Documentazione PasswordSecurityChecker

Erik Pelloni, SAMT I3BB 2021

Introduzione

Informazioni sul progetto

- Allievo e docente coinvolti nel progetto: Erik Pelloni, Luca Muggiasca (docente)
- SAM Trevano sezione informatica I3BB 2021/2022
- Data di inizio del progetto: 9 Settembre 2021
- Data di consegna del progetto: 23 Dicembre 2021
- **Background/Situazione iniziale**:

Un utente vuole controllare la sicurezza della propria password.

- **Descrizione del problema e motivazione**:

Il programma, nel modo spiegato seguentemente permette di verificare il grado di sicurezza della password in base a dei dati personali. È importante utilizzare delle password efficaci in modo da evitare il furto di account o dati.

- **Approccio/Metodi**: Sulla base di alcuni dati personali eventualmente passati al programma, quest'ultimo proverà a risalire alla password.
- **Risultati**: Il programma stampera l'esito finale (password trovata/non trovata), il numero di tentativi eseguiti per trovare la password e il tempo impiegato.

Abstract

> *Nowadays, everyone has many different accounts for which a password is required. Choosing a secure password is very important to avoid data theft. This program checks the security of the password entered, based on optional information passed in as a parameter.*
> *Passwords are often the only barrier between you and your personal information like data, money, privacy or even our identities.*

Scopo

Lo scopo del progetto è quello di creare un programma che sia in grado di dare un feedback riguardante la sicurezza di una password. Questo viene fatto cercando di trovare la password utilizzando delle combinazioni tra i dati passati come parametri e nel caso in cui la password non venisse trovata, con un attacco brute force.

Analisi

Analisi del dominio

Questo capitolo descrive il contesto in cui il prodotto verrà utilizzato:

- Il prodotto verrà utilizzato per avere un riscontro riguardo la sicurezza della propria password.

Prerequisiti

- Il prodotto può essere utilizzato da chiunque
- La competenza richiesta per l'utilizzo di questo programma è saper lanciare un programma tramite linea di comando. Non sono richieste altre conoscenze o competenze.
- È necessario avere Java installato sul dispositivo che si vuole utilizzare per eseguire il programma.

Analisi e specifica dei requisiti

Seguono una descrizione dell'analisi e la specifica dei requisiti.

- La funzione del prodotto è quella di controllare la sicurezza di una password, cercando di trovarla utilizzando dei dati relativi alla persona.
- Il prodotto è eseguibile tramite linea di comando e stampa i risultati a terminale.
- I dati verranno utilizzati unicamente per il funzionamento del programma e vengono eliminati una volta completata l'operazione.

In base alla lista dei requisiti e all'analisi degli stessi, è stata redatta una *specifica dei requisiti* nella quale sono elencate e descritte in modo dettagliato le funzionalità che il prodotto fornisce. I requisiti non rappresentano delle attività bensì delle caratteristiche che il prodotto possiede.

ID	Titolo	Descrizione	Priorità	Vers	Note
Req-01	Linea di comando	Il programma sarà eseguibile tramite linea di comando	1	1.0	
Req-02	Dati input	Al programma verranno passati i seguenti dati: nome e cognome, data di nascita e un terzo dato a propria scelta	1	1.1	I dati sono facoltativi
Req-03	Controllo validità	Verrà eseguito un controllo della validità dei dati inseriti	1	1.0	
Req-04	Help	Sarà possibile visualizzare un help che mostra il funzionamento del programma	1	1.0	
Req-05	Controllo semplice	Verranno controllate delle password combinate in modo semplice	1	1.0	
Req-06	Controllo complesso	Verranno controllate delle password combinate in modo più complesso	1	1.0	

ID	Titolo	Descrizione	Priorità	Vers	Note
Req-07	Controllo password frequenti	Verrà controllata una lista contenente delle password frequenti	1	1.0	
Req-08	Brute force	Se la password non viene trovata con i controlli precedenti si prosegue con un attacco brute force, se l'utente lo decide	2	1.0	
Req-09	Stampa valori	Vengono stampati i tentativi effettuati e il tempo impiegato	1	1.0	
Req-10	Script	Sarà possibile passare i dati a uno script tramite file csv, lo script invocherà automaticamente il programma	3	1.2	Sostituito dal requisito Req-11
Req-11	Gestione parametri	Il passaggio di parametri viene gestito all'interno del programma, utilizzando una libreria.	2	2.0	

Legenda elementi tabella dei requisiti:

ID: identificativo univoco del requisito

Titolo: titolo dato al requisito

Nome: descrizione del requisito

Priorità: indica l'importanza di un requisito nell'insieme del progetto, definita assieme al committente. Sono defininiti al massimo di 3 livelli di priorità in questo modo:

