

VER. 1.0.0
27/05/2023

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



MANUALE MANUALE MANUALE MANUALE TECNICO

INDICE CONTENUTI

01

Introduzione progetto

(Librerie usate/Ambiente di sviluppo)

02

Struttura generale delle classi usate/File

03

Descrizione classi usate

04

Scelte algoritmiche

05

Formato File e Struttura dati

INTRODUZIONE

"**Climate Monitoring**" è un progetto sviluppato in ambito accademico durante il '**Laboratorio A**' al fine di dimostrare le capacità acquisite nel corso di 'Programmazione Java' (3).

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).
3. Nel corso di 'Programmazione Java' non è stata affrontata la parte grafica (di GUI) per questo nelle successive slide di presentazione non verrà trattata nello specifico la parte grafica delle Classi create.



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 'OperatoriRegistrati.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.dati'**.

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.dati'** e sia del file **'OperatoriRegistrati.dati'** mediante il metodo **'FileWriter'** (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.dati' e 'OperatoriRegistrati' per associare l'user
 */
FileWriter fw = new FileWriter("data"+sep+"CentroMonitoraggio.dati",true);
FileWriter fw2 = new FileWriter("data"+sep+"OperatoriRegistrati.dati",true);
/**
 * Inserisco ad uno ad uno i parametri forniti nel form dall'utente, nel file 'CentroMonitoraggio.dati'
 */
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.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 **'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();
}
```



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 **separatore 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
...
...



CLIMATE MONITORING

Università degli Studi dell'Insubria – Laurea Triennale in Informatica

Crediti stesura: Raffaele Biavaschi

Crediti informazioni:

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