**WatchNeighbors - JAVA**



**Membri:**

* **Marco di Capua (matricola: 727673)**
* **Filippo Pelosi (matricola: 726602)**
* **Riccardo Zorzi (matricola: 726748)**

INDICE

1.DESCRIZIONE DEL PROGETTO

2.SCELTE PROGETTUALI

3.COMPONENTI UTILIZZATI

4.ANALISI DELLE FUNZIONALITÀ

5. GUIDA

6. SUDDIVISIONE DEL LAVORO

**1.DESCRIZIONE DEL PROGETTO**

Il progetto in Java prevede la creazione di un programma in grado di gestire una rete sociale di condivisione e segnalazione in broadcast di allarmi e pericoli per i quartieri di tre città.

L’applicazione è stata creata con le seguenti caratteristiche:

* Registrazione di un profilo utente con scelta della relativa città considerando la possibilità in seguito di modificare o cancellare il profilo in questione;
* Possibilità di login da parte degli utenti già registrati;
* Un utente registrato è in grado di inserire una nuova segnalazione caratterizzata dalla città in questione e localizzata in base a latitudine e longitudine, in seguito a questo processo viene aggiornato il relativo file di testo per le segnalazioni. Inoltre è stata implementata l’opzione di settare la latitudine e la longitudine attraverso l’utilizzo della mappa;
* Le informazioni relative agli utenti e alle segnalazioni sono state riportate in due file .csv;
* Possibilità di visualizzare la mappa dei quartieri in base alla zona selezionata durante l’utilizzo dell’applicazione. Inoltre è stata aggiunta la possibilità di interagire con la mappa tramite l’apposito zoom, oppure l’identificazione di una segnalazione cliccando sulla zona interessata della mappa (user-friendly);
* Possibilità per gli utenti registrati di prendersi carico e chiudere una segnalazione, se soddisfano i requisiti necessari, e di visualizzare la cronologia delle segnalazioni chiuse;
* Possibilità anche per gli utenti non registrati di effettuare una ricerca in base alla città, al quartiere e al motivo delle segnalazioni attive che si desidera controllare;

**2.SCELTE PROGETTUALI**

Inizialmente il problema è stato attentamente analizzato, successivamente è stata decisa la suddivisione dei ruoli ed è stata scelta la struttura dati più adatta per memorizzare temporaneamente i dati degli utenti e delle segnalazioni. In seguito sono state create le classi della logica del programma e del salvataggio permanente dei dati su file .csv e sono state testate utilizzando un’interfaccia a riga di comando. Infine si è deciso di procedere con la progettazione e implementazione della GUI per rendere più facile all’utente l’interazione con il programma.

La javadoc è stata realizzata in parallelo al processo di sviluppo delle classi.

Dopo aver completato la stesura del codice del programma è stato fatto un test approfondito dell’applicazione finale che ha portato alla correzione di alcuni malfunzionamenti e a una revisione del codice per renderlo più leggero e di facile comprensione.

Nella fase di realizzazione del progetto è stato deciso di suddividere in tre moduli l’applicazione assegnando a ciascuna classe le relative funzioni.

I moduli del programma contengono le seguenti classi:

* Logic:

1. FileManager: classe che contiene i metodi per gestire la scrittura e lettura del file .csv;
2. Main: classe che contiene il metodo main per avviare il programma WatchNeighbors;
3. Reporting: classe che permette di istanziare nuovi oggetti di tipo Reporting e che contiene i metodi per gestire le relative funzionalità;
4. User: classe che permette di istanziare nuovi oggetti di tipo User e contiene i metodi per gestire le relative funzionalità;
5. Utility: classe che contiene metodi utili per il corretto funzionamento del programma WatchNeighbors;

- Enums:

1. City: enumerativo che contiene le tre città (Como, Lecco, Varese) e i metodi associati;
2. ReportingOutcome: enumerativo che contiene i possibili esiti delle segnalazioni (non ancora risolto, falso allarme, intervento delle forze dell’ordine, la persona sospetta si è allontanata, incendio estinto) e i metodi associati;
3. ReportingReason: enumerativo che contiene i possibili motivi delle segnalazioni (furto, incendio, omicidio, scippo, allarme, persona sospetta) e i metodi associati;
4. District: enumerativo che contiene i quartieri delle città (nord-est, nord-ovest, sud-est, sud-ovest) e i metodi associati;
5. ReportingStatus: enumerativo che contiene i possibili stati delle segnalazioni (da verificare, in corso di accertamento, verificato) e i metodi associati;

- GUI

1. GUIHistory: classe che permette la visualizzazione dell’interfaccia con la cronologia delle segnalazioni chiuse e i relativi metodi;
2. GUIGuide: classe che permette la visualizzazione dell’interfaccia con la guida del programma e contiene i relativi metodi;
3. GUIModifyProfile: classe che permette la visualizzazione dell’interfaccia per la modifica del profilo e contiene i relativi metodi;
4. GUIModifyPassword: classe che permette la visualizzazione dell’interfaccia per la modifica della password e contiene i relativi metodi;
5. GUIMain: classe che permette la visualizzazione dell’interfaccia principale del programma e contiene i relativi metodi;
6. GUIRegistration: classe che permette la visualizzazione dell’interfaccia per la registrazione di un nuovo utente e contiene i relativi metodi;
7. GUIUser: classe che permette la visualizzazione dell’interfaccia utente e contiene i relativi metodi;

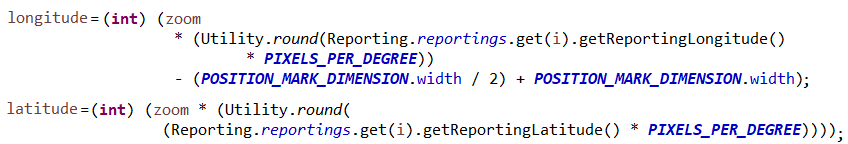
La struttura dati che è stata scelta per memorizzare temporaneamente i dati degli utenti e delle segnalazioni è l’ArrayList, una struttura dati dinamica che implementa l’interfaccia List in un array ridimensionabile. Ogni volta che viene aggiunto un elemento, se l’array è pieno, esso viene automaticamente riallocato con una dimensione maggiore del 50%, i dati vengono copiati nel nuovo array e il nuovo elemento viene aggiunto nella prima posizione disponibile. Questa operazione avviene in modo trasparente per l’utente. Dato che implementa l’interfaccia List è possibile utilizzare i metodi forniti da questa interfaccia per manipolare i dati nell’array. Il vantaggio principale di questa struttura dati è che ogni operazione di accesso posizionale richiede tempo costante O(1), mentre l’operazione di rimozione/aggiunta di un elemento ha complessità O(1) nel caso migliore (rimozione/aggiunta nell’ultima posizione) e O(n) nel caso peggiore (rimozione/aggiunta nella prima posizione).

Durante la fase di progettazione per garantire una migliore sicurezza è stato deciso che le informazioni inserite dagli utenti nei loro profili debbano soddisfare i seguenti requisiti per essere accettate dal programma:

* Userid: deve essere composto da almeno un carattere diverso dallo spazio vuoto e deve essere diverso dagli userid di tutti gli altri utenti già registrati su WatchNeighbors per evitare ambiguità relative all’utente che ha aperto, preso in carico o chiuso una segnalazione;
* Nome e cognome: devono essere composti da almeno un carattere;
* Mail: deve essere composta da almeno otto caratteri, contenere almeno una volta i caratteri ‘.’ e ‘@’ e non contenere spazi vuoti;
* Password: deve essere composta da almeno sette caratteri;
* Città: deve essere scelta una delle tre possibilità Como, Lecco, Varese;
* Quartiere: deve essere scelta una delle quattro possibilità Nord-Ovest, Nord-Est, Sud-Ovest, Sud-Est, e la scelta deve essere coerente con a latitudine e la longitudine specificate;
* Latitudine: deve essere compresa tra 0.0 e 10.0;
* Longitudine: deve essere compresa tra 0.0 e 20.0;

La stringa password è stata criptata con MD5 per proteggerla in modo più efficiente. MD5 è una funzione di hash crittografica unidirezionale e irreversibile, ciò significa che data in input una stringa di lunghezza arbitraria la funzione ne restituisce un’altra a 128 bit dalla quale è impossibile risalire alla stringa di input. Il processo di generazione è molto veloce e assicura che è altamente improbabile ottenere con due diverse stringhe in input uno stesso valore hash in output.

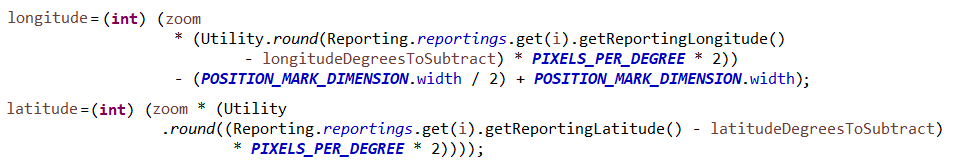
Per facilitare l’utente è stata inserita nell’interfaccia grafica una mappa con l’indicatore di ogni segnalazione. La cartina ha dimensione 1000x500 pixel, ogni grado equivale a cinquanta pixel, per un totale di venti gradi di longitudine e dieci di latitudine. La mappa è stata poi divisa in quattro parti uguali corrispondenti ai quartieri Nord-Ovest, Nord-Est, Sud-Ovest e Sud-Est. Le funzioni che si occupano di calcolare la posizione dei segna posizione delle segnalazioni se non c’è alcun quartiere specificato sono le seguenti:



Nelle quali:

* zoom è un double compreso tra uno e due che rappresenta il livello di zoom;
* round(double x) è un metodo che serve ad arrotondare il risultato alla prima cifra dopo la virgola;
* PIXELS\_PER\_DEGREE è una costante intera pari a cinquanta che indica da quanti pixel è formato ogni grado;
* POSITION\_MARK\_DIMENSION.width è una costante pari alla larghezza del segna posizione della segnalazione, che nella formula della longitudine viene prima divisa per due e sottratta in modo tale che il centro dell’etichetta corrisponda al punto della segnalazione, e poi aggiunta perché corrisponde alla distanza tra il pannello e la mappa;

Se invece viene selezionato un quartiere tra i quattro specificati precedentemente la mappa ha dimensione 1000x500 pixel e ogni grado equivale a cento pixel per un totale di dieci gradi di longitudine e cinque di latitudine. Le funzioni che calcolano la posizione dei segna posizione sono le seguenti:



Nelle quali, a differenza delle precedenti, la costante PIXELS\_PER\_DEGREE viene moltiplicata per due e, a seconda del quartiere, vengono sottratti dei gradi in modo che le segnalazioni siano visibili sulla mappa. Nello specifico, il numero di gradi sottratti in relazione al quartiere è il seguente:

* Nord-Ovest: zero gradi di latitudine e longitudine;
* Nord-Est: zero gradi di latitudine e dieci di longitudine;
* Sud-Ovest: cinque gradi di latitudine e zero di longitudine;
* Sud-Est: cinque gradi di latitudine e dieci di longitudine;

Per identificare a quale segnalazione corrisponde un segna posizione è stato impostato come nome di ogni etichetta l’id della segnalazione a cui si riferisce. In questo modo si può ricavare l’id della segnalazione corrispondente al segna pozione cliccato grazie al metodo evt.getComponent().getName() che restituisce il nome dell’etichetta selezionata.