- 1: priorità maggiore
- 3: priorità minore

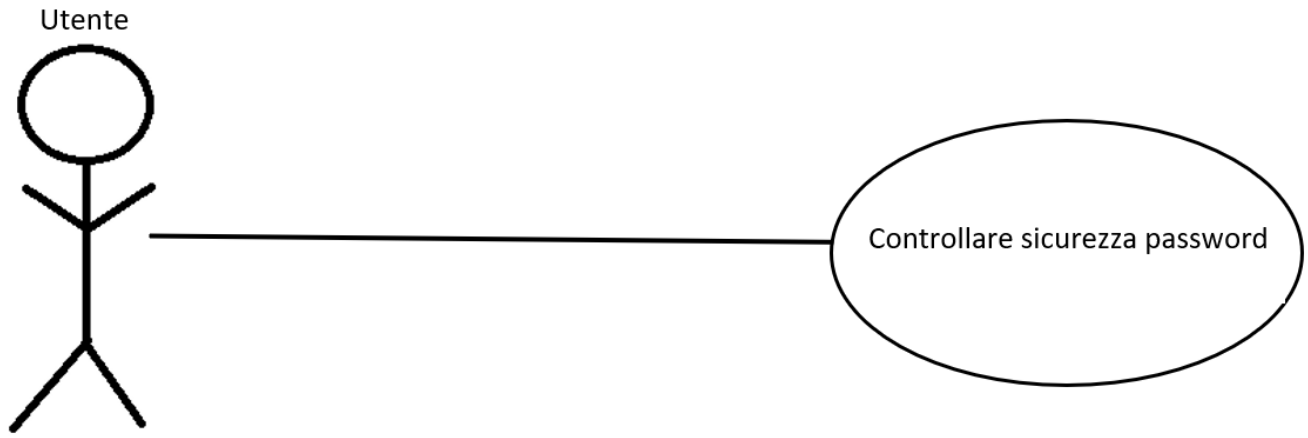
Versione: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l'ultima versione.

Note: eventuali osservazioni importanti o riferimenti ad altri requisiti.

Sotto requisiti: elementi che compongono il requisito.

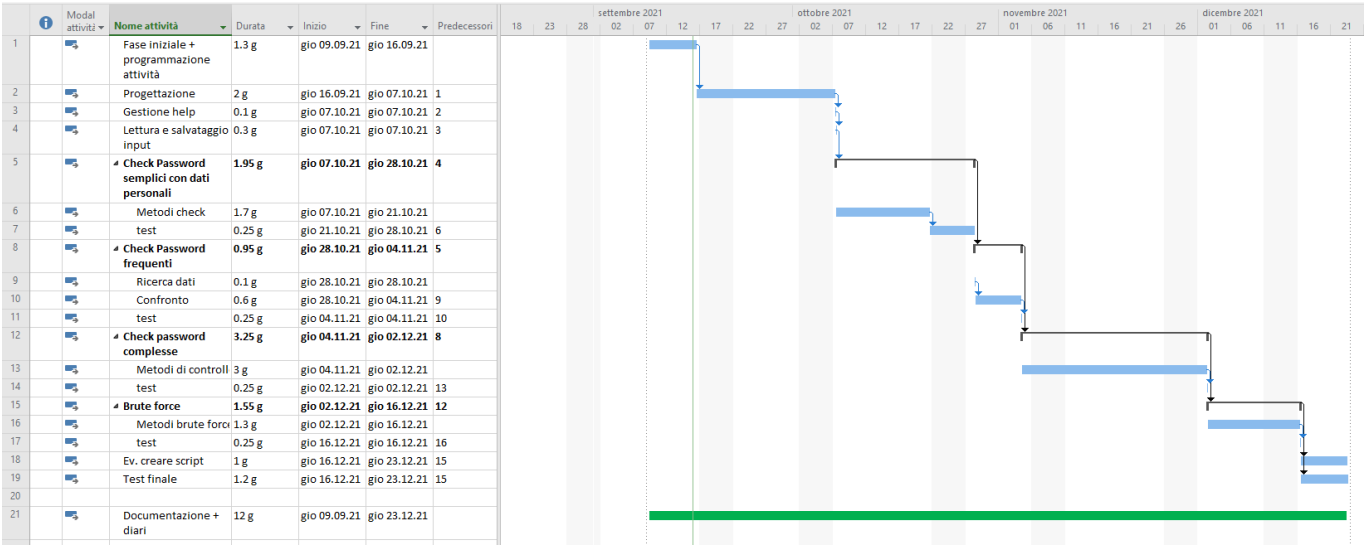
Use case



L'utente è in grado (tramite linea di comando) di verificare la sicurezza della propria password, anche in relazione con i propri dati personali.

