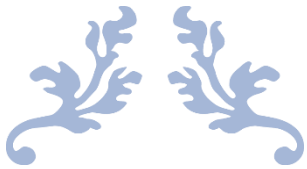


VAXCENTER

Manuale Tecnico

Università degli Studi dell'Insubria – Laurea Triennale in Informatica

Progetto Laboratorio A: VaxCenter

Sviluppato da:

- ***Alessandro Cassani,*** *matricola 744512*
- ***Damiano Ficara****, matricola 744958*
- ***Paolo Bruscagin,*** *matricola 744703*
- ***Simone Torno****, matricola 746592*

Versione 1.0

Sommario

Introduzione

VaxCenter è un progetto relativo al corso di laboratorio A della facoltà di scienze informatiche dell'Università degli Studi dell'Insubria, svolto durante l'anno accademico **2020-2021**.

Il progetto è sviluppato in Java 14, è stato sviluppato su Windows 10 e testato su **Windows 10, Linux Mint: 20.2, MacOS: 10.13**

Si tratta di un'applicazione **stand-alone**, la quale non necessita di alcun tipo di installazione specifica nel sistema operativo.

Si è deciso di non utilizzare librerie esterne, ma unicamente le classi standard di Java.

Struttura generale del programma

Il progetto è diviso in due **package**:

- **centrivaccinali**, che si occupa di fornire le funzionalità utilizzate dagli operatori vaccinali, oltre che alla memorizzazione su file dei dati relativi.
- **cittadini**, che si occupa di fornire le funzionalità utilizzate dagli utenti cittadini e la loro memorizzazione su file.

❖ centrivaccinali

- CentriVaccinali;
- Vaccinato;
- FileCentriVaccinali;
- FileDatiCV;
- Path;
- CreazioneCornice;
- Main.

❖ cittadini

- Cittadini;
- EventiAvversi;
- CifrarioDiCesare;
- FileDatiCittadini;
- PathCittadini.

Inoltre, come da specifiche, i file di salvataggio sono stati salvati in una cartella `/data` e nominati rispettivamente:

- *Vaccinati_NomeCentroVaccinale.dati*, dove *NomeCentroVaccinale* varia dinamicamente con il nome del centro vaccinale, contenente i vaccinati di uno specifico centro vaccinale;
- *Cittadini_Registrati.dati*, contenente i cittadini registrati a sistema;
- *CentriVaccinali.dati*, contenente i centri vaccinali registrati a sistema.

Il punto di avvio del programma risiede all'interno del package **centrivaccinali**, nello specifico nella classe **Main**.

Si presentano in seguito le classi contenenti le funzionalità richieste e sviluppate, nei due package dettagliatamente. Divideremo la discussione in **classi principali e classi accessorie**, le quali non contengono le funzioni principali richieste dalle specifiche, ma di fondamentale importanza per il corretto funzionamento dell'applicazione.

Non verrà affrontata una discussione sulla parte grafica del progetto, in quanto non essendo espressamente richiesta nelle specifiche di progetto.

Premesse

- Per poter testare tutte le funzionalità del software non è possibile registrare in sequenza il vaccinato, per poi registrare il cittadino e subito dopo inserire gli eventi avversi del cittadino, perché l'inserimento nella posizione corretta di quest'ultimi può avvenire unicamente quando viene aggiornato il campo linea dell'oggetto vaccinato ossia all'avvio dell'applicazione.
- Per poter gestire gli input è stata utilizzata la classe **Scanner** e i suoi relativi metodi
- Le strutture dati utilizzate appartengono alla libreria standard di Java, nello specifico la java.util.* e la java.io.*;
- All'avvio dell'applicazione è presente il metodo **caricamentoDati()** (successivamente approfondito nella rispettiva classe Main) che permette il caricamento delle informazioni memorizzate su file nelle specifiche strutture dati.
- Tutte le funzionalità dell'applicazione accedono almeno una volta alla struttura dati dei centri vaccinali per estrarre un oggetto "CentriVaccinali" memorizzandone o visualizzandone le informazioni.
- Come convenzione, si è deciso di salvare tutti le stringhe in maiuscolo eccetto password e userid, al momento della registrazione del cittadino.
- Per quanto riguarda la gestione del path, si è deciso di sviluppare due classe specifiche (Path e PathCittadini) che gestiscono, a seconda del sistema operativo utilizzato, il percorso.
- Il tipo di file scelto è .**txt**, affinché non si creino problemi di compatibilità tra i differenti sistemi operativi.
- I cittadini registrati e i vaccinati non sono memorizzati in due strutture dati globali, bensì sono registrati in due diverse strutture dati, entrambe appartenenti al singolo oggetto CentriVaccinali (rappresentante il centro vaccinale nella quale è avvenuta la vaccinazione).
- Per ovviare alla mancanza della parte grafica, è stata sviluppata una classe CreazioneCornice, la quale stampa una cornice di asterischi nel terminale.
- Per poter eseguire la nostra applicazione, il progetto è stato esportato nell'estensione **.jar,** in questa maniera basterà richiamare la jar dal prompt dei comandi
- Per la visione completa di tutti i metodi di ciascuna classe, si consiglia la visione della **javaDoc.**

Classi e funzionalità principali

1. La classe **CentriVaccinali**

La classe *CentriVaccinali* si occupa della registrazione a sistema di un centro vaccinale, nonché del caricamento e aggiornamento da file a struttura dati delle informazioni dei centri vaccinali memorizzati.

