

PROGETTO LABORATORIO A
CLIMATE MONITORING
SVILUPPATO DA: BADROUS
GIORGIO, BIAVASCHI
RAFFAELE, BONACINA DAVIDE,
CASATI SIMONE.



MANUALE MANUALE MANUALE MANUALE TECNICO

INDICE

01

Introduzione progetto

(Librerie usate/

Ambiente di sviluppo

02

Struttura generale

delle classi usate/File

03

Descrizione classi usate

04

Descrizione classi con
database

05

Scelte algoritmiche

06

Formato File e Struttura dati

07

GUI

INTRODUZIONE

"**Climate Monitoring**" è un progetto sviluppato in ambito accademico durante il '**Laboratorio A e B**' al fine di dimostrare le capacità acquisite nel corso dei primi due anni di studi. Il progetto consiste in un **sistema di monitoraggio/salvataggio di parametri climatici** fornito da centri di monitoraggio gestiti da Operatori abilitati e in grado di essere mostrati anche ad utenti comuni.

Il progetto è stato sviluppato in **Java 17** usando l'interfaccia grafica già presente nelle librerie base di Java (**libreria Swing**). E' stato utilizzato per lo sviluppo l'IDE: **NetBeans 17**.

Il progetto è stato strutturato per **funzionare su più piattaforme possibili** (1), ed è stato testato direttamente su:

- **Windows 10 Pro / Windows 11** (Architettura x64)
- **MacOS** (Architettura x64 e ARM)

La **classi principali** del progetto sono:

- **Accesso**
- **AreaParametri**
- **CentroMonitoraggio**
- **ClimateMonitor** (classe main)
- **Home** (2)
- **Parametri**
- **Registrazione**
- **Arealnt**

Note sul progetto

1. Durante la progettazione del software è stato scelto di **non usare una libreria grafica esterna** per alleggerire il carico e permettere a **tutte le piattaforme con Java installato** (e quindi librerie base) di eseguirlo.
2. La classe '**Home**' costituisce la classe base della struttura grafica ed infatti da quest'ultima che viene poi ereditata la finestra dell'utente con privilegi (Operatore).



STRUTTURA

CLASSI

- Classe 'Accesso'
- Classe 'AreaParametri'
- Classe 'CentroMonitoraggi'
- Classe 'ClimateMonitor' (M)
- Classe Home
- Classe Parametri
- Classe Registrazione
- Classe AreaInt

FILE (DATA)

- File CentroMonitoraggio
- File CoordinateMonitoraggio
- File OperatoriRegistrati
- File ParametriClimatici

CLASSI

• Accesso

La classe **Accesso** svolge la funzione di **eseguire l'accesso alla modalità privilegiata** per gli utenti '**Operatori**' registrati all'interno del file '**OperatoriRegistrati.dat**'.

Gli utenti possono essere stati precedentemente salvati all'interno del file o inseriti successivamente mediante l'utilizzo della classe **Registrazione**.

Subito dopo la dichiarazione della Classe, viene dichiarato un'oggetto di 'Home' (nome 'hh') per **costruire una finestra speculare** alla finestra di base. Questo serve ad evitare di ricostruire una finestra da capo e ad **ereditare i metodi/attributi della finestra base**.

Il **costruttore parametrizzato** attiva quindi la finestra, **richiamando la classe principale, settando il titolo e la corretta dimensione in base al display utilizzato** (viene disabilitata la possibilità di ridimensionare la schermata).

Il metodo '**AccediActionPerformed**' al click del bottone '**Accedi**' **verifica la presenza delle credenziali richieste**, se ne manca una o entrambe **rilascia un messaggio d'errore** (tramite un **JOptionPane**) specificando l'errore, in caso di esito positivo prosegue invece richiamando il metodo '**Accedi**' per **eseguire l'accesso** (viene fatto all'interno di un costrutto '**Try-Catch**' per evitare l'innalzarsi di eccezioni).

Il metodo '**Accedi**', che implementa la gestione dell'eccezione '**IOException**' in caso di **errore durante la lettura del file**, permette l'accesso recuperati i parametri Username/Password dalle **TextField**. Nel metodo viene eseguita la lettura del file mediante il metodo '**FileReader**' (da libreria) e impostando il **corretto separatore** (che coincide con quello usato nella struttura dati). **Ad accesso eseguito** vengono mostrati sulla pagina 'Home' **gli elementi** (nello specifico, i bottoni) **da utente 'Operatore'** loggato. In caso di credenziali mancati nel file, viene mostrato l'errore di '**Credenziali errate**'.

```
public void accedi() throws IOException{
    /**
     * Imposto la linea e il lettore su valore 'nullo' iniziale
     */
    String line = null;
    FileReader in = null;
    try {
        /**
         * Imposto il lettore di riga con l'apposito separatore (dichiarato inizialmente)
         * Leggo dal file 'OperatororiRegistrati.dat'
         */
        in = new FileReader("data"+sep+"OperatoriRegistrati.dat");
```



CLASSI

• AreaParametri

La classe **AreaParametri** svolge la funzione di **visualizzare i parametri climatici** una volta scelta **una specifica località** e registrati all'interno del file **'ParametriClimatici.dati'**.