Pianificazione

Gantt preventivo



	Modal attività	Nome attività	Durata	Inizio	Fine	Predecessori
1		Fase iniziale + programmazione attività	1.3 g	gio 09.09.21	gio 16.09.21	
2		Progettazione	2 g	gio 16.09.21	gio 07.10.21	1
3		Gestione help	0.1 g	gio 07.10.21	gio 07.10.21	2
4		Lettura e salvataggio input	0.3 g	gio 07.10.21	gio 07.10.21	3
5		**Check Password semplici con dati personali**	**1.95 g**	**gio 07.10.21**	**gio 28.10.21**	**4**
6		Metodi check	1.7 g	gio 07.10.21	gio 21.10.21	
7		test	0.25 g	gio 21.10.21	gio 28.10.21	6
8		**Check Password frequenti**	**0.95 g**	**gio 28.10.21**	**gio 04.11.21**	**5**
9		Ricerca dati	0.1 g	gio 28.10.21	gio 28.10.21	
10		Confronto	0.6 g	gio 28.10.21	gio 04.11.21	9
11		test	0.25 g	gio 04.11.21	gio 04.11.21	10
12		**Check password complesse**	**3.25 g**	**gio 04.11.21**	**gio 02.12.21**	**8**
13		Metodi di controll	3 g	gio 04.11.21	gio 02.12.21	
14		test	0.25 g	gio 02.12.21	gio 02.12.21	13
15		**Brute force**	**1.55 g**	**gio 02.12.21**	**gio 16.12.21**	**12**
16		Metodi brute forc	1.3 g	gio 02.12.21	gio 16.12.21	
17		test	0.25 g	gio 16.12.21	gio 16.12.21	16
18		Ev. creare script	1 g	gio 16.12.21	gio 23.12.21	15
19		Test finale	1.2 g	gio 16.12.21	gio 23.12.21	15
20						
21		Documentazione + diari	12 g	gio 09.09.21	gio 23.12.21	

Documento originale (.mpp)

Gantt consuntivo

Analisi dei mezzi

Per la creazione del progetto è stato utilizzato un PC con Windows 10, Java 16 e Visual Studio Code, non ci sono altri requisiti a livello hardware per la creazione e lo svolgimento di questo progetto.

Hardware

Nel mio caso specifico le specifiche del computer sul quale ho lavorato sono le seguenti:

- Nome SO Microsoft Windows 10 Enterprise
- Processore Intel(R) Core(TM) i5-7360U CPU @ 2.30GHz, 2301 Mhz, 2 core, 4 processori logici
- Memoria fisica installata (RAM) 16.0 GB

Software

- Visual Studio Code (version 1.63)

È stata utilizzata una libreria scritta da Paolo Bettelini, che permette una migliore gestione dei parametri passati come argomenti da linea di comando.

Il prodotto potrà essere eseguito su un qualsiasi dispositivo che abbia Java installato, indipendentemente dal sistema operativo.

