CAESAR CIPHER

Claudio Nave 6101907 claudio.nave@stud.unifi.it

PREFAZIONE

Il programma è scritto in Scala e basa il suo funzionamento sul framework Akka. L'interfaccia grafica è invece realizzata tramite l'uso combinato delle librerie ScalaFX e ScalaFXML, che wrappano il componente 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 sarà il funzionamento dell'algoritmo implementato.

INTRODUZIONE

SCALA

Scala è un linguaggio di programmazione ad alto livello sviluppato a partire dal 2001 da Martin Odersky e dal suo team. Dal momento che compila nativamente in bytecode risulta totalmente compatibile con il ben più famoso e diffuso linguaggio Java, discostandosene però per una maggiore complessità di sintassi ed un approccio totalmente orientato agli oggetti con una forte integrazione alla programmazione funzionale. Nato per superare alcuni anacronismi e criticità 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 linguaggi 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. Una trattazione approfondita della sintassi Scala non sarà oggetto di questo testo. 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 un framework scritto 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. Sostanzialmente il programma viene 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 gli 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 ed evitando controlli espliciti di flusso tramite monitor o semafori. 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 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 ha introdotto una forte tipizzazione della maggior parte dei componenti del framework. 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 del framework 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 del primo, di cui ufficialmente Oracle ne ha terminato lo sviluppo. Benché in Scala si possa usare direttamente JavaFX è stata sviluppata una libreria che permette di tradurre tale componente Java in un DSL più simile al linguaggio di Scala. ScalaFXML invece è una libreria che permette di sfruttare la struttura grafica di foglio fxml, un file scritto in un particolare tipo di xml rappresentante una schermata, e controller, rappresentante il codice interagente con i componenti grafici di tale schermata. Tutta la parte grafica non essendo direttamente correlata con l'algoritmo implementato non sarà presa in esame. Maggiori informazioni su queste due librerie possono essere reperite rispettivamente su 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

Per implementare il cifrario di Cesare 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 corrispondente. Tale messaggio conterrà sia l'indice del carattere 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 la corretta cifratura del carattere preso in esame. A questo punto l'attore risponderà al guardiano segnalando 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 e mostrandola all'utente.

Come esempio criptiamo la stringa “abc” con l’algoritmo appena descritto. Per semplicità il flusso dei messaggi viene presentato in una forma serializzata, ma nella realtà è lo scheduler del sistema operativo a stabilire di volta in volta l’esecuzione di un particolare attore generando un interleaving nel flusso dei messaggi arbitrario:

- Il guardiano viene creato in modalità di cifratura, con un alfabeto latino minuscolo e una lunghezza di shift pari a 3. Vengono creati 26 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 “abc” tramite messaggio. Leggendo carattere per carattere invierà un messaggio all’attore (a) con indice 0 e lunghezza di shift di 3, un messaggio all’attore (b) con indice 1 e lunghezza di shift di 3, un messaggio all’attore (c) con indice 2 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’attore (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’attore (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’attore (c) e accortosi della lunghezza di shift pari a 0 inoltrerà al guardiano un messaggio contenente il proprio carattere (d) e l’indice 0.
- Il guardiano ricevuto il messaggio dall’attore (d) registrerà il carattere ricevuto e il relativo indice in un apposito buffer.
- L’attore (b) ricevuto il messaggio dal guardiano ne manderà a sua volta uno all’attore (c) con stesso indice 1 e lunghezza di shift pari a 2.
- L’attore (c) ricevuto il messaggio dall’attore (b) ne manderà a sua volta uno all’attore (d) con stesso indice 1 e lunghezza di shift pari a 1.
- L’attore (d) ricevuto il messaggio dall’attore (c) ne manderà a sua volta uno all’attore (e) con stesso indice 1 e lunghezza di shift pari a 0.
- L’attore (e) ricevuto il messaggio dall’attore (d) e accortosi della lunghezza di shift pari a 0 inoltrerà al guardiano un messaggio contenente il proprio carattere (e) e l’indice 1.
- Il guardiano ricevuto il messaggio dall’attore (e) registrerà il carattere ricevuto e il relativo indice in un apposito buffer.
- L’attore (c) ricevuto il messaggio dal guardiano ne manderà a sua volta uno all’attore (d) con stesso indice 2 e lunghezza di shift pari a 2.
- L’attore (d) ricevuto il messaggio dall’attore (c) ne manderà a sua volta uno all’attore (e) con stesso indice 2 e lunghezza di shift pari a 1.
- L’attore (e) ricevuto il messaggio dall’attore (d) ne manderà a sua volta uno all’attore (f) con stesso indice 2 e lunghezza di shift pari a 0.

- L’attore (f) ricevuto il messaggio dall’attore (e) e accortosi della lunghezza di shift pari a 0 inoltrerà al guardiano un messaggio contenente il proprio carattere (f) e l’indice 0.
- Il guardiano ricevuto il messaggio dall’attore (f) registrerà il carattere ricevuto e il relativo indice in un apposito buffer.
- Il guardiano accortosi di aver ricevuto tanti messaggi quanti ne aveva mandati crea la stringa di output “def” grazie 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.