I parametri possono essere stati precedentemente salvati all'interno del file o inseriti successivamente mediante l'utilizzo della classe **Parametri**.

Subito dopo la dichiarazione della Classe, viene dichiarato un'oggetto di 'Home' (nome 'hh') per **costruire una finestra speculare** alla finestra di base. Questo serve ad evitare di ricostruire una finestra da capo, e ad **ereditare i metodi/attributi della finestra base**.

Il **costruttore parametrizzato** attiva quindi la finestra, **richiamando la classe principale, settando il titolo e la corretta dimensione in base al display utilizzato** (viene disabilitata la possibilità di ridimensionare la schermata).

Viene inoltre richiamato subito il metodo **'visualizzaParametriClimatici'** che svolge la funzione di effettiva estrazione dei parametri da file.

Il metodo **'visualizzaParametriClimatici'** che implementa la gestione dell'eccezione **'IOException'** (in questo caso sfruttando il costrutto **'Try-catch'**) in caso di errore durante la lettura del file, permette l'accesso ai parametri climatici presenti nel file. Nel metodo viene eseguita la lettura del file mediante il metodo **'FileReader'** (da libreria) e impostando il corretto separatore (che coincide con quello usato nella struttura dati). Una volta estratti i parametri vengono aggiunti all'interno di una **'Table'** per **semplificarne la visualizzazione**, inoltre **in caso di mancata presenza** dei parametri viene gestito l'errore mediante un **'JOptionPane'** con l'errore corrente. Viene richiamato il metodo di **'AddRowTable'** per l'aggiunta di righe.

```
* Verifica corrispondenza GeoID inserito con dati estratti dal file 'ParametriClimatici.dati'
*/
if(geo==Long.parseLong(param[0])){
    /**
     * Creazione tabella con 'Parametri Climatici' estratti dal file 'ParametriClimatici.dati'
     * Utilizzo metodo 'addRowTable' per creare le righe
     */
    addRowTable(new String[]{param[2],param[3],param[4],param[5],param[6],param[7],param[8]});
    noteArea.setText(param[9]); ck=true;
}
```



CLASSI

• CentroMonitoraggio

La classe **CentroMonitoraggio** svolge la funzione di **inserire i centri di monitoraggio, solo ad accesso eseguito** e utente **'Operatore'** e registrati all'interno del file **'CentroMonitoraggio.dat'**.

Subito dopo la dichiarazione della Classe, viene dichiarato un'oggetto di **'Home'** (nome **'hh'**) per **costruire una finestra speculare** alla finestra di base. Questo serve ad evitare di ricostruire una finestra da capo, e ad **ereditare i metodi/attributi della finestra base**.

Il **costruttore parametrizzato** attiva quindi la finestra, **richiamando la classe principale, settando il titolo e la corretta dimensione in base al display utilizzato** (viene disabilitata la possibilità di ridimensionare la schermata).

Il metodo **'inserisciActionPerformed'** al click del bottone **'Inserisci'** verifica la presenza delle informazioni richieste nelle **TextField**, se ne manca una o tutte **rilascia un messaggio d'errore** (tramite un **JOptionPane**) specificando l'errore. In caso di esito positivo prosegue invece richiamando il metodo **'registraCentroAree'** **per eseguire l'inserimento** (viene fatto all'interno di un costrutto **'Try-Catch'** per evitare l'innalzarsi di eccezioni).

Il metodo **'registraCentroAree'** che implementa la gestione dell'eccezione **'IOException'** in caso di **errore durante la lettura del file, permette l'inserimento delle informazioni del centro** appena inserite. Nel metodo viene eseguita la scrittura sia del file **'CentroMonitoraggio.dat'** e sia del file **'OperatoriRegistrati.dat'** mediante il metodo **'FileReader'** (da libreria) e impostando il corretto separatore (che coincide con quello usato nella struttura dati). Questo per **collegare il Centro inserito all'Operatore**.

```
/**
 * Imposto lo scrittore di riga con l'apposito separatore (dichiarato inizialmente)
 * Scrivo sul file 'CentroMonitoraggio.dat' e 'OperatoriRegistrati' per associare l'user
 */
FileWriter fw = new FileWriter("data"+sep+"CentroMonitoraggio.dat",true);
FileWriter fw2 = new FileWriter("data"+sep+"OperatoriRegistrati.dat",true);
/**
 * Inserisco ad uno ad uno i parametri forniti nel form dall'utente, nel file 'CentroMonitoraggio.dat'
 */
fw.write("\n");
fw.append(nomeCentro.getText()+sp);
fw.append(indirizzo.getText()+sp);
fw.append(area.getText()+s);
```



CLASSI

• ClimateMonitor

La classe **ClimateMonitor** costituisce la classe **main** del programma. Svolge due funzione principali: **creare la pagina di Home** all'avvio del programma e di settare la variabile globale '**sep**' che contiene il **separatore di Directory del sistema operativo in uso** (utilizzato poi all'interno dei metodi di lettura/scrittura).