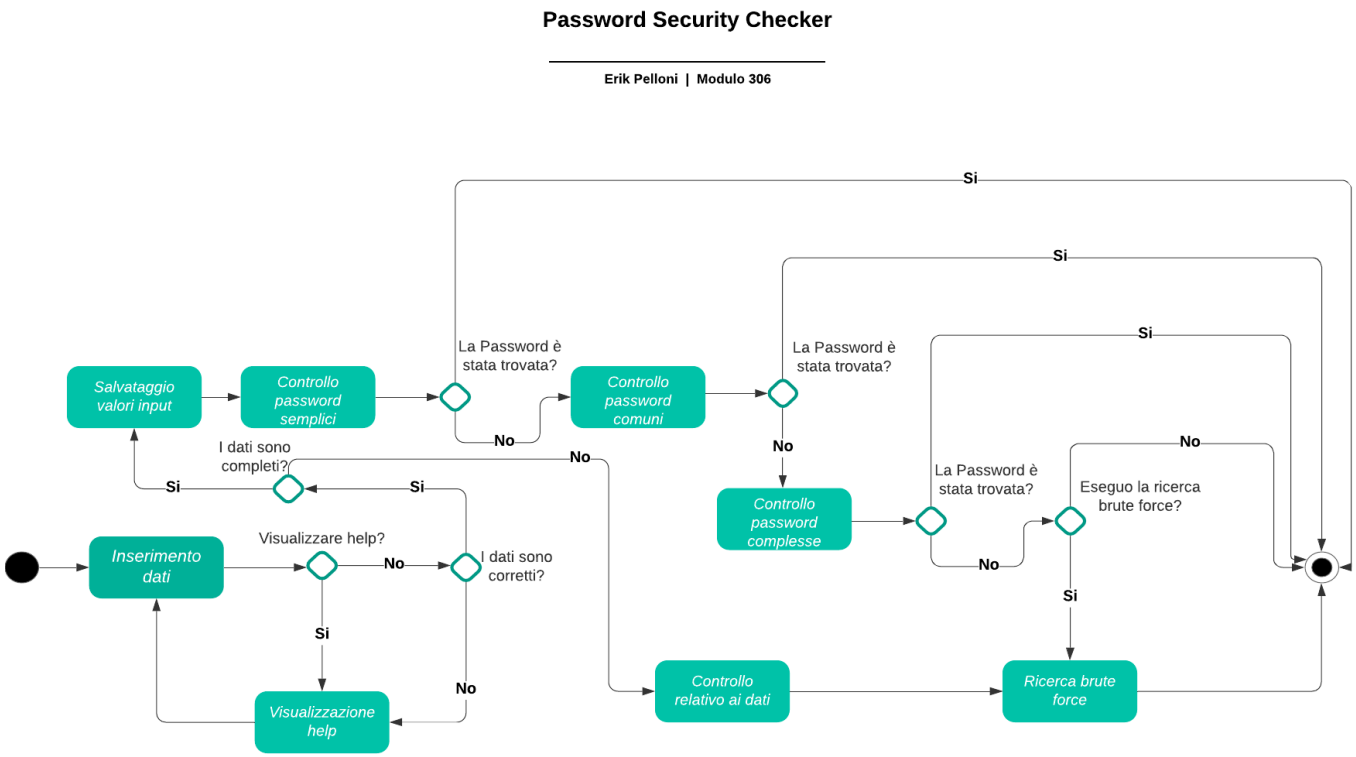
Progettazione

Questo capitolo descrive esaustivamente come deve essere realizzato il prodotto fin nei suoi dettagli. (Una buona progettazione permette all'esecutore di evitare fraintendimenti e imprecisioni nell'implementazione del prodotto.)

Design dell'architettura del sistema

- Si tratta di un programma che viene invocato tramite linea di comando e con il passaggio di argomenti.
- Composto da 1 classe scritta in java.

Activity Diagram



Descrizione

Lanciando l'applicazione si passano i dati necessari per il funzionmento.
Se le informazioni non sono passate nel modo corretto o se l'utente ha scento di visualizzare l'help, l'applicazione mostra l'help (che spiega la formattazine dei valori di input).

Una volta inseriti correttamente i dati il programma comincia a provare a forzare la password.

- In un primo momento vengono provate le combinazioni più semplici. Se la password viene trovata, il programma giunge a termine, se no si passa al controllo successivo.
- Il secondo controllo che viene effettuato è quello tra la password fornita e una lista di password frequentemente utilizzate. Anche in questo caso la procedura di funzionamento del programma è la medesima: se la password viene trovata, il programma giunge a termine, se no si passa al controllo successivo.
- Se la password non è presente nemmeno all'interno della lista, il controllo da compiere è quello delle combinazioni più complesse. Nuovamente la procedura finale è la stessa.
- Infine, se l'utente ha deciso di effettuarla, l'ultima ricerca effettuata è quella brute force. Una volta trovata la password o raggiunto l'eventuale limite (di tempo o di tentativi, stabilito dall'utente) il

programma giunge al termine e stampa l'output finale.

- *N.B. I controlli verranno effettuati solamente sui dati passati come parametro all'esecuzione del programma.*