Oggetto **CentriVaccinali**:

Un oggetto della classe *CentriVaccinali* contiene le informazioni di base di un centro vaccinale come: nome, indirizzo, tipologia, comune e cap.

Inoltre, sono salvati i cittadini vaccinati e registrati in quello specifico centro vaccinale.

Le informazioni essenziali inserite dall'operatore sono salvate in campi di tipo **String** per evitare il sollevamento di eccezioni nel caso in cui input non sia compatibile

Per evitare che un utente inserisca stringhe vuote o informazioni non consone, si è deciso di applicare una serie di controlli logici per limitarne gli errori.

Hashtable dei **CentriVaccinali**:

```
Hashtable<String, CentriVaccinali> ht = new Hashtable<>(2841, 0.9f)
```

L'applicazione sviluppata si base principalmente sulla struttura dati **Hashtable dei centri vaccinali**. In essa sono contenuti tutti i centri vaccinali registrati con un'unica accortezza, non possono essere presenti più centri con lo stesso nome. Questo perché viene usato come chiave dell'Hashtable, il nome del centro vaccinale per prevenire possibili collisioni e ottenere un tempo di accesso al dato in memoria pari a $O(1)$.
La capacità iniziale di questa struttura dati è di **2841**, pari al numero di centri vaccinali presenti al $05/09/2021$ in Italia, per massimizzare le prestazioni della struttura dati

Viene posto un fattore di carico a $0{,}9$f poiché non essendoci collisioni, i costi vengono minimizzati.

Metodi principali utilizzati nella classe:

- **`getVaxCenter()`**

  Il metodo **`getVaxCenter()`** è un metodo fondamentale, in quanto permette , data una lista di centri vaccinali, di selezionarle uno. Si è scelto di stampare la lista di centri vaccinali, combinata con un variabile contatore, che indicizza tutti i centri vaccinali di quella determinata lista. In seguito, sarà possibile selezionare il centro vaccinale desiderato.

  Particolare importanza è stata data alla gestione delle eccezioni IndexOutOfBoundsException() e NullPointerException(), per ovviare ad inserimenti non corretti da parte dell'utente.

  `Complessità:` $O(1)$

- **registraCentroVaccinale()**

```
protected static void
registraCentroVaccinale(Hashtable<String,CentriVaccinali> ht)
{
CentriVaccinali centro = new CentriVaccinali(ht);
ht.put(centro.getNomeCentroVaccinale(),centro);
//blocco switch per inserire il cv appena creato nella corretta lista
per tipologia //(hub,aziendale ospedaliero)
    switch (centro.getTipologia()){
            case "HUB": hub.add(centro); break;
            case "OSPEDALIERO": ospedaliero.add(centro); break;
            case "AZIENDALE": aziendale.add(centro); break;
        }
   //stampo il centro vaccinale appena registrato nel file
CentriVaccinali.dati
    new FileCentriVaccinali(centro);
   System.out.println("Operazione eseguita con successo! \n");
}
```

Si è scelto di definire il metodo registraCentroVaccinale() statico in quanto permette di essere utilizzato da tutti gli oggetti della classe. Ciò significa che questo metodo non si riferisce ad una istanza della classe, ma bensì alla classe stessa. All'interno del metodo, viene istanziato un oggetto di tipo CentriVaccinali e lo si inserisce all'interno della struttura dati dei centri vaccinali "ht", con chiave stringa *nomeCentroVaccinale* e come valore oggetto CentriVaccinali.

Successivamente si inserisce l'oggetto nella lista specifica, in base alla tipologia inserita (ossia hub-aziendale-ospedaliero), mediante un semplice **switch.**

Una volta aver stabilita la tipologia , si può procedere alla stampa su file "*CentriVaccinali.dati*" delle informazioni salvate, tramite la creazione di un oggetto FileCentriVaccinali, che verrà approfondito nella classe relativa.

Complessità: La memorizzazione dell'oggetto salvato nell'hashtable e nella lista designata presenta un costo $O(1)$.

2. La classe **Vaccinato**

La classe *Vaccinato* contiene le funzioni necessarie per la memorizzazione e il caricamento da file in memoria di un oggetto di tipo *Vaccinato*.

Oggetto **Vaccinato**:

Un oggetto *Vaccinato* contiene le informazioni di base di un cittadino vaccinato, quali nome del centro vaccinale, nome, cognome, codice fiscale, la data di somministrazione del vaccino, il tipo di vaccino avuto, id numerico.

