Laboratorio interdisciplinare B

Climate Monitoring

Un sistema di monitoraggio di parametri climatici

CATTANEO LUCA

FICARA PAOLO  
MAURI ANDREA

Indice

[Introduzione 5](#_Toc170830430)

[Requisiti per l’avvio 5](#_Toc170830431)

[Librerie esterne utilizzate 5](#_Toc170830432)

[Dipendenze 5](#_Toc170830433)

[Maven Embedder 5](#_Toc170830434)

[Plexus Utils 5](#_Toc170830435)

[Maven Wagon Provider API 5](#_Toc170830436)

[Maven Wagon File 6](#_Toc170830437)

[JUnit 6](#_Toc170830438)

[PostgreSQL JDBC Driver 6](#_Toc170830439)

[Codice Fiscale 6](#_Toc170830440)

[Maven Invoker 6](#_Toc170830441)

[SLF4J API 6](#_Toc170830442)

[SLF4J Simple 6](#_Toc170830443)

[Commons Compress 7](#_Toc170830444)

[Apache POI OOXML 7](#_Toc170830445)

[Commons CSV 7](#_Toc170830446)

[Plugin di Build 7](#_Toc170830447)

[Maven Compiler Plugin 7](#_Toc170830448)

[SQL Maven Plugin 7](#_Toc170830449)

[STRUTTURA DEL DATABASE 8](#_Toc170830450)

[TABELLE PRINCIPALI 8](#_Toc170830451)

[parametriclimatici 8](#_Toc170830452)

[coordinatemonitoraggio 8](#_Toc170830453)

[centromonitoraggio 8](#_Toc170830454)

[operatoreregistrato 9](#_Toc170830455)

[lavora 9](#_Toc170830456)

[RELAZIONI 9](#_Toc170830457)

[parametriclimatici - coordinatemonitoraggio: 9](#_Toc170830458)

[parametriclimatici - centromonitoraggio: 9](#_Toc170830459)

[operatoreregistrato - centromonitoraggio: 9](#_Toc170830460)

[lavora - coordinatemonitoraggio e centromonitoraggio: 9](#_Toc170830461)

[Struttura generale del sistema di classi 9](#_Toc170830462)

[Gestione dei dati e rappresentazioni oggetti 9](#_Toc170830463)

[Gestione dell’interfaccia grafica 10](#_Toc170830464)

[Gestione connessione client/server 10](#_Toc170830465)

[Classi per la gestione dei dati 10](#_Toc170830466)

[InterestingAreas 10](#_Toc170830467)

[Metodi principali della classe: 10](#_Toc170830468)

[MonitoringStation 10](#_Toc170830469)

[User 11](#_Toc170830470)

[Forecast 11](#_Toc170830471)

[Attributi della Classe 11](#_Toc170830472)

[Metodi principali della Classe 12](#_Toc170830473)

[DBManager 12](#_Toc170830474)

[Metodi principali della Classe 12](#_Toc170830475)

[DatiCondivisi 14](#_Toc170830476)

[Attributi della classe 14](#_Toc170830477)

[Metodi Principali della Classe 14](#_Toc170830478)

[Classi per la gestione dell’interfaccia grafica 16](#_Toc170830479)

[Casse Setting 16](#_Toc170830480)

[Funzionamento della Classe 16](#_Toc170830481)

[Interfaccia Grafica 16](#_Toc170830482)

[Impostazione Predefinita 16](#_Toc170830483)

[Connessione al Server 16](#_Toc170830484)

[Gestione della Finestra 16](#_Toc170830485)

[Avvio dell'Applicazione 16](#_Toc170830486)

[Classe Menu 16](#_Toc170830487)

[Descrizione delle Funzionalità 16](#_Toc170830488)

[Costruttori 16](#_Toc170830489)

[Metodi 17](#_Toc170830490)

[Gestione degli Eventi 17](#_Toc170830491)

[Variabili di Istanza 17](#_Toc170830492)

[Componenti Swing 17](#_Toc170830493)

[Flusso di Esecuzione 17](#_Toc170830494)

[Eccezioni Gestite 17](#_Toc170830495)

[Dettagli sui Componenti dell’Interfaccia 18](#_Toc170830496)

[Login 18](#_Toc170830497)

[Descrizione delle Funzionalità 18](#_Toc170830498)

[Costruttore 18](#_Toc170830499)

[Metodo grafica 18](#_Toc170830500)

[Inizializzazione dei Componenti 18](#_Toc170830501)

[Gestione degli Eventi 18](#_Toc170830502)

[Eccezioni 18](#_Toc170830503)

[Dettagli sui Componenti dell'Interfaccia 19](#_Toc170830504)

[Variabili di Istanza 19](#_Toc170830505)

[Metodi Privati 19](#_Toc170830506)

[Flusso di Esecuzione 19](#_Toc170830507)

[Eccezioni Gestite 19](#_Toc170830508)

[Register 19](#_Toc170830509)

[Descrizione delle Funzionalità 20](#_Toc170830510)

[Costruttore 20](#_Toc170830511)

[Metodo grafica 20](#_Toc170830512)

[Metodo createComboMonitoringStation 20](#_Toc170830513)

[Inizializzazione dei Componenti 20](#_Toc170830514)

[Gestione degli Eventi 20](#_Toc170830515)

[Metodi Privati 20](#_Toc170830516)

[Variabili di Istanza 21](#_Toc170830517)

[Flusso di Esecuzione 21](#_Toc170830518)

[Eccezioni Gestite 21](#_Toc170830519)

[Dettagli sui Componenti dell'Interfaccia 21](#_Toc170830520)

[AddNotes 22](#_Toc170830521)

[Descrizione delle Funzionalità 22](#_Toc170830522)

[Costruttore 22](#_Toc170830523)

[Metodo grafica 22](#_Toc170830524)

[Inizializzazione dei Componenti 22](#_Toc170830525)

[Gestione degli Eventi 22](#_Toc170830526)

[Metodi Privati 23](#_Toc170830527)

[Variabili di Istanza 23](#_Toc170830528)

[Flusso di Esecuzione 23](#_Toc170830529)

[Eccezioni Gestite 23](#_Toc170830530)

[Dettagli sui Componenti dell'Interfaccia 23](#_Toc170830531)

[CreateMonitoringStation 24](#_Toc170830532)

[Descrizione delle Funzionalità 24](#_Toc170830533)

[Costruttori 24](#_Toc170830534)

[Metodo grafica 24](#_Toc170830535)

[Inizializzazione dei Componenti 24](#_Toc170830536)

[Gestione degli Eventi 24](#_Toc170830537)

[Variabili di Istanza 25](#_Toc170830538)

[Componenti Swing 25](#_Toc170830539)

[Flusso di Esecuzione 25](#_Toc170830540)

[Inizializzazione: 25](#_Toc170830541)

[Interazione con l'Utente: 25](#_Toc170830542)

[Eccezioni Gestite 26](#_Toc170830543)

[Dettagli sui Componenti dell'Interfaccia 26](#_Toc170830544)

[Etichette (JLabel): 26](#_Toc170830545)

[Campi di Testo (JTextField): 26](#_Toc170830546)

[Menu a Tendina (JComboBox): 26](#_Toc170830547)

[Pulsanti (JButton): 26](#_Toc170830548)

[MenuOperatore 26](#_Toc170830549)

[Descrizione delle Funzionalità 26](#_Toc170830550)

[Costruttori 26](#_Toc170830551)

[Metodi 26](#_Toc170830552)

[Variabili di Istanza 27](#_Toc170830553)

[Componenti Swing 27](#_Toc170830554)

