Università degli Studi dell'Insubria
Laurea Triennale in Informatica

+Immuni

Manuale Tecnico

Progetto per Laboratorio interdisciplinare A

AUTORI:

Davide Mainardi (mat. 746490)
Brenno Re (mat. 747060)
Luca Muggiasca (mat.744565)
Marc Cepraga (mat. 744101)

Sommario

Introduzione

+Immuni è un progetto sviluppato nell'ambito del progetto di Laboratorio A per il corso di laurea in Informatica dell'Università degli Studi dell'Insubria.

Il progetto è stato sviluppato in Java 8 (per esattezza il jdk 8.311), usa un'interfaccia grafica ideata con il tool SceneBuilder (sulla base del funzionamento della libreria javafx) ed è stato sviluppato e testato sui sistemi operativi Windows, MacOs e Linux (varie distro).

Elenco tecnologie (software e librerie) utilizzate

- **Intellij IDEA**: è un IDE per il linguaggio di programmazione java, tramite il quale è stato possibile sviluppare l'intero codice sorgente dal backend fino ai controller per il frontend ed è stato utilizzato anche per la creazione del jar.

- **Github**: Github, o meglio Git è lo strumento di versioning che abbiamo usato per collaborare a scrivere l'applicativo e condividere codice visionato tra di noi.

- **SceneBuilder**: è un IDE (ambiente di sviluppo integrato) che consente di evitare la creazione manuale di un file FXML e rappresenta la via più ottimale dal punto di vista del tempo richiesto e della complessità del lavoro. Con SceneBuilder è stata realizzata l'interfaccia grafica del progetto.

- **JavaFx**: è una libreria che permette di caricare dei file fxml e di usarli come interfaccia grafica, dando anche metodi, classi e interfacce per scrivere i controller associati a ogni pagina del frontend

- **Javadoc**: è un applicativo incluso nel Java Development Kit ed è utilizzato per la generazione automatica della documentazione del codice sorgente scritto in linguaggio Java.

Struttura generale del sistema di classi Java

Il progetto è strutturato fondamentalmente in due gruppi ideali: le classi backend, che si occupano della vera e propria elaborazione dei dati, e le classi frontend adibite alla gestione dell'interfaccia grafica.

Fisicamente le classi java sono strutturate a package, sotto la cartella del codice sorgente (src), a seconda della loro mansione.

- Classi BackEnd
  - Pakage centrivaccinali
    - CentriVaccinali (main class)
    - CentroVaccinale
    - CittadinoVaccinato
    - EventoAvverso
    - GestioneCentriVaccinali
    - GestioneVaccinati
    - InfoEventoAvversoAnonimo

- Package cittadini
  - Cittadini
  - CittadinoRegistrato
  - GestioneCittadinoRegistrato
- Package criptazione
  - AlgoritmoMD5
- Package gestionefile
  - GestioneCsv

Classi FrontEnd

- Pakage controllers
  - CittadinoViewController
  - FixInput
  - InserisciEventoAvversoController
  - LoginCittadinoController
  - MainUIController (launcher UI)
  - OperatoreViewController
  - RegistraCentroVaccinaleController
  - RegistraVaccinatoController
  - RegistrazioneCittadinoController
  - RicercaCentro1Controller
  - RicercaCentro2Controller
  - RicercaViewController
  - VisualizzaInfoController
  - WelcomeController

Sono presenti anche altri tre package nel codice, che però non contengono classi Java, ma altri file correlati all'interfaccia grafica. I package in questione sono css, fxml e images.

Verranno presentate ora le classi BackEnd nel dettaglio, non verrà affrontata invece una discussione dettagliata delle classi adibite alla gestione dell'interfaccia grafica poiché di interesse marginale e non essendo l'interfaccia grafica espressamente richiesta nei requisiti del progetto.

Classi Backend

GestioneCsv

GestioneCsv è sicuramente una delle classi più importanti all'interno del codice, poiché si occupa di delineare i metodi per la scrittura, lettura, ricerca e verifica dei dati sui file di memoria CSV.

Ogni classe della famiglia Gestione estende GestioneCsv, così da usufruire dei metodi di lettura/scrittura e di ricerca delineati nella classe padre.

Piccola spiegazione di una classe Gestione

Prendiamo come esempio la classe GestioneCentriVaccinali, in questa classe sono presenti i seguenti metodi:

- **registraCentroVaccinale**: che si occupa di preparare l'oggetto CentroVaccinale, passato dall'interfaccia grafica, alla scrittura sul file apposito richiamando il metodo scritturaFile della classe GestioneCsv.
- **cercaCentroEsiste**: questo metodo effettua una veloce verifica se il centro vaccinale inserito esiste.
- **getCentriVaccinali**: legge il file contenete tutti i centri vaccinali e restituisce una lista di oggetti CentroVaccinale.
- **searchCentroByName**: effettua una ricerca per Nome e restituisce una lista di oggetti CentroVaccinale che rispetti la condizione.
- **searchCentroByComuneAndTipologia**: effettua una ricerca per Comune e Tipologia e restituisce una lista di oggetti CentroVaccinale che rispetti la condizione.

Le altre classi figlio sono analoghe, alcune con metodi più specifici e altre con meno metodi dettagliati.

Complessità stimate

Andiamo a presentare una panoramica delle complessità dei vari metodi all'interno di GestioneCsv.