Si è deciso, di porre i campi sopraelencati della classe come **String**, per evitare il sollevamento di eccezioni, nel caso in cui l'input non rispecchiasse una tipologia compatibile. Sono stati applicati una serie di controlli logici, per verificare che gli inserimenti dell'operatore vaccinali siano coerenti con le caratteristiche del dato da inserire. Particolare importanza è stata data alla creazione dell' **id numerico** su 16 bit, in quanto si è stabilito un metodo apposito che, a seconda del numero totale di vaccinati di tutti i centri vaccinali, concatena una stringa di zeri al numero di vaccinati, generando un valore su 16 bit.

```
protected static String idnumerico(){
    String nVaccinati = String.valueOf(vaccinati);
    String str = null;
    //se il numero di vaccinati totale è un numero a 2 cifre, allora
    concatenerò 14 zeri all'inizio
    //se il numero di vaccinati totali è formato da 10 cifre, allora
    concatenerò 6 zeri all'inizio
    switch (nVaccinati.length()) {
           case 0: str = "0000000000000000";
           break;
           case 1: str = "000000000000000" + nVaccinati;
           break;
           case 2: str = "00000000000000" + nVaccinati;
           break;
           case 3: str = "0000000000000" + nVaccinati;
           break;
           case 4: str = "000000000000" + nVaccinati;
           break;
           case 5: str = "00000000000" + nVaccinati;
           break;
           case 6: str = "0000000000" + nVaccinati;
           break;
           case 7: str = "000000000" + nVaccinati;
           break;
           case 8: str = "00000000" + nVaccinati;
           break;
           case 9: str = "0000000" + nVaccinati;
           break;
           case 10: str = "000000" + nVaccinati;
           break;
           case 11: str = "00000" + nVaccinati;
           break;
           case 12: str = "0000" + nVaccinati;
```

```
            break;
            case 13: str = "000" + nVaccinati;
            break;
            case 14: str = "00" + nVaccinati;
            break;
            case 15: str = "0" + nVaccinati;
            break;
            case 16: str = nVaccinati;
            break;
  }
  return str;
}
```

Metodi principali utilizzati nella classe:

- **registraVaccinato()**

```
protected static void
registraVaccinato(Hashtable<String,CentriVaccinali> ht)
{
  Vaccinato vac = new Vaccinato();
  try {
   //il nome del centro vaccinale contenuto nell'oggetto vaccinato viene
   //usato come key per estrarre dalla struttura dati ht il centro
   //vaccinale dove è avvenuta la vaccinazione
   CentriVaccinali cv = ht.get(vac.getNomecentrovaccinale());
   //l'oggetto Vaccinato viene inserito in struttura dati con key il
   codice fiscale
        cv.vaccinati.put(vac.getCodfisc(), vac);
        new FileDatiCV(vac);
        System.out.println("Operazione eseguita con successo! \n");
        System.out.println();
    } catch(NullPointerException e) {
        e.printStackTrace();
        System.out.println("il nome del centro vaccinale inserito è
        inesistente\n");
    }
 }
}
```

Per ideare la `registraVaccinato()`, si è deciso di procedere mediante la creazione di un oggetto *Vaccinato*. Utilizzando il nome del centro vaccinale appena inserito nell'oggetto vaccinato, viene estratto dalla struttura dati dei centri vaccinali il centro avente come chiave il nome del centro vaccinale.
Si prosegue memorizzando nell' Hashtable dei vaccinati, contenuta nell'oggetto appena estratto, l'oggetto vaccinato con key il codice fiscale, essendo univoco.

Una volta aver aggiunto il vaccinato nella struttura dati , si può procedere alla stampa su file "*Vaccinati_NomeCentroVaccinale.dati*" delle informazioni salvate, tramite la creazione di un oggetto FileDatiCV, che verrà approfondito nella classe relativa. Viene lanciata La **NullPointerException** , nel caso in cui il nome del centro vaccinale inserito dall'operatore vaccinale sia inesistente.

Complessità: L'estrazione dell'oggetto *CentriVaccinali* avviene in tempo $O(1)$, poichè si conosce la chiave. Il caricamento nell'oggetto centro vaccinale avviene in tempo $O(1)$.

3. La classe **Path**

La classe Path, come si può intendere facilmente dal nome, si occupa della gestione dei percorsi di salvataggio dei file utilizzati. Affinché l'applicazione possa essere eseguita su differenti sistemi operativi, all'interno della classe Path sono stati creati una serie di metodi, con modificatore di visibilità **protected**, che forniscono il path corretto a seconda del sistema operativo utilizzato, poichè sono presenti una serie di differenze nel separatore, infatti:

- Windows presenta : //
- Unix/Mac/Linux presentano : \

Particolare rilevanza viene data all'utilizzo della classe System, la quale mediante una serie di istruzioni, fornisce le informazioni di base necessarie:

- **`System.getProperty("user.dir");`** permette di fornire, come stringa, il percorso della cartella dove si sta eseguendo l'applicazione
- **`System.getProperty("os.name");`** permette di fornire, come stringa, il nome del sistema operativo in uso.

In seguito, mediante un semplice "os-detector", ci si occupa di stabilire il path a seconda della stringa del sistema operativo. Si utilizza istruzione *contains*, per verificare che la stringa del sistema operativo corrisponda ad uno specifico **S.O**.

```
if (os.contains("win")) {
    //Sistema Operativo Windows
    path = cartella + "\\" + "data" + "\\" + "Dati Centri Vaccinali" +
   "\\" + nomeFile;
    } else if (os.contains("nix") || os.contains("nux") ||
os.contains("mac")) {
   //Sistema Operativo Mac / Unix
    path = cartella + "/" + "data" + "/" + "Dati Centri Vaccinali" +
"/" + nomeFile ;
    } else {
    //Sistema Operativo Sconosciuto
    path = cartella + "/" + "data" + "/" + "Dati Centri Vaccinali" +"/"
+ nomeFile;
    }
```

Per poter gestire nel migliore dei modi, il cambiamento automatico di *NomeCentroVaccinale*, sono stati creati una serie di metodi aventi come parametro o un oggetto *CentriVaccinali* o un oggetto *Vaccinato* oppure direttamente una stringa, in maniera tale da sfruttare nei primi due il rispettivo campo contenente il nome del centro vaccinale, mentre nell'ultimo la stringa stessa.