[Flusso di Esecuzione 27](#_Toc170830555)

[Eccezioni Gestite 27](#_Toc170830556)

[Dettagli sui Componenti dell’Interfaccia 27](#_Toc170830557)

[SearchResult 28](#_Toc170830558)

[Descrizione delle Funzionalità 28](#_Toc170830559)

[Costruttori 28](#_Toc170830560)

[Metodi 28](#_Toc170830561)

[Variabili di Istanza 28](#_Toc170830562)

[Componenti Swing 28](#_Toc170830563)

[Flusso di Esecuzione 28](#_Toc170830564)

[Eccezioni Gestite 28](#_Toc170830565)

[Dettagli sui Componenti dell’Interfaccia 29](#_Toc170830566)

[Classi per la gestione del sistema distribuito 29](#_Toc170830567)

[ServerMain 29](#_Toc170830568)

[Struttura della Classe 29](#_Toc170830569)

[Pacchetti Importati 29](#_Toc170830570)

[Dichiarazione della Classe 29](#_Toc170830571)

[Metodo Principale 29](#_Toc170830572)

[Metodi Implementati 29](#_Toc170830573)

[Metodi Privati 30](#_Toc170830574)

[Utilizzo 30](#_Toc170830575)

[ClientHandler 30](#_Toc170830576)

[Attributi della classe 30](#_Toc170830577)

[Metodi Principali della Classe 31](#_Toc170830578)

# Introduzione

Climate monitoring è un progetto sviluppato nell’ambito del progetto di Laboratorio Interdisciplinare A e poi in seguito aggiunte funzionalità nel Laboratorio Interdisciplinare B per il corso di laurea in Informatica dell’Università degli Studi dell’Insubria.

Il progetto è stato sviluppato in Java 17, usa un’interfaccia grafica costruita con Java Swing, libreria inclusa negli ambienti di sviluppo Java apposita per lo sviluppo del design delle applicazioni desktop, ed è stato sviluppato e testato su Windows 11.

# Librerie esterne utilizzate

Sono state utilizzate varie librerie esterne grazie alle possibilità offerte da maven. Nel file pom.xml sono contenute tutte le risorse utilizzate. Ecco un resoconto di tutto quello utilizzato.  
Dipendenze

## Dipendenze

### Maven Embedder

* **Gruppo**: org.apache.maven
* **Artifact**: maven-embedder
* **Versione**: 3.8.6
* **Descrizione**: Utilizzato per incorporare Maven in un'applicazione Java.

### Plexus Utils

* **Gruppo**: org.codehaus.plexus
* **Artifact**: plexus-utils
* **Versione**: 3.4.0
* **Descrizione**: Fornisce utilità comuni utilizzate da Maven e altri progetti Plexus.

### Maven Wagon Provider API

* **Gruppo**: org.apache.maven.wagon
* **Artifact**: wagon-provider-api
* **Versione**: 3.5.0
* **Descrizione**: API per il trasferimento di artefatti tra Maven e repository remoti.

### Maven Wagon File

* **Gruppo**: org.apache.maven.wagon
* **Artifact**: wagon-file
* **Versione**: 3.5.0
* **Descrizione**: Implementazione di Wagon che permette il trasferimento di file locali.

### JUnit

* **Gruppo**: junit
* **Artifact**: junit
* **Versione**: 3.8.1
* **Scope**: test
* **Descrizione**: Framework per il testing unitario in Java.

### PostgreSQL JDBC Driver

* **Gruppo**: org.postgresql
* **Artifact**: postgresql
* **Versione**: 42.7.3
* **Descrizione**: Driver JDBC per la connessione a database PostgreSQL.

### Codice Fiscale

* **Gruppo**: it.kamaladafrica
* **Artifact**: codice-fiscale
* **Versione**: 1.6.0
* **Descrizione**: Libreria per la gestione e la validazione del codice fiscale italiano.

### Maven Invoker

* **Gruppo**: org.apache.maven.shared
* **Artifact**: maven-invoker
* **Versione**: 3.2.0
* **Descrizione**: Fornisce funzionalità per eseguire progetti Maven.

### SLF4J API

* **Gruppo**: org.slf4j
* **Artifact**: slf4j-api
* **Versione**: 1.7.36
* **Descrizione**: API per il logging in Java, permette l'astrazione delle varie implementazioni di logging.

### SLF4J Simple

* **Gruppo**: org.slf4j
* **Artifact**: slf4j-simple
* **Versione**: 1.7.36
* **Descrizione**: Implementazione semplice di SLF4J per il logging.

### Commons Compress

* **Gruppo**: org.apache.commons
* **Artifact**: commons-compress
* **Versione**: 1.21
* **Descrizione**: Libreria per la compressione e decompressione di archivi come zip, tar, etc.

### Apache POI OOXML

* **Gruppo**: org.apache.poi
* **Artifact**: poi-ooxml
* **Versione**: 5.2.3
* **Descrizione**: Libreria per la manipolazione di documenti Office Open XML (Excel, Word, etc.).

### Commons CSV

* **Gruppo**: org.apache.commons
* **Artifact**: commons-csv
* **Versione**: 1.9.0
* **Descrizione**: Libreria per la lettura e scrittura di file CSV.

## Plugin di Build

### Maven Compiler Plugin

* **Gruppo**: org.apache.maven.plugins
* **Artifact**: maven-compiler-plugin
* **Versione**: 3.8.1
* **Configurazione**:
  + **source**: 1.8
  + **target**: 1.8
* **Descrizione**: Utilizzato per compilare il codice sorgente del progetto.

### SQL Maven Plugin

* **Gruppo**: org.codehaus.mojo
* **Artifact**: sql-maven-plugin
* **Versione**: 1.5
* **Descrizione**: Plugin per eseguire comandi SQL durante le varie fasi del ciclo di vita di Maven.
* **Configurazione**:
  + **driver**: org.postgresql.Driver
  + **url**: jdbc:postgresql://localhost:5432/
  + **username**: postgres
  + **password**: postgres
  + **settingsKey**: sensibleKey
* **Esecuzioni**:
  + **drop database**:
    - **Fase**: compile
    - **Comando SQL**: drop database if exists dbcm
  + **create database**:
    - **Fase**: compile
    - **Comando SQL**: create database dbcm
  + **create database schemas**:
    - **Fase**: compile
    - **File SQL**: src/main/sql/db.sql

# STRUTTURA DEL DATABASE

## CREAZIONE

La creazione del database avviene tramite Maven, cioè nel file pom.xml è rappresentata la dichiarazione di comandi sql per cancellare se esiste il db e di crearlo mediante la definizione dello schema da parte dell’invocazione di un file sql denominato db.sql. Questo fa in modo tale che ogni volta che tramite shell e uso del comando maven “mvn clean install” viene eseguito il file pom e questo creerà un database sul proprio server postgresql, nella porta 5432 e come utente ci sarà postgres con password postgres(questa password va modificata tramite shell di postegresql, vedasi ReadMe.txt). Una volta che la build del pom è stata eseguita, all’avvio del codice verranno inseriti, se non presenti, i record relativi alle aree di interesse tramite immissione da file csv.

## TABELLE PRINCIPALI

### parametriclimatici

Descrizione: Memorizza i dati climatici raccolti dai vari centri di monitoraggio.

Campi:

* idcitta: Identificativo della città (chiave primaria).
* nome\_centro: Nome del centro di monitoraggio.
* data: Data della registrazione dei parametri.
* ora: Ora della registrazione dei parametri.
* vento: Velocità del vento.
* umidita: Livello di umidità.
* pressione: Pressione atmosferica.
* temperatura: Temperatura.
* precipitazioni: Quantità di precipitazioni.
* altitudine: Altitudine.
* massa: Massa dei ghiacciai.