- **findfile()**
  Questo metodo fa una ricerca in cascata e verifica se esistono i file che servono all'app per gestire i dati. Definiamo F come il numero di file contenuti nella directory, si delineano 3 casi:
  - caso migliore: la directory contiene solo il file jar, quindi la complessità è uguale a O(F) = O(1)
  - caso medio: la directory contiene solo file, quindi la complessità è O(F)
  - caso peggiore: nella directory sono presenti delle sotto-directory, in questo caso il metodo viene richiamato per ogni sotto-directory. Quindi la complessità O(F) viene sommata n volte, dove n sono le volte che viene richiamato il metodo.
- **controllaNomiColonne()**
  In questo metodo le operazioni rilevanti sono:
  - In scrittura su file, se definiamo L il numero di righe nel file abbiamo tre possibili scenari:

        - caso medio: il file è vuoto, quindi il metodo scriverà solo una linea -> O(5*1) = O(1) Le operazioni in questo scenario dell'if sono 5 ma la complessità essendo costante è O(1).
        - caso migliore: il file non è vuoto e la prima linea è corretta, quindi il metodo non scriverà nessuna linea sul file -> O(4*1) = O(1)
        - caso peggiore: il file non è vuoto e la prima linea non soddisfa la condizione, quindi il metodo riscrive tutte le righe del file spostando tutte le righe di 1. Questo scenario prevede un ciclo while di complessità -> O(2*L) = O(L) e un ciclo for di complessità -> O(2*L) = O(L). La complessità peggiore dell'if è O(L) + O(L) = O(L)
    - Le altre operazioni hanno complessità uguale a O(1)

    La complessità del metodo nel caso peggiore è O(L) , invece nel caso medio e nel caso migliore è O(1)

- **letturaFile()**

  In questo metodo le operazioni rilevanti sono:
    - Definiamo L come il numero di righe nel file:
        - caso migliore: la linea è vuota, dopo aver verificato la condizione nel ciclo while si passa a reader.close() ed infine, dopo il catch, a return listaRighe. Si ha quindi una complessità O(1)
        - caso peggiore: la linea non è vuota, viene eseguito il ciclo while per L volte. Si ha quindi una complessità O(1*L) = O(L)
- **scritturaFile()**

  Il metodo ha complessità O(1)
- **ricercaAttrEsiste()**

  In questi metodi le operazioni rilevanti sono:
    - Definiamo L come il numero di righe nel file:
        - caso migliore: la linea è vuota, dopo aver verificato la condizione nel ciclo while si passa a reader.close() ed infine, dopo il catch, a return idExist. Si ha quindi una complessità O(1)
        - caso peggiore: la linea non è vuota, viene eseguito il ciclo while per L volte. Ad ogni iterazione viene eseguito String.split() per n volte, dove n è il numero separatori CSV(virgole) che è un valore costante per ogni linea. Si ha quindi una complessità = O(L)
- **getNomeCentroByIdVaccinato()**

  In questi metodi le operazioni rilevanti sono:
    - Definiamo L come il numero di righe nel file:
        - caso migliore: la linea è vuota, dopo aver verificato la condizione nel ciclo while si passa a reader.close() ed infine, dopo il catch, a return nomeCentro. Si ha quindi una complessità O(1)
        - caso peggiore: la linea non è vuota, viene eseguito il ciclo while per L volte. Ad ogni iterazione viene eseguito String.split() per n volte, dove n è il numero separatori

CSV(virgole) che è un valore costante per ogni linea. Si ha quindi una complessità = O(L)

GestioneVaccinati

GestioneVaccinati è una classe che si occupa di registrare i cittadini vaccinati e di gestire gli eventi avversi

Complessità stimate

Andiamo a presentare una panoramica delle complessità dei vari metodi all'interno di GestioneVaccinati.

- **nextIdUniv()**
  Questo metodo restituisce un nuovo id non utilizzato.
  - Dato n come numero di righe presenti nel file Vaccinati.dati.CSV:
    - Caso migliore: quando n = 0, dopo aver verificato la condizione del ciclo while si passa a return idUniv. La complessità è quindi O(1)
    - Caso peggiore: le iterazioni del while dipendono da n. la complessità è quindi O(n)
- **registraVaccinato()**
  Questo metodo dato un oggetto cittadinoVaccinato prepara la riga che verrà scritta nel file.
  La sua complessità è O(1) perché è composto da semplici istruzioni
- **inserisciEventiAvversi()**
  Questo metodo si occupa di inserire l'evento avverso per ogni cittadinoVaccinato.
  La sua complessità è O(n) dove n è il numero di cittadini vaccinati
- **getCittadiniVaccinati()**
  Questo metodo restituisce la lista dei cittadini vaccinati.
  La sua complessità è O(n) dove n è il numero di cittadini vaccinati
- **eventiAvversiToString()**
  Questo metodo formatta l'array degli eventi avversi in una stringa.
  La sua complessità è O(e) dove e è il numero degli eventi avversi di un cittadinoVaccinato
- **eventiAvversiToArray()**
  Questo metodo converte la stringa degli eventi avversi del cittadinoVaccinato in un array.
  La sua complessità è O(e) dove e è il numero degli eventi avversi di un cittadinoVaccinato
- **getAllEventiAvversi()**
  Questo metodo restituisce la lista di tutti eventi avversi.
  La sua complessità è O(n*e) dove n è il numero di cittadini vaccinati ed e è il numero di eventi avversi per ogni cittadinoVaccinato

AlgoritmoMD5

AlgoritmoMD5 è una classe che si occupa di criptare la password del cittadinoRegistrato.

Complessità stimate

Andiamo a presentare una panoramica delle complessità dei vari metodi all'interno di AlgoritmoMD5.

- **Converti()**

  Data una password in chiaro, questo metodo si occupa di convertirla in criptazione MD5.

  La sua complessità è O(a), dove a è la lunghezza dell'array