Subito dopo la dichiarazione della Classe, viene dichiarata una variabile di nome '**sep**' che tramite il metodo '**File.separator**' **ricava il separatore di Directory di sistema**. Il **metodo main** attiva quindi la finestra principale **richiamando** la classe di tipo '**Home**' e creando un oggetto di nome '**a**'.

La **struttura** della finestra e i **metodi** vengono ripresi dalla classe '**Home**' che rappresenta la struttura base del programma.

```
/**
 * <strong>Classe Main</strong>
 */
/**
 * Richiamo origine progetto.
 */
package climatemonitoring;
/**
 * Richiamo Librerie di Java
 */
import java.io.File;
/**
 * @author 753546 Badrous Giorgio William
 * @author 753540 Casati Simone
 * @author 754772 Biavaschi Raffaele
 * @author 755531 Bonacina Davide
 * @version Software 1.0.0
 * @version JDK 17
 */
public class ClimateMonitor {
    /**
     * Dichiarazione variabile 'sep' (separatore) di tipo stringa, uso il metodo 'File.separator' per recuperarlo dal
     * sistema operativo corrente
     */
    static String sep = File.separator;
    /**
     * Metodo 'main' che crea la prima finestra del programma.
     * @param args di tipo String, unica accettato dal metodo 'main' per debug
     */
    public static void main(String[] args){
        Home a = new Home();
    }
}
```



CLASSI

• Home

La classe **Home** rappresenta la **struttura base della finestra** del programma, contiene gli **elementi grafici** e i **metodi principali** (che vengono poi ereditati).

Subito dopo la dichiarazione della Classe, vengono dichiarati i **parametri necessari all'autenticazione e registrazione** dell'utente '**Operatore**'.

Il **costruttore parametrizzato** attiva quindi la finestra, **richiamando la classe principale, settando il titolo e la corretta dimensione in base al display utilizzato** (viene disabilitata la possibilità di ridimensionare la schermata).

Il metodo '**nomeButtonActionPerformed**' al click del bottone '**nomeButton**' verifica la presenza del **nome** (della località) richiesto nelle **TextField**, se ne manca **rilascia un messaggio d'errore** (tramite un **JOptionPane**) specificando l'errore, in caso di esito positivo prosegue invece richiamando il metodo '**cercaAreaGeografica**' per eseguire la **ricerca tramite nome**.

Il metodo '**coordButtonActionPerformed**' al click del bottone '**coordButton**' verifica la presenza delle coordinate (latitudine/longitudine, della località) richieste nelle **TextField**, se ne manca una o entrambe rilascia un messaggio d'errore (tramite un **JOptionPane**) specificando l'errore, in caso di esito positivo prosegue invece richiamando il metodo '**cercaAreaGeografica**' per eseguire la **ricerca tramite coordinate**. Il metodo viene richiamato all'interno di un costrutto '**Try-catch**' per evitare l'innalzarsi di eccezioni **nel caso i parametri inseriti non siano numerici**.

Sono presenti inoltre i metodi:

- '**offsetSlideStateChanged**' che al cambio di stato ha il **compito di ricavare l'offset di ricerca** (nel caso venga eseguita una ricerca tramite coordinate) dallo slider '**offsetSlide**'
- '**accediActionPerformed**' che al click del pulsante richiama la classe '**Accesso**' per eseguire la **procedura d'accesso** come utente '**Operatore**'
- '**registratiActionPerformed**' che al click del pulsante richiama la classe '**Registrazione**' per eseguire la **procedura di registrazione** come utente '**Operatore**'
- '**logoutActionPerformed**' che al click del pulsante esegue la **procedura d'uscita** dalla modalità '**Operatore**' **nascondendo le funzioni riservate** (pulsanti non accessibili per gli utenti comuni) e **settando le variabili dell'utente 'Operatore'** sullo stato iniziale ('null')
- '**addCentroActionPerformed**' che al click del pulsante richiama la classe '**CentroMonitoraggio**' per eseguire la procedura d'inserimento degli '**Centri di Monitoraggio**', visibile solo come utente '**Operatore**'

CLASSI

- **'cancelActionPerformed'** che al click del pulsante ha il **compito di pulire la tabella dai risultati precedenti** di ricerca, inoltre imposta su vuoto le **TextField** e resetta la posizione dello **Slider** sulla posizione di partenza.
- **'addParamActionPerformed'** che al click del pulsante richiama la classe **'Parametri'** per **eseguire l'inserimento dei parametri climatici** come utente **'Operatore'**
- **'resTableMouseClicked'** che al click del pulsante richiama la classe **'AreaParametri'** per **visualizzare i parametri climatici** di una data località.
- **'nomeFieldKeyPressed'** implementazione **tramite tastiera** della ricerca con nome.
- **'lonFieldKeyPressed'** e **'latFieldKeyPressed'** implementazione **tramite tastiera** della ricerca con coordinate.
- **'offsetSlideKeyPressed'** implementazione **tramite pressione dello slider** della ricerca con coordinate.