### coordinatemonitoraggio

Descrizione: Contiene le informazioni geografiche e identificative delle aree di interesse disponibili per i centri di monitoraggio.

Campi:

* id: Identificativo unico del centro (chiave primaria).
* name: Nome della città.
* name\_ascii: Nome della città in caratteri ASCII.
* country\_code: Codice del Paese.
* country\_name: Nome del Paese.
* lat: Latitudine.
* lon: Longitudine.

### centromonitoraggio

Descrizione: Memorizza informazioni sui centri di monitoraggio.

Campi:

* name: Nome del centro di monitoraggio (chiave primaria).
* address: Indirizzo del centro.

### operatoreregistrato

Descrizione: Contiene le informazioni degli operatori registrati che gestiscono i centri di monitoraggio.

Campi:

* cf: Codice fiscale dell'operatore (chiave primaria).
* nome: Nome dell'operatore.
* cognome: Cognome dell'operatore.
* mail: Indirizzo email dell'operatore.
* nick: Nome utente.
* password: Password per l'accesso.
* nome\_centro: Nome del centro di monitoraggio presso cui l'operatore lavora.

### lavora

Descrizione: Rappresenta la relazione N:N tra i centri di monitoraggio e le loro coordinate.

Campi:

id\_coordinate: Identificativo delle coordinate (chiave esterna verso coordinatemonitoraggio).

nome\_centro: Nome del centro di monitoraggio (chiave esterna verso centromonitoraggio).

## RELAZIONI

### parametriclimatici - coordinatemonitoraggio:

La tabella parametriclimatici è collegata alla tabella coordinatemonitoraggio tramite l'identificativo della città (idcitta).

### parametriclimatici - centromonitoraggio:

La tabella parametriclimatici è collegata alla tabella centromonitoraggio tramite il campo nome\_centro.

### operatoreregistrato - centromonitoraggio:

La tabella operatoreregistrato è collegata alla tabella centromonitoraggio tramite il campo nome\_centro.

### lavora - coordinatemonitoraggio e centromonitoraggio:

La tabella lavora stabilisce una relazione tra la tabella coordinatemonitoraggio e la tabella centromonitoraggio tramite i campi id\_coordinate e nome\_centro.

# Struttura generale del sistema di classi

Nel progetto si possono suddividere le classi in tre macro-rami: classi adibite per la gestione dei dati e la rappresentazione degli oggetti in questione, classi impiegate alla gestione dell’interfaccia grafica e classi per la gestione e scambio dati client/server.

## Gestione dei dati e rappresentazioni oggetti

* DatiCondivisi
* DBManager
* Forecast
* InterestingAreas
* MonitoringStation
* User
* ServerInterface

## Gestione dell’interfaccia grafica

* AddNotes
* CreateMonitoringStation
* Login
* Menu(main)
* MenuOperatore
* Register
* SearchResult

## Gestione connessione client/server

* ServerMain
* ClientHandler

Verranno ora presentate le classi adibite alla gestione dei dati dettagliamente, non verranno invece affrontate nel particolare le classi per la gestione dell’interfaccia grafica essendo che non sono espressamente richieste nei requisiti del progetto.

# Classi per la gestione dei dati

# InterestingAreas

InterestingAreas è la classe che rappresenta le aree di interesse ai quali si riferisce un utente.

L’oggetto che descrive l’area di interesse viene identificato con i seguenti attributi:

* Id: identificativo numerico univoco rappresentato dal tipo String
* name: nome dell’area di interesse rappresentato dal tipo String
* countryCode: il codice del paese dell’area di interesse rappresentato dal tipo String
* lat: la latitudine dell’area di interesse rappresentata dal tipo String
* lon: la longitudine dell’area di interesse rappresentata dal tipo String

## Metodi principali della classe:

1. Metodo contains(String s)
   * Descrizione: Verifica se il nome dell'area contiene una determinata stringa (ignorando maiuscole/minuscole).
   * Parametri:
     + s: la stringa da cercare
   * Complessità: O(n), dove 'n' è la lunghezza del nome dell'area

Nota: La complessità degli accessi e delle modifiche agli attributi è O(1), in quanto si tratta di operazioni dirette sugli attributi dell'oggetto.

# MonitoringStation

MonitoringStation è la classe che rappresenta le stazioni di monitoraggio che un operatore può creare.

L’oggetto che descrive la stazione di monitoraggio viene identificato con i seguenti attributi:

* name: il nome della stazione di monitoraggio rappresentato dal tipo String
* address: l’indirizzo della stazione di monitoraggio rappresentato dal tipo String
* interestingAreas: le aree di interesse della stazione di monitoraggio rappresentato da un vettore di String

Tutti i metodi hanno complessità O(1), poiché eseguono operazioni dirette sugli attributi dell'oggetto, senza iterazioni o algoritmi complessi.

# User

User è la classe che rappresenta un operatore.

L’oggetto che descrive un operatore viene identificato con i seguenti attributi:

* name: il nome dell’operatore rappresentato dal tipo String
* surname: il cognome dell’operatore rappresentato dal tipo String
* cf: il codice fiscale dell’operatore rappresentato dal tipo String
* mail: l’indirizzo mail dell’operatore rappresentato dal tipo String
* nick: il nickname dell’operatore rappresentato dal tipo String
* password: la password dell’operatore rappresentato dal tipo String
* station: la stazione di monitoraggio dell’operatore rappresentato dal tipo String

Nota: Tutti i metodi hanno complessità O(1), poiché eseguono operazioni dirette sugli attributi dell'oggetto, senza iterazioni o algoritmi complessi.

# Forecast

La classe Forecast rappresenta una previsione meteorologica associata a una specifica città e stazione di rilevamento. Questa classe include attributi per descrivere vari parametri meteorologici, oltre a metodi per impostare e ottenere questi parametri. La classe implementa l'interfaccia Serializable, permettendo quindi la serializzazione degli oggetti.

## Attributi della Classe

* idCitta: L'ID della città a cui si riferisce la previsione. Tipo: String
* NomeStazione: Il nome della stazione meteorologica. Tipo: String
* data: La data della previsione. Tipo: Date
* ora: L'ora della previsione. Tipo: Timestamp
* vento: I dati relativi al vento. Tipo: String[]
* umidita: I dati relativi all'umidità. Tipo: String[]
* pressione: I dati relativi alla pressione. Tipo: String[]
* temperatura: I dati relativi alla temperatura. Tipo: String[]
* precipitazioni: I dati relativi alle precipitazioni. Tipo: String[]
* altitudine: I dati relativi all'altitudine. Tipo: String[]
* massa: I dati relativi alla massa. Tipo: String[]

## Metodi principali della Classe

1. Metodo toQuery()
   * Descrizione: Restituisce una stringa da utilizzare come statement per la costruzione di una query per inserire i dati all’interno di un database.

Note: Accesso e Modifica degli Attributi: Gli attributi della classe possono essere accessi e modificati utilizzando i metodi getter e setter. La complessità di queste operazioni è O(1).

Rappresentazione per interfacciarsi al database: Il metodo toQuery consente di ottenere una stringa che rappresenta la previsione in un formato compatibile con l'inserimento in un database, facilitando l'integrazione con sistemi di archiviazione dati.

# DBManager

La classe DBManager è responsabile della gestione delle operazioni di lettura e scrittura sui dati del database relativi agli utenti, aree di interesse, stazioni di monitoraggio e previsioni climatiche. Utilizza connessioni JDBC per interagire con un database PostgreSQL.