Inoltre abbiamo deciso di assegnare all’id delle segnalazioni chiuse il numero zero per poterle distinguere da quelle aperte durate la lettura del file .csv attraverso un semplice controllo e inserirle nell’ArrayList corrispondente.

Le segnalazioni chiuse verranno poi utilizzate per costruire la tabella della cronologia delle segnalazioni.

**3.COMPONENTI UTILIZZATI**

* IDE utilizzato per la progettazione e la stesura del corpo del programma:



* WindowBuilder: estensione di Eclipse che permette di sviluppare interfacce grafiche tramite semplici operazioni di drag-and-drop;
* Documentazione ufficiale java reperibile al seguente link: https://docs.oracle.com/javase/8/docs/api/index.html



* Sitografia:

1. http://[www.stackoverflow.com](http://www.stackoverflow.com) : sito con domande e risposte relative alla programmazione java;
2. <http://forum.html.it/forum> : sito con guide relative alla programmazione java;

**4.ANALISI DELLE FUNZIONALITÀ**

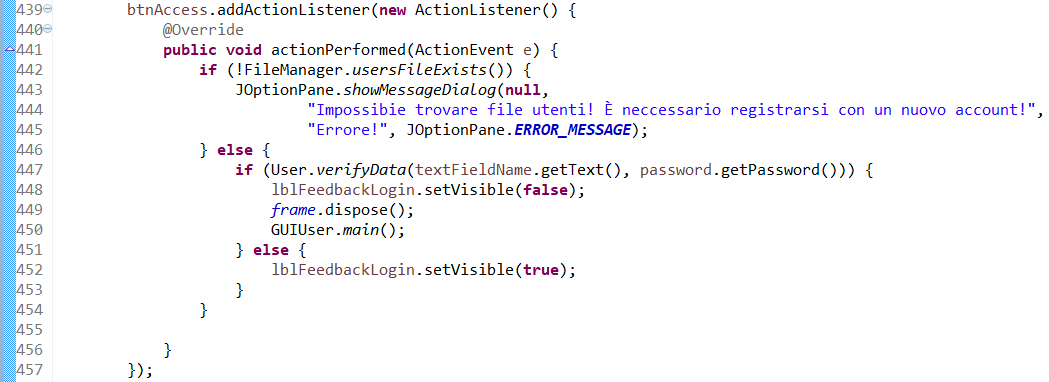
In questo paragrafo sono state analizzate le funzionalità principali del programma WatchNeighbors.

N.B: le funzionalità che si ripetono sono state analizzate una volta sola.

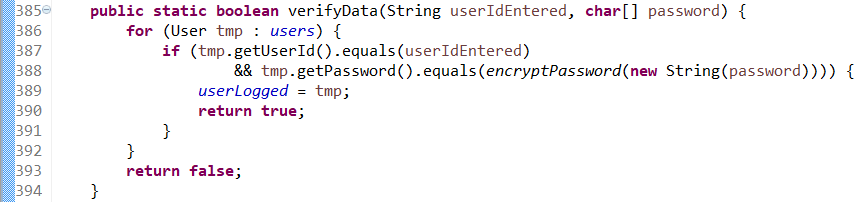
1. **Interfaccia principale**

* Login:

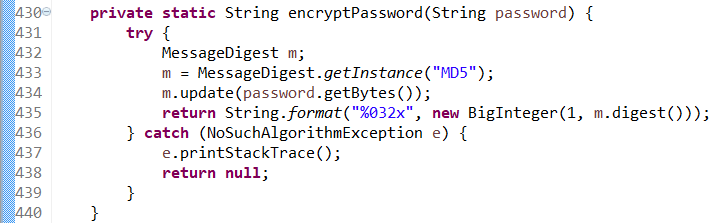
Il codice seguente viene eseguito quando l’utente clicca sul pulsante “Accedi”



* Si controlla se il file utenti.csv esiste, e in caso negativo viene visualizzato un messaggio d’errore;
* Se i file esiste, si controllano i campi compilati dall’utente tramite il metodo verifyData(String userIdEntered, char[] password) che verrà analizzato successivamente;
* Se i dati inseriti corrispondo a quelli di un utente registrato vengono eseguite le seguenti operazioni:
  + - viene resa invisibile l’etichetta con l’avviso di errore;
    - viene distrutta l’interfaccia principale;
    - viene avviata l’interfaccia utente;
* Altrimenti se i dati non corrispondono viene resa visibile l’etichetta con l’avviso di errore;
* Metodo verifyData(String userIdEntered, char[] passsword):

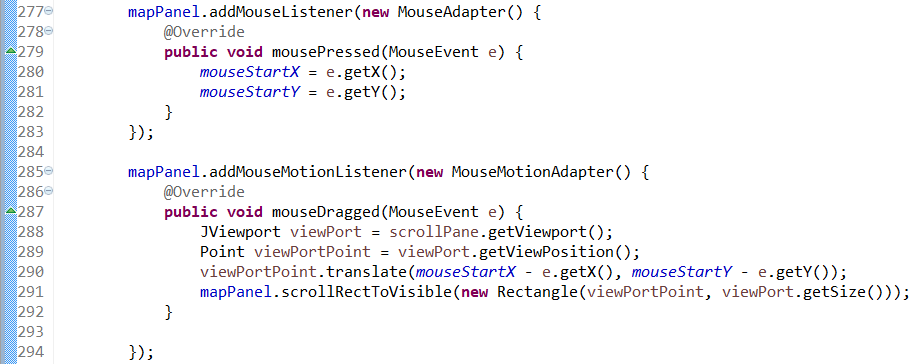


* Per ogni oggetto Utente dell’ArrayList utenti si controlla se l’userId e la password (quest’ultima viene prima convertita in una stringa e criptata con il metodo che verrà analizzato successivamente) passati come argomenti corrispondo ai corrispondenti campi dell’oggetto;
* Se i due parametri corrispondono vengono eseguite le seguenti azioni:
  + - Il campo userLogged viene aggiornato con il riferimento all’utente che soddisfa la condizione;
    - Viene restituito il valore true;
  + se nessun oggetto User soddisfa le condizioni viene restituito il valore false;
* Metodo encryptPassword(String password)



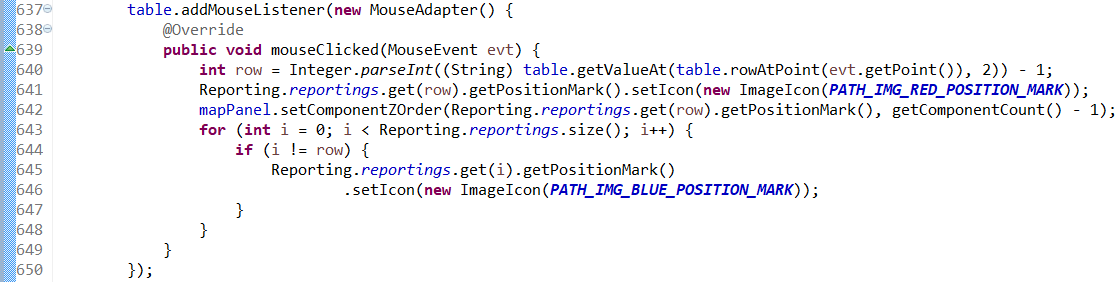
* + Viene creato un oggetto di tipo MessageDigest;
  + Viene chiamato il metodo statico getInstance(String algorithm) che restituisce un oggetto MessageDigest che implementa l’algoritmo specificato, in questo caso MD5;
  + Attraverso i metodo update(byte[] input) la password viene processata attraverso l’oggetto MessageDigest;
  + Viene restituita una stringa composta da 32 caratteri che rappresenta la stringa passata come argomento criptata con l’algoritmo MD5;
  + Nel caso venga sollevata l’eccezione NoSuchAlgorithmException viene stampato l’errore e restituito il valore null;
* Movimento mappa:

Il codice seguente viene eseguito quando l’utente interagisce con la mappa



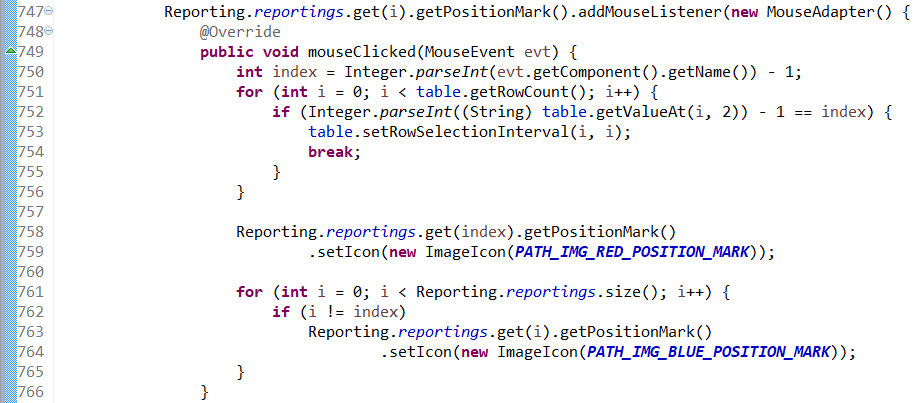
* + Quando l’utente preme il tasto sinistro del mouse sopra la mappa, vengono salvate nelle variabili statiche mouseStartX e mouseStartY le coordinate x e y del punto in cui è avvenuto il click;
  + Quando l’utente trascina il mouse vengono eseguite le seguenti azioni:
    - Viene creato un oggetto di tipo JViewPort con il metodo getViewport() della classe ScrollPane che restituisce la JViewPort dello scrollPane;
    - Viene creato un oggetto di tipo Point chiamato vvp utilizzando il metodo getViewPosition() che rappresenta le coordinate x e y della view che appare nell’angolo in alto a sinistra della viewPort;
    - Il punto viewPortPoint viene traslato della differenza tra la posizione iniziale del mouse (memorizzata in mouseStartX e mouseStartY) e la posizione attuale;
    - Il pannello viene spostato in modo che sia visibile la vista del nuovo rettangolo che viene creato con punto in alto a sinistra il punto viewPortPoint e dimensioni pari a quelle della viewPort;
* Selezione segnalazione dalla tabella:

Il codice seguente viene eseguito quando l’utente seleziona una riga della tabella



