CAESARE CIPHER

Il programma CaesarCipher è distribuito sotto licenza GPL3. Una copia completa del progetto può essere reperita all'indirizzo https://github.com/Errore418/CaesarCipher.

PREFAZIONE

Il programma è scritto in Scala e basa il suo funzionamento sulla libreria Akka. L'interfaccia grafica è invece realizzata tramite l'uso combinato delle librerie ScalaFX e ScalaFXML, che wrappano la libreria JavaFX e ne espongono un DSL specifico per Scala. Una disamina accurata di tutte queste componenti è ben al di sopra delle capacità di questo elaborato, perciò verranno passate brevemente in rassegna senza entrare troppo nei particolari. Ciò che verrà esaminato ed illustrato a fondo sarà il funzionamento dell'algoritmo implementato ad alto livello e di come questo sia stato tradotto in Scala.

INTRODUZIONE

SCALA

Scala è un linguaggio di programmazione di alto livello sviluppato a partire dal 2001 da Martin Odersky e dal suo team. In quanto compila nativamente in bytecode risulta totalmente compatibile con il ben più famoso e diffuso linguaggio Java, ma da quest'ultimo si discosta per una maggiore complessità di sintassi ed un approccio totalmente orientato agli oggetti con una forte integrazione alla programmazione funzionale. Nato per superare alcune criticità e anacronismi del linguaggio Java, primo fra tutti l'assenza di un supporto di sintassi all'approccio funzionale, si è affermato in una cerchia ristretta ma molto affiatata di appassionati stufi dell'eccessiva verbosità del linguaggio della Oracle. Nonostante non sia mai stato adottato su larga Scala alcuni suoi aspetti hanno fatto così presa sui suoi utilizzatori da spingere Java stesso ad introdurli nella propria sintassi. Tale stretto legame tra i due linguaggio affonda già nel concepimento di Scala: Odersky era un ingegnere alla Oracle che ha guidato lo sviluppo e il rilascio al pubblico dell'intero meccanismo dei Generics in Java. Benché si possa scrivere codice Scala banalmente simile a quello Java, ci sono tutta una serie di feature che possono essere usate per trasformare il codice in qualcosa di incomprensibile per un puro programmatore Java. Una trattazione approfondita della sintassi Scala non sarà oggetto di questo testo, ci si limiterà ad illustrare brevemente i costrutti e gli idiomi usati per implementare l'algoritmo scelto. Maggiori informazioni posso essere reperite sul sito ufficiale del linguaggio https://www.scala-lang.org/ oppure nel testo Programming in Scala scritto dallo stesso Odersky.

AKKA

Akka è una libreria scritta dagli stessi sviluppatori di Scala. Benché si possa usare sia per Scala che per Java, l'utilizzo in Scala grazie ai maggiori costrutti sintattici presenti nel linguaggio risulta più chiaro e pulito. Implementa il paradigma concettuale della programmazione ad attori. Sostazialmente il programma è modellato in una seria di componenti, gli attori, che interagiscono tra di loro scambiandosi messaggi. I concetti chiave di tale paradigma sono l'inaccessibilità diretta degli attori, cioè agli attori non possono ricavarsi in nessun modo referenze dirette di altri attori, e l'immutabilità dei messaggi, cioè ogni messaggio non può essere alterato ma solo letto. Ciò permette di creare programmi concorrenti anche di una certa complessità mantenendo sotto controllo la sincronizzazione esplicita tramite monitor o semafori tra i componenti. Nel caso specifico di Akka la comunicazione tra attori non è vincolata a un'architettura specifica, un programma scritto con attori presenti tutti sulla stessa macchina più essere facilmente adattato al caso in cui alcuni attori risiedano su un macchine distinte e comunichino tramite protocolli di rete. Nel programma CaesarCipher è stata usata l'ultima versione rilasciata di Akka "akka-typed",

chiamata così in quanto a introdotto una forte tipizzazione della maggior parte delle componenti della libreria. Ad esempio se prima un attore poteva ricevere qualunque oggetto di qualunque tipo sotto forma di messaggio ed era suo compito valutarne l'utilità o la fruibilità, in akka-typed ogni attore dichiara esplicitamente la gerarchia dei tipi accettati. Maggiori informazioni possono essere reperite sul sito ufficiale della libreria https://akka.io/ oppure nel testo Akka in action (non aggiornato alla versione akka-typed).

