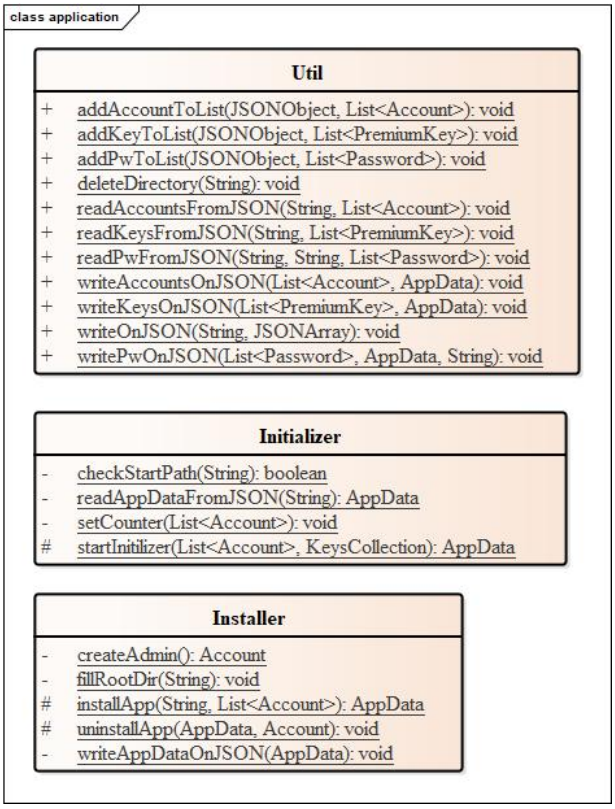


Figura 4: diagramma UML delle classi relativo al package *application*, parte 2.


```
  "root_path":"C:\\Users\\loren\\Desktop\/Pw_C0ll3ct0r",
  "passwords_path":"C:\\Users\\loren\\Desktop\/Pw_C0ll3ct0r\/Collections",
  "installation_datetime":"2023\/04\/30 16:01:44"}
 }
]
```


- **premium_keys.json**: contiene l'elenco delle coppie chiave-indirizzo delle *premium key* generate dall'amministratore. Se il campo `"activation_status"` è `"true"`, allora l'utente il cui indirizzo di posta elettronica è `"linked_email_address"` ha creato il proprio account utilizzando la chiave a lui riservata;


```
[
 {
  "premiumKey":{
   "activation_status":"true",
   "linked_email_address":"~!$w ,!@~w! {CKKJRy\u007Fs{~@u!\u007F\\",
   "key_value":"PcP[P]cP<@H?FM"}
 },
 {
  "premiumKey":{
   "activation_status":"false",
   "linked_email_address":"k{~wx7kj{krn{rI}r|ljur7r}\/",
   "key_value":"LrjxxjrLh;9;<\/"}
 }
]
```


- **accounts_data.json**: contiene l'elenco di tutti gli account che sono stati creati. Da notare che le informazioni sensibili sono state criptate utilizzando la versione modificata del cifrario di Cesare;

```
[
 {
  "account":{
   "account_email":"}?}v \u007FzCQ$%&uv\u007F%z?&\u007Fzsx?z%W",
   "account_type":"ADMINISTRATOR",
   "account_ID":"AD_0",
   "surname":"Leoni",
   "account_pw":"Puz{o{9@<<>",
   "name":"Lorenzo",
   "phone_number":";<?==88;88*"}
  },
  {
   "account":{
    "account_email":"} #v\u007F+ ?}v \u007FzBJJIQx~rz}?t ~W",
    "account_type":"PREMIUM_USER",
    "account_ID":"US_1",
    "surname":"Leoni",
    "account_pw":"X}\"#w#AHDHf",
    "name":"Lorenzo",
    "phone_number":"BCFDD??B??M"}
   },
   {
    "account":{
     "account_email":"&#v-B+|}(yT{!u} Bw#!f",
     "account_type":"USER",
     "account_ID":"US_2",
     "surname":"Bianchi",
     "account_pw":"ZyzzHHw99z",
     "name":"Roberto",
     "phone_number":"LNRRJKPNMO\u007F"}
    }
   ]
```