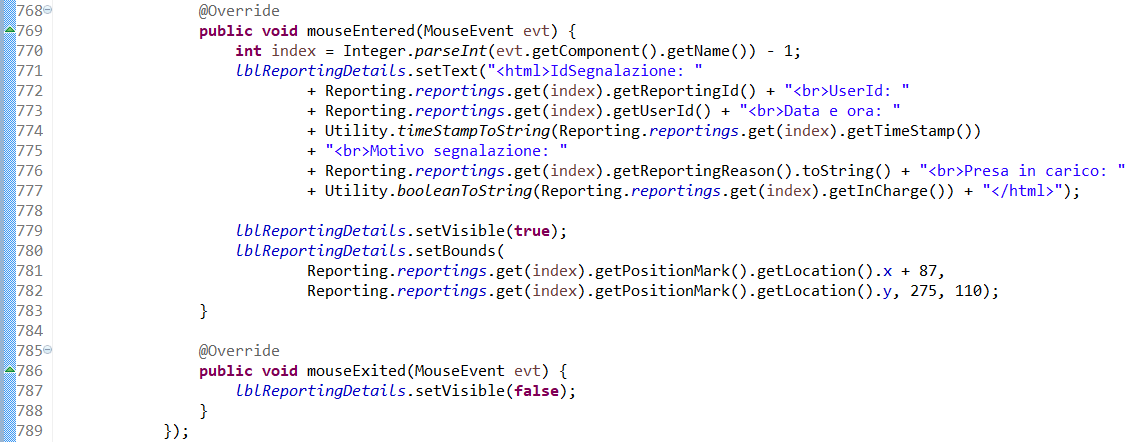
* + Viene creata una variabile intera a cui viene assegnato il numero contenuto nella terza colonna alla riga selezionata della tabella (corrispondente all’id della segnalazione), diminuito di uno in modo che sia corrispondente alla posizione della segnalazione selezionata nell’ArrayList delle segnalazioni;
  + Si ottiene dall’ArrayList la segnalazione corrispondente a quella selezionata e si imposta come immagine dell’etichetta il segna posizione rosso;
  + Viene organizzato l’ordine z dei componenti in modo che il segna posizione della segnalazione selezionata sia sopra le altre in caso di sovrapposizione;
  + Viene effettuato un ciclo for per impostare come immagine dell’etichetta sulla mappa il segna posizione blu per tutte le segnalazioni tranne quella selezionata;
* Selezione segnalazione dalla mappa:

Il codice seguente viene eseguito quando l’utente clicca sul segna posizione di una segnalazione sulla mappa:



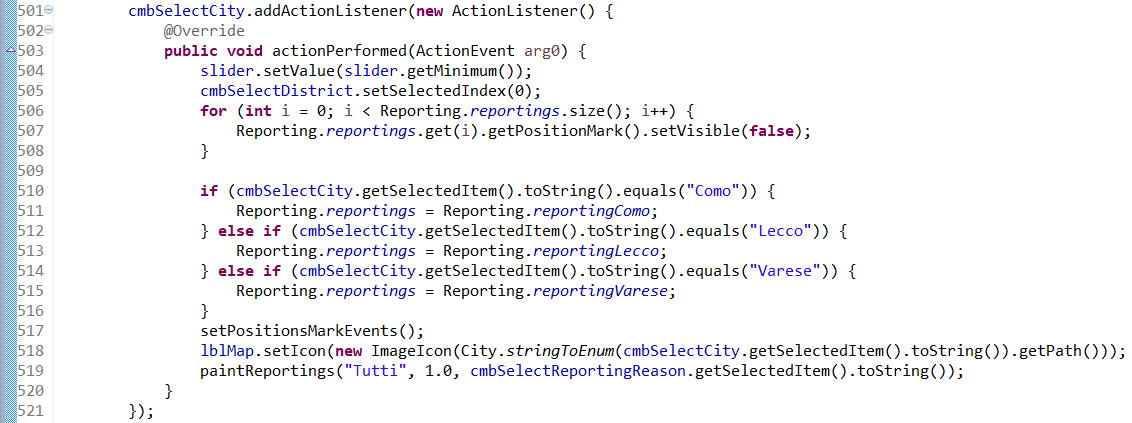
* Viene individuata la posizione all’interno dell’ArrayList della segnalazione corrispondente al segna posizione selezionato (il nome di un segna posizione corrisponde all’id della segnalazione a cui si riferisce, mentre l’indice all’interno dell’ArrayList corrisponde all’id della segnalazione diminuito di uno);
* Viene selezionata la riga della tabella che corrisponde alla segnalazione;
* Viene posta come immagine dell’etichetta il segna posizione rosso;
* L’immagine dell’etichetta di tutte le altre segnalazioni viene impostata con il segna posizione blu;
* Visualizzare dettagli segnalazione:

Il codice seguente viene eseguito quando l’utente entra con il puntatore del mouse sul segna posizione di una segnalazione sulla mappa:



* + Viene individuata la posizione all’interno dell’ArrayList della segnalazione corrispondente al segna posizione selezionato (il nome di un segna posizione corrisponde all’id della segnalazione a cui si riferisce, mentre l’indice all’interno dell’ArrayList corrisponde all’id della segnalazione diminuito di uno);
  + Viene inserito il testo nell’etichetta contenente le informazioni dei campi principali della segnalazione, sfruttando i tag html per andare a capo all’interno dell’etichetta;
  + Viene resa visibile l’etichetta con le informazioni della segnalazione;
  + Vengono impostate le dimensioni dell’etichetta e la sua posizione in base a quella del pannello;
  + Quando il mouse esce dal segna posizione viene resa invisibile l’etichetta con i dettagli della segnalazione;
* Selezionare una città:

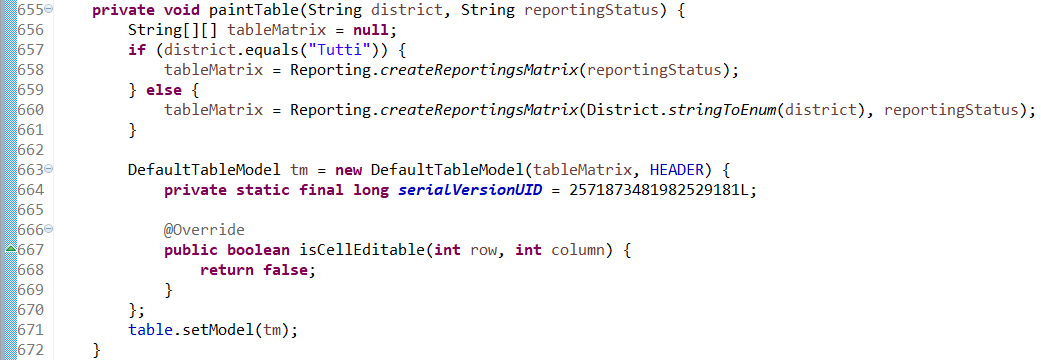
Il codice seguente viene eseguito quando l’utente seleziona una città dalla relativa combo box:



* + Viene settato il valore dello slider al minimo, perché l’immagine della città selezionata verrà mostrata senza zoom;
  + Viene settato il valore della combo box relativo ai quartieri a zero, che corrisponde al quartiere “Tutti”;
  + Vengono resi invisibili tutti i segna posizione delle segnalazioni attualmente visualizzate sulla mappa attraverso un ciclo for;
  + Viene aggiornato il riferimento dell’ArrayList segnalazioni in base alla città selezionata;
  + Vengono assegnati gli eventi visti in precedenza ai segna posizione della città selezionata;
  + Viene impostata l’immagine della mappa a seconda della città scelta;
  + Viene chiamato il metodo paintReportings(String district, double zoom, String reportingReason) che verrà analizzato in seguito;
* Metodo paintReportings(String district, double zoom, String reportingReason):



* + Viene chiamato il metodo paintTable(String district, String reportingReason) che verrà analizzato in seguito;
  + Viene eseguito un ciclo for che effettua le operazioni descritte successivamente per ogni segnalazione;
  + Se la stringa district passata come parametro corrisponde a “Tutti” vengono eseguite le seguenti operazioni:
    - Il segna posizione della segnalazione presa in considerazione viene reso visibile e la sua immagine viene impostata con il segna posizione blu;
    - Viene impostata la posizione del segna posizione utilizzando le funzioni descritte nel paragrafo relativo alle scelte progettuali;
    - Il segna posizione viene aggiunto al pannello e viene organizzato l’ordine z dei componenti in modo che il segna posizione della segnalazione selezionata sia sopra le altre in caso di sovrapposizione;
  + Se invece il quartiere della segnalazione corrisponde a quello passato come argomento vengono eseguite le seguenti operazioni:
    - Il segna posizione della segnalazione presa in considerazione viene reso visibile;
    - Vengono inizializzate due variabili a seconda del quartiere passato come argomento che rappresentano i gradi di latitudine e longitudine da sottrarre come spiegato nel paragrafo relativo alle scelte progettuali;
    - Viene impostata la posizione del segna posizione utilizzando le funzioni descritte nel paragrafo relativo alle scelte progettuali;
    - Altrimenti il segna posizione viene reso invisibile;
  + Se il motivo della segnalazione è diverso da “Tutti” e dal motivo della segnalazione passato come argomento il segna posizione viene reso invisibile;
* Metodo paintTable(String district, String reportingReason):



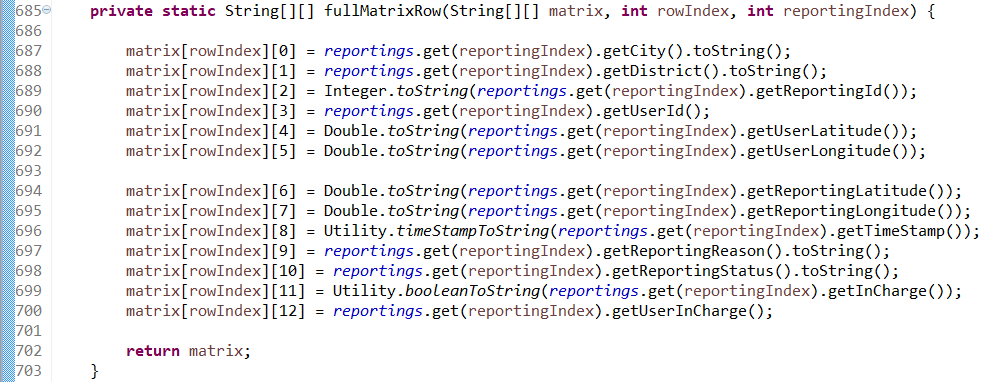
* + Viene creta una matrice di stringhe che viene inizialmente posta uguale a null;
  + Se la stringa district passata come argomento è uguale a “Tutti” la matrice viene inizializzata chiamando il metodo statico createReportingsMatrix(String reportingReason) della classe Reporting che verrà analizzato in seguito;
  + Altrimenti la matrice viene inizializzata chiamando il metodo statico createReportingsMatrix (District district, String reportingReason) della casse Reporting che verrà analizzato in seguito;
  + Viene creato un oggetto di tipo DefaultTableModel passando come parametri la costante HEADER, un array di stringhe che rappresentano i nomi delle colonne della tabella, e la matrice contente le stringhe da posizionare all’interno delle righe e colonne;
  + Viene definita una classe locale anonima per ridefinire il metodo isCellEditable(int row, int column) in modo che restituisca sempre false rendendo impossibile modificare le celle della tabella;
  + Viene impostato il modello della tabella con il DefaultTableModel creato in precedenza;
* Metodo createReportingsMatrix(String reportingReason):



