ClimaScope
Manuale Tecnico

May 2023

Introduzione

ClimaScope è un un sistema di monitoraggio di parametri climatici fornito da centri di monitoraggio sul territorio italiano, in grado di rendere disponibili, ad operatori ambientali e comuni cittadini, i dati relativi alla propria zona di interesse.

Per un'introduzione riguardo le funzionalità del tool, si veda *Climascope: Manuale Utente*. Il codice sorgente è disponibile qui.

Dipendenze

Le librerie esterne che vengono usate nel progetto sono le seguenti:

- **Java Swing - JGoodies** per la gestione grafica;
- **opencsv** per leggere file di formato `.csv`;
- **json-simple** per leggere file di formato `.json`;
- **lucene-spatial**, per il calcolo delle distanze tra due punti su una sfera con la formula di Haversine.

Software Design

Struttura del database

I requisiti di progetto prevedono di usare dei file di testo come databases. Tali file sono collocati nella cartella `data` e sono:

- `climate_parameters.data`, che colleziona le informazioni sulle misurazioni dei parametri climatici per area di interesse; in particolare, questo file contiene una lista in formato `json` in cui ogni elemento ha la seguente struttura:

```
{
  "state": "state_name", /* string */
```

```
  "geoname_id": geoname_id, /* int */
  "ascii_name": "ascii_name", /* string */
  "wind": [], /* list of int */
  "humidity": [], /* list of int */
  "pressure": [], /* list of int */
  "temperature": [], /* list of int */
  "rainfall": [], /* list of int */
  "glaciers_alt": [], /* list of int */
  "glaciers_mass": [], /* list of int */
  "tot_measure": tot_measure, /*int */
  "notes": "notes", /* string */
  "date": [],  /* list of string */
  "center": [],  /* list of string */
  "who": [] /* list of string */
}
```

L'elemento i-esimo delle liste indica alla i-esima registrazione.

- `monitoring_centers.data`, che colleziona le informazioni sui centri di monitoraggio; anch'esso contiene una lista in formato `json`, i cui elementi hanno la seguente struttura:

```
{
    "name": "name", /* string */
    "address": "address", /* string */
    "monitored_areas": [] /* list of int */
}
```

L'elemento i-esimo delle liste indica alla i-esima registrazione.

- `monitoring_coordinates.csv`, colleziona le informazioni geografiche delle aree di interesse in formato *comma separated values*. Ogni riga del file assume i valori descritti nella prima riga:

```
ID;Name;Ascii_name;State Code;State;Coordinates
```

L'ultima colonna, corrispondente alle coordinate del luogo, segue la seguente convezione

```
LAT(signed), LONG(signed)
```

Ad esempio, una riga del file potrebbe essere:

```
3173435;Milan;Milan;IT;Italy;45.46427, 9.18951
```

- `registered_operators.data`, che registra le informazioni degli operatori registrati in una list in formato `json`, i cui elementi hanno la seguente struttura:

```
{
    "name": "name", /* string */
    "tax_code": "tax_code", /* string */
    "email": "email", /* string */
    "username": "username", /* string */
    "password": "pwd", /* string */
    "monitoring_center": "monitoring_center" /* string */
}
```

Nota: nel descrivere le strutture dei file si è usata la seguente convenzione: "`/* commento */`" per indicare un commento, non supportato in formato `json`. Tale convezione è stata introdotta solo per miglior comprensione del contenuto di ogni elemento del dizionario `json`.

Organizzazione dei file

Le componenti del software sono contenute nelle cartelle `src` (codice sorgente) e `lib` (librerie di terze parti). In particolare, la cartella `src/main/java/uni/climatemonitor` contiene il file `ClimateMonitor.java` che contiene il metodo `main` dell'intero progetto.