Il metodo **'main'** **crea e rende visibile** all'avvio del programma la finestra principale.

Il metodo **'cercaAreaGeografica'** con parametro **'nome'**, che implementa la gestione dell'eccezione **'IOException'** (in questo caso sfruttando il costrutto **'Try-catch'**) in caso di **errore durante la lettura del file**, permette la ricerca della località tramite nome all'interno del file **'CoordinateMonitoraggio.dati'**. Nel metodo viene eseguita la lettura del file mediante il metodo **'FileReader'** (da libreria) e impostando il corretto separatore (che coincide con quello usato nella struttura dati). Una volta estratte, le località vengono aggiunte all'interno di una **'Table'** per semplificarne la visualizzazione, mediante l'utilizzo del metodo **'addRowTable'** per l'aggiunta delle righe.

Il metodo **'cercaAreaGeografica'** con parametri **'lat'**, **'lon'** e **'offset'**, che implementa la gestione dell'eccezione **'IOException'** (in questo caso sfruttando il costrutto **'Try-catch'**) in caso di **errore durante la lettura del file**, permette la ricerca della località tramite coordinate (e eventuale offset) all'interno del file **'CoordinateMonitoraggio.dati'**. Nel metodo viene eseguita la lettura del file mediante il metodo **'FileReader'** (da libreria) e impostando il corretto separatore (che coincide con quello usato nella struttura dati). Una volta estratte, le località vengono aggiunte all'interno di una **'Table'** per semplificarne la visualizzazione, mediante l'utilizzo del metodo **'addRowTable'** per l'aggiunta delle righe.

CLASSI

Parametri

La classe **Parametri** svolge la funzione di inserire i **parametri climatici** di una data località, solo ad accesso eseguito come utente '**Operatore**' e registrati all'interno del file '**ParametriClimatici.dat**'. Inoltre esegue la **conversione** dei parametri climatici **nei range di score prefissati** in fase di presentazione del progetto.

Il **costruttore parametrizzato** attiva quindi la finestra, **richiamando la classe principale, settando il titolo** e la **corretta dimensione in base al display** utilizzato (viene disabilitata la possibilità di ridimensionare la schermata).

Il metodo '**inserisciActionPerformed**' al click del bottone '**inserisci**' verifica **la presenza dei parametri richiesti** (della località) richiesto nelle TextField/DropDown, se ne manca uno o tutte rilascia un messaggio d'errore (tramite un JOptionPane) specificando l'errore, in caso di esito positivo prosegue invece eseguendo l'inserimento dei dati sul file '**ParametriClimatici.dat**'. Prima di essere inseriti però **i dati vengono convertiti in score** mediante l'utilizzo dei vari metodi, **ognuno specifico per il dato**.


Sono presenti inoltre i metodi:

- '**calcolaScoreVento**' metodo specifico per il **calcolo dello score del parametro Vento**, il range preso in esame è **tra 1 e 120** (minimo d'intensità/massimo d'intensità).
- '**calcolaScoreUmidita**' metodo specifico per il **calcolo dello score del parametro Umidità**, il range preso in esame è **tra 0 e 100** (minimo d'intensità/massimo d'intensità).
- '**calcolaScorePressione**' metodo specifico per il **calcolo dello score del parametro Pressione**, il range preso in esame è **tra 970 e 1047** (minimo d'intensità/massimo d'intensità).
- '**calcolaScoreTemperatura**' metodo specifico per il **calcolo dello score del parametro Temperatura**, il range preso in esame è **tra -30 e 45** (minimo di temperatura/massimo di temperatura).
- '**calcolaScorePrecipitazioni**' metodo specifico per il **calcolo dello score del parametro Precipitazioni**, il range preso in esame è **tra 1 e 12** (minimo livello pioggia/massimo livello pioggia).
- '**calcolaScoreAltitudineGhiacciai**' metodo specifico per il **calcolo dello score del parametro Altitudine dei Ghiacciai**, il range preso in esame è tra 0 e 1000 (minimo livello altezza/massimo livello altezza).
- '**calcolaScoreMassaGhiacciai**' metodo specifico per il **calcolo dello score del parametro della Massa dei Ghiacciai**, il range preso in esame è tra 0 e 1000 (minimo livello massa/massimo livello massa).