Per ottenere il path finale, a seguito dell' "os-detector", basterà concatenare tali stringhe. Si noti, mediante il consulto della **javaDoc**, come i metodi di questa classe sono tutti i statici, in questa maniera appartengono direttamente alla classe, invece che all'oggetto stesso.

4. La classe **PathCittadini**

La classe PathCittadini è una classe analoga alla classe Path, ma per ragioni di sicurezza all'interno del package , affinché i metodi siano **protected** si è deciso di creare un ulteriore classe, il cui fine è lo stesso della classe Path, ma rivolto verso il file *Cittadini_Registrati.dati*.

I metodi, all'interno di questa classe, sono analoghi alla classe Path e seguono lo stesso ragionamento e scelte applicate in precedenza.

Per maggiori dettagli, si consulti la **javaDoc**.

5. La classe **Cittadini**

La classe Cittadini è una classe fondamentale dell'applicazione creata, in quanto contiene la maggior parte delle funzioni richieste dalle specifiche si progetto, nello specifico:

- **`registraCittadino():`** ossia consente di registrare in uno specifico centro vaccinale un cittadino;
- **`inserisciEventiAvversi():`** inserire eventuali eventi avversi post-vaccinazione in uno specifico centro vaccinale;
- **`visualizzaEventiAvversi():`** visualizzare la severità media e il numero di segnalazioni totali di uno specifico centro vaccinale.

Oltre ad una serie di metodi ausiliari per il corretto funzionamento dell'applicazione:

- **`logIn()`** : concede l'accesso al cittadino registrato per l'inserimento degli eventi avversi, nel caso in cui le credenziali inserite corrispondano a quelle registrate a sistema.
- **`cercaCentroVaccinale()`** : permette al sistema di ricercare il centro vaccinale.

Oggetto **Cittadino**:

Un oggetto Cittadini contiene le informazioni di base di un cittadino quali: nome, cognome, codice fiscale, e-mail, userid, password, e il codice identificativo.

Si è deciso, di porre i campi sopraelencati della classe come **String**, per evitare il sollevamento di eccezioni, nel caso in cui l'input non fosse di una tipologia compatibile. Anche in questo caso nella creazione dell'oggetto, sono stati applicati una serie di controlli logici, in modo da verificarne la coerenza dei dati inseriti.

Metodi ausiliari utilizzati nella classe:

Si è deciso di impostare la discussione, partendo dalle scelte applicate nei metodi fondamentali, per poi spiegare le funzioni richieste dalle specifiche.

- **`logIn()`**

  La funzione `logIn()` è una funzione che, come si può intendere dal nome, permette di verificare se le credenziali di accesso inserite dall'utente corrispondono a quelle registrate a sistema.

  Si è scelto di passare come parametri:

  - Una stringa `NomeCentroVaccinale`: che identifica il nome del centro vaccinale in cui si è svolta la vaccinazione del cittadino
  - `Hashtable ht` contenente i centri vaccinali

  Idea di questo metodo è di estrarre l'oggetto *CentriVaccinali*, rappresentante il centro vaccinale dove è avvenuta la vaccinazione del cittadino, tramite la chiave **String** passata come parametro e , utilizzando la struttura dati **cittadiniRegistrati** (appartenente all'oggetto *CentriVaccinali* appena estratto), con `key` lo `userid` e come valore l'oggetto *Cittadini*, si verifica con l'istruzione `containsKey` la presenza o meno dello `userid`, inserito dal cittadino vaccinato.

  Dopo aver verificato la presenza del nome utente, è stato necessario gestire la password.

  Si è scelto di proseguire con l'idea utilizzata in precedenza, in questo caso però dopo aver confermato la correttezza dello `user id` si è estratto dalla struttura dati **cittadiniRegistrati** (`key = userid`) l'oggetto *Cittadini* contenente le informazioni del cittadino che desidera inserire gli eventi avversi comparsi post-vaccinazione.

  La strategia consiste nel controllare se il campo `password` dell'oggetto *Cittadini* corrisponde alla password inserita dall'utente. Se l'esito è positivo , viene "restituito" `il codice fiscale`, che sarà fondamentale nell'**inserimento degli eventi avversi**, poichè univoco e in grado di stabilire la posizione della linea nel file *Vaccinati_NomeCentroVaccinale.dati* delle informazioni del cittadino vaccinato e quindi la posizione nella quale verranno inseriti gli eventi avversi registrati.

  Inizialmente, si era pensato di mettere il metodo **booleano**, poichè sostanzialmente fornisce un risultato positivo o negativo all'accesso al sistema. Per poter ottenere il codice fiscale , come quanto detto in precedenza, è stato stabilito che venga restituita una stringa **false** se l'accesso non avviene correttamente, altrimenti viene "restituito" direttamente il codice fiscale.

  Inoltre , è stato stabilito a **cinque**, il numero di tentativi possibili per poter inserire le credenziali(5 per userid - 5 per password), scaduti questi tentativi non sarà possibile accedervi ed il metodo, come detto in precedenza, restituirà **false**.