Tale cartella contiene inoltre altre sottocartelle secondo la seguente alberatura

```
climatemonitor
        |_ data
        |_ generics
        |_ graphics
```

Nella cartella `data` sono collezionate le classi necessarie per la lettura, la scrittura e la gestione dei dati:

- `CentersData`: contiene informazioni e metodi riguardo i centri di monitoraggio e gli operatori registrati;
- `ClimateParams`: è la classe che descrive i parametri climatici per una località;
- `Coordinates`: implementa dei metodi per gestire le coordinate di una località;
- `GeoData`: contiene informazioni e metodi riguardo le registrazioni dei parametri climatici e riguardo le località;
- `Location`: descrive una località;
- `MonitoringCenter`: descrive un centro di monitoraggio;
- `Operator`: descrive un operatore registrato;
- `FileHandler`: gestione file, con le sottoclassi
    * `ClimateParametersFileHandler`

    * `LocationsFileHandler`
    * `MonitoringCentersFileHandler`
    * `OperatorsFileHandler`

Nella cartella `generics` è contenuta una singola classe, usata per la definizione delle costanti nel progetto.
Nella cartella `graphics` sono infine contenute le classi relative alla grafica, che implementano la logica necessaria per il funzionamento della GUI:

- `AboutDialog`: implementa un popup che contiene le informazioni riguardo il progetto;
- `DetailsPage`: implementa la pagina di dettaglio;
- `MainPage`: implementa la pagina principale;
- `MainWindow`: implementa la finestra principale che contiene pagina di dettaglio e pagina principale;
- `NewArea`: implementa il popup che permette di creare una nuova località;
- `UtilsSingleton`: descritta nella prossima sezione.

Gestione della grafica

Ogni classe nella cartella `graphics` controlla una parte di grafica del programma. La classe `MainWindow` è un *container* mutualmente esclusivo che può contenere o la pagina principale (implementata in `MainPage`) o la pagina di dettaglio (implementata in `DetailsPage`). La classe `UtilsSingleton` è un singleton[1] che viene usato per rendere disponibili a tutte le altre classi: (i) alcune informazioni grafiche (ad esempio quale pagina è attiva nel container mutualmente esclusivo), un metodo per cambiare pagina visibile (`switchPage()`); (ii) metodi e informazioni per il controllo dell'accesso (tra cui, ad esempio `giveAccessTo()` e `getWhoisLoggedIn()`); (iii) due attributi (`geoData` e `centersData`) per la gestione di operatori, centri di monitoraggio, località e parametri climatici.

Le classi `AboutDialog` e `NewArea` sono invece le implementazioni dei popup che si aprono cliccando sul tasto "About" e sul tasto "Add Area" (si veda *ClimaScope: Manuale Utente* per altri dettagli su tale funzionalità).

L'approccio *event-driven* per la parte grafica prevede di associare ad ogni evento una *callback* che possa essere eseguita ogni volta che l'evento scatenante accade. Descriveremo quindi nella prossima sezione le principali funzionalità, associando l'evento che le attiva agli algoritmi che vengono eseguiti.

[1] L'uso del singleton nel framework Java Swing è ritenuto un metodo efficace per rendere disponibili informazioni e metodi alle altre classi: *Swing or other client-side UIs are another case where singletons may be a valid choice. In the case of a Java desktop app, it is likely that you have more complete control over your deployment environment than you do in a server-side application. In this case, it can be acceptable to have a manager class managing some state as a singleton in the VM.* (`https://puredanger.github.io/tech.puredanger.com/2007/07/03/pattern-hate-singleton/`)

Implementazione delle funzionalità principali

Inizializzazione dei dati

I dati riguardanti le aree di interesse contenuti nel file `monitoring_coordinates.csv` e i dati riguardanti i parametri climatici contenuti nel file `climate_parameters.data` devono essere disponibili all'uso dall'avvio del programma. Per soddisfare questo requisito, all'avvio del programma viene istanziato il singleton della classe `UtilsSingleton`, che nel suo costruttore istanzia l'oggetto `geoData` di tipo `GeoData`. Il costruttore di tale oggetto legge i due file tramite i rispettivi file handler e salva il loro contenuto in due `ArrayList`: un `ArrayList<Location>` per collezionare tutte le aree di interesse e un `ArrayList<ClimateParams>` per i parametri climatici.

Allo stesso modo, all'avvio del programma, i dati riguardanti le aree di interesse contenuti nel file `monitoring_centers.data` e i dati riguardanti i parametri climatici contenuti nel file `registered_operators.data` devono essere disponibili. Come sopra, il singleton di tipo `UtilsSingleton` istanzia l'attributo `centersData` di tipo `CentersData`. Il costruttore di tale oggetto legge i due file tramite i rispettivi file handler e salva il loro contenuto in due `ArrayList`: un `ArrayList<Operator>` per collezionare gli operatori registrati e un `ArrayList<MonitoringCenter>` per i centri di monitoraggio.