CLASSI

Il metodo '**centriDropItemStateChanged**' che al **cambio di stato** della DropDown '**CentriDrop**' importa dal file '**CentroMonitoraggio.dati**' i centri associati all'utente '**Operatore**' corrente, utili al fine di inserire i parametri climatici. Il metodo di ricerca implementa la gestione dell'eccezione '**IOException**' (in questo caso sfruttando il costrutto '**Try-catch**') in caso di **errore durante la lettura del file**.

```
try {  
    /**  
    * Imposto il lettore di riga con l'apposito separatore (dichiarato inizialmente)  
    * Leggo dal file 'CentroMonitoraggio.dati'  
    */  
    in = new FileReader("data"+sep+"CentroMonitoraggio.dati");  
    /**  
    * Buffer per la lettura  
    */  
    BufferedReader br = new BufferedReader(in);  
    String splitBy = ",";  
    /**  
    * Ciclo di lettura del file per ricerca corrispondenza valore, conclusione a riga 'nulla'  
    */  
    while ((line = br.readLine()) != null) {  
        if(ck){  
            String[] centro = line.split(splitBy);  
            String nomeCentro = centro[0];  
            //System.out.println("Nome: "+nomeCentro);  
            /**  
            * Verifica corrispondenza valore inserito con dati estratti dal file 'CentroMonitoraggio.dati'  
            */  
            if(NomeCentro.equals(nomeCentro)){  
                String aree[] = centro[2].split(split);  
                for(String s : aree){  
                    /**  
                    * Inserimento valore estratto dal file 'CentroMonitoraggio.dati' nella DropDown  
                    */  
                    areaDrop.addItem(s);  
                }  
            }  
        }  
    }  
}
```



CLASSI

• Registrazione

La classe **Registrazione** svolge la funzione di **eseguire la registrazione alla modalità privilegiata** per gli utenti '**Operatori**' registrati all'interno del file '**OperatoriRegistrati.dat**'.

Subito dopo la dichiarazione della Classe, viene dichiarato un oggetto di 'Home' (nome 'reg') per **costruire una finestra speculare** alla finestra di base. Questo serve ad evitare di ricostruire una finestra da capo, e ad **ereditare i metodi/attributi della finestra base**.

Il **costruttore parametrizzato** attiva quindi la finestra, **richiamando la classe principale, settando il titolo e la corretta dimensione in base al display utilizzato** (viene disabilitata la possibilità di ridimensionare la schermata).

Il metodo '**RegistratiActionPerformed**' al click del bottone '**Registrati**' **verifica la presenza delle credenziali richieste**, se ne manca una o tutte **rilascia un messaggio d'errore** (tramite un **JOptionPane**) specificando l'errore, in caso di esito positivo prosegue invece richiamando il metodo '**registrazione**' per **eseguire la registrazione** (viene fatto all'interno di un costrutto '**Try-Catch**' per evitare l'innalzarsi di eccezioni).

Il metodo '**registrazione**' che implementa la gestione dell'eccezione '**IOException**' in caso di **errore durante la lettura del file**, permette la registrazione dei parametri dalle **TextField**. Nel metodo viene eseguita la scrittura del file mediante il metodo '**FileWriter**' (da libreria) e impostando il **corretto separatore** (che coincide con quello usato nella struttura dati). Viene eseguita inoltre una lettura anche del file '**CentroMonitoraggio.dat**' per **eseguire l'associazione tra Operatore e Centro**.

Ad inserimento eseguito correttamente la finestra di registrazione si chiude, in caso di mancato inserimento viene mostrato invece su riga di comando **l'errore corrente**.

```
/**
 * Verifica corrispondenza tra selezione e valore estratto da file
 */
if(nome.equals(FileNome)){
    /**
     * Stampa su riga di comando usato in fase di debug, poi rimosso
     */
    //System.out.println("Siamo Arrivati 2 "+centri[3]);
    /**
     * Assegnazione IDCentro da valore di file
     */
    IDCentro = centri[3];
```



CLASSI

• Arealnt

La classe **Arealnt** svolge la funzione di **inserire una nuova area d'interesse, solo** per gli utenti '**Operatori**' registrati, all'interno del file '**CoordinateMonitoraggio.dati**'.

Subito dopo la dichiarazione della Classe, viene dichiarato un oggetto di 'Home' (nome 'reg') per **costruire una finestra speculare** alla finestra di base. Questo serve ad evitare di ricostruire una finestra da capo, e ad **ereditare i metodi/attributi della finestra base**.

Il **costruttore parametrizzato** attiva quindi la finestra, **richiamando la classe principale, settando il titolo e la corretta dimensione in base al display utilizzato** (viene disabilitata la possibilità di ridimensionare la schermata).