Diagramma delle classi

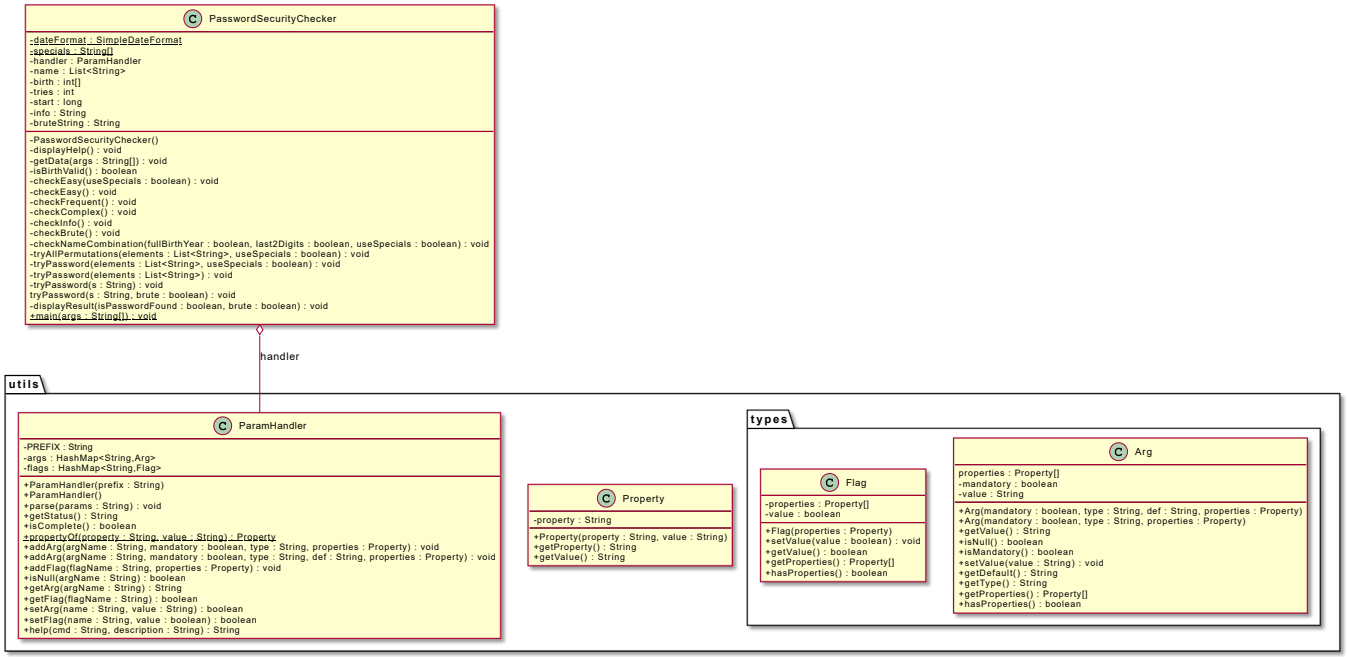


Immagine originale diagramma delle classi

Implementazione

Per questo progetto ho creato un'unica classe "`PasswordSecurityChecker`" in java, la quale si occupa di gestire i dati, creare le combinazioni, controllare se la password è stata trovata e stampare il messaggio finale a terminale.

Gestione parametri

Per la gestione dei parametri ho utilizzato la libreria ParamHandler di Paolo Bettelini e l'ho implementata in questo modo

```
public class PasswordSecurityChecker {
  ...
  ParamHandler handler;
  ...
  handler.addFlag(
                "b",
                ParamHandler.propertyOf("Name", "Brute Force"),
                ParamHandler.propertyOf("Description", "Performs the brute force
attack"));

  handler.addArg(
                "name", false, "String",
                ParamHandler.propertyOf("Name", "Name"),
                ParamHandler.propertyOf("Description", "Name and Surname"),
                ParamHandler.propertyOf("Format", "Name Surname"),
                ParamHandler.propertyOf("Example", "John Doe"));
  ...
  handler.parseArg();
  ...
  handler.getArg("name");
}
```

la sintassi per l'aggiunta di un parametro a un handler è la seguente

```
handler.addArg("argName", mandatory, "type", properties);
```

il primo parametro da passare è il nome dell'argomento da aggiungere ad handler. Il secondo è un boolean che definisce se l'argomento è obbligatorio o meno, il terzo specifica il tipo di dato che si aspetta il parametro (per mostrarlo nell'help) e infine vanno aggiunte le varie proprietà (composte da nome e descrizione).

Per controllare la validità della data di nascita ho utilizzato un oggetto SimpleDateFormat del package java.text e gli oggetti Calendar e Date del package java.util come mostrato qui sotto

```
...
SimpleDateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy");
...
Date birthDate = dateFormat.parse(handler.getArg("birth"));
Calendar c = Calendar.getInstance();
c.setTime(birthDate);
```

Con il comando parse provo ad inserire i dati dell'utente all'interno della variabile `birthDate`. Nel caso in cui quest'ultimi non fossero validi, verrà sollevata un'eccezione di tipo `ParseException`, quindi verranno richiesti nuovamente i dati.

L'accesso ai dati può essere effettuato utilizzando come nel seguente esempio

```
c.get(Calendar.DAY_OF_MONTH);
c.get(Calendar.YEAR);
```

All'interno del programma non ho tenuto valide le date di nascita con l'anno inferiore a 1000.

Controllo delle password

Metodi tryPassword

Per eseguire il controllo vero e proprio di una password ho creato diverse "versioni" del metodo `tryPassword`: ognuna delle quali accetta dei parametri diversi dall'altra. Il metodo utilizzato da tutti gli altri è quello mostrato qui sotto

```
void tryPassword(String s, boolean brute, boolean common){
    boolean flag = false;
    tries++;
    if(password.equals(s)){
        flag = true;
    }else if(!(brute || common)){
        for (String special : specials) {
            tries++;
            if(password.equals(s + special)){
                flag = true;
                break;
            }
        }
    }
    if (flag) {
        // trovata, fermo il tempo di ricerca
        end = System.currentTimeMillis();
        // metodo finale
```

```
            displayResult(true, brute);
        }
    }
```

Questo metodo confronta la password con la stringa passata come parametro e se queste corrispondono, allora invoca il metodo `displayResult()` per stampare il risultato e terminare il programma. Se invece la password non è la stringa passata, allora prova a concatenare dei caratteri speciali in coda a quest'ultima. Questa cosa viene ripetuta per ogni stringa controllata (tranne per quelle ricavate dall'attacco brute force e dalla lista di password comuni). Il parametro `brute` specifica se il tentativo è stato effettuato tramite l'attacco brute force.

Altre versioni dello stesso metodo sono ad esempio quelle riportate seguentemente

```
private void tryPassword(String s) {
    tryPassword(s, false);
}
```

Questo si tratta di un metodo che richiama il suo omonimo (il quale possiede però dei parametri differenti) con il secondo parametro fisso su `false`, in modo da poter definire un valore di "default" e dare quindi la possibilità di invocare il metodo senza dover specificare il secondo parametro.

È presente anche un metodo `tryPassword` che accetta come parametro una lista di stringhe. Quest'ultimo invocherà il metodo `tryPassword(String s)` concatenando tutti gli elementi della lista, anche utilizzando l'altro metodo `tryPassword`.