Ricerca località

L'interfaccia grafica mette a disposizione dell'utente una barra di ricerca per la ricerca delle aree di interesse. Questa deve ammettere la modalità di ricerca per nome e per stato di appartenenza e deve mostrare i risultati in una lista di suggerimenti.

Per soddisfare questo requisito, per ogni lettera inserita nella barra di ricerca viene chiamata una funzione di filtro per nome, che controlla che il nome (case sensitive e nel seguente formato: "name, country_code") sia contenuto in una struttura d'appoggio. Alla prima lettera inserita, tale struttura coincide con la lista `ArrayList<Location>` instanziata al momento della lettura del file delle aree di interesse come descritto nella sezione Inizializzazione dei dati, perciò il filtro dovrà controllare l'esistenza della prima lettera in ognuna degli N elementi dell'array. La struttura di appoggio viene quindi aggiornata in modo che solo gli elementi il cui nome contiene la lettera inserita siano presenti e vengano mostrati nella lista dei suggerimenti. Alla seconda lettera inserita, il numero di elementi che il filtro dovrà controllare è $\leq N$ e così via. Si noti che per la ricerca per stato d'appartenenza, basta cercare il *country code* relativo dello stato (case sensitive, es: Italia, IT).

La funzione di filtro di cui si è accennato sopra è la seguente:

```
/**
 * Filter by name
 * @param filter
 */
private void filterModel(String filter) {
    UtilsSingleton utils = UtilsSingleton.getInstance();
    for (Location l : utils.getGeoData().getGeoLocationsList()) {
        if (!l.toString().contains(filter)) {
            if (searchListModel.contains(l)) {
                searchListModel.removeElement(l);
            }
```

```
        } else {
            if (!searchListModel.contains(l)) {
                searchListModel.addElement(l);
            }
        }
    }
}
```

`searchListModel` è un oggetto di tipo `DefaultListModel<Location>` (nativo di Java Swing). La classe `DefaultListModel` permette di rimuovere o aggiungere elementi alla lista dei suggerimenti (che è visibile quando si inizia a scrivere nella barra di ricerca). Se la stringa inserita nella barra di ricerca dall'utente è contenuta nel nome dell'elemento i-esimo della lista delle località, tale elemento va aggiunto alla lista (se non è ancora presente); al contrario, l'elemento va rimosso (se presente).

Un'altra modalità di ricerca supportata deve essere quella per coordinate: come requisito è stato imposto che i risultati devono essere mostrati in ordine ascendente di distanza dalle coordinate inserite, entro un raggio di 50 km, in una lista di suggerimenti. La scelta del raggio entro cui rendere visibili gli elementi è una scelta di design.
Nel caso in cui venga inserita nella barra di ricerca una stringa con il seguente formato

```
LAT° N|S LONG° E|W
```

il codice rileva il pattern e chiama un filtro che utilizza per ogni elemento della lista `ArrayList<Location>` la formula di Haversine tra il punto cercato e le coordinate dell'elemento; la formula di Haversine è chiamata dal metodo `Coordinates.distance`. Colleziona dunque nella struttura d'appoggio l'elemento se tale distanza è minore di 50 km. In questo caso, per tutti gli N elementi della lista `ArrayList<Location>` devono essere calcolate le distanze. La struttura di appoggio è in questo caso una `HashMap`, che viene transformata in una `TreeMap` (che implementa un albero Red-Black). Essendo questa struttura un'implementazione di un albero binario di ricerca, è quindi infine possibile mostrare i dati ordinati all'utente. Il codice che implementa questo filtro è il seguente:

```
/**
 * Filter by coordinates
 * @param coordinates
 */
private void filterModelByCoordinates(Coordinates coordinates) {
    /* local variables */
    UtilsSingleton utils = UtilsSingleton.getInstance();

    double dist;
    Map<Double, Location> sortedMap;
    Map<Double, Location> unsortedMap = new HashMap<>();

    /* remove all elements just to be sure the list is empty */
    searchListModel.removeAllElements();

    for (Location l : utils.getGeoData().getGeoLocationsList()) {
        dist = coordinates.distance(l.getCoordinates());
        /* accept only those locations with a distance <= 50km */
```

```
            if (dist <= Constants.MAX_DIST) {
                unsortedMap.put(dist, l);
            }
        }
        /* get a sorted map by keys: keys are distances from the
           searched coordinates to the ones in the locations' file. */
        sortedMap = new TreeMap<Double, Location>(unsortedMap);
        /* at the end we can add to the suggestion list in the right
           order, from the nearest to the furthest */
        for (Map.Entry<Double, Location> entry : sortedMap.entrySet()){
            searchListModel.addElement(entry.getValue());
        }
}
```