ScalaFX e ScalaFXML

Le shell grafiche in Java vengono realizzate principalmente in Swing o in JavaFX, dove quest'ultimo risulta più moderno e ingloba al suo interno le componenti di Swing di cui ufficialmente Oracle ne ha terminato lo sviluppo. Benché in Scala si possa usare direttamente JavaFX è stata sviluppata una llibreria che permette di tradurre in un DSL più simile al linguaggio di Scala le componenti di JavaFX. ScalaFXML invece è una libreria che permette di sfruttare la struttura grafica di foglio fxml, rappresentante una finestra, e controller, rappresentante il codice eseguito dai componenti grafici di tale finestra. Tutta la parte grafica non essendo direttamente correlata con l'algoritmo implementato non sarà presa in esame, limitandosi a illustrare il funzionamento delle varie schermate. Maggiori informazioni su queste due librerie possono essere reperite rispettivamente a http://www.scalafx.org/ e https://github.com/vigoo/scalafxml .

Testo assegnato
Il cifrario di Cesare costituisce uno tra i più antichi algoritmi crittografici conosciuti. Esso si basa sull'utilizzo dell'aritmetica modulare e ogni carattere x viene criptato secondo la seguente relazione C(x) = (x + s) mod a, (1) Dove C è la funzione di cifratura, a il numero di lettere che compongono l'alfabeto considerato (a = 21, da 0 a 20 se consideriamo l'alfabeto italiano) e s indica il numero di spostamenti verso destra che devono essere compiuti per effettuare la sostituzione di un carattere. Supponiamo ad esempio di lavorare con l'alfabeto italiano, quindi a = 21, e di volere trovare C(x) dove x = 0 b 0 , sapendo che s = 3. Avremo dunque che C(x) = 0 e 0 . Implementare la procedura di cifratura di Cesare considerando l'alfabeto italiano, in modo da poter cifrare una qualunque stringa di caratteri appartenenti a tale alfabeto. L'implementazione dell'esercizio richiesto deve essere accompagnata da relazione che comprenda una breve introduzione al tipo di tecnica utilizzata e la spiegazione del codice.

Implementazione ad alto livello
Per implementare l'algoritmo di Cesare sopradescritto è stato scelto un approccio originale in cui ogni carattere dell'alfabeto è modellato come un attore a cui viene indicato chi sia l'attore corrispondente del prossimo carattere dell'alfabeto preso in esame. Posto quindi un attore guardiano che vigila sul funzionamento di ogni altro attore, all'avvio della procedura creerà tanti attori quanti sono i caratteri dell'alfabeto considerato. Successivamente ad ognuno di questi attori indicherà l'attore suo prossimo tramite opportuno messaggio. L'attore guardiano perciò ricevuta la stringa da criptare la spezzetterà nei singoli caratteri e per ogni carattere manderà un messaggio all'attore corripondente. Tale messaggio conterrà sia l'indice del caratere all'interno della stringa di input sia il numero di spostamenti a destra a cui sottoporre il carattere. L'attore del carattere riceverà pertanto tale messaggio e si limiterà a rispedirlo a sua volta al suo prossimo attore diminuendo la lunghezza di shift di uno. Tale procedura si ripeterà identica finché un attore non riceverà un messaggio con lunghezza pari a zero, ciò infatti segnalerà il termine dello operazione di shifting e il corretto criptaggio del carattere preso in esame. A questo punto l'attore risponderà al guardiano segnalano il proprio carattere e l'indice nella stringa di output. Dopo che il guardiano riceverà tanti messaggi quanti ne avrà spediti, capirà di avere a disposizione tutti i caratteri correttamente criptati e potrà riordinarli in base al loro indice formando la stringa di output, che potrà quindi essere mostrata all'utente. Come esempio criptiamo la stringa "a" con l'algoritmo appena descritto:

- Il guardiano viene creato in modalità di cifratura e con un alfabeto latino minuscolo. Vengono creati 24 attori e ad ognuno viene assegnata una lettera. Successivamente il primo attore (a) riceverà un messaggio di linking contenente l'attore (b) in quanto suo prossimo. All'attore (b) arriverà un messaggio contenente l'attore (c) e così via per tutti i caratteri fino all'attore (z) a cui verrà indicato l'attore (a).
- Il guardiano riceverà la stringa "a" tramite messaggio. Leggendo carattere per carattere invierà un messaggio all'attore (a) con indice 0 e lunghezza di shift di 3.
- L'attore (a) ricevuto il messaggio dal guardiano ne manderà a sua volta uno all'attore (b) con stesso indice 0 e lunghezza di shift pari a 2.
- L'attore (b) ricevuto il messaggio dall'atttore (a) ne manderà a sua volta uno all'attore (c) con stesso indice 0 e lunghezza di shift pari a 1.
- L'attore (c) ricevuto il messaggio dall'atttore (b) ne manderà a sua volta uno all'attore (d) con stesso indice 0 e lunghezza di shift pari a 0.
- L'attore (d) ricevuto il messaggio dall'atttore (c) e accortosi della lunghezza di shift pari a zero inoltrerà al guardiano un messaggio contenten il proprio carattere (d) e l'indice 0.
- Il guardiano ricevuto il messaggio dall'attore (d) e accortosi di aver ricevuto tanti messaggi quanti ne aveva mandati crea la stringa di output "d" grazie anche agli indici corrispondenti ai caratteri ricevuti e la mostra all'utente.
- Il guardiano ferma la sua esecuzione, chiudendo in cascata tutti gli attori che aveva fatto partire.

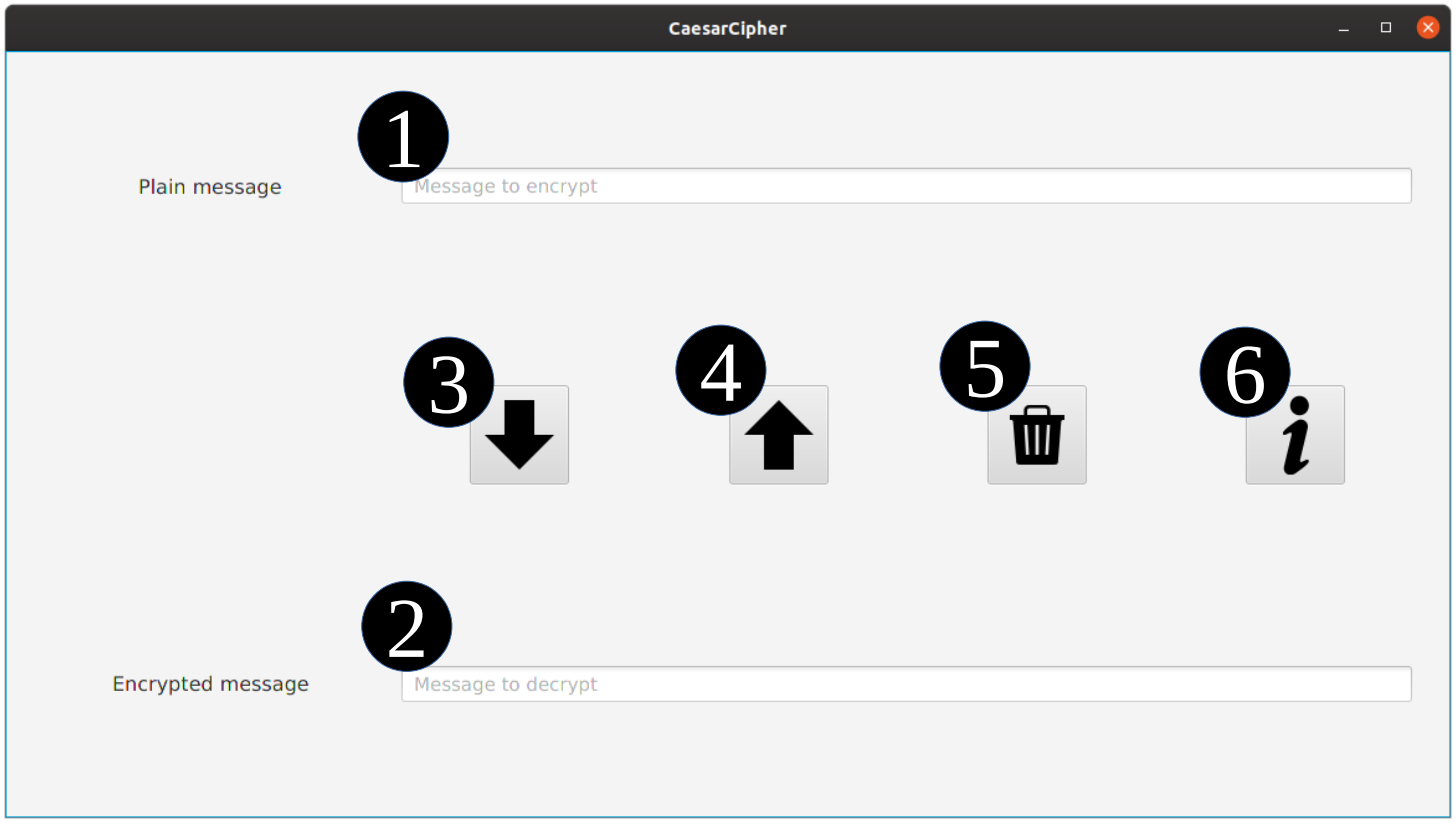
Ovviamente nel caso banale di un solo carattere l'invio e la ricezione dei messaggi si svolge in maniera serializzata, ma quando il messaggio è composto da più caratteri l'invio e la ricezione dei messaggi sono arbitrari e dipendono dall'interleaving dei vari thread degli attori generato dallo scheduler del sistema operativo. Inoltre se il guardiano trova dei caratteri nella stringa di input fuori dall'alfabeto considerato a cui quindi non corrisponde nessun attore, invia a se stesso un messaggio contentente il carattere estraneo trovato e il suo indice, così da apparire invariati nella stessa posizione anche nella stringa di output. Per la decifratura il meccanismo è esattamente lo stesso, l'unica cosa che cambia è il linking dei caratteri: se agli attori dei caratteri si inviano gli attori corrispondenti ai caratteri precedenti invece che successivi lo shifting avverrà a sinistra, così da invertire la cifratura e produrre una stringa in chiaro.