* + Viene creta una matrice di stringhe che viene inizialmente posta uguale a null;
  + Se la stringa reportingReason passata come argomento è uguale a “Tutti” la matrice viene inizializzata con numero di righe pari al numero di segnalazioni e tredici colonne, e viene riempita chiamando il metodo fillMatrixRow(String[][] matrix, ArrayList<Reporting> reportings, int indexRow, int indexReporting) per ogni segnalazione dell’ArrayList;
  + Altrimenti vengono eseguite e seguenti azioni:
    - Si inizializza un contatore uguale a zero;
    - Per ogni segnalazione si controlla se il motivo della segnalazione corrisponde a quello passato come argomento e nel caso in cui la condizione sia soddisfatta si incrementa il contatore;
    - Viene inizializzata la matrice con numero di righe pari al numero del contatore e tredici colonne, e viene riempita chiamando il metodo fillMatrixRow(String[][] matrix, ArrayList<Reporting> reportings, int indexRow, int indexReporting) per ogni segnalazione dell’ArrayList con motivo uguale a quello passato come parametro;
  + Viene restituita la matrice;
* Metodo createReportingsMatrix(District district, String reportingReason):

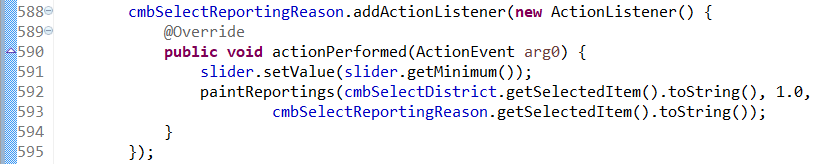


* + Viene creta una matrice di stringhe che viene inizialmente posta uguale a null;
  + Se la stringa reportingReason passata come parametro è uguale a “Tutti” vengono eseguite le seguenti azioni:
    - Si inizializza un contatore uguale a zero;
    - Per ogni segnalazione si controlla se il quartiere della segnalazione corrisponde a quello passato come argomento e nel caso in cui la condizione sia soddisfatta si incrementa il contatore;
    - Viene inizializzata la matrice con numero di righe pari al numero del contatore e tredici colonne, e viene riempita chiamando il metodo fillMatrixRow(String[][] matrix, ArrayList<Reporting> reportings, int indexRow, int indexReporting) per ogni segnalazione dell’ArrayList con quartiere uguale a quello passato come parametro;
  + Altrimenti vengono eseguite le azioni descritte nel metodo precedente con la differenza che insieme al controllo del motivo della segnalazione viene controllato anche che il quartiere corrisponda a quello passato come parametro;
* Metodo fillMatrixRow(String[][] matrix, ArrayList<Reporting> reportings, int indexRow, int indexReporting):



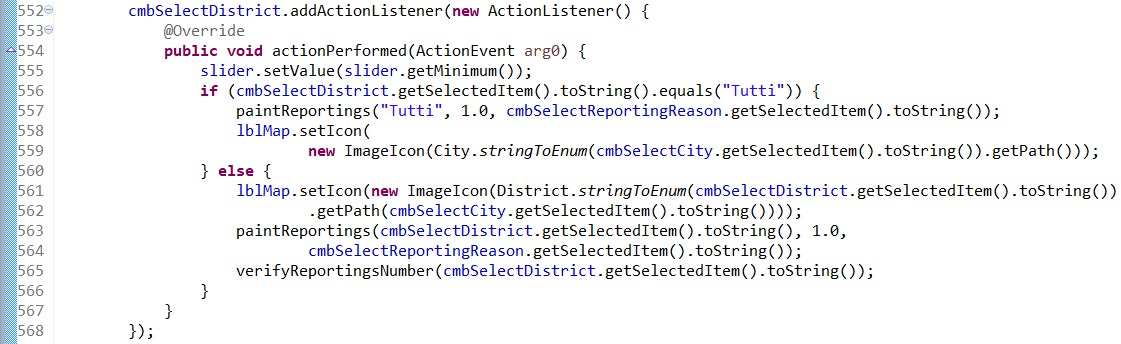
* Vengono riempite le colonne della matrice, passata come parametro, alla riga passata come parametro con i campi della segnalazione alla posizione passata come parametro;
* Viene restituita la matrice;
* Selezionare un motivo della segnalazione:

Il codice seguente viene eseguito quando l’utente seleziona una città dalla relativa combo box:

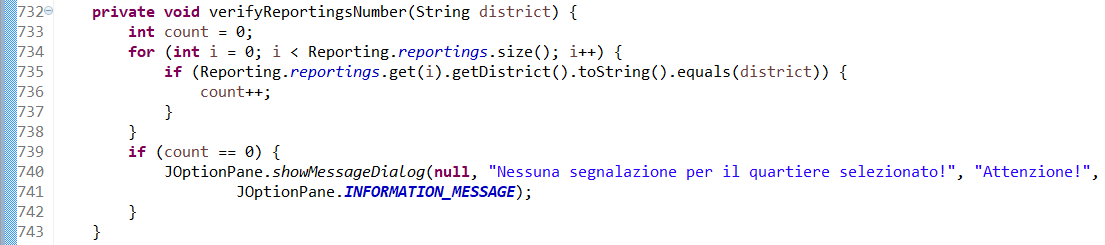


* + Viene settato il valore dello slider al minimo;
  + Viene chiamato il metodo paintReportings, analizzato in precedenza, passando come parametri il quartiere attualmente selezionato, il double dello zoom pari a 1.0 e il motivo della segnalazione scelto;
* Selezionare un quartiere:

Il codice seguente viene eseguito quando l’utente seleziona un quartiere dalla relativa combo box:



* + Viene settato il valore dello slider al minimo, perché l’immagine del quartiere selezionato verrà mostrata senza zoom;
  + Se il quartiere selezionato corrisponde a “Tutti” vengono eseguite le seguenti azioni:
    - Viene chiamato il metodo paintReportings, analizzato in precedenza con parametro “Tutti” come quartiere selezionato, 1.0 come livello di zoom e il motivo selezionato come motivo della segnalazione;
    - Viene impostata l’immagine della mappa con quella della città attualmente selezionata;
  + Altrimenti vengono eseguite le seguenti operazioni:
    - Viene impostata l’immagine della mappa con quella del quartiere della città attualmente selezionati;
    - Viene chiamato il metodo paintReportings, analizzato in precedenza con parametri: il quartiere selezionato, 1.0 come livello di zoom e il motivo selezionato come motivo della segnalazione;
    - Viene chiamato il metodo verifyReportingsNumber(String district), che verrà analizzato successivamente passando il quartiere selezionato come parametro;
* Metodo verifyReportingsNumber(String district):



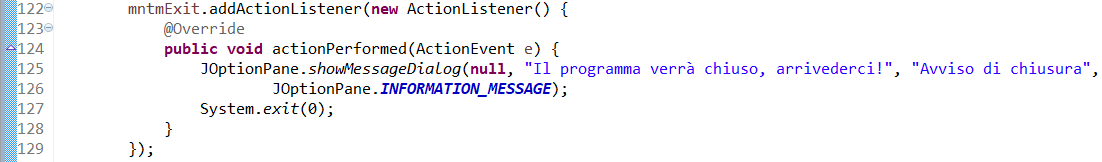
* + Viene inizializzato un contatore uguale a zero;
  + Per ogni segnalazione si controlla se il quartiere della segnalazione corrisponde a quello passato come parametro e nel caso in cui la condizione sia soddisfatta si incrementa il contatore;
  + Se, dopo aver effettuato il ciclo for, il contatore è uguale a zero viene mostrato un avviso all’utente;
* Zoom:

Il codice seguente viene eseguito quando l’utente regola il livello di zoom attraverso lo slider:



* + Viene inizializzato una variabile double, chiamata zoom, dividendo per dieci il valore attuale dello slider (lo slider viene creato in modo che il suo valore sia sempre compreso tra dieci e venti);
  + Viene inizializzato un BufferedImage contenente l’immagine della città o del quartiere attualmente selezionati;
  + Viene inizializzato un nuovo BufferedImage di grandezza pari a quella del precedente BufferedImage moltiplicato per il ridimensionamento;
  + Viene creato un oggetto di tipo Graphics2D, grazie al quale si può disegnare dentro il destinationBuffer, attraverso il metodo createGraphics();
  + Viene creato un oggetto di tipo AffineTrasform che indica di quanto scalare l’immagine;
  + Viene disegnata nel destinationBuffer l’immagine contenuta nel sourceBuffer dopo aver applicato la trasformazione indicata nell’oggetto attribute creato in precedenza;
  + Viene creato un nuovo oggetto di tipo ImageIcon che rappresenta l’immagine contenuta nel destinationBuffer;
  + Viene impostata come immagine della mappa la nuova immagine ottenuta e viene ingrandita la mappa in modo che la contenga;
  + Viene chiamato il metodo paintReportings passando come parametri il quartiere e il motivo della segnalazione selezionati e il ridimensionamento;
  + Nel caso venga sollevata una IOException viene mostrato un messaggio di errore;
* Chiudere il programma:

Il codice seguente viene eseguito quando l’utente seleziona la voce “Esci” del menu “File”



* + Viene mostrato un messaggio che avvisa l‘utente che il programma verrà chiuso;
  + Viene chiamato il metodo System.exit(0) che si occupa della chiusura de programma;

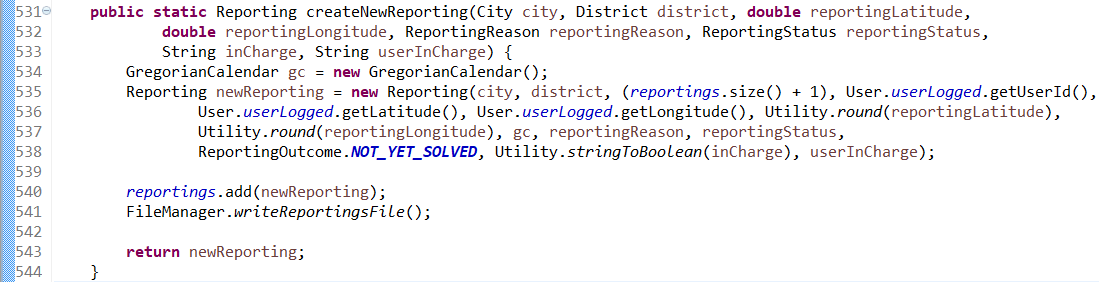
1. **Interfaccia utente**

* Creare una nuova segnalazione:

Il codice seguente viene eseguito quando l’utente clicca sul tasto “conferma”:

****

* + Viene controllato tramite il metodo belongs(double latitude, double longitude) se le coordinate della nuova segnalazione appartengono al quartiere selezionato, e in caso la condizione non sia soddisfatta viene mostrato un messaggio d’errore;
  + Altrimenti si eseguono le seguenti operazioni:
    - Se l’utente ha selezionato “Sì” nella combo box relativa alla presa in carico viene aggiornata la stringa relativa all’utente incaricato con il suo userId, altrimenti viene lasciata vuota;
    - Viene creata una nuova segnalazione attraverso il metodo statico createNewReporting della classe Reporting, che verrà analizzato in seguito, passando come parametri i campi compilati dall’utente;
    - Viene impostato segna posizione blu come immagine dell’etichetta della nuova segnalazione;
    - Vengono chiamati i metodi paintTable() e setPositionsMarkEvent() analizzato in precedenza;
    - Il segna posizione viene aggiunto al pannello e viene organizzato l’ordine z dei componenti in modo che il segna posizione della nuova segnalazione sia sopra le altre in caso di sovrapposizione;
    - Viene chiamato il metodo paintReportings() analizzato in precedenza;
    - Viene resa invisibile l’etichetta della segnalazione di prova e viene mostrato un avviso;
* Metodo createNewReporting (City city, District district, double reportingLatitude, double reportingLongitude, ReportingReason reportingReason, ReportingSatus reportingStatus, String inCharge, String userInCharge):