Il metodo '**inserisciActionPerformed**' al click del bottone '**inserisci**' **verifica la presenza delle credenziali richieste**, se ne manca una o tutte **rilascia un messaggio d'errore** (tramite un **JOptionPane**) specificando l'errore e il parametro mancante, in caso di esito positivo prosegue invece richiamando il metodo '**inserisciArealnt**' per **eseguire l'inserimento** (viene fatto all'interno di un costrutto '**Try-Catch**' per evitare l'innalzarsi di eccezioni).

Il metodo '**inserisciArealnt**' che implementa la gestione dell'eccezione '**IOException**' in caso di **errore durante la lettura del file**, permette la registrazione dei parametri della località (dalle **TextField**) all'interno del file, opportunamente convertite. Nel metodo viene eseguita la scrittura del file mediante il metodo '**FileWriter**' (da libreria) e impostando il **corretto separatore** (che coincide con quello usato nella struttura dati).

Il metodo '**toAscii**' che converte il **nome della località** in **formato ASCII**, sfruttando la funzione '**Normalizer**' da libreria.

```
private static String toAscii(String accented){
    final String normalized = Normalizer.normalize( accented, Normalizer.Form.NFD );
    StringBuilder sb = new StringBuilder( accented.length() );
    for ( int i = 0; i < normalized.length(); i++ )
    {
        char c = normalized.charAt( i );
        if ( Character.getType( c ) != Character.NON_SPACING_MARK )
        {
            sb.append(c);
        }
    }
    return sb.toString();
}
```



DATABASE

Il database utilizzato nel progetto è un database PostgreSQL. Viene utilizzato per memorizzare e gestire dati relativi a utenti, aree geografiche, parametri climatici, centri di monitoraggio e rapporti tra operatori e centri di monitoraggio.

Struttura del Database

Tabelle principali:

- **CoordinateMonitoraggio:** Memorizza le coordinate geografiche e le informazioni relative alle aree di monitoraggio.
- **ParametriClimatici:** Memorizza i parametri climatici raccolti dai centri di monitoraggio.
- **CentroMonitoraggio:** Memorizza i dettagli dei centri di monitoraggio.
- **Operatori:** Memorizza le informazioni sugli operatori che utilizzano il sistema.
- **Rapporti:** Memorizza le relazioni tra operatori e centri di monitoraggio.

Applicazione del Database nelle Classi

1. Home.java

Questa classe rappresenta la finestra principale dell'applicazione e fornisce funzionalità per la ricerca di aree geografiche per nome o coordinate.

Interazioni con il database:

Metodo `cercaAreaGeografica(String nome)`:

- Stabilisce una connessione con il database.
- Esegue una query SQL per cercare le aree geografiche che corrispondono al nome fornito.
- I risultati della query vengono aggiunti alla tabella della GUI per la visualizzazione.

Metodo `cercaAreaGeografica(double lat, double lon, int offset)`:

- Simile al metodo sopra, ma esegue una query basata su coordinate geografiche e un offset di distanza.

2. Parametri.java

Questa classe gestisce l'inserimento di nuovi parametri climatici associati a una specifica area e un centro di monitoraggio.

Interazioni con il database:

Metodo `inserisciActionPerformed(java.awt.event.ActionEvent evt)`:

- Recupera i dati inseriti dall'utente nella GUI.
- Stabilisce una connessione con il database.
- Inserisce i parametri climatici nel database.

Metodo `centroANDareaDropInitialize()`:

- Popola i dropdown `centriDrop` e `areaDrop` con i dati recuperati dal database.

3. Registrazione.java

Questa classe gestisce la registrazione di nuovi operatori nel sistema, inclusi i dettagli personali e l'associazione con un centro di monitoraggio.

Interazioni con il database:

Metodo reg():

- Recupera i dati inseriti dall'utente nella GUI.
- Stabilisce una connessione con il database.
- Inserisce i dati dell'operatore e l'associazione con un centro di monitoraggio nel database.

Metodo centriDropSelector():

- Popola il dropdown centriDrop con i nomi dei centri di monitoraggio recuperati dal database.

4. CentroMonitoraggio.java

Gestisce le informazioni relative ai centri di monitoraggio.

Interazioni con il database:

Metodo registraCentroAree():

- Inserisce nuovi centri di monitoraggio nel database.
- Utilizza query SQL per aggiungere dettagli del centro nel sistema.

5. AreaParametri.java

Visualizza i parametri climatici di specifiche aree geografiche.

Interazioni con il database:

Metodo visualizzaParametriClimatici():

- Recupera e visualizza i parametri climatici associati a un GeoNameID specifico.
- Esegue una query SQL per ottenere i dati climatici dal database e li mostra nella GUI.

6. Accesso.java

Questa classe gestisce l'accesso degli utenti al sistema "Climate Monitoring", permettendo loro di autenticarsi per accedere a funzionalità avanzate.