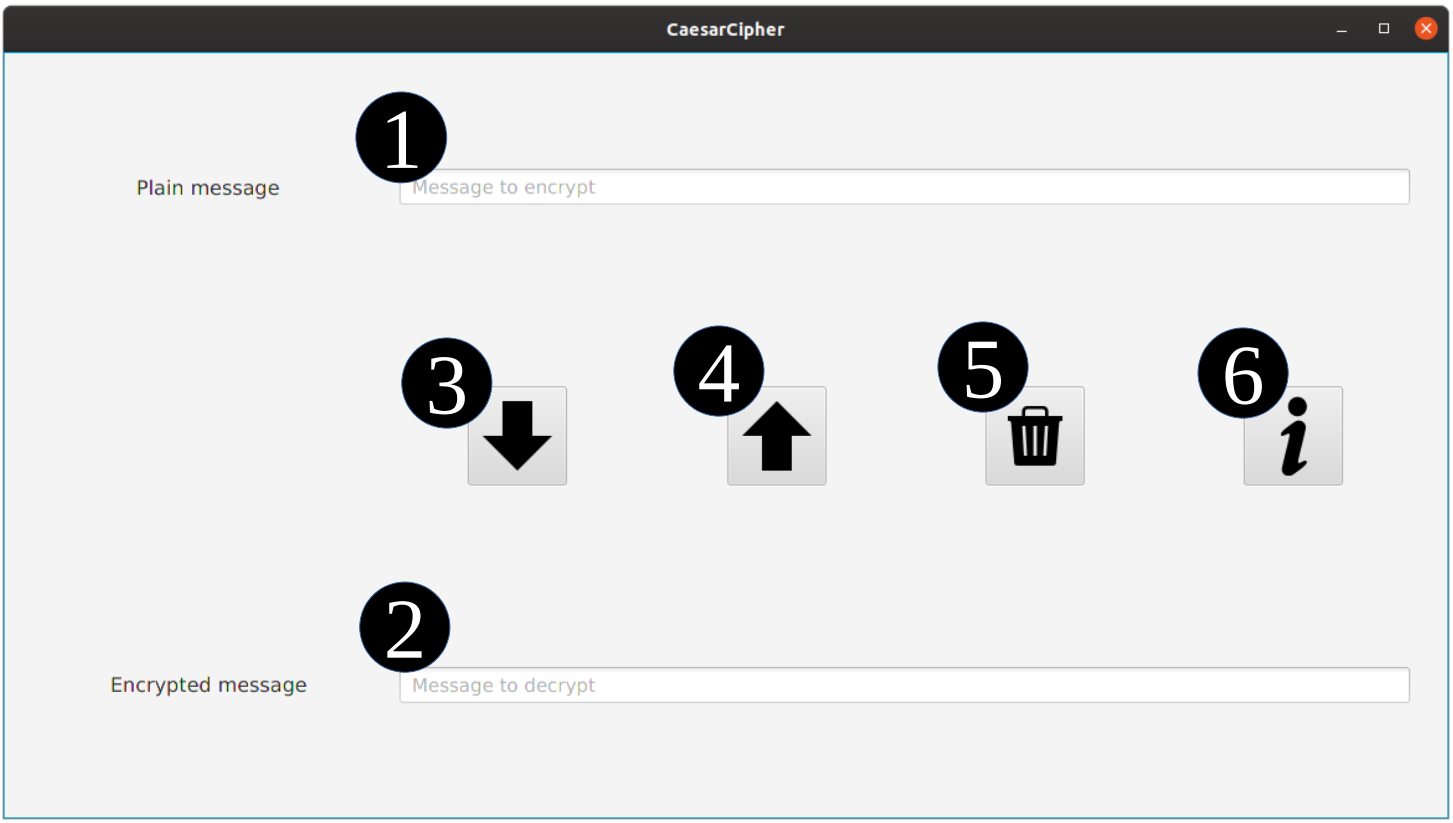
In caso di caratteri nella stringa di input fuori dall’alfabeto considerato a cui quindi non corrisponde nessun attore, il guardiano invia a se stesso un messaggio contenente il carattere estraneo trovato e il suo indice, così da apparire invariati e nella stessa posizione anche nella stringa di output. Per la decifrazione 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 appositi plugin 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 (su Windows si sconsiglia l’uso di PowerShell) e verificare che l’output del comando “*java -version*” sia regolare. Nel caso in cui si sappia di essere sprovvisti di java, il comando non sia stato eseguito in quanto “*java non è un comando riconosciuto*” oppure la versione di java presente sia 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 si trova file *CaesarCipher-X.X-os.jar* ed eseguirlo con “*java -jar CaesarCipher-X.X-os.jar*”. Dopo l’apertura della finestra principale sul terminale verranno cominciati ad essere stampati i log dell’esecuzione. Per controllarne il livello lanciare il programma con “*java -Dorg.slf4j.simpleLogger.defaultLogLevel=level -jar CaesarCipher-X-X-os.jar*” dove *level* può assumere uno dei seguenti valori: *error*, il livello più alto che lascerà stampare a schermo solo le eccezioni gravi; *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.

MODALITÀ 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 utilizzato.
- 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.