Complessità: $O(1)$, poichè sono presenti un numero massimo di tentativi pari a dieci, inoltre si accede al confronto delle informazioni ricercate tramite Hashtable , che presenta una complessità di ricerca pari a $O(1)$.

- **`cercaCentroVaccinale()`**

La funzione `cercaCentroVaccinale()` permette all'utente di compiere una ricerca tra i centri vaccinali disponibili a seconda di determinati parametri forniti in ingresso.

Poichè sono presenti due tipologie di ricerca, si è scelto di procedere con **l'overloading** del metodo ottenendo:

- una ricerca **per nome**, la quale presenta come parametro *una stringa* contenuta nel nome del centro vaccinale ricercato e la struttura dati Hashtable *ht*, contenente tutti i possibili centri vaccinali
- una ricerca **per comune e tipologia**, la quale presenta come parametro due stringhe(comune e tipologia) e verifica la presenza dei possibili centri vaccinali con queste caratteristiche.

Nella **ricerca per nome** si è utilizzata la classe `Enumeration`, nella quale sono state caricate le chiave di `ht`(ossia i nomi dei centri vaccinali). Nel nostro specifico caso, sono stati utilizzati i metodi `hasMoreElements()` e `nextElement()` per iterare le chiavi contenute nell'Enumeration.

Nel caso in cui la key contenga la stringa passata come parametro, si estrae l'oggetto indicizzato dalla key e lo si inserisce all'interno di una lista di centri vaccinali. Si è scelto di utilizzare una Lista poichè non sappiamo con certezza quante chiavi conterranno quella particolare stringa.

Complessità: $O(n)$, dove n è il numero di centri vaccinali inseriti nella struttura dati

Per quanto concerne **la ricerca per comune e tipologia** è stata applicata una strategia differente, in quanto sono presenti tre *ArrayList statiche* appartenenti alla classe *CentriVaccinali* nelle quali sono inseriti *CentriVaccinali* a seconda della loro tipologia (hub-ospedaliero-aziendale). Si è deciso di scandire la lista della tipologia inserita mediante un ciclo **for-each** e verificare se il comune dell'oggetto preso in considerazione coincida con la stringa comune inserita in input. In caso di esito positivo si aggiunge tale centro ad una lista di *CentriVaccinali*.

Complessità: $O(n)$, dove n è il numero di centri vaccinali della tipologia ricercata.

Metodi principali della classe:

- **`registraCittadino()`**
  La funzione `registraCittadino()` permette al cittadino di selezionare il centro vaccinale nella quale è avvenuta la vaccinazione e successivamente registrarsi all'interno del centro selezionato.
  Per poter implementare questa funzione, si è scelto di utilizzare la `cercaCentroVaccinale()`, in questa maniera sarà possibile scegliere dove registrare il cittadino scelto.

Dopo aver ottenuto la lista di *CentriVaccinali*, mediante il metodo `getVaxCenter()` è possibile selezionare il centro vaccinale desiderato, gestendo le eccezioni:
- NullPointerException;
- **IndexOutOfBoundException**.

delegate dal metodo `getVaxCenter()`, in caso di errori nella ricerca/selezione.
In seguito alla selezione del centro vaccinale, viene creato un oggetto *Cittadini*, in modo da contenere tutte le informazioni principali del cittadino. Si è scelto di passare come parametri, `il nome del centro vaccinale` e la `struttura dati cittadiniRegistrati`.
Inoltre, mediante la creazione dell'oggetto, viene controllato in automatico che lo `user id` non sia già stato inserito nella struttura dati **cittadiniRegistrati**.
Infine, viene gestita la stampa delle informazioni del cittadino, avvalendosi del file *Cittadini_Registrati.dati* e dell'oggetto della classe FileDatiCittadini.
Si tiene a precisare che la ricerca del centro vaccinale è utilizzabile in entrambe nelle sue versioni, sarà l'utente a decidere come utilizzarla.

Complessità:

- $O(n)$, dove n è il numero di centri vaccinali della tipologia ricercata, nel caso in cui si utilizzi il metodo di ricerca tramite comune e tipologia.
- $O(m)$, dove m è il numero di centri vaccinali inseriti in struttura dati, nel caso in cui si utilizzi il metodo di ricerca tramite stringa.

- **`inserisciEventiAvversi()`**
La funzione `inserisciEventiAvversi()`, permette al cittadino registrato di inserire eventuali eventi avversi comparsi post-vaccinazione.

Per la realizzazione di questa funzione si è scelto di procedere prima di tutto ricercando e selezionando il proprio centro vaccinale dove è stata eseguita la vaccinazione e dove si è registrati, mediante la funzione `cercaCentroVaccinale()`(entrambe le versioni) e il metodo `getVaxCenter()`.

Dopo la ricerca/selezione è necessario eseguire il `logIn`, (metodo precedentemente trattato).
Se il `logIn` ha esito positivo, viene salvato nel file *Vaccinati_NomeCentroVaccinale.dati* elenco personalizzato di eventi avversi creato dall'utente, vengono aggiornati i campi appartenenti all'oggetto;
$\text{CentriVaccinali} - \text{numeroSegnalazioni}$ e sommaSeverita tramite metodi appartenenti alla classe *EventiAvversi*.