Pagina di dettaglio: utenti non registrati

Dopo aver cliccato un elemento della lista dei suggerimenti nella modalità utente non registrato, l'interfaccia mostra la pagina di dettaglio dei parametri climatici dell'area di interesse scelta. Nella pagina di dettaglio sono visibili le informazioni riguardo l'ultimo record e una statistica delle misurazioni precedenti.

Come descritto nella sezione Software Design, nel file `climate_parameters.data` sono collezionati tutti i record relativi ad una qualsiasi area di interesse. Questi dati vengono letti e immagazzinati nella lista `climateParamsList: ArrayList<ClimateParams>`. Un oggetto di tipo `ClimateParams` ha come attributi i parametri climatici (come `ArrayList`): le nuove registrazioni vengono appese in testa all'array. In questo modo ogni parametro contiene le registrazioni in ordine cronologico, dalla più recente alla più vecchia.

Se la lista costituita dagli oggetti `ClimateParams` (di tipo `ArrayList<ClimateParams>`) contiene registrazioni per l'area di interesse selezionata, cioè se contiene un oggetto `cp: ClimateParams` tale che `cp.geonameID` è uguale all'ID dell'area selezionata, vengono mostrati i valori dell'ultima registrazione e la media aritmetica tra tutte le registrazioni per ogni parametro. In particolare si devono fare al più N confronti per controllare se l'oggetto ha registrazioni associate. Infatti, supponendo che la lista delle località `ArrayList<Location>` ha N elementi e sapendo che ad ogni località può essere associato solo un oggetto di tipo `ClimateParams` che contiene la storia delle registrazioni, la lunghezza massima della lista `ArrayList<ClimateParams>` è N.

Come specificato, la statistica è mostrata sotto forma di media aritmetica tra le varie misure; vengono mostrati dei sottolivelli a seconda della parte decimale della media, secondo la seguente regola:

```
[...]
/* set sublevels according to the decimal part */
if (isDecimalPartValid){
    String subLevel;
    if (decimalPart == 0) {
        subLevel = "";
    } else if (decimalPart < 10){
        subLevel = "RARELY";
```

```
    } else if (decimalPart < 40){
        subLevel = "OCCASIONALLY";
    } else if (decimalPart < 70) {
        subLevel = "FREQUENTLY";
    } else {
        subLevel = "NORMALLY";
    }
}
```

Nota: per rendere più coerenti i sottolivelli, è stata fatta una scelta di design per cui il livello 5 equivale al punteggio peggiore (CRITICAL) e il livello 1 equivale al punteggio migliore (EXCELLENT).
Nota 2: in eventuali future versioni sarebbe interessante pensare a parametri statistici di maggior interesse.

Procedura di registrazione

Nella pagina principale, all'attivazione del bottone "Register", compare un form di registrazione; i dati da inserire per registrarsi sono:

- nome e cognome
- codice fiscale
- e-mail
- username
- password
- centro di monitoraggio di afferenza, se presente

Se durante la procedura di registrazione il centro di monitoraggio di afferenza non è disponibile tra quelli già esistenti, viene attivata la modalità di creazione. In particolare vengono mostrati i campi necessari alla creazione:

- nome del centro;
- indirizzo;
- lista delle località seguite.