Metodo tryAllPermutations

Il metodo `tryAllPermutations`, data una lista di stringhe prova le password composte dalla combinazione degli elementi all'interno della stessa.

```
private void tryAllPermutations(List<String> elements) {
    List<String> copy = new ArrayList<>(elements);
    int[] indexes = new int[copy.size()];
    tryPassword(copy);

    int i = 0;
    while (i < copy.size()) {
        if (indexes[i] < i) {
            Collections.swap(copy, i % 2 == 0 ? 0 : indexes[i], i);
            tryPassword(copy);
            indexes[i]++;
            i = 0;
        } else {
            indexes[i] = 0;
            i++;
        }
    }
}
```

Per funzionare, come prima cosa eseguo una copia della lista, per poterla manipolare senza cambiare poi l'ordine della lista che viene passata come parametro. In seguito viene eseguito un ciclo che in base a un criterio scambia tra di loro gli elementi della lista, creando così sempre combinazioni differenti.

All'interno di questo metodo invoco `tryPassword` in modo da controllare subito se la nuova combinazione dei dati corrisponde alla password.

Metodi di controllo

Nel metodo `checkEasy` controllo password semplici come semplicemente il nome, il cognome, l'anno di nascita e semplici combinazioni tra queste informazioni. Questi dati però sono opzionali, quindi questi controlli potrebbero essere saltati.

Il secondo metodo ad effettuare una ricerca è il metodo `checkFrequent`. Questo controlla se la password si trova all'interno di una lista di password utilizzate di frequente con qualche mia aggiunta al milione di password già presenti. Le password sono salvate in un file, il metodo le legge, le inserisce all'interno di una lista e le prova una ad una utilizzando il metodo tryPassword, mostrato in precedenza.

Il terzo metodo, `checkComplex` controlla sempre le password utilizzando i dati passati come parametro, creando però anche delle combinazioni più complicate ed articolate concatenando diversi dati.

L'ultimo "metodo di controllo" è `checkBrute`, si tratta di un metodo che esegue una ricerca brute force utilizzando un array di caratteri che creo, inserendone solo alcuni (94 per la precisione), in modo da velocizzare il controllo

```
private void checkBrute() {
    // creo un array contenente tutti i caratteri da utilizzare
    char[] characters = new char[94];
    for (int i = 0; i < characters.length; i++) {
        characters[i] = (char)(i + 33);
    }
    int base = characters.length + 1;
    StringBuilder guess = new StringBuilder();
    int tests = 1;
    int c = 0;
    int m = 0;

    while (true) {
        int y = tests;
        while (true) {
            c = y % base;
            m = (int) Math.floor((y - c) / (double) base);
            y = m;
            int index = (c - 1);
            if (index < 0) {
                index = characters.length - 1 -(-index % characters.length);
            }
            guess.insert(0, characters[index]);
            if (m == 0) {
```

```
                    break;
                }
            }
            tryPassword(guess.toString(), true);
            // se non trova la password
            tests++;
            guess.setLength(0);
        }
    }
```

Display risultato

Il metodo `displayResult` è il metodo che serve per stampare il risultato della ricerca e terminare il programma.

Per terminare il programma ho usato il comando `System.exit(0)`.

Riceve i parametri per gestire il fatto che la password sia stata trovata e nel caso in cui fosse così se fosse stata trovata utilizzando l'attacco brute force oppure no.

Test

Protocollo di test

I test sono realizzati per garantire l'adempimento delle richieste formulate nei requisiti e fungono da garanzia di qualità del prodotto. Ogni test è ripetibile alle stesse condizioni.

N.B. I dati inseriti come parametro sono dei dati di esempio, per testare il corretto funzionamento del programma.

Test Case	**TC-01**
Nome	Funzionamento da terminale
Riferimento	REQ-01
Descrizione	Il programma funziona tramite linea di comando
Prerequisiti	Java installato sul computer, essere in possesso del programma jar e un terminale da cui eseguire il programma
Procedura	Aprire il terminale e digitare il comando scritto nella guida per eseguire il programma
Risultati attesi	Il programma si avvia

Test Case	**TC-02**
Nome	Dati obbligatori
Riferimento	REQ-03
Descrizione	I dati obbligatori sono richiesti
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma senza aggiungere l'argomento -password
Risultati attesi	L'errore viene gestito, il programma mostra il messaggio di help e termina.

Test Case	**TC-03**
Nome	Controllo dati
Riferimento	REQ-03
Descrizione	I dati inseriti verranno controllati
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma passando dei dati sbagliati, ad esempio la data di nascita con valore "02.06.10" oppure "prova"
Risultati attesi	L'errore viene gestito, il programma mostra il messaggio di help e termina.