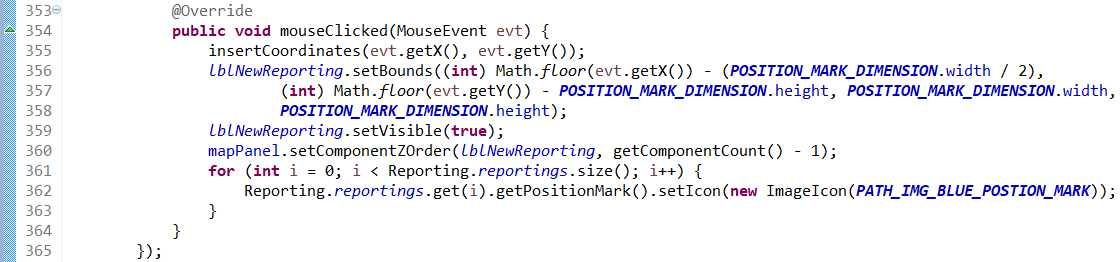


* + Viene creato un nuovo oggetto di tipo GregorianCalendar che rappresenta la data e l‘ora attuali;
  + Viene invocato il costruttore della classe passando i seguenti parametri:
    - città, quartiere, latitudine e longitudine, motivo, stato, presa in carico e l’id dell’utente incaricato della segnalazione, passati al metodo come parametri, che corrispondono alle informazioni inserite dall’utente;
    - l’id della segnalazione che viene assegnato dal programma in modo progressivo;
    - l’id, la latitudine e la longitudine dell’utente loggato che ha aperto la segnalazione;
    - l’oggetto di tipo GregorianCalendar creato in precedenza che rappresenta la data e l’ora attuali;
    - l’esito della segnalazione, impostato a “Non ancora risolto” di default;
  + Viene aggiunta la nuova segnalazione all’ArrayList;
  + Viene chiamato il metodo statico writeReportingsFile() della classe FileManager che verrà analizzato in seguito;
  + Viene restituita la segnalazione creata;
* Metodo writeReportingsFile():



* + Viene creato un nuovo oggetto di tipo BufferedWriter per la scrittura sul file “segnalazioni.csv”;
  + Vengono effettuati quattro cicli for-each (uno per ogni ArrayList) che si occupano di scrivere sul file per ogni segnalazione la stringa che la rappresenta separate dal carattere “\n”;
  + Viene chiuso il BufferedWriter creato in precedenza;
  + Nel caso venga sollevata un’eccezione di tipo IOException viene mostrato un messaggio d’errore;
* Inserire segnalazione sulla mappa:

Il codice seguente viene eseguito quando l’utente clicca sulla mappa

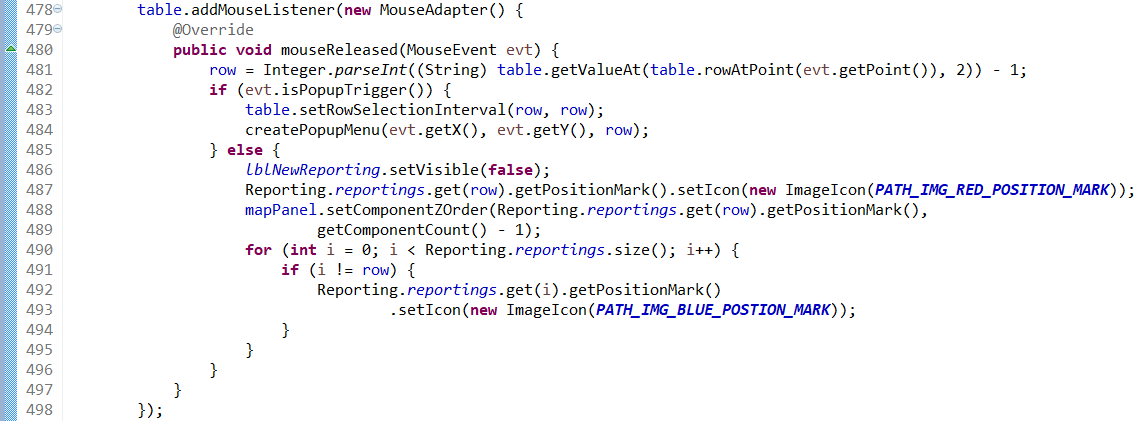


* + Viene chiamato il metodo insertCoordinates(int x, int y) che si occupa di inserire negli spinners le coordinate in gradi del punto cliccato sulla mappa;
  + Viene posizionato il segna posizione della segnalazione di prova nel punto in cui l’utente ha cliccato con il mouse in modo che il centro del lato inferiore dell’etichetta corrisponda al punto della segnalazione;
  + Viene reso visibile il segna posizione e viene organizzato l’ordine z dei componenti in modo che l’etichetta della segnalazione di prova sia sopra le altre in caso di sovrapposizione;
  + Si effettua un ciclo for che si occupa di impostare l’immagine del segna posizione blu sull’etichetta di ogni segnalazione;
* Metodo insertCoordinates(int x, int y):



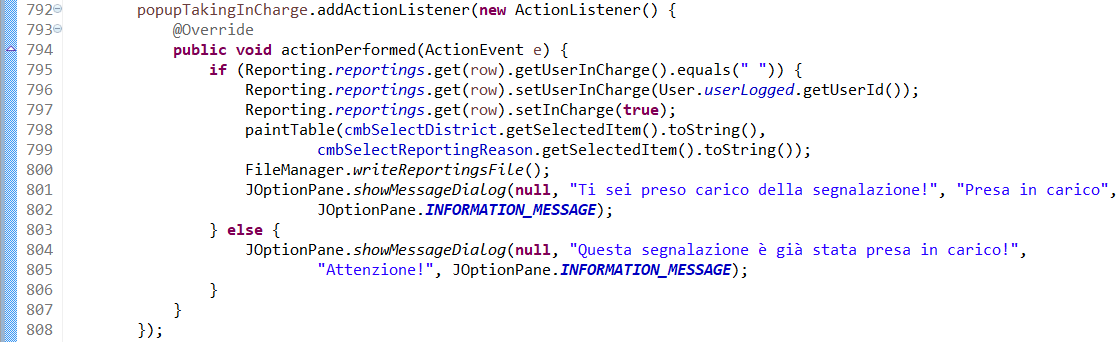
* Viene inizializzata una variabile double, chiamata zoom, dividendo per dieci il valore attuale dello slider (lo slider viene creato in modo che il suo valore sia sempre compreso tra dieci e venti);
* Vengono inizializzate a 0.0 due variabili double che rappresentano la longitudine e la latitudine del punto di coordinate x e y cliccato sulla mappa;
* Se il quartiere selezionato corrisponde a “Tutti”, si ricavano le coordinate in gradi del punto cliccato nel modo seguente:
  + La longitudine si ottiene sottraendo alla coordinata x del punto cliccato la dimensione del segna posto, il risultato viene diviso per lo zoom moltiplicato per la costante PIXELS\_PER\_DEGREE. Il risultato ottenuto viene arrotondato alla prima cifra dopo la virgola;
  + La latitudine si ottiene seguendo lo stesso procedimento utilizzato per la longitudine, utilizzando però la coordinata y del punto cliccato;
  + Vengono controllati i valori ottenuti, e nel caso in cui non rientrino nelle coordinate della mappa vengono impostati al valore minimo o massimo consentiti;
* Altrimenti vengono eseguite e seguenti operazioni:
  + - Vengono inizializzate due variabili a seconda del quartiere passato come parametro che rappresentano i gradi di latitudine e longitudine da aggiungere come spiegato nel paragrafo relativo alle scelte progettuali;
    - Si ricavano le coordinate in gradi del punto ciccato nel modo seguente:
      * La longitudine si ottiene sottraendo alla coordinata x del punto cliccato la dimensione del segna posto, il risultato viene diviso lo zoom e moltiplicato per la costante PIXELS\_PER\_DEGREE moltiplicata per due. Il risultato ottenuto viene arrotondato alla prima cifra dopo la virgola e vengono aggiunti i gradi di longitudine da aggiungere ricavati precedentemente;
      * La latitudine si ottiene seguendo lo stesso procedimento utilizzato per la longitudine, utilizzando però la coordinata y del punto cliccato;
      * Vengono controllati i valori ottenuti, e nel caso in cui non rientrino nelle coordinate de quartiere selezionato vengono impostati al valore minimo o massimo consentiti;
  + Infine gli spinner vengono impostati con le coordinate ottenute, e la combo box del quartiere viene settata con il quartiere che contiene latitudine e longitudine ricavate;
* Selezionare segnalazione dalla tabella:

Il codice seguente viene eseguito quando l’utente seleziona una riga della tabella cliccando con il tasto destro o sinistro de mouse



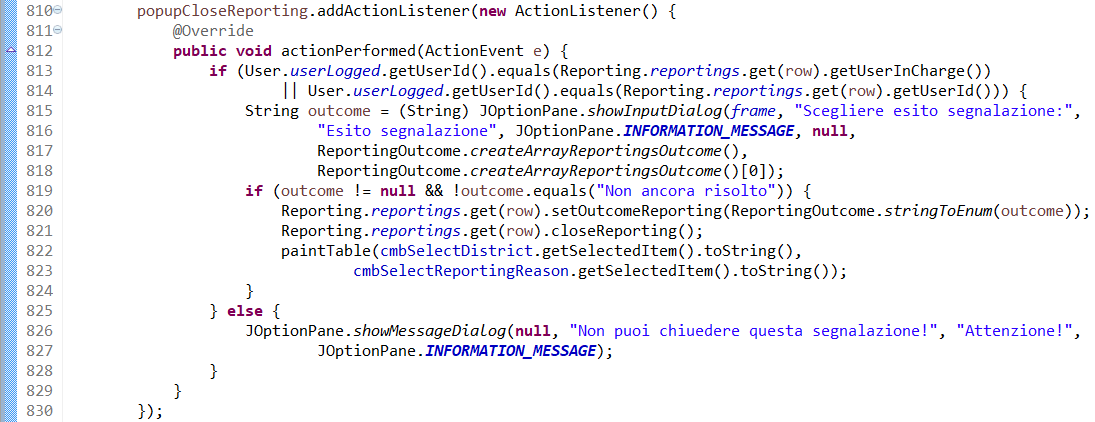
* Viene creata una variabile intera a cui viene assegnato il valore che viene letto nella terza colonna della tabella alla riga selezionata (cioè l’id della segnalazione) diminuito di uno in modo da ottenere la posizione della segnalazione selezionata nell’ArrayList delle segnalazioni;
* Se l’evento è stato generato dalla pressione del tasto destro si eseguono le seguenti operazioni:
  + Si evidenzia la riga selezionata della tabella;
  + Viene chiamato il metodo createPopupMenu(int x, int y, int row) che si occupa di creare un menù nel punto dello schermo in cui è stato generato l’evento. Le voci del menù permettono di prendere in carico o chiudere una segnalazione e verranno analizzate in seguito;
* Altrimenti il codice eseguito è lo stesso che permette di selezionare una riga della tabella dell’interfaccia principale descritto in precedenza;
* Prendere in carico una segnalazione:

Il codice seguente viene eseguito quando l’utente, dopo aver cliccato con il tasto destro del mouse su una riga della tabella, seleziona la voce “Prenderti carico della segnalazione”

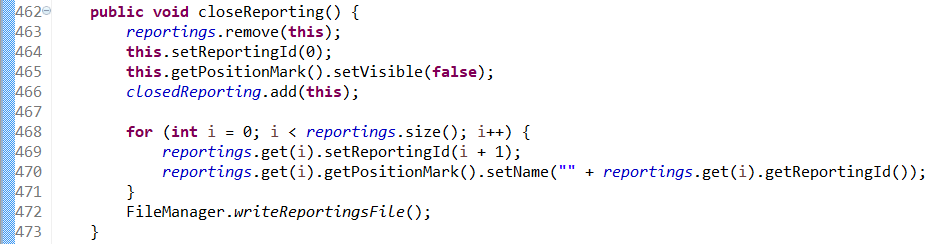


* + Se l’id dell’utente incaricato della segnalazione selezionata è uguale alla stringa “ “, vengono eseguite le seguenti operazioni:
    - Viene impostato come userInCharge della segnalazione l’id dell’utente loggato e viene impostato a true il valore del campo inCharge;
    - Vengono chiamati i metodi paintTable(String district, String reportingReason) e writeReportingsFile() descritti in precedenza;
    - Viene mostrato un messaggio che avvisa l’utente che l’operazione è avvenuta con successo;
  + Altrimenti viene mostrato un messaggio d’errore che indica che la segnalazione è già stata presa in carico da un utente;
* Chiudere una segnalazione:

Il codice seguente viene eseguito quando l’utente, dopo aver cliccato con il tasto destro del mouse su una riga della tabella, seleziona la voce “Chiudi segnalazione”

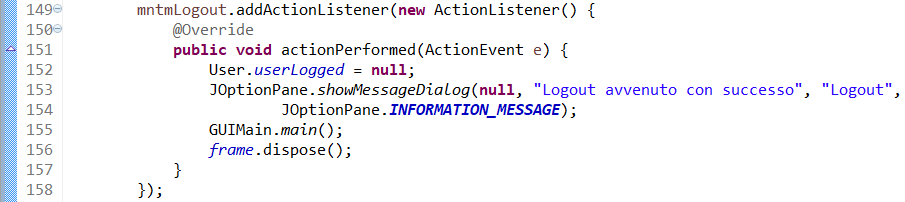


* + Viene controllato che l’id dell’utente loggato corrisponda all’id dell’utente che ha preso in carico o aperto la segnalazione selezionata, e in caso affermativo vengono eseguite le seguenti operazioni:
    - Viene inizializzata una stringa, chiamata outcome, con il valore selezionato dall’utente nella finestra contente una combo box con i possibili esiti della segnalazione;
    - Se la stringa outcome è diversa da null e dalla stringa “Non ancora risolto”, il suo valore viene impostato nel campo reportingOutcome, e vengono chiamati i metodi closeReporting(), che verrà analizzato in seguito, e paintTable descritto in precedenza;
  + Altrimenti viene mostrato un messaggio d’errore che indica che l’utente non soddisfa i requisiti per chiudere la segnalazione selezionata;
* Metodo closeReporting():



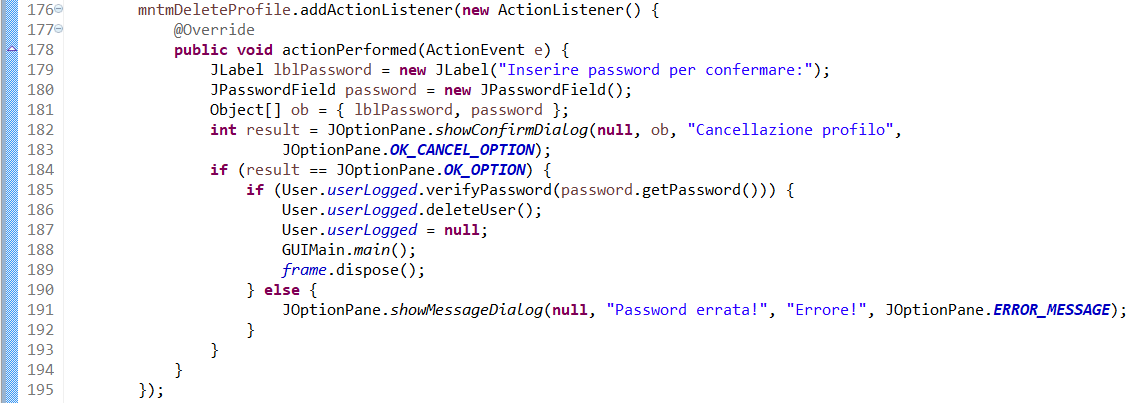
* + Viene rimossa dall’ArrayList la segnalazione che esegue il metodo;
  + Viene impostato a zero l’id della segnalazione (per maggiori informazioni consultare il paragrafo relativo alle scelte progettuali) e viene reso invisibile il suo segna posizione;
  + La segnalazione viene aggiunta all’ArrayList delle segnalazioni chiuse;
  + Viene eseguito un ciclo for che si occupa di impostare l’id di tutte le segnalazioni aperte in modo progressivo e il nome del segna posizione come spiegato nel paragrafo relativo alle scelte progettuali;
  + Viene chiamato il metodo statico writeReportingsFile() analizzato in precedenza;
* Logout:

Il codice seguente viene eseguito quando l’utente seleziona la voce “Logout” del menu “File”

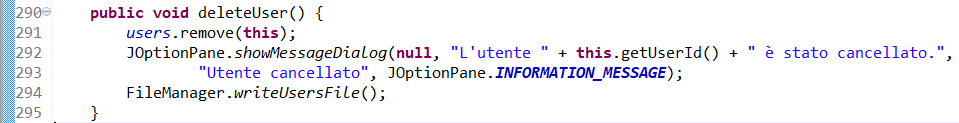


* + UserLogged viene impostato al valore null;
  + Viene mostrato un messaggio che avvisa l‘utente che l’operazione è avvenuta con successo;
  + Viene avviata l’interfaccia principale e distrutta l’interfaccia utente;
* Cancellare profilo:

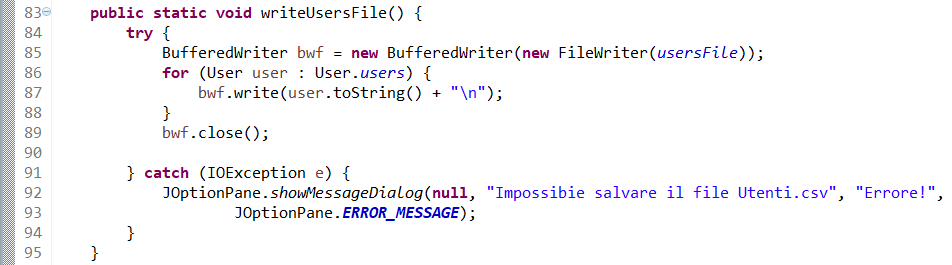
Il codice seguente viene eseguito quando l’utente seleziona la voce “Cancella profilo” del menu “Opzioni”



* Viene mostrata una finestra di dialogo in cui viene richiesto di inserire la password del profilo e confermare o annullare;
* Se l’utente conferma la cancellazione del profilo, viene controllato tramite il metodo verifyPassword(char[] password) se la password inserita corrisponde a quella dell’utente loggato, e in caso affermativo viene chiamato il metodo deleteUser(), che verrà analizzato in seguito, altrimenti viene mostrato un messaggio di errore;
* Metodo deleteUser():



* Viene rimosso dall’ArrayList l’utente che esegue il metodo e viene mostrato un messaggio che indica che l’operazione è avvenuta con successo;
* Viene chiamato il metodo statico writeUsersFIle() della classe FileManager;
* Metodo writeUsersFile():



* + Viene creato un nuovo oggetto di tipo BufferedWriter per la scrittura su file “utenti.csv”;
  + Viene effettuato un ciclo for-each che si occupa di scrivere sul file per ogni utente la stringa che lo rappresenta separate dal carattere “\n”;
  + Viene chiuso il BufferedWriter creato in precedenza;
  + Nel caso venga sollevata un’eccezione di tipo IOException viene mostrato un messaggio d’errore;

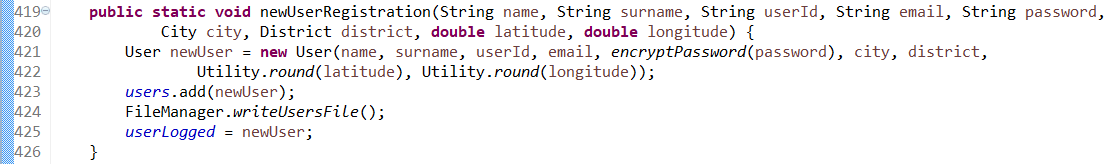
1. **Interfaccia registrazione**

* Registrare un nuovo utente:

Il codice seguente viene eseguito quando l’utente clicca sul pulsante “Registrati”



* + Si controlla se le coordinate inserite dall’utente appartengono al quartiere selezionato, e in caso negativo viene mostrato un messaggio d’errore;
  + Si controlla che i dati inseriti soddisfino le condizioni descritte nel paragrafo relativo alle scelte progettuali, se la condizione non è soddisfatta viene mostrato un messaggio d’errore, altrimenti si eseguono le seguenti operazioni:
    - Viene chiamato il metodo statico newUserRegistration che verrà analizzato in seguito;
    - Vengono distrutte l’interfaccia registrazione e l’interfaccia principale;
    - Viene avviata l’interfaccia utente;
* Metodo newUserRegistration (String name, String surname, String userId, String email, String password, City city, District district, double latitude, double longitude):



* + Viene creato un nuovo oggetto User passando al costruttore i dati passati come parametri;
  + Viene aggiunto il nuovo utente all’ArrayList;
  + Viene chiamato il metodo writeUsersFile() analizzato precedentemente;
  + Viene aggiornato il campo userLogged con il nuovo utente registrato;

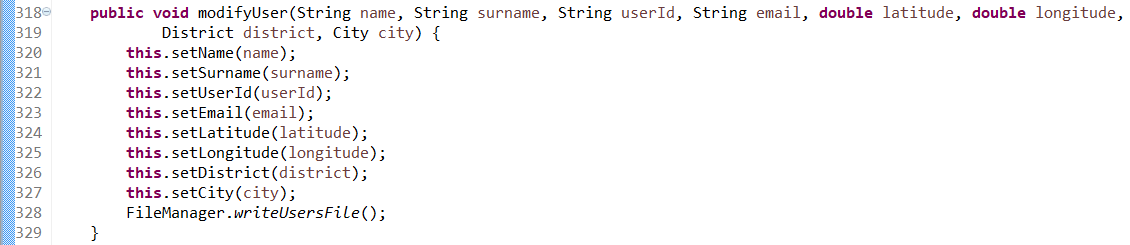
1. **Interfaccia modifica profilo**

* Modificare il profilo:

Il codice seguente viene eseguito quando l’utente clicca sul pulsante “Conferma modifiche”



* + Si controlla se le coordinate inserite dall’utente appartengono al quartiere selezionato, e in caso negativo viene mostrato un messaggio d’errore;
  + Si controlla che i dati inseriti soddisfino le condizioni descritte nel paragrafo relativo alle scelte progettuali. Se la condizione non è soddisfatta viene mostrato un messaggio d’errore, altrimenti viene chiamato il metodo statico modifyUser che verrà analizzato in seguito, viene mostrato un messaggio e distrutta l’interfaccia modifica profilo;
* Metodo modifyUser(String name, String surname, String userId, String email, double latitude, double longitude, District district, City city):

****

* + Vengono impostati i campi dell’oggetto su cui viene eseguito il metodo con i corrispondenti parametri passati al metodo;
  + Viene chiamato il metodo statico writeUsersFile() analizzato in precedenza;

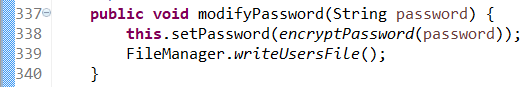
1. **Interfaccia modifica password**

* Modificare la password:

Il codice seguente viene eseguito quando l’utente clicca sul pulsante “Conferma”



* + Si verifica che la password attuale inserita corrisponda a quella dell’utente loggato, e in caso negativo viene mostrato un messaggio d’errore;
  + Si verifica che la nuova password inserita soddisfi i requisiti illustrati nel paragrafo relativo alle scelte progettuali, e in caso negativo viene mostrato un messaggio d’errore;
  + Si verifica che le due password inserite siano uguali, e in caso negativo viene mostrato un messaggio d’errore;
  + Se il messaggio d’errore non è visibile viene chiamato il metodo modifyPassword(String password) che verrà analizzato in seguito;
  + Viene avvisato l’utente che la password è stata modificata con successo, e viene distrutta l’interfaccia modifica profilo;
* Metodo modifiyPassword(String password):



* Viene impostata come password dell’utente la stringa passata come parametro criptata attraverso il metodo encryptPassword(String password) analizzato in precedenza;
* Viene chiamato il metodo writeUsersFile() illustrato in precedenza;

1. **Interfaccia guida**

* Selezionare paragrafo:

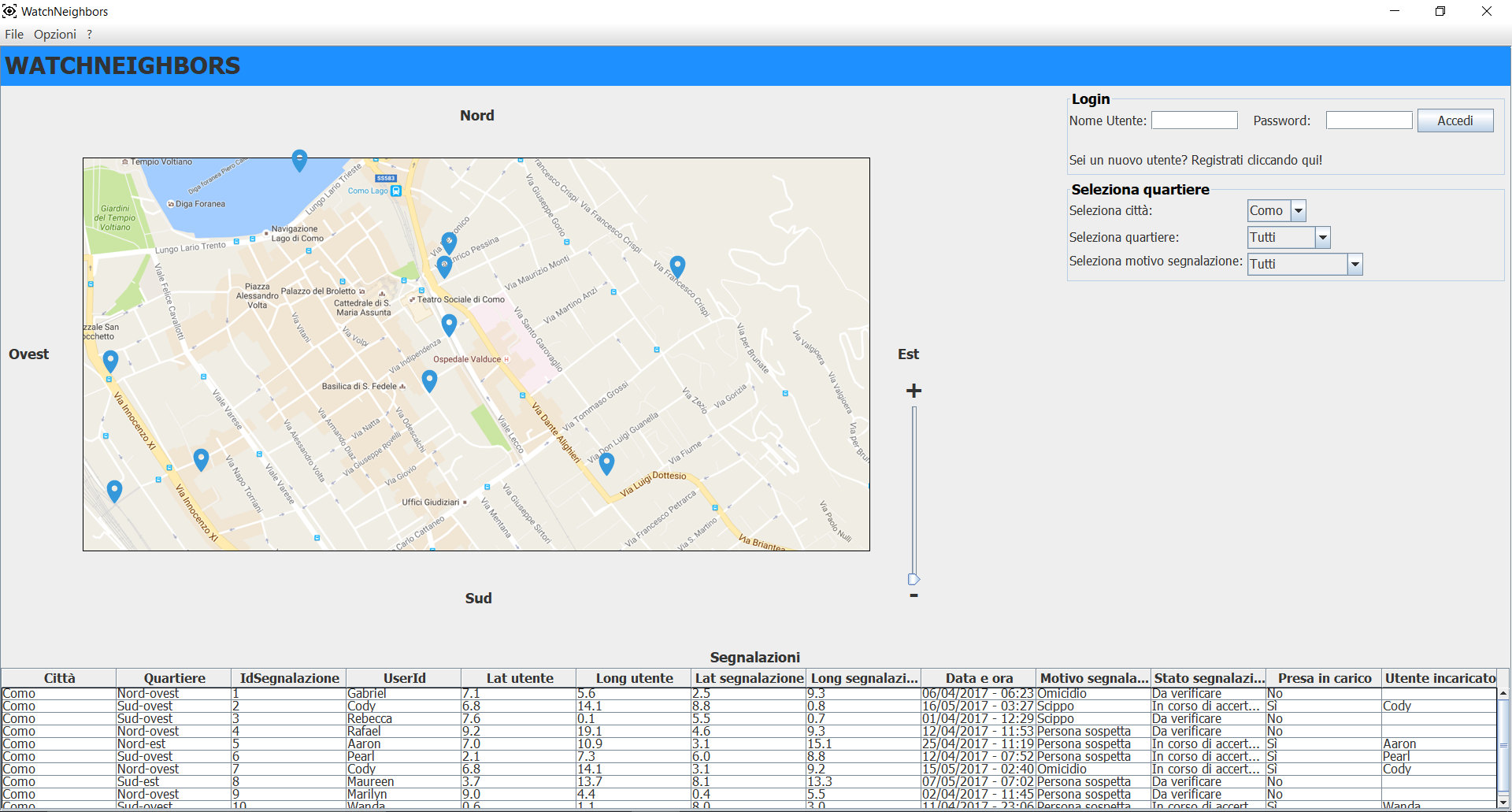
Il codice seguente viene eseguito quando l’utente seleziona un paragrafo della guida:



* + Viene inizializzato un oggetto di tipo treePath attraverso il metodo getPathForLocation(int x, int y) che restituisce il path del nodo alle coordinate specificate;
  + Se il TreePath è diverso da null, si risale dal path al nodo selezionato grazie al metodo toString(), e viene impostato e reso visibile il testo relativo al paragrafo scelto;