## Metodi principali della Classe

1. readUser(Connection conn)

* Descrizione: Legge i dati dalla tabella operatoreregistrato e li trasforma in una lista di oggetti User.
* Parametri:
  + conn - Connessione al database.
* Ritorna: Una lista di oggetti User.
* Eccezioni:
  + SQLException - Errore durante l'esecuzione della query.
  + ClassNotFoundException - Classe non trovata.

1. readAreas(Connection conn)

* Descrizione: Legge i dati dalla tabella interesting\_areas e li trasforma in una lista di oggetti InterestingAreas.
* Parametri:
  + conn - Connessione al database.
* Ritorna: Una lista di oggetti InterestingAreas.
* Eccezioni:
  + SQLException - Errore durante l'esecuzione della query.

1. readStation(Connection conn)

* Descrizione: Legge i dati dalla tabella centromonitoraggio e li trasforma in una lista di oggetti MonitoringStation.
* Parametri:
  + conn - Connessione al database.
* Ritorna: Una lista di oggetti MonitoringStation.
* Eccezioni:
  + SQLException - Errore durante l'esecuzione della query.

1. readForecast(Connection conn)

* Descrizione: Legge i dati dalla tabella parametriclimatici e li trasforma in una lista di oggetti Forecast.
* Parametri:
  + conn - Connessione al database.
* Ritorna: Una lista di oggetti Forecast.
* Eccezioni:
  + SQLException - Errore durante l'esecuzione della query.

1. writeUser(User u, Connection conn)

* Descrizione: Scrive un oggetto User nel database.
* Parametri:
  + u - Oggetto User da scrivere.
  + conn - Connessione al database.
* Ritorna: Nessuno.
* Eccezioni:
  + SQLException - Errore durante l'esecuzione della query.
  + RemoteException - Errore di connessione remota.

1. writeForecast(Forecast f, Connection conn)

* Descrizione: Scrive un oggetto Forecast nel database.
* Parametri:
  + f - Oggetto Forecast da scrivere.
  + conn - Connessione al database.
* Ritorna: Nessuno.
* Eccezioni:
  + SQLException - Errore durante l'esecuzione della query.
  + ClassNotFoundException - Classe non trovata.

1. writeStation(MonitoringStation ms, Connection conn, List<String> areas)

* Descrizione: Scrive un oggetto MonitoringStation e le relative aree di interesse nel database.
* Parametri:
  + ms - Oggetto MonitoringStation da scrivere.
  + conn - Connessione al database.
  + areas - Lista di nomi delle aree di interesse.
* Ritorna: Nessuno.
* Eccezioni:
  + SQLException - Errore durante l'esecuzione della query.
  + ClassNotFoundException - Classe non trovata.
* Dettagli di Implementazione
* Connessioni al Database: La classe utilizza connessioni JDBC per eseguire le operazioni di lettura e scrittura sul database.
* PreparedStatement: Utilizzato per prevenire SQL Injection e migliorare le performance delle query.

1. insertAreas(Connection conn)

* Descrizione: Legge il file csv geonames-and-coordinates.csv e inserisce ogni record alla tabella coordinatemonitoraggio nel database rispettando la struttura della stessa.
* Parametri:
  + conn - Connessione al database.
* Ritorna: Nessuno.
* Eccezioni: Nessuna.
* Dettagli di Implementazione

Utilizzo di un BufferedReader per la lettura riga per riga del file considerato. Il passaggio alla cella successiva è dettata del riconoscimento del ;  
L’inserimento dei dati nel database è fatto tramite statement per garantire una sicurezza maggiore.

Note

Assicurarsi che le librerie JDBC appropriate siano incluse nel classpath del progetto. Le tabelle nel database devono esistere e avere la struttura corretta come previsto dai metodi. Gestire le eccezioni in modo appropriato per garantire la robustezza dell'applicazione. Questa documentazione fornisce una panoramica delle funzionalità offerte dalla classe DBManager e dei dettagli di implementazione per l'uso efficace e sicuro delle sue capacità di gestione del database.

# DatiCondivisi

La classe DatiCondivisi è progettata come un singleton per leggere, memorizzare e gestire i dati necessari per un'applicazione di monitoraggio climatico. Questa classe utilizza JDBC per connettersi a un database PostgreSQL e implementa UnicastRemoteObject per supportare operazioni RMI.

## Attributi della classe

instance: Singleton che rappresenta l'istanza unica della classe. Tipo: DatiCondivisi.

monitoringStations: Lista di oggetti MonitoringStation. Tipo: ArrayList<MonitoringStation>.

users: Lista di oggetti User. Tipo ArrayList<User>.

areas: Lista di oggetti InterestingAreas. Tipo: ArrayList<InterestingAreas>.

forecasts: Lista di oggetti Forecast. Tipo: ArrayList<Forecast>.

operatore: L'utente attualmente in sessione. Tipo: User.

conn: Connessione al database PostgreSQL. Tipo: Connection.

dBManager: Istanza della classe DBManager per gestire operazioni sul database. Tipo: DBManager.

JDBC\_DRIVER = "org.postgresql.Driver": Nome del driver JDBC. Tipo: String

## Metodi Principali della Classe

1. getInstance()
   * Descrizione: Restituisce l'unica istanza della classe DatiCondivisi, creando l'istanza se non esiste.
   * Parametri: Nessuno
   * Complessità: O(1)
2. refresh()
   * Descrizione: Ricarica i dati degli utenti, previsioni e stazioni di monitoraggio dal database.
   * Parametri: Nessuno
   * Complessità: O(n), dove n è la somma del numero di utenti, previsioni e stazioni di monitoraggio.
3. cercaAreaGeografica(String s)
   * Descrizione: Cerca l'area geografica interessata e restituisce un array di stringhe contenenti le città nell'area d'interesse.
   * Parametri:
     + s: la stringa che rappresenta il nome o le coordinate dell'area
   * Complessità: O(n), dove n è il numero di aree di interesse.
4. cercaLimitrofo(double lat1, double lon1)
   * Descrizione: Cerca le città vicine ad un determinato punto dato in latitudine e longitudine.
   * Parametri:
     + lat1: latitudine
     + lon1: longitudine
   * Complessità: O(n), dove n è il numero di aree di interesse.
5. existForecast(String area)
   * Descrizione: Controlla se una determinata area di interesse ha delle rilevazioni.
   * Parametri:
     + area: nome dell'area
   * Complessità: O(n), dove n è il numero di previsioni.
   * Eccezioni:
     + SQLException
6. convertNameToId(String name)
   * Descrizione: Converte il nome di una città nel suo ID corrispondente nel database.
   * Parametri:
     + name: nome della città
   * Complessità: O(1)
   * Eccezioni:
     + SQLException
7. sortAreas()
   * Descrizione: Ordina in modo crescente le aree di interesse.
   * Parametri: Nessuno
   * Complessità: O(n log n), dove n è il numero di aree di interesse.
8. writeForecast(Forecast f)
   * Descrizione: Scrive una previsione climatica nel database.
   * Parametri:
     + f: oggetto Forecast
   * Complessità: O(1)
   * Eccezioni: SQLException, ClassNotFoundException
9. writeUser(User u)
   * Descrizione: Scrive un utente nel database.
   * Parametri:
     + u: oggetto User
   * Complessità: O(1)
   * Eccezioni:
     + SQLException, RemoteException
10. writeStation(MonitoringStation ms, List<String> areas)
    * Descrizione: Scrive una stazione di monitoraggio nel database.
    * Parametri:
      + ms: oggetto MonitoringStation
      + areas: lista di aree di interesse
    * Complessità: O(1)
    * Eccezioni: SQLException, RemoteException, ClassNotFoundException

Nota

La complessità degli accessi e delle modifiche agli attributi è O(1), in quanto si tratta di operazioni dirette sugli attributi dell'oggetto.