- **pw_US_1.json**: contiene l'elenco delle password e le informazioni a esse associate dell'account avente l'identificativo US_1. Se il campo `"information"` è `"empty field"`, allora significa che l'utente non ha voluto specificare alcun dettaglio aggiuntivo per la password corrispondente.


```
[
 {
  "password":{"pw":"WD!$&Dh)(((D5FDFDf",
  "name":"Amazon.it",
  "information":"empty field",
  "ID":"PW_0",
  "username":"!v\"{z'\/$#k"}
 },
 {
  "password":{"pw":"k$!v'zzHBGEGH6k",
  "name":"Navigraph.com",
  "information":"Navigraph.com is a web site where
    you can create a Flight Plan (FP) for FS2020",
  "ID":"PW_1",
  "username":"%~!$)az$NMk"}
```

```
    }
]
```

5 Versione modificata del cifrario di Cesare

L'algoritmo di Cesare consiste nello spostare in avanti di 3 posizioni ogni lettera di una parola, riferendosi alla tabella ASCII. Per esempio, "**Lorenzo-1998**" diventa "**Oruhq}r04«;**". Nella sua versione modificata, invece, lo spostamento (*lag*) è scelto casualmente tra 7 e 25 (15 in PasswordCollector). Tuttavia, per decifrare la parola è necessario conoscere il *lag*, pertanto in coda a essa viene aggiunto un carattere dal quale è possibile risalire allo spostamento casuale generato dall'algoritmo. Le formule di cifratura e di decifrazione del *lag* sono rispettivamente

```
char charLag = (char) (lag*5 + 2)
lag = ((int) charLag - 2)/5;
```

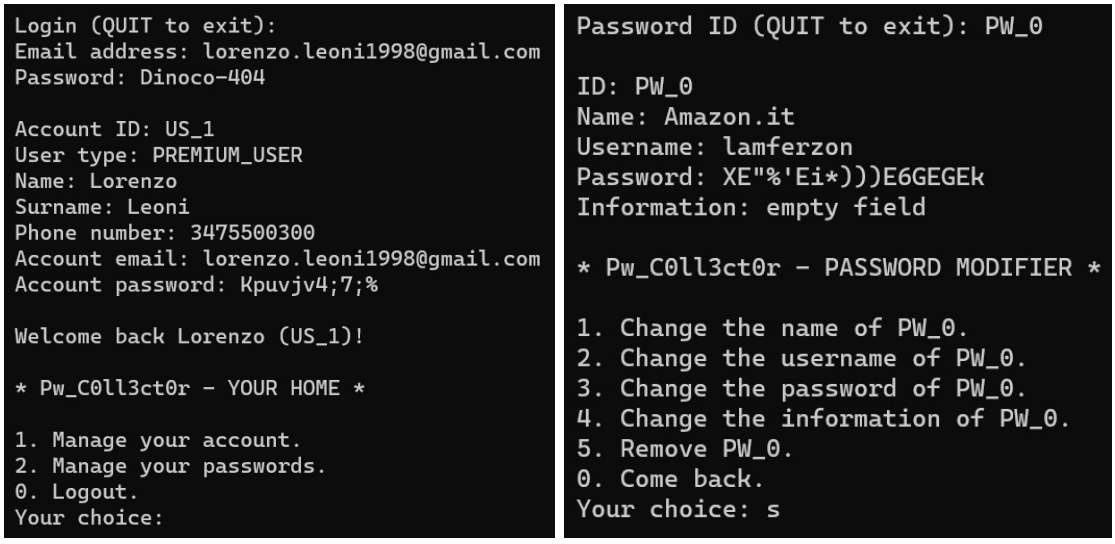
dove `charLag` rappresenta il simbolo in tabella ASCII la cui posizione risulta dall'applicazione della formula al `lag`. Se lo spostamento vale 7, allora $7 \cdot 5 + 2 = 37$, il quale corrisponde al simbolo "%" in tabella ASCII. Se, invece, il *lag* è uguale a 25, allora $25 \cdot 5 + 2 = 127$; esso coincide con l'ultimo simbolo utilizzabile in quanto, per questioni rappresentative e di codifica, è stato deciso di utilizzare soltanto i simboli in tabella ASCII che vanno dalla posizione 32 alla 127. Di seguito è riportato un esempio di decifrazione di una password criptata:

"**]vshyll:4979:(%**"
"%" corrisponde a 37 in tabella ASCII
$lag = (37 - 2)/5 = 7$
-7 per ogni carattere: "**Volaree3-2023!**"

6 Repository GitHub

Il codice sorgente di PasswordCollector è disponibile per il download al seguente repository:

`https://github.com/lamferzon/Password-collector-in-Java`



(a) (b)

Figura 5: un paio di screenshot dell'applicazione in esecuzione.