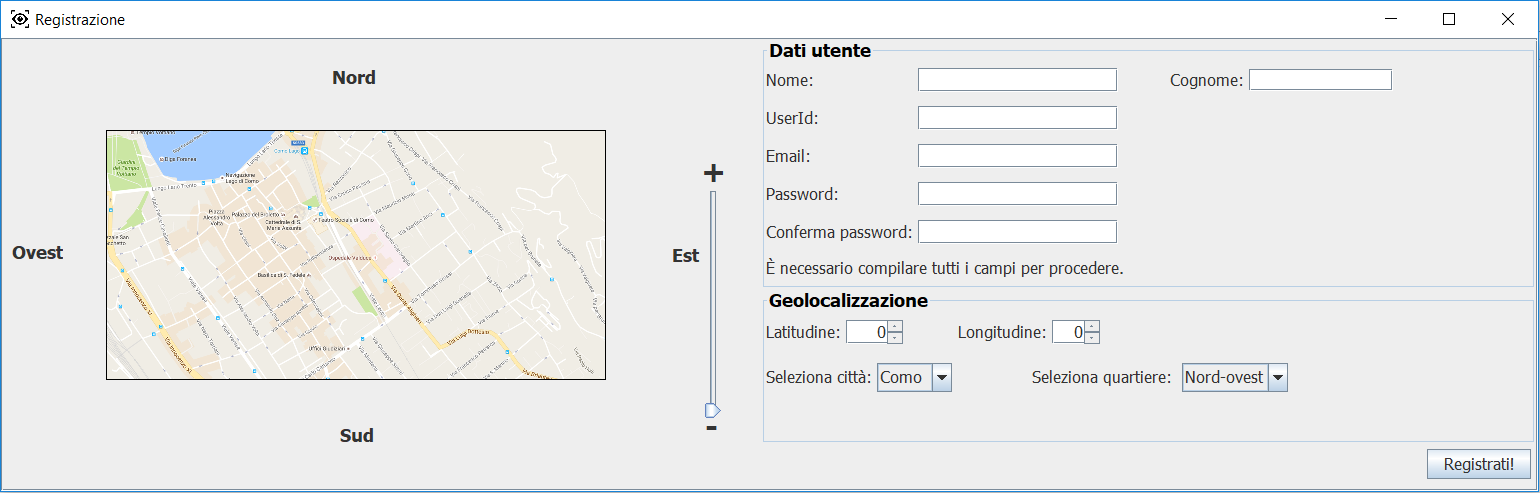
**5.GUIDA**

**a) Interfaccia principale**



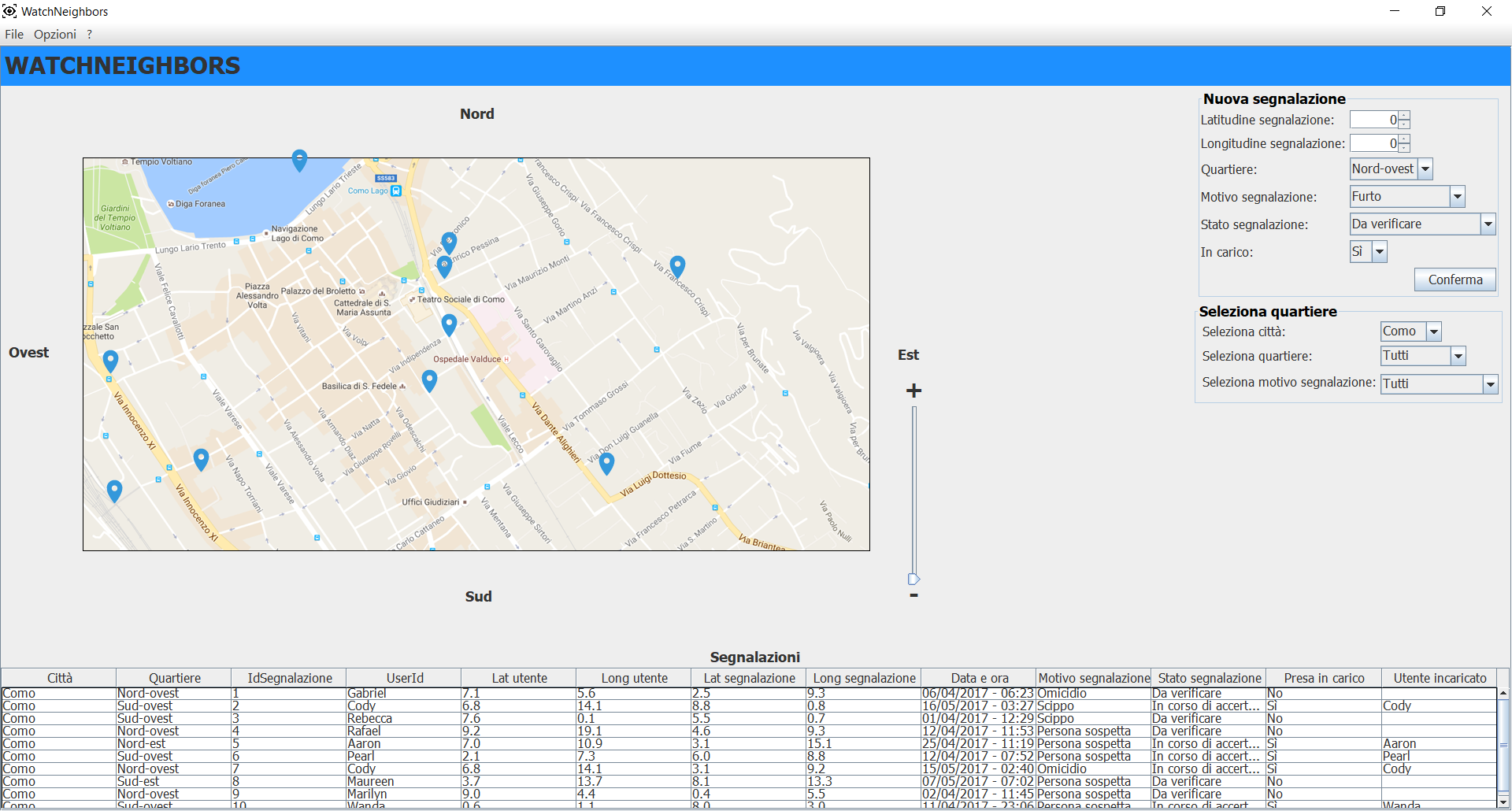
All’avvio del programma si aprirà una schermata iniziale, contenente alcune tra le funzioni principali di esso. A sinistra è collocata una mappa della città fornita di appositi puntatori indicanti le segnalazioni precedentemente effettuate. Puntando col cursore su uno di essi, si otterranno le informazioni principali della segnalazione. Cliccando invece col tasto sinistro, il puntatore si colorerà di rosso e la segnalazione verrà evidenziata nella parte inferiore della schermata all’interno della tabella delle segnalazioni attive. È possibile utilizzare lo zoom della mappa attraverso l’apposita barra situata alla destra di essa. In alto a destra della schermata principale è situata una finestra in cui si ha la possibilità di effettuare il login (nel caso si sia già in possesso di un profilo), oppure di registrarsi cliccando su “Registrati cliccando qui!”. Vi è inoltre l’opportunità di filtrare le segnalazioni per città, quartiere e tipo di segnalazione e, sia la mappa che si trova a sinistra della schermata sia la tabella delle segnalazioni attive, cambieranno a seconda delle vostre scelte. Infine, nella parte inferiore della schermata, è stata predisposta una tabella contenente le segnalazioni attive nella zona da noi selezionata. Cliccando su una riga di questa tabella, verrà evidenziato in rosso il puntatore della segnalazione corrispondente sulla mappa.

**b) Registrazione nuovo utente**



Cliccando su “Registrati cliccando qui!” (situato nell’interfaccia principale) e compilando correttamente tutti i campi, si aprirà una nuova finestra di dimensioni ridotte nella quale potrete inserire i vostri dati personali e creare il vostro profilo-utente. A sinistra comparirà nuovamente la mappa della città da noi selezionata con la stessa funzione di zoom citata precedentemente. Per facilitare l’utilizzo dell’applicazione, sarà possibile cliccare all’interno della mappa, ottenendo così automaticamente la latitudine e la longitudine dell’abitazione del nuovo utente.

**c) Interfaccia utente**

****

Una volta effettuato il login dall’ interfaccia principale, si aprirà una nuova finestra simile alla precedente. Le funzioni della mappa posta sulla sinistra sono le medesime della mappa presente nell’ interfaccia principale. Si ha inoltre la possibilità di cliccare all’interno della mappa, ottenendo così automaticamente la latitudine e la longitudine della nuova segnalazione. Nella parte superiore della schermata è stata predisposta una barra che ci fornirà diverse opzioni:

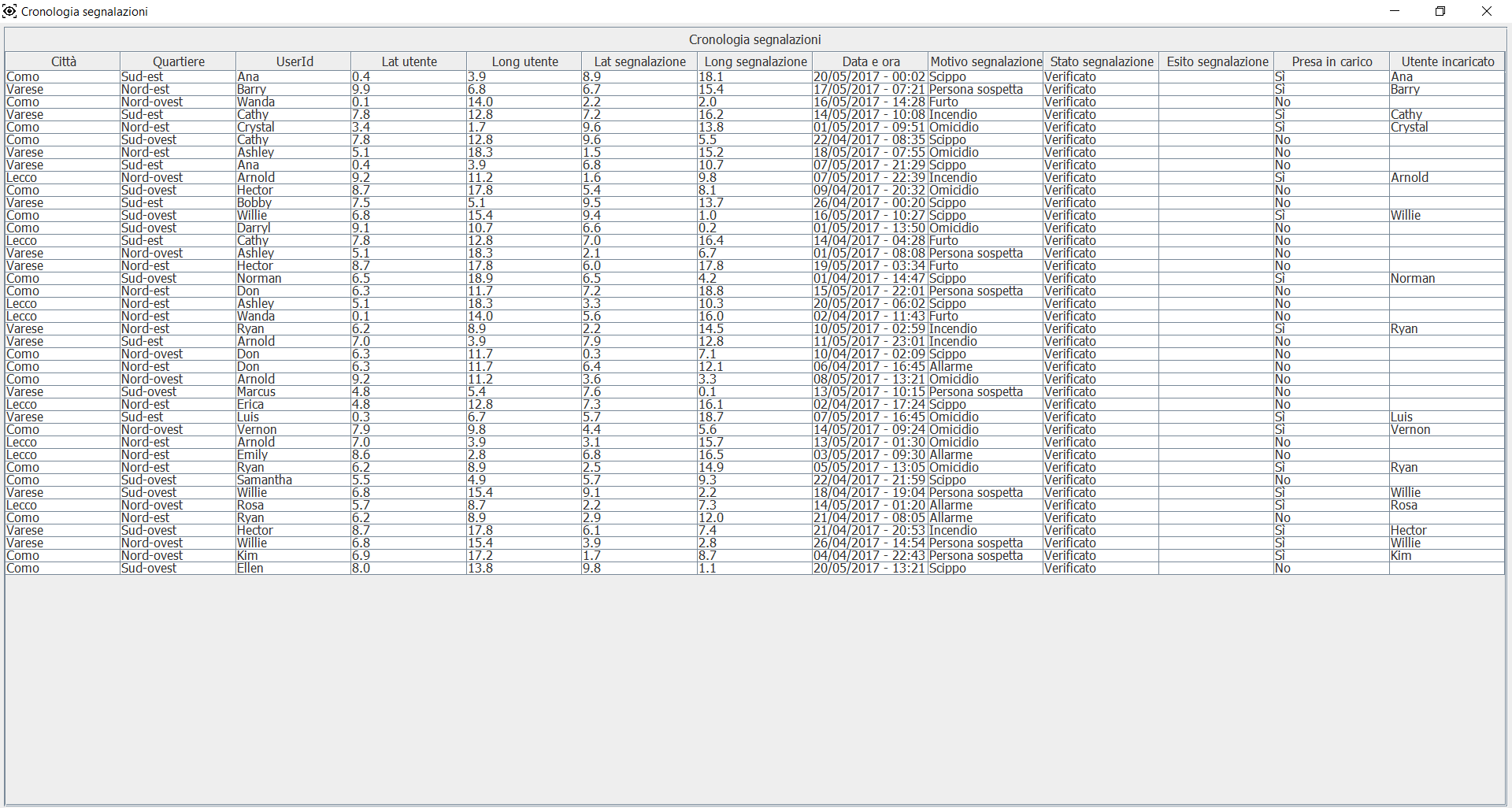
• in “File” si trova l’opzione di effettuare il logout del nostro profilo-utente oppure quella di uscire dal programma chiudendolo;

• in “Opzioni” si può accedere alla modifica o alla cancellazione del nostro profilo e alla cronologia delle segnalazioni chiuse;

• in “?” si accede a questa guida.

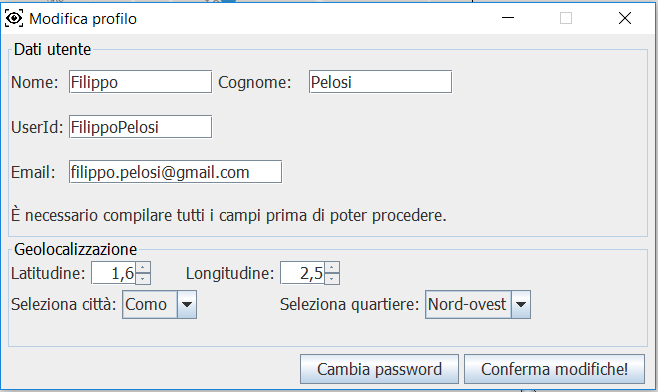
Sulla parte destra della finestra avremo dei campi da compilare riguardanti l’inserimento di una nuova segnalazione. Una volta confermata la segnalazione, essa verrà inserita all’interno della mappa. Si ha inoltre l’opzione di filtrare le segnalazioni presenti sulla mappa e sulla tabella delle segnalazioni attive (menzionata precedentemente). Infine, nella parte inferiore della finestra, si trova la tabella delle segnalazioni attive che ha le stesse funzioni citate precedentemente, con l’aggiunta della opzione di click con il tasto destro del mouse su una segnalazione, per accedere alla opzione di presa in carico o chiusura della segnalazione. L’opzione di presa in carico di una segnalazione è disponibile solo se essa non è già stata presa in carico, mentre l’opzione di chiusura della segnalazione solo all’utente che ha preso in carico o ha aperto la segnalazione.

**d) Cronologia delle segnalazioni**

****

Cliccando su cronologia segnalazioni, presente all’interno di “Opzioni”, si aprirà una nuova finestra contenente una tabella con le segnalazioni chiuse dagli utenti, con i relativi dati.

**e) Modifica profilo e password**

****

Cliccando su “Modifica profilo”, presente all’interno di “Opzioni”, si aprirà una nuova finestra che permetterà di accedere ai nostri dati personali e di modificarli. Se si è interessati solo alla modifica della propria password è sufficiente cliccare sull’apposito pulsante “Cambia password” che comporterà l’apertura di una nuova finestra in cui sarà possibile modificare i campi per inserire una nuova password.

**6.SUDDIVISIONE DEL LAVORO**

Filippo Pelosi si è occupato dei sistemi di lettura e scrittura del file .csv, dei metodi di utilità generale, della progettazione e dello sviluppo delle interfacce grafiche e ha scritto la relazione;

Marco di Capua si è occupato della progettazione e dello sviluppo del codice delle interfacce grafiche;

Riccardo Zorzi ha sviluppato gli enumerativi e logica dell’applicazione;

Tutti i componenti del gruppo hanno collaborato alla fase di progettazione del programma, alla realizzazione della javadoc e alle fasi di test.