# Classi per la gestione dell’interfaccia grafica

Tra le classi per la gestione dell’interfaccia grafica troviamo principalmente due tipologie: la classe main e le finestre modali che vengono rese visibili durante l’applicazione a seconda del tipo di operazione che un utente deve svolgere.

# Casse Settings

La classe Settings estende javax.swing.JFrame, fornendo così una finestra GUI per impostare l'IP e la porta del server.

## Funzionamento della Classe

### Interfaccia Grafica

La finestra Settings presenta un'interfaccia grafica semplice con etichette e campi di testo per l'IP e la porta del server, oltre a un pulsante di conferma.

### Impostazione Predefinita

Alla creazione della finestra, i campi di testo per l'IP e la porta del server vengono impostati con valori predefiniti. Se sono già state salvate configurazioni precedenti tramite ClientHandler, queste vengono caricate nei rispettivi campi.

### Connessione al Server

Quando l'utente preme il pulsante "Conferma", il metodo associato tenta di stabilire una nuova connessione al server con l'IP e la porta specificati. Se la connessione ha successo, viene aperta la finestra del menu principale; altrimenti, viene visualizzato un messaggio di errore.

### Gestione della Finestra

Il metodo grafica imposta un'icona per la finestra e la posiziona al centro dello schermo per una migliore esperienza utente.

### Avvio dell'Applicazione

Il metodo main imposta il look and feel dell'applicazione e avvia la finestra Settings.

# Classe Menu

La classe Menu estende javax.swing.JFrame e gestisce la finestra del menu principale dell’applicazione.

## Descrizione delle Funzionalità

### Costruttori

* Menu(): Crea l’oggetto della finestra del menu e inizializza tutti i suoi componenti. Configura la visibilità dei pulsanti in base all’operatore corrente.

### Metodi

* grafica(): Configura l’icona e la posizione della finestra.
* initComponents(): Inizializza e configura i componenti dell’interfaccia grafica, inclusi etichette, campi di testo e pulsanti.

## Gestione degli Eventi

* jButton2ActionPerformed(evt): Gestisce l’evento di pressione del pulsante per accedere alla finestra di login.
* jButton3ActionPerformed(evt): Gestisce l’evento di pressione del pulsante per accedere alla finestra di registrazione.
* jButton1ActionPerformed(evt): Gestisce l’evento di pressione del pulsante per cercare un’area geografica.
* jList1MouseClicked(evt): Gestisce l’evento di selezione di un elemento dalla lista dei risultati di ricerca.
* jButton4MouseClicked(evt): Gestisce l’evento di pressione del pulsante per accedere al menu dell’operatore.
* jButton5MouseClicked(evt): Gestisce l’evento di pressione del pulsante per disconnettere l’operatore corrente.
* jTextField1KeyTyped(evt): Gestisce l’evento di pressione del tasto invio nel campo di testo per la ricerca.

## Variabili di Istanza

Non ci sono variabili di istanza in questa classe.

## Componenti Swing

* Etichette (JLabel): jLabel1, jLabel3
* Campi di Testo (JTextField): jTextField1
* Liste (JList): jList1
* Pulsanti (JButton): jButton1, jButton2, jButton3, jButton4, jButton5

## Flusso di Esecuzione

1. La finestra del menu viene creata e i componenti vengono inizializzati.
2. L’icona e la posizione della finestra vengono configurate.
3. L’utente può cercare un’area geografica, accedere alla finestra di login o registrazione, o accedere al menu dell’operatore se è già connesso.

## Eccezioni Gestite

* ClassNotFoundException: Errore nel caricamento dei driver JDBC.
* SQLException: Errore nella connessione al database o nell’esecuzione della query.
* RemoteException: Errore nelle operazioni di rete.

## Dettagli sui Componenti dell’Interfaccia

* **Etichette (JLabel)**: Visualizzano il testo statico.
* **Campi di Testo (JTextField)**: Permette all’utente di inserire il nome dell’area geografica da cercare.
* **Liste (JList)**: Visualizza i risultati della ricerca di un’area geografica.
* **Pulsanti (JButton)**: Avviano il processo di ricerca, accedono alle finestre di login e registrazione, accedono al menu dell’operatore, o disconnettono l’operatore corrente.

# Login

La classe Login gestisce la finestra di autenticazione dell'applicazione, permettendo agli utenti di effettuare il login tramite e-mail e password.

## Descrizione delle Funzionalità

### Costruttore

* **Login()**:
  + Inizializza i componenti dell'interfaccia grafica.
  + Imposta l'icona e la posizione della finestra al centro dello schermo.
  + Carica la lista degli utenti tramite il metodo readUser() di ClientHandler.

### Metodo grafica

* Imposta l'icona della finestra e la posiziona al centro dello schermo.

### Inizializzazione dei Componenti

* **initComponents()**:
  + Inizializza i componenti dell'interfaccia grafica, inclusi etichette, campi di testo, pulsanti, e altri elementi.
  + Configura le proprietà di base della finestra, come il titolo, le dimensioni e il layout.

### Gestione degli Eventi

* **jButton1ActionPerformed**:
  + Gestisce l'evento di pressione del bottone "Accedi".
  + Verifica l'e-mail e la password inserite.
  + Se le credenziali sono valide, effettua l'autenticazione e apre la finestra del menu operatore.
  + Se le credenziali non sono valide, mostra un messaggio di errore.
* **jButton2ActionPerformed**:
  + Gestisce l'evento di pressione del bottone "Cancel".
  + Ritorna al menù principale senza effettuare il login.

## Eccezioni

* Il costruttore e i metodi che interagiscono con il database lanciano eccezioni di tipo:
  + ClassNotFoundException
  + SQLException
  + RemoteException

## Dettagli sui Componenti dell'Interfaccia

* **Etichette (JLabel)**:
  + Visualizzano il testo statico, come "E-Mail" e "Password".
* **Campi di Testo (JTextField, JPasswordField)**:
  + Permettono all'utente di inserire l'e-mail e la password.
* **Pulsanti (JButton)**:
  + "Accedi": avvia il processo di autenticazione.
  + "Cancel": annulla l'operazione di login e ritorna al menu principale.

## Variabili di Istanza

* **users**: Lista degli utenti caricati dal database, utilizzata per verificare le credenziali di login.

## Metodi Privati

* **grafica()**: Configura l'icona e la posizione della finestra.
* **initComponents()**: Inizializza e configura i componenti dell'interfaccia grafica.
* **jButton1ActionPerformed(evt)**: Gestisce l'evento di autenticazione.
* **jButton2ActionPerformed(evt)**: Gestisce l'evento di annullamento del login.

## Flusso di Esecuzione

1. **Inizializzazione**:
   * La finestra di login viene creata e i componenti vengono inizializzati.
   * L'icona e la posizione della finestra vengono impostate.
   * La lista degli utenti viene caricata dal server.
2. **Interazione con l'Utente**:
   * L'utente inserisce e-mail e password nei campi di testo.
   * Premendo "Accedi", il sistema verifica le credenziali:
     + Se valide, l'utente viene autenticato e viene aperta la finestra del menu operatore.
     + Se non valide, viene mostrato un messaggio di errore.
   * Premendo "Cancel", il sistema ritorna al menu principale senza effettuare il login.