Interazioni con il database:

Metodo verificaCredenziali():

- Controlla le credenziali inserite dall'utente contro quelle memorizzate nel database.
- Esegue una query SQL per verificare se esiste un match tra username (o email) e password forniti dall'utente.
- Se le credenziali sono corrette, consente l'accesso all'utente e carica il profilo associato.

Metodo caricaProfiloUtente():

- Una volta verificate le credenziali, questo metodo recupera ulteriori informazioni sull'utente dal database.
- Esegue una query SQL per ottenere dettagli come il nome, cognome e altre preferenze utente che possono essere utilizzate per personalizzare l'interfaccia utente.

Queste classi dimostrano come il database viene utilizzato per gestire vari aspetti del progetto "Climate Monitoring", inclusi la ricerca di dati geografici, l'inserimento di nuovi parametri climatici e la gestione delle registrazioni degli operatori. Le interazioni con il database vengono gestite principalmente tramite JDBC, con query SQL preparate per garantire la sicurezza e l'integrità dei dati.

ESEMPIO

Il metodo `cercaAreaGeografica(String nome)` nella classe `Home` è progettato per cercare aree geografiche nel database basandosi sul nome fornito dall'utente. Ecco una spiegazione dettagliata del codice, passo dopo passo:

```
public void cercaAreaGeografica(String nome) {
    // Dichiarazione delle variabili per la gestione della connessione e l'esecuzione delle query.
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;

    try {
        // Stabilire una connessione con il database utilizzando i dati di accesso specificati.
        conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);

        // Preparazione della query SQL per cercare le aree geografiche che contengono il nome fornito.
        // Utilizza il placeholder '?' per inserire dinamicamente il nome nell'istruzione SQL.
        String sql = "SELECT GeoNameID, Name, CountryName, CountryCode FROM CoordinateMonitoraggio WHERE LOWER(Name) LIKE LOWER(?)";
        pstmt = conn.prepareStatement(sql);

        // Impostazione del parametro per la query SQL, convertendo il nome in lowercase e aggiungendo i wildcard '%' per la ricerca parziale.
        pstmt.setString(1, "%" + nome + "%");

        // Esecuzione della query e salvataggio dei risultati in un oggetto ResultSet.
        rs = pstmt.executeQuery();

        // Iterazione sui risultati ottenuti dalla query.
        while (rs.next()) {
            // Estrazione dei dati di ciascuna area geografica dal ResultSet.
            String id = rs.getString("GeoNameID");
            String nomeArea = rs.getString("Name");
            String nomeStato = rs.getString("CountryName");
            String codiceStato = rs.getString("CountryCode");

            // Aggiunta dei dati estratti alla tabella nella GUI per la visualizzazione.
            addRowTable(new String[]{id, nomeArea, nomeStato, codiceStato});
        }
    } catch (SQLException e) {
        // Gestione delle eccezioni SQL mostrando un messaggio di errore in una finestra di dialogo.
        JOptionPane.showMessageDialog(null, e.getMessage(), "Database Error!", JOptionPane.ERROR_MESSAGE);
    } finally {
        // Blocco finally per assicurarsi che tutte le risorse (ResultSet, PreparedStatement, Connection) vengano chiuse.
        try {
            if (rs != null) rs.close();
            if (pstmt != null) pstmt.close();
            if (conn != null) conn.close();
        } catch (SQLException e) {
            // Gestione delle eccezioni durante la chiusura delle risorse.
            JOptionPane.showMessageDialog(null, e.getMessage(), "Database Error!", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Dettagli Aggiuntivi:

Connection: La connessione al database è gestita tramite `DriverManager.getConnection()`, che accetta URL, nome utente e password del database. Questo garantisce che ogni operazione sul database sia eseguita attraverso una connessione

SCELTE ALGORITMICHE

In fase di progettazione si è scelto di semplificare il modello di sviluppo, **partendo da una grafica standard** presente nelle librerie di Java. Questo ci ha permesso di standardizzare gli elementi grafici e **gli algoritmi presenti nel background per la gestione dei dati** ricavati da questi elementi. **È infatti comune trovare all'interno delle classi elementi simili di gestione**, ad esempio in 'Accedi' e 'Registrati' l'algoritmo di controllo dei parametri inseriti, al fine della verifica della mancanza di uno/tutti/o alcuni di questi, è uguale (cambia solo il numero di parametri verificati).

Un altro algoritmo di rilievo è la **conversione dei parametri climatici in range**, che sono stati schematizzati in base a dei parametri tabellari d'intensità/quantità.