Test Case	**TC-04**
Nome	Help
Riferimento	REQ-04
Descrizione	Sarà possibile visualizzare un help che mostra il funzionamento del programma
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma aggiungendo il parametro "-h"
Risultati attesi	Verrà mostrato il messaggi di help e il programma terminerà

Test Case	**TC-05**
Nome	Password semplice
Riferimento	REQ-05
Descrizione	La password viene trovata senza utilizzare l'attacco brute force
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma inserendo il nome e con password uguale al nome, ad esempio -name "John Doe" -password "John"
Risultati attesi	La password viene trovata senza l'attacco brute force, velocemente con pochi tentativi

Test Case	**TC-06**
Nome	Password complessa
Riferimento	REQ-06
Descrizione	La password viene trovata senza utilizzare l'attacco brute force
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma inserendo i seguenti parametri:-name "John Doe" -birth "01.01.1970" -password "Doe70John!!"
Risultati attesi	La password viene trovata senza l'attacco brute force.

Test Case	**TC-07**
Nome	Password comune
Riferimento	REQ-07
Descrizione	La password viene trovata senza utilizzare l'attacco brute force
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma inserendo il parametro -password "Password" (presente all'interno della lista)
Risultati attesi	La password viene trovata senza l'attacco brute force.

Test Case	**TC-08**
Nome	Brute force
Riferimento	REQ-08
Descrizione	La password viene trovata utilizzando l'attacco brute force
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma inserendo i parametri seguenti: -password "}k" -b (in modo da non aspettare troppo tempo)
Risultati attesi	La password viene trovata utilizzando l'attacco brute force.

Test Case	**TC-09**
Nome	Stampa risultato positivo
Riferimento	REQ-08
Descrizione	Se la password viene trovata, dev'essere stampato il risultato a terminale
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma inserendo i parametri seguenti: -password "Password"
Risultati attesi	"Password found in ... minutes,... and with ... tries without using brute force"

Test Case	**TC-10**
Nome	Stampa risultato negativo
Riferimento	REQ-08
Descrizione	Se la password non viene trovata, dev'essere stampato il risultato a terminale
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma inserendo i parametri seguenti: -password "Kaoabsoj" (senza il parametro "-b" per brute force)
Risultati attesi	"Password not found in ... minutes,... and with ... tries without using brute force"

Test Case	**TC-11**
Nome	Combinazione con caratteri maiuscoli
Riferimento	REQ-05 / REQ-06
Descrizione	Viene trovata una password con una combinazione particolare di caratteri in minuscolo e maiuscolo
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma passando i dati e una password con una combinazione di maiuscole e minuscole, ad esempio "jOhNDoE" avendo passato il dato "John Doe" come nome
Risultati attesi	La password viene trovata senza l'utilizzo dell'attacco brute force

Test Case	**TC-12**
Nome	Secondo nome e/o secondo cognome
Riferimento	REQ-02
Descrizione	È possibile inserire più nomi e cognomi
Prerequisiti	Vedi prerequisiti TC-01
Procedura	Avviare il programma inserendo come parametro -name una stringa che contenga più nomi "John Junior Doe Smith"
Risultati attesi	I nomi e cognomi vengono salvati

Risultati test

Test passati	Test falliti
TC-01	
TC-02	
TC-03	
TC-04	
TC-05	
TC-06	
TC-07	
TC-08	
TC-09	
TC-10	
	TC-11
TC-12	

Mancanze/limitazioni conosciute

Mi sono accorto troppo tardi, verso la fine del progetto, di aver non aver lavorato nel modo migliore, in quanto non ho dato abbastanza importanza alla fase di progettazione e ho fatto troppe cose "sul momento". Un motivo sicuramente è la mia poca esperienza, visto che questo si tratta del mio primo progetto con una durata simile. Questo errore mi ha fatto capire quanto sia importante una progettazione completa e precisa, **prima** di iniziare a scrivere il codice.

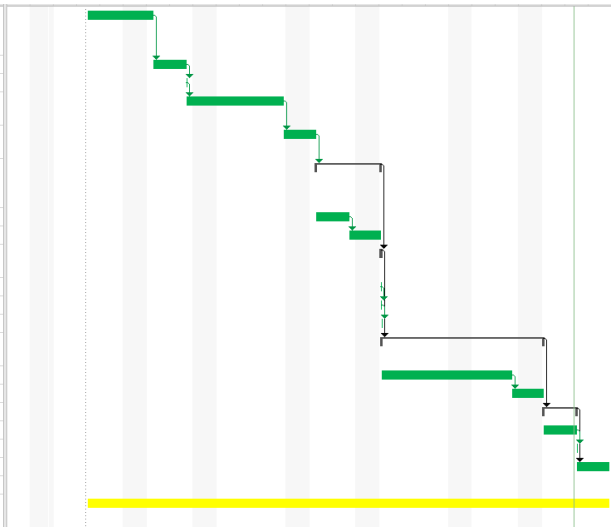
Sempre per lo stesso motivo, il mio codice in alcuni punti è molto "hard coded" e a volte parecchio ripetitivo. Con la dovuta progettazione non sarebbe successo.

Alcune cose sono riuscito a metterle a posto alla fine, ma altre non sono riuscito.