Come detto in precedenza, si è deciso di restituire il codice fiscale, a seguito del `logIn`, poichè univoco e in grado di stabilire la posizione della linea nel file .
Si è scelto di estrarre dalla struttura dati **vaccinati**, l'elemento con chiave pari al codice fiscale, della quale viene richiamato il campo contenente il nome del centro per comporre in automatico il *NomeCentroVaccinale* del file *Vaccinati_NomeCentroVaccinale.dati*.

In seguito per poter stampare sia logicamente sia visivamente le linee all'interno del file, si legge mediante la Scanner il file e, mediante un ciclo for, vengono memorizzate in una lista di tipo String tutte le linee del file fino alla riga dalla quale dovremo iniziare a stampare gli eventi avversi appena segnalati dal Cittadino.

Successivamente si è stabilito di aggiungere nella lista gli eventi avversi e mediante un ulteriore ciclo for, lo scanner è avanzato superando le righe bianche nella quale inseriremo gli eventi avversi segnalati, per poi terminare questa fase inserendo le restanti linee del file.
Come ultimo passaggio le informazioni nella lista vengono stampate su file , aggiornando il contenuto del file. Si è deciso di utilizzare la classe *FileWriter* ed i suoi metodi.

Complessità: $O(n)$, dove , dove n è il numero di vaccinati presenti nel file *Vaccinati_NomeCentroVaccinale.dati*

- **`visualizzaInfoCentriVaccinali()`**
  La funzione `visualizzaEventiAvversi()` permette di ottenere un prospetto riassuntivo degli eventi riportando il numero di segnalazioni e la severità media.

  Per l'implementazione di questa funzione è stata applicata la stessa strategia applicata nella funzione descritta in precedenza, procedendo prima di tutto ricercando e selezionando il proprio centro vaccinale dove è stata eseguita la vaccinazione e dove si è registrati, mediante la funzione `cercaCentroVaccinale()` (entrambe le versioni) e il metodo `getVaxCenter()`.

  In seguito, vengono richiamati i metodi statici presenti all'interno della classe *CentriVaccinali* , i quali permetto di:
  - calcolare **la severità media** degli eventi avversi registrati, tramite una semplice divisione tra i campi sommaSeverita e $\text{numeroSegnalazioni}$.
  - visualizzare il **numero di segnalazioni**, semplicemente prelevando la prima linea del file *Vaccinati_NomeCentroVaccinale.dati*.

  Per rendere visivamente migliore la visualizzazione dei risultati viene utilizzata l'istruzione `String.format("%0.1f",mediaSeverita)` , la quale permette di stampare in uscita il valore con una sola cifra decimale.

  Per maggiori dettagli, si consiglia di visionare i metodi della **javaDoc**, fondamentali per il corretto funzionamento.
  - ✓ caricaProspettoRiassuntivo();
  - ✓ prospettoRiassuntivo();
  - ✓ mediaseverita()

6. La classe **EventiAvversi**

La classe EventiAvversi permette al cittadino registrato in un centro vaccinale di creare un elenco personalizzato di sintomi comparsi post-vaccinazione.

Un oggetto di questa classe è formato da tre campi, il campo “nome” contenente la denominazione del sintomo analizzato, il campo “severita” contenente la gravità, espressa in scala da 1 a 5, dell’evento avverso in questione e un’ulteriore campo “note”, contenente dei commenti opzionali inseriti dall’utente (max 256 caratteri).

Si è deciso di inizializzare i campi “note” e “severita” ad una stringa vuota in modo tale da riconoscere facilmente quando effettivamente quest’ultimi siano stati modificati e quindi registrati dall’utente.

Metodi utilizzati nella classe:

- **`CreazioneElenco()`**

Questo è il metodo principale della classe, permette di creare l’elenco personalizzato di sintomi registrati dall’utente inserendo le segnalazioni in una *ArrayList* di oggetti di tipo *EventiAvversi.*

I sintomi registrabili sono inseriti in automatico utilizzando il costruttore della classe, il quale associa una stringa ricevuta in input al campo “nome” dell’oggetto *EventiAvversi* e quindi al nome dell’evento registrabile.

I sintomi inseriti sono i seguenti: mal di testa, febbre, dolori articolari e muscolari, linfoadenopatia, tachicardia e crisi ipertensiva.

Attraverso un ciclo for-each viene iterata la lista di oggetti appena creata (ogni oggetto ha solo il campo nome diverso dal valore di inizializzazione), e per ogni oggetto scandito viene chiesto all’utente se è comparso il sintomo rappresentato. Se il cittadino conferma l’avvenuto, viene richiesto obbligatoriamente di inserire la severità con la quale si è presentato il sintomo, successivamente viene richiesto di inserire delle note opzionali di commento (facoltativo).

Il metodo ritorna quindi la lista di oggetti *EventiAvversi* personalizzata dal cittadino.

`Complessità:` $O(1)$, numero di operazioni costanti

Attraverso i metodi `contaSegnalazioni() e sommaSeverita()` e la lista di tipo *EventiAvversi* fornita come parametro è possibile contare nel primo caso il numero di volte in cui un sintomo si è presentato al cittadino, e nel secondo caso la somma delle severità registrate. Questi valori saranno fondamentali per poter stampare in output con la funzione visualizzaInfoCentriVaccinali() il prospetto riassuntivo di uno specifico centro vaccinale.