```
public static int calcolaScoreVento(int vento) {  
    int score_vento=0;  
    if (vento >= 1 && vento <= 10) {  
        score_vento=1;  
    } else if (vento <= 20) {  
        score_vento=2;  
    } else if (vento <= 30) {  
        score_vento=3;  
    } else if (vento <= 60) {  
        score_vento=4;  
    } else if (vento <= 120) {  
        score_vento=5;  
    } else {  
        JOptionPane.showMessageDialog(null, "Valore inserito per il vento non valido!", "Errore!",  
JOptionPane.ERROR_MESSAGE);  
        ck=false;  
    }  
    return score_vento;  
}
```

STRUTTURA DATI

La **struttura dati** del progetto comprende **più file** (di tipo 'data') **ognuno con specifiche funzioni di archiviazione**. Si è scelto di usare come **separatore** tra i campi il carattere ';' (carattere speciale). I file utilizzati sono:

- **CentroMonitoraggio.dati**, che contiene i Centri di Monitoraggio (nome; indirizzo; area_interesse; id_centro)
- **CoordinateMonitoraggio.dati** (Geoname ID;Name;ASCII Name;Country Code;Country Name;Coordinates)
- **OperatoriRegistrati.dati** (nome; cognome; cod_fisc; email; userid; password; id_centro)
- **ParametriClimatici.dati** (Geoname ID;id_centro;Vento;Umidita;Pressione;Temperatura;Precipitazioni;AltitudineGhiacciai;MassaGhiacciai;Note)

Per rendere compatibile il metodo di lettura/scrittura di Java (FileReader/FileWriter) con Windows/Mac/Linux si è deciso di usare la '**Path**' di ricerca con il **seperatore di percorso di sistema** (tramite il metodo '**File.separator**').

ESEMPIO 'CENTROMONITORAGGIO.DATI'



Nome	INDIRIZZO	AREA INTERESSE	ID CENTRO
Pilfer	Via Melchiorre 48	Arese,Como	1
...
...

GUI

La GUI (Graphical User Interface) del progetto "Climate Monitoring" è progettata per fornire un'interfaccia utente interattiva e visivamente accattivante che facilita il monitoraggio e la gestione dei dati climatici. La GUI è costruita utilizzando il framework Swing di Java, che offre una vasta gamma di componenti per la creazione di applicazioni desktop. Ecco alcuni dettagli **chiave** sulla GUI di questo progetto:

1. Struttura e Layout

Multi-finestra: Il progetto utilizza diverse finestre (JDialog e JFrame) per separare le funzionalità, come la registrazione degli utenti, l'accesso, la visualizzazione e l'inserimento dei dati climatici.

Organizzazione chiara: Ogni finestra ha sezioni ben definite per l'inserimento dei dati, la visualizzazione dei risultati e la navigazione, migliorando l'usabilità per l'utente.

2. Componenti Chiave

JTable: Utilizzato per visualizzare i dati climatici e altre informazioni in forma tabellare, permettendo agli utenti di vedere chiaramente vari parametri in una sola vista.

JComboBox: Usato per selezionare opzioni da un elenco dropdown, come nella selezione di aree geografiche o centri di monitoraggio, facilitando l'interazione con il database.

JButton: Pulsanti per eseguire azioni come la ricerca, l'inserimento dei dati, e il logout.

JTextField e JPasswordField: Per l'inserimento di testo, come nome utente, password, e parametri di ricerca.

JLabel: Etichette che forniscono informazioni o istruzioni all'utente, migliorando la comprensione delle varie sezioni dell'interfaccia.

3. Interattività

Eventi e Listener: La GUI reagisce agli input degli utenti attraverso l'uso di listener per eventi, come azioni su pulsanti o selezioni da combobox, permettendo una dinamica interazione con il database.

Feedback visivo: L'interfaccia fornisce risposte immediate alle azioni degli utenti, come conferme di successo, avvisi di errori, o aggiornamenti dei dati visualizzati.

4. Accessibilità e Usabilità

Design intuitivo: La GUI è progettata per essere intuitiva, con controlli facilmente riconoscibili e un layout logico che segue le convenzioni comuni di UI design.

Gestione degli errori: La GUI gestisce gli errori in modo efficace, mostrando messaggi di errore che aiutano gli utenti a risolvere i problemi durante l'interazione.

5. Estetica e Personalizzazione

Stile e Tema: L'interfaccia utilizza un tema coeso che è piacevole esteticamente e offre una consistente esperienza utente attraverso le diverse finestre e pannelli.

Icone e Colori: Utilizza icone descrittive e un schema di colori ben scelto per migliorare la navigazione e l'esperienza complessiva.

6. Scalabilità

Modularità: La GUI è costruita in modo modulare, permettendo facile estensione o modifica di componenti specifici senza influenzare il resto dell'interfaccia.



CLIMATE MONITORING

Università degli Studi dell'Insubria – Laurea Triennale in Informatica

Crediti informazioni:

- *Programma: Climate Monitoring*
- *<https://docs.oracle.com/javase/8/docs/api/>*
- *<https://netbeans.apache.org/wiki/index.html>*