Consuntivo

Gantt consuntivo

1			Fase iniziale + programmazione attività	1.9 g	gio 09.09.21	gio 23.09.21	
2			Progettazione	1.1 g	gio 23.09.21	gio 30.09.21	1
3			Gestione help	0.15 g	gio 30.09.21	gio 30.09.21	2
4			Lettura e salvataggio input	2 g	gio 30.09.21	gio 21.10.21	3	È stata modificata la logica, utilizzando una libreria
5			Metodo confronto Password	0.4 g	gio 21.10.21	gio 28.10.21	4
6			**Check Password semplici con dati personali**	**1.6 g**	**gio 28.10.21**	**gio 11.11.21**	**5**
7			Metodi check	1.4 g	gio 28.10.21	gio 04.11.21	
8			test	0.2 g	gio 04.11.21	gio 11.11.21	7
9			**Check Password frequenti**	**0.6 g**	**gio 11.11.21**	**gio 11.11.21**	**6**
10			Ricerca dati	0.1 g	gio 11.11.21	gio 11.11.21	
11			Confronto	0.4 g	gio 11.11.21	gio 11.11.21	10
12			test	0.1 g	gio 11.11.21	gio 11.11.21	11
13			**Check password complesse**	**3.4 g**	**gio 11.11.21**	**gio 16.12.21**	**9**
14			Metodi di controll	3.2 g	gio 11.11.21	gio 09.12.21	
15			test	0.2 g	gio 09.12.21	gio 16.12.21	14
16			**Brute force**	**1.3 g**	**gio 16.12.21**	**gio 23.12.21**	**13**
17			Metodi brute forc	1.1 g	gio 16.12.21	gio 23.12.21	
18			test	0.2 g	gio 23.12.21	gio 23.12.21	17
19			Test finale	0.7 g	gio 23.12.21	gio 30.12.21	16	Vari test sono stati eseguiti durante lo svilupp
20							
21			Documentazione + diari + manuale	13 g	gio 09.09.21	gio 30.12.21	



Documento originale (.mpp)

Gantt preventivo

In termini di tempo, con il progetto sono rimasto in grandi linee nei tempi. Questo progetto in particolare, avendo il tempo può essere sempre portato in avanti, aggiungendo dei tentativi di password diversi.

L'unica cosa che per ciò che riguarda la gestione del tempo non è andata proprio bene è stata la documentazione, sono arrivato a concentrare una grandissima parte del lavoro nelle ultime settimane invece che documentare passo passo durante il lavoro e durante lo sviluppo.

Conclusioni

Purtroppo devo dire di non essere soddisfatto al 100% del mio prodotto finale, sono sicuro che con una progettazione più completa e precisa avrei potuto ottenere un risultato migliore. C'è anche da dire che questo è il primo progetto con una durata simile al quale lavoro; nonostante non sia totalmente soddisfatto sono molto contento di averci lavorato e di averlo comunque portato a termine, ottenendo un prodotto funzionante.

Per questo, credo che il progetto sia riuscito.

È stato abbastanza impegnativo, com'è giusto che sia, ma sento di aver appreso molto da quest'esperienza e che mi sarà molto utile in futuro.

Dubito altamente che cambierà il mondo, però ha cambiato me, in particolare il mio modo di pensare e lavorare a un progetto.

Sviluppi futuri

- Modifica della logica per rimuovere parti "hard coded" e provare più combinazioni, ad esempio con le maiuscole/minuscole
- Aggiunta password con caratteri speciali al posto di lettere, ad esempio 'a' --> '@'
- Aggiunta input password non in chiaro in modo interattivo
- Aggiunta di un'interfaccia grafica

Considerazioni personali

Come già scritto in precedenza, questo si tratta del mio primo progetto con una durata simile. Ho imparato molte cose, alcune più legate direttamente al linguaggio di programmazione, quindi conoscenze più tecniche per ciò che riguarda java, altre invece più per quanto riguarda la gestione di un progetto. Sicuramente quest'esperienza mi ha aiutato a capire bene, provandolo in prima persona, quanto sia fondamentale che la fase di progettazione sia svolta bene e fino ai più piccoli particolari per una buona riuscita del progetto. Bisogna cominciare a produrre codice solo una volta che si ha una progettazione completa e specifica.

Personalmente mi è piaciuto lavorare da solo, mi piace avere indipendenza, anche se devo ammettere che, paricolarmente per la documentazione, sono arrivato un po' alla fine a recuperare il lavoro non completato durante i mesi di lavoro.

Sono sicuro che quest'esperienza mi sarà utile in fururo, per i prossimi progetti.

Sitografia

- https://www.baeldung.com/java-array-permutations, *Permutations of an Array in Java*, 28.10.2021
- https://stackoverflow.com/questions/50215907/python-brute-force-password-guesser, 02.12.2021
- https://github.com/LuMug/Mod306, *Modulo 306*, in diverse date

Allegati

- Diari di lavoro
- Codici sorgente
- Manuale d'uso
- Qdc

@ErikPelloni