All'attivazione del bottone di conferma della registrazione, i dati riguardo il nuovo operatore e il suo centro di monitoraggio devono essere salvati nei relativi file. In particolare, quando un nuovo operatore si registra, bisogna aggiungerlo alla lista degli operatori e bisogna: 1) aggiungere un nuovo centro di monitoraggio alla lista dei centri, se è stato creato; 2) aggiungere l'operatore al centro di monitoraggio di appartenenza nella lista dei centri.
La procedura per soddisfare il requisito è quindi la seguente. Alla conferma:

1. aggiornare `operators: ArrayList<Operator>`, appendendo una nuova istanza di `Operator` che contiene le nuove informazioni dell'operatore appena registrato;

2. aggiornare `monitoringCenters: ArrayList<MonitoringCenter>`:
    * se il centro di monitoraggio è nuovo: creando una nuova istanza di `MonitoringCenter` che contenga le informazioni del nuovo utente;
    * se il centro di monitoraggio esiste: aggiornando l'elemento `MonitoringCenter` corrispondente al centro di monitoraggio del nuovo utente;
3. aggiornare `monitoring_centers.data` e `registered_operators.data` secondo la struttura presentata nella sezione Struttura del database.

Procedura di login

Quando il bottone di login viene attivato e un utente immette i suoi dati, la classe singleton `UtilsSingleton` si occupa di validare nome utente e password, cercando nella lista degli utenti registrati `operators: ArrayList<Operator>` se l'utente esiste. Se esiste e la password è corretta, la procedura di login finisce, altrimenti viene segnalato un errore. Quando la procedura finisce correttamente, lo stato corrente viene memorizzato nella classe singleton `UtilsSingleton`, in modo che qualsiasi sia in grado di avere le informazioni necessarie sull'utente loggato.

Procedura di logout

Attivato il bottone di logout, la classe singleton `UtilsSingleton` si occupa di ripristinare lo stato relativo all'utente attualmente loggato a `null`.

Pagina di dettaglio: operatori

Dopo aver cliccato un elemento della lista dei suggerimenti nella modalità utente registrato, l'interfaccia mostra la pagina di dettaglio dei parametri climatici dell'area di interesse scelta. In questo caso nella pagina di dettaglio l'operatore può registrare una misura dei parametri climatici, solo se la località selezionata è tra quelle seguite dal centro di monitoraggio di appartenenza dell'operatore.

La procedura da seguire quando un operatore loggato clicca su una località dalla lista dei suggerimenti è la seguente.
Al click su un elemento della lista dei suggerimenti:

1. controllare se l'operatore loggato appartiene ad un centro di monitoraggio che segue la località selezionata;
2. se è abilitato: aprire la pagina in modalità registrazione parametri;
3. altrimenti: aprirla in modalità utente normale[2].

La procedura da seguire per registrare nuovi parametri climatici è la seguente.
All'attivazione del bottone "Save":

1. aggiornare la lista dei parametri climatici:

[2]In realtà viene aggiunto un bottone alla pagina, che permette all'operatore di aggiungere la località a quelle seguite dal proprio centro di monitoraggio

    * se la località esiste già, trovarla e appendere in testa alle liste rappresentanti i parametri l'ultimo valore inserito;
    * se la località non esiste ancora, crearla e registrare i parametri;

2. aggiornare il file `climate_parameters.data` di conseguenza.

Aggiunta località esistente

Un operatore loggato ha la possibilità di aggiungere una località a quelle seguite dal proprio centro di interesse. In particolare, nella pagina di dettaglio di una località non ancora seguita dal centro di interesse a cui l'operatore appartiene verrà mostrato il bottone "Add". All'attivazione del bottone bisogna eseguire la seguente procedura:

1. aggiungere la località alla lista di località seguite dal centro di monitoraggio di appartenenza dell'operatore
2. aggiornare il file `monitoring_centers.data` di conseguenza

Creazione di una nuova località

Un operatore registrato può creare una nuova località usando la modalità "Add Area". In questo caso, verrà aggiornata la lista delle località e il rispettivo file. La procedura seguita è la seguente. Quando la creazione viene confermata:

1. se una località con le *stesse* coordinate esiste già, la procedura finisce con un errore;
2. al contrario, si aggiorna la lista delle località, appendendo un nuovo oggetto di tipo `Location`;
3. si aggiorna il file `monitoring_coordinates.csv` di conseguenza.