7. La classe **Main**

La classe *Main* corrisponde alla home dell'applicazione, in essa sono contenute tutte le funzionalità utilizzabili dagli operatori vaccinali e dai cittadini. All'avvio del programma l'utente inserisce se è un operatore vaccinale o un cittadino, distinguendo così il tipo di funzionalità utilizzabili.
Tutte le scelte possibili sono gestite tramite switch-case, dove l'utilizzatore inserisce una stringa numerica rappresentante la funzionalità desiderata.

Metodi utilizzati nella classe:

- **`CaricamentoDati()`**

All'avvio del programma vengono caricate le informazioni salvate precedentemente sui file utilizzati dall'applicazione nelle strutture dati designate.
In esso è presente la funzione `CaricaDaticv()`, il cui compito è quello di caricare i dati dei centri vaccinali nella Hashtable contenente i centri vaccinali. Prima di questo viene effettuato il controllo che il file in questione esista (caso in cui non sia ancora stato registrato alcun centro vaccinale), altrimenti il metodo lancerebbe l'eccezione **FileNotFoundException.**
Successivamente, sempre controllando che ci sia almeno un centro vaccinale registrato a sistema, si procede al caricamento del prospetto riassuntivo di ogni centro vaccinale mediante il metodo `prospettoRiassuntivo().`
Quest'ultimo permette di aggiornare i campi numeroSegnalazioni e sommaSeverita di ogni centro vaccinale, tramite la funzione `caricaProspettoRiassuntivo()` la quale estrae le informazioni dalla prima linea dei file *"Vaccinati_NomeCentroVaccinale.dati"*, dove sono contenute il numero di segnalazioni riscontrate e la somma delle severità segnalate.
Successivamente vengono caricate nelle Hashtable dei vaccinati, contenute dentro gli specifici oggetti CentriVaccinali, le informazione dei vaccinati nei rispettivi centri vaccinali.
I dati sono presi dai diversi file *"Vaccinati_NomeCentroVaccinale.dati"* , dove *NomeCentroVaccinale* viene sostituito dinamicamente dal nome del centro vaccinale dove è avvenuta la vaccinazione.
Infine, seguendo la stessa logica del metodo `Caricadaticv(),` vengono caricati i dati dei cittadini registrati nelle Hashtable **cittadiniRegistrati** contenute dentro i singoli oggetti CentriVaccinali.

`Complessità:`

- `Caricadaticv()` = $O(n)$, dove n è il numero di centri vaccinali registrati.
- `prospettoRiassuntivo()` = $O(n)$, dove n è il numero di centri vaccinali registrati a sistema.
- `caricaDativaccinati()` = $O(n)$, dove n è il numero di centri vaccinali registrati con almeno un vaccinato registrato.
- `caricadaticittadini()` = $O(n)$, dove n è il numero di cittadini registrati presente nel file "Cittadini_Registrati.dati"

8. Le classi **Accessorie**

Le **Classi Accessorie** sono classi fondamentali per la corretta esecuzione del programma, ma che non presentano le funzioni specificamente richieste dalle specifiche di progetto. Per questo motivo si è deciso, in fase di spiegazione di differenziare i due concetti.

- **FileCentriVaccinali:**

  Per rendere più ordinato il progetto, si è deciso di creare una classe apposita per "stampare" sul file "*CentriVaccinali.dati*" i centri vaccinali registrati.

  All'interno di questa classe si è deciso di procedere creando un oggetto di tipo *FileCentriVaccinali*, il quale si occupa di verificare la presenza del file ed eventualmente crearlo e richiamare il secondo metodo di questa classe ossia `FaseScritturaCV`(), dove mediante utilizzo delle classi *FileWriter, FileReader e PrintWriter* , aggiungiamo **in append**, ossia alla fine, il contenuto dell'oggetto CentriVaccinali. In questa maniera si evita di sovrascrive ogni volta il file ed evitare di eliminare dati sensibili.

  Si è deciso di applicare il costrutto **try-catch**, così da catturare e gestire le possibili eccezioni facenti parte di **IOException**.

- **FileDatiCV:**

  La classe *FileDatiCV*, segue l'idea della classe *FileCentriVaccinali*, ma a differenza di agire sul file *CentriVaccinali.dati* agisce sul file *Vaccinati_NomeCentroVaccinale.dati.* Nello specifico, si occupa di stampare i dati dei vaccinati.
  Grazie alla classe *Path* definita in precedenza, e sfruttando i suoi metodi, è possibile determinare il nome esatto del centro vaccinale gestendo quindi in maniera automatica il cambiamento del nome.
  Anche in questo caso, il costruttore si occupa della verifica della presenza del File, e oltre a stampare i dati dei vaccinati, mediante il metodo *FaseScritturaDatiCV*, alla creazione del file stampa il numero di segnalazioni e la somma severità, che verranno aggiornati dopo l'inserimento di eventi avversi da parte del cittadino.

  Per poter permettere al vaccinato di aggiungere eventuali eventi avversi, si predispone la stampa di sei righe bianche, affinché il cittadino vaccinato potrà aggiungere gli effetti post-vaccinazione in un secondo momento.