## Eccezioni Gestite

* **ClassNotFoundException**: Errore nel caricamento dei driver JDBC.
* **SQLException**: Errore nella connessione al database o nell'esecuzione della query.
* **RemoteException**: Errore nelle operazioni di rete.

# Register

La classe Register gestisce la finestra di registrazione dell'applicazione, consentendo agli utenti di registrarsi fornendo i loro dati personali.

## Descrizione delle Funzionalità

### Costruttore

* **Register()**:
  + Inizializza i componenti dell'interfaccia grafica.
  + Imposta l'icona e la posizione della finestra al centro dello schermo.
  + Popola il menu a tendina delle stazioni di monitoraggio.
  + Inizializza il gestore del database (DBManager).

### Metodo grafica

* Imposta l'icona della finestra e la posiziona al centro dello schermo.

### Metodo createComboMonitoringStation

* Popola il menu a tendina delle stazioni di monitoraggio.

### Inizializzazione dei Componenti

* **initComponents()**:
  + Inizializza i componenti dell'interfaccia grafica, inclusi etichette, campi di testo, pulsanti, e altri elementi.
  + Configura le proprietà di base della finestra, come il titolo, le dimensioni e il layout.

## Gestione degli Eventi

* **jButton1ActionPerformed**:
  + Gestisce l'evento di pressione del bottone "Registrati".
  + Verifica i campi del modulo di registrazione tramite il metodo controlloCampi().
  + Se tutti i campi sono validi, crea un nuovo utente e lo salva nel database.
  + Se ci sono errori nei campi, mostra un messaggio di errore.
* **jButton2ActionPerformed**:
  + Gestisce l'evento di pressione del bottone "Cancel".
  + Ritorna al menù principale senza completare la registrazione.
* **txtDBirthFocusGained** e **txtDBirthFocusLost**:
  + Gestiscono il focus sul campo di testo della data di nascita, mostrando un suggerimento di formato.

## Metodi Privati

* **grafica()**: Configura l'icona e la posizione della finestra.
* **createComboMonitoringStation(monitoringStations)**: Popola il menu a tendina delle stazioni di monitoraggio.
* **initComponents()**: Inizializza e configura i componenti dell'interfaccia grafica.
* **jButton1ActionPerformed(evt)**: Gestisce l'evento di registrazione.
* **jButton2ActionPerformed(evt)**: Gestisce l'evento di annullamento della registrazione.
* **txtDBirthFocusGained(evt)**: Gestisce il focus acquisito sul campo di testo della data di nascita.
* **txtDBirthFocusLost(evt)**: Gestisce il focus perso sul campo di testo della data di nascita.
* **controlloCampi()**: Verifica la validità dei campi di registrazione e restituisce eventuali errori.

## Variabili di Istanza

* **DBManager dBManager**: Gestore del database per le operazioni di salvataggio degli utenti.
* **Componenti Swing**:
  + Etichette (JLabel): Per visualizzare il testo statico, come "Nome", "Cognome", "Email", ecc.
  + Campi di Testo (JTextField, JPasswordField): Per l'inserimento dei dati da parte dell'utente.
  + Pulsanti (JButton): "Registrati" e "Cancel" per gestire le azioni dell'utente.
  + Menu a Tendina (JComboBox<String>): Per selezionare la stazione di monitoraggio.
  + Gruppo di Pulsanti (ButtonGroup): Per gestire la selezione del sesso.

## Flusso di Esecuzione

1. **Inizializzazione**:
   * La finestra di registrazione viene creata e i componenti vengono inizializzati.
   * L'icona e la posizione della finestra vengono impostate.
   * Il menu a tendina delle stazioni di monitoraggio viene popolato.
   * Il gestore del database viene inizializzato.
2. **Interazione con l'Utente**:
   * L'utente inserisce i propri dati nei campi di testo.
   * Premendo "Registrati", il sistema verifica i campi tramite controlloCampi():
     + Se validi, crea un nuovo utente e lo salva nel database.
     + Se non validi, mostra un messaggio di errore.
   * Premendo "Cancel", il sistema ritorna al menu principale senza completare la registrazione.

## Eccezioni Gestite

* **ClassNotFoundException**: Errore nel caricamento dei driver JDBC.
* **SQLException**: Errore nella connessione al database o nell'esecuzione della query.
* **RemoteException**: Errore nelle operazioni di rete.
* **ParseException**: Errore nel parsing della data di nascita.

## Dettagli sui Componenti dell'Interfaccia

* **Etichette (JLabel)**:
  + Visualizzano il testo statico, come "Nome", "Cognome", "Data di nascita", "Comune Nascita", "Provincia", "Sesso", "Email", "NickName", "Password", "Centro di monitoraggio di afferenza".
* **Campi di Testo (JTextField, JPasswordField)**:
  + Permettono all'utente di inserire i propri dati personali.
* **Pulsanti (JButton)**:
  + "Registrati": Avvia il processo di registrazione.
  + "Cancel": Annulla l'operazione di registrazione e ritorna al menu principale.
* **Menu a Tendina (JComboBox)**:
  + Permette di selezionare la stazione di monitoraggio di afferenza.

# AddNotes

La classe AddNotes gestisce la finestra dell'applicazione che consente agli utenti di aggiungere note relative alle rilevazioni meteorologiche. Questa finestra permette l'inserimento di varie informazioni come vento, umidità, pressione, temperatura, precipitazioni, altitudine e massa dei ghiacciai.

## Descrizione delle Funzionalità

### Costruttore

#### AddNotes()

* Inizializza i componenti dell'interfaccia grafica tramite il metodo initComponents().

#### AddNotes(String idCitta, String nomeStazione, String date, String time, int wind, int humidity, int pressure, int temperature, int rainfall, int glacierAltitude, int massGlaciers) throws ParseException

* Inizializza i componenti dell'interfaccia grafica tramite il metodo initComponents().
* Chiama il metodo grafica() per configurare l'icona e la posizione della finestra.
* Crea un oggetto Forecast con i dati forniti.

### Metodo grafica

#### grafica()

* Configura l'icona della finestra.
* Imposta la posizione della finestra al centro dello schermo.

### Inizializzazione dei Componenti

#### initComponents()

* Inizializza i componenti dell'interfaccia grafica, inclusi etichette, campi di testo e pulsanti.
* Configura le proprietà di base della finestra, come il titolo, le dimensioni e il layout.

### Gestione degli Eventi

#### jButton1ActionPerformed(evt)

* Gestisce l'evento di pressione del pulsante "Inserisci la rilevazione".
* Aggiorna i dati di rilevazione nel campo temp con i valori inseriti dall'utente.
* Invoca il metodo remoto writeForecast(temp) per salvare i dati nel database.
* Passa al MenuOperatore e chiude la finestra corrente.

#### txtVentoActionPerformed(evt), txtUmiditaActionPerformed(evt), txtTempActionPerformed(evt), txtPrecActionPerformed(evt), txtPressActionPerformed(evt), txtAltActionPerformed(evt), txtMassActionPerformed(evt)

* Gestiscono l'evento di azione sui campi di testo.
* Limitano la lunghezza del testo inserito a 256 caratteri.

#### txtVentoKeyPressed(evt)

* Gestisce l'evento di pressione dei tasti sul campo di testo del vento.
* Limita la lunghezza del testo inserito a 256 caratteri.

### Metodi Privati

* grafica(): Configura l'icona e la posizione della finestra.
* initComponents(): Inizializza e configura i componenti dell'interfaccia grafica.
* jButton1ActionPerformed(evt): Gestisce l'evento di inserimento della rilevazione.
* txtVentoActionPerformed(evt), txtUmiditaActionPerformed(evt), txtTempActionPerformed(evt), txtPrecActionPerformed(evt), txtPressActionPerformed(evt), txtAltActionPerformed(evt), txtMassActionPerformed(evt): Gestiscono l'evento di azione sui campi di testo.