ESECUZIONE

Nonostante il programma sia scritto in Scala tramite l'uso di plugin appositi il jar finale è compilato in modo tale da risultare eseguibile da una semplice jre versione 8 o superiore. Per controllare di avere correttamente installato java sulla propria macchina basterà aprire una finestra di terminale e verificare che l'output del comando "java -versione" sia regolare. Nel caso in cui si sappia di essere sprovvisti di java, il comando non sia stato esegutio in quanto "java" non è un comando riconosciuto oppure la versione di java presente è inferiore alla 8 si consigliano le guide ufficiali della oracle in merito: https://java.com/en/download/help/download_options.xml e https://www.java.com/it/download/help/path.xml. Una volta installato correttamente java aprire una finestra di terminale e spostarsi nella cartella in cui è presente il file CaesarCipher-X.X-os.jar ed eseguirlo con "java -jar CaesarCipher-X.X-os.jar". Dopo l'apertura della finestra del programma sul terminale verranno stampati i log del programma. Per controllarne il livello lanciare il programma con "java -Dorg.slf4j.simpleLogger.defaultLogLevel=level -jar CaesarCipher-X-X-os.jar" dove level può assumenre uno dei seguenti valori: error, il livello più alto che lascerà stampare a schermo solo le eccezioni gravi lanciate durante l'esecuzione del programma; info, il livello di default che stamperà a schermo qualche informazione utile sull'esecuzione del programma; debug, il livello più basso che stamperà a schermo una gran quantità di informazioni utili per ricostruire con precisione il flusso d'esecuzione del programma.

MODALITA’ D’USO

Quando si avvierà il programma ci si troverà di fronte alla seguente schermata:



- Per criptare del testo inserirlo in (1) e premere il bottone (3). Il testo criptato verrà inserito in (2).
- Per decriptare del testo inserirlo in (2) e premere il bottone (4). Il testo decriptato verrà inserito in (1).
- Premere il bottone (5) per pulire (1) e (2).
- Premere il bottone (6) per aprire la schermata riportante la licenza del programma e i crediti del materiale terzo usato.
- Sia le stringhe da criptare che quelle da decriptare verranno epurate di eventuali spazi iniziali o finali. Una stringa vuota o composta da soli spazi non verrà criptata.