- **CreazioneCornice:**

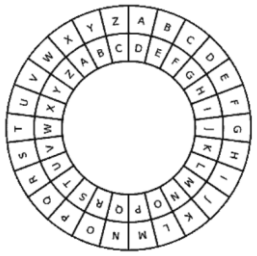
  La classe *CreazioneCornice* si occupa, come si intende facilmente dal nome della creazione della cornice. Questa classe è stata sviluppata, in quanto in assenza di una parte grafica, per rendere il terminale visivamente più piacevole all'avvio del software. Per una visione specifica dei metodi, si consiglia la visione della **javaDoc.**

- **FileDatiCittadini:**

  La classe *FileDatiCittadini* segue idea della classe FileDatiCV e della *FileCentriVaccinali*, ma a differenza di queste due, agisce sul File

Cittadini_Registrati.dati; nello specifico si occupa di stampare i dati dei cittadini registrati in un centro vaccinale.
Anche in questo caso, si utilizza il costruttore per verificare la presenza del file e mediante i metodi definiti nella classe *PathCittadini*, viene scelto in automatico il percorso.
É presente, inoltre, il metodo `FaseScritturaCittadini(),` il cui funzionamento è analogo ai metodi delle classi citate in precedenza. Il metodo mediante ausilio, delle classi *FilreWriter, FileReader* e *PrintWriter*, stampa le informazioni di un cittadino all'interno del file Cittadini_Registrati.dati, aggiungendo come in precedenza , in **append**, il contenuto di ogni cittadino. Per rendere visivamente più piacevole il file, tra ogni cittadino è presente una linea di separazione tratteggiata.

- **CifrarioDiCesare:**
  La classe *CifrarioDiCesare*, come si può intuire dal famoso algoritmo di crittografia, si occupa di cifrare le password degli utenti registrati. Il cifrario di Cesare è uno dei più antichi algoritmi pervenuti che permettono di criptare un messaggio.



Si è deciso di applicare questo sistema di cifratura, in quanto analizzato e visto durante il corso di *programmazione* ed inoltre per garantire un livello di sicurezza in più, nonostante il cifrario di cesare, per quanto possa essere molto intuitivo può essere facilmente *perforato* e non in grado di cifrare i numeri.

Si è scelto di creare un oggetto *CifrarioDiCesare*, il quale presenta come campi la chiave che viene stabilita come **standard**.
Viene presentato *algoritmo di cifratura*, si noti come si è scelto di sfruttare come il char sia un tipo derivante dall'intero, in particolare grazie allo standard **ASCII**.

```
public String enc(String cleartext){
    String text=cleartext.toLowerCase(); //porto la stringa minuscola
    char[] charText=text.toCharArray(); //converto la stringa in un
    array di char
    for(int i=0;i<charText.length;i++){
   if((charText[i]>='a' && charText[i]<='z')) //il valore è compreso tra
   a=97 e z = 122
     charText[i]=(int) charText[i]+chiave%26>'z'?(char)('a'+charText[i]-
                 ('z'+1)+chiave%26):(char)(charText[i]+chiave%26);
    }
    return new String(charText);
  }
}
```

Strutture dati utilizzate

- **`Hashtable<String,Vaccinato> vaccinati`**: all'interno di questa struttura dati le informazioni sono organizzate nel seguente modo: come chiave viene utilizzato il codice fiscale(*String*) del cittadino vaccinato e come valore l'oggetto *Vaccinato*, che contiene tutte le informazioni del cittadino vaccinato; questo concetto verrà ripreso e approfondito nella classe Vaccinato
    - Come chiave è stato scelto il codice fiscale, poichè essendo univoco, è impossibile il verificarsi di collisioni nel caso in cui l'utente inserisca il codice fiscale corretto.
    - L'uso di un' Hashtable, permette di ottenere prestazioni elevate in termini di complessità temporale, infatti il tempo di accesso ad un dato, mediante la chiave, avviene in $O(1)$

- **`Hashtable<String,Cittadini> cittadiniRegistrati`**: all'interno di questa struttura dati le informazioni sono organizzate nel seguente modo: come chiave viene utilizzato lo user id(String) inserito al momento della registrazione del cittadino nello specifico centro vaccinale e com valore oggetto Cittadini, che contiene tutte le informazione del cittadino. Questi concetti verranno ripresi nella classe Cittadini.
    - Come chiave è stato scelto lo user id, poichè dopo specifiche implementazioni nei controlli, è impossibile il verificarsi di collisioni nel caso in cui l'utente inserisca uno user id differente. Queste implementazioni verranno commentante nella classe Cittadini
    - l'uso di un'Hashtable , permette di ottenere prestazioni elevate in termini di complessità temporale, infatti il tempo di accesso ad un dato, mediante la chiave, avviene in $O(1)$

- **`ArrayList <CentriVaccinali> aziendale`**: all'interno di questa struttura dati sono immagazzinati i centri vaccinali di tipologia aziendale.
    - Questa implementazione su lista consente un tempo di accesso ad un dato pari a $O(n)$, dove n è il numero di centri vaccinali inseriti.

- **`ArrayList<CentriVaccinali> ospedaliero`**: all'interno di questa struttura dati sono immagazzinati i centri vaccinali di tipologia ospedaliero.
    - Questa implementazione su lista consente un tempo di accesso ad un dato pari a $O(n)$, dove n è il numero di centri vaccinali inseriti.

- **`ArrayList<CentriVaccinali> hub:`** all'interno di questa struttura dati sono immagazzinati i centri vaccinali di tipologia hub.
    - Questa implementazione su lista consente un tempo di accesso ad un dato pari a $O(n)$, dove n è il numero di centri vaccinali inseriti.