### Variabili di Istanza

* Forecast temp: Oggetto che rappresenta le previsioni meteorologiche.
* DBManager dBManager: Gestore del database per le operazioni di salvataggio delle rilevazioni.
* Componenti Swing:
  + Etichette (JLabel): lblTemp, lblPrec, lblAlt, lblUm, lblPress, lblMassa, jLabel1, jLabel2
  + Campi di Testo (JTextField): txtVento, txtUmidita, txtTemp, txtPrec, txtPress, txtAlt, txtMass
  + Pulsante (JButton): jButton1

## Flusso di Esecuzione

1. **Inizializzazione:**
   * La finestra di aggiunta note viene creata e i componenti vengono inizializzati.
   * L'icona e la posizione della finestra vengono impostate.
   * Se viene utilizzato il costruttore con parametri, viene creato un oggetto Forecast con i dati di rilevazione forniti.
2. **Interazione con l'Utente:**
   * L'utente inserisce i dati nei campi di testo.
   * Premendo "Inserisci la rilevazione", il sistema aggiorna i dati di rilevazione e li salva nel database tramite il metodo remoto writeForecast(temp).
   * Il sistema passa al MenuOperatore e chiude la finestra corrente.

## Eccezioni Gestite

* ParseException: Errore nella scrittura della data o dell'ora.
* RemoteException: Errore nelle operazioni di rete.
* SQLException: Errore nella connessione al database o nell'esecuzione della query.

## Dettagli sui Componenti dell'Interfaccia

* **Etichette (JLabel):**
  + Visualizzano il testo statico, come "Temperatura", "Precipitazioni", "Altitudine dei ghiacciai", "Vento", "Umidità", "Pressione", "Massa dei ghiacciai", "Inserisci una nota".
* **Campi di Testo (JTextField):**
  + txtVento, txtUmidita, txtTemp, txtPrec, txtPress, txtAlt, txtMass: Permettono all'utente di inserire i dati di rilevazione.
* **Pulsante (JButton):**
  + jButton1: Avvia il processo di inserimento della rilevazione.

# CreateMonitoringStation

La classe CreateMonitoringStation gestisce la finestra dell'applicazione che consente agli utenti di creare una nuova stazione di monitoraggio. Questa finestra permette l'inserimento del nome della stazione, dell'indirizzo e la selezione delle aree di monitoraggio.

## Descrizione delle Funzionalità

### Costruttori

* **CreateMonitoringStation():**
  + Crea l'oggetto della finestra dove verrà creata una nuova stazione di monitoraggio e inizializza tutti i suoi componenti.
  + Legge le aree di monitoraggio dal database e popola il menu a tendina delle aree.
  + Inizializza l'oggetto User.
  + Inizializza la lista delle aree.
* **CreateMonitoringStation(User u):**
  + Crea l'oggetto della finestra dove verrà creata una nuova stazione di monitoraggio e inizializza tutti i suoi componenti.
  + Legge le aree di monitoraggio dal database e popola il menu a tendina delle aree.
  + Inizializza l'oggetto User con le credenziali dell'operatore.
  + Inizializza la lista delle aree.

### Metodo grafica

* **grafica():**
  + Configura l'icona della finestra.
  + Imposta la posizione della finestra al centro dello schermo.

### Inizializzazione dei Componenti

* **initComponents():**
  + Inizializza e configura i componenti dell'interfaccia grafica, inclusi etichette, campi di testo e pulsanti.
  + Configura le proprietà di base della finestra, come il titolo, le dimensioni e il layout.
  + Popola il menu a tendina delle aree di monitoraggio.

## Gestione degli Eventi

* **buttonAddActionPerformed(evt):**
  + Gestisce l'evento di pressione del pulsante "Registrati".
  + Verifica che tutti i campi siano compilati.
  + Aggiorna le informazioni della stazione di monitoraggio nel database.
  + Passa al Menu e chiude la finestra corrente.
* **buttonAdd1ActionPerformed(evt):**
  + Gestisce l'evento di pressione del pulsante "Cancel".
  + Passa al Menu e chiude la finestra corrente.
* **btnAggiungiActionPerformed(evt):**
  + Gestisce l'evento di pressione del pulsante "Aggiungi area".
  + Aggiunge l'area selezionata alla lista delle aree di monitoraggio.
  + Visualizza un messaggio di conferma.

## Variabili di Istanza

* **User current:** Oggetto che rappresenta l'operatore corrente.
* **List<String> areas:** Lista delle aree di monitoraggio.
* **DBManager dBManager:** Gestore del database per le operazioni di salvataggio delle stazioni di monitoraggio.

## Componenti Swing

* **Etichette (JLabel):**
  + jLabel1, jLabel2, jLabel3, jLabel4
* **Campi di Testo (JTextField):**
  + name, address
* **Menu a Tendina (JComboBox):**
  + InterestingAreas
* **Pulsanti (JButton):**
  + buttonAdd, buttonAdd1, btnAggiungi

## Flusso di Esecuzione

### Inizializzazione:

1. La finestra per la creazione di una nuova stazione di monitoraggio viene creata e i componenti vengono inizializzati.
2. L'icona e la posizione della finestra vengono configurate.
3. Le aree di monitoraggio vengono lette dal database e popolate nel menu a tendina.

### Interazione con l'Utente:

1. L'utente inserisce il nome, l'indirizzo e seleziona le aree di monitoraggio.
2. Premendo "Registrati", il sistema verifica che tutti i campi siano compilati.
3. Se i campi sono compilati correttamente, le informazioni sulla stazione di monitoraggio vengono aggiornate nel database.
4. Il sistema passa al Menu e chiude la finestra corrente.
5. Premendo "Cancel", il sistema passa al Menu e chiude la finestra corrente.

## Eccezioni Gestite

* ClassNotFoundException: Errore nel caricamento dei driver JDBC.
* SQLException: Errore nella connessione al database o nell'esecuzione della query.
* RemoteException: Errore nelle operazioni di rete.

## Dettagli sui Componenti dell'Interfaccia

### Etichette (JLabel):

Visualizzano il testo statico, come "Nome", "Indirizzo", "Elenco aree" e "Registra centro aree".

### Campi di Testo (JTextField):

* name, address: Permettono all'utente di inserire il nome e l'indirizzo della stazione di monitoraggio.

### Menu a Tendina (JComboBox):

* InterestingAreas: Permette di selezionare le aree di monitoraggio.

### Pulsanti (JButton):

* buttonAdd: Avvia il processo di registrazione.
* buttonAdd1: Annulla l'operazione di registrazione e ritorna al Menu.
* btnAggiungi: Aggiunge l'area selezionata alla lista delle aree di monitoraggio.

# MenuOperatore

La classe MenuOperatore estende javax.swing.JFrame e gestisce la finestra dell’applicazione che consente agli operatori di visualizzare e aggiungere rilevazioni meteorologiche. Questa finestra permette la visualizzazione delle rilevazioni esistenti e l’inserimento di nuove rilevazioni.

## Descrizione delle Funzionalità

### Costruttori

* MenuOperatore(): Crea l’oggetto della finestra dove si andrà ad aggiungere una rilevazione, riempie la combobox con i relativi dati e popola la tabella con i dati salvati in precedenza. Inizializza l’oggetto DBManager.

### Metodi

* grafica(): Configura l’icona e la posizione della finestra.
* refreshTable(String area, String stazione): Aggiorna la tabella delle rilevazioni in base all’area e alla stazione specificate.
* createComboMonitoringStation(): Legge le stazioni di monitoraggio dal database e popola il menu a tendina delle stazioni.
* initComponents(): Inizializza e configura i componenti dell’interfaccia grafica, inclusi etichette, campi di testo e pulsanti.

## Variabili di Istanza

* List<Forecast> f: Lista delle previsioni.
* DBManager dBManager: Gestore del database per le operazioni di salvataggio delle rilevazioni.

## Componenti Swing

* Etichette (JLabel): lblWelcome, jLabel1, lblVento, lblUm, lblUmm, lblPress, lblPressi, lblTemp, lblTempe, lblPrec, lblPreci, lblAlt, lblAlti, lblMassa, lblMas, jLabel9
* Campi di Testo (JSlider): sldVento, sldUm, sldPres, sldTemp, sldPrec, sldAlt, sldMassa
* Menu a Tendina (JComboBox): cmbAreas
* Pulsanti (JButton): btnInsert, jButton1, jButton5
* Tabelle (JTable): tblRilevazioni

## Flusso di Esecuzione

1. La finestra per la visualizzazione e l’aggiunta di rilevazioni viene creata e i componenti vengono inizializzati.
2. L’icona e la posizione della finestra vengono configurate.
3. Le stazioni di monitoraggio vengono lette dal database e popolate nel menu a tendina.
4. L’utente può visualizzare le rilevazioni esistenti e aggiungere nuove rilevazioni.

## Eccezioni Gestite

* ClassNotFoundException: Errore nel caricamento dei driver JDBC.
* SQLException: Errore nella connessione al database o nell’esecuzione della query.
* RemoteException: Errore nelle operazioni di rete.

## Dettagli sui Componenti dell’Interfaccia

* **Etichette (JLabel)**: Visualizzano il testo statico, come “Benvenuto”, “Vento”, “Umidità”, “Pressione”, “Temperatura”, “Precipitazioni”, “Altitudine”, “Massa”.
* **Campi di Testo (JSlider)**: Permettono all’utente di inserire i valori delle rilevazioni.
* **Menu a Tendina (JComboBox)**: Permette di selezionare l’area di monitoraggio.
* **Pulsanti (JButton)**: Avviano il processo di inserimento di una nuova rilevazione o annullano l’operazione.
* **Tabelle (JTable)**: Visualizzano le rilevazioni esistenti.

# SearchResult

La classe SearchResult estende javax.swing.JFrame e gestisce la finestra dell’applicazione che consente agli operatori di visualizzare i risultati di una ricerca di previsioni meteorologiche.

## Descrizione delle Funzionalità

### Costruttori

* SearchResult(): Crea l’oggetto della finestra dei risultati di ricerca e inizializza tutti i suoi componenti.
* SearchResult(String areaName, Menu me): Crea l’oggetto della finestra dei risultati di ricerca con il nome dell’area di ricerca e il menu associato. Inizializza i componenti, configura la grafica e aggiorna la tabella con i risultati della ricerca.

### Metodi

* grafica(): Configura l’icona e la posizione della finestra.
* refreshTable(String areaName): Aggiorna la tabella dei risultati di ricerca in base al nome dell’area specificata.
* initComponents(): Inizializza e configura i componenti dell’interfaccia grafica, inclusi etichette, campi di testo e pulsanti.

## Variabili di Istanza

* List<Forecast> f: Lista delle previsioni.
* Menu m: Menu associato alla finestra dei risultati di ricerca.

## Componenti Swing

* Etichette (JLabel): Non presenti in questa classe.
* Campi di Testo (JTextField): Non presenti in questa classe.
* Menu a Tendina (JComboBox): Non presenti in questa classe.
* Pulsanti (JButton): jButton20
* Tabelle (JTable): tblRilevazioni

## Flusso di Esecuzione

1. La finestra per la visualizzazione dei risultati di ricerca viene creata e i componenti vengono inizializzati.
2. L’icona e la posizione della finestra vengono configurate.
3. I risultati della ricerca vengono letti dal database e popolati nella tabella.
4. L’utente può visualizzare i risultati della ricerca.

## Eccezioni Gestite

* ClassNotFoundException: Errore nel caricamento dei driver JDBC.
* SQLException: Errore nella connessione al database o nell’esecuzione della query.

## Dettagli sui Componenti dell’Interfaccia

* **Tabelle (JTable)**: Visualizza i risultati della ricerca. Ogni riga rappresenta una previsione meteorologica e ogni colonna rappresenta un attributo della previsione (data, ora, vento, umidità, pressione, temperatura, precipitazioni, altitudine dei ghiacciai, massa dei ghiacciai).
* **Pulsanti (JButton)**: Il pulsante “Torna alla home” permette all’utente di chiudere la finestra dei risultati di ricerca e tornare al menu principale.

# Classi per la gestione del sistema distribuito

# ServerMain

La classe ServerMain rappresenta il componente principale del server per un'applicazione di monitoraggio climatico. Implementa l'interfaccia ServerInterface e fornisce metodi per la gestione dei dati e la comunicazione remota tramite RMI (Remote Method Invocation).

## Struttura della Classe

### Pacchetti Importati

La classe importa diversi pacchetti per la gestione delle eccezioni, il logging, le operazioni di I/O, la gestione di processi, la normalizzazione delle stringhe e l'uso di Maven per la costruzione e gestione dei progetti.

### Dichiarazione della Classe

La classe ServerMain estende UnicastRemoteObject e implementa l'interfaccia ServerInterface. La porta di comunicazione RMI è definita come 1234. Il costruttore richiama l'eccezione RemoteException.

### Metodo Principale

Il metodo main avvia il server RMI.

### Metodi Implementati

#### Metodi di Lettura

* **readUser**: Restituisce una lista di utenti.
* **readAreas**: Restituisce una lista di aree di interesse.
* **readStation**: Restituisce una lista di stazioni di monitoraggio.
* **readForecast**: Restituisce una lista di previsioni.

#### Metodi di Ordinamento e Ricerca

* **sortAreas**: Ordina le aree di interesse.
* **cercaAreaGeografica**: Cerca un'area geografica basata su parametri specifici.

#### Metodi di Verifica

* **existForecast**: Verifica l'esistenza di una previsione basata su un parametro specifico.

#### Metodi di Gestione Operatore

* **setOperatore**: Imposta un operatore.
* **getOperatore**: Restituisce l'operatore corrente.

#### Metodi di Aggiornamento e Scrittura

* **refresh**: Aggiorna i dati condivisi.
* **writeForecast**: Scrive una previsione.
* **writeUser**: Scrive un utente.
* **writeStation**: Scrive una stazione di monitoraggio e le aree ad essa associate.

#### Metodi di Conversione e Normalizzazione

* **convertNameToId**: Converte un nome in un ID.
* **normalizeStrings**: Normalizza una stringa rimuovendo i segni diacritici.

## Utilizzo

Il ServerMain è progettato per essere eseguito come applicazione standalone. Il metodo main avvia il server.

I metodi della classe sono sincronizzati per garantire la sicurezza dei thread durante l'accesso ai dati condivisi. Gli utenti possono interagire con il server tramite chiamate RMI per leggere dati, eseguire ricerche, scrivere nuove informazioni e aggiornare i dati esistenti.

# ClientHandler

La classe ClientHandler estende UnicastRemoteObject e gestisce la connessione del client al server. Questa classe è progettata come un singleton per garantire che esista una sola istanza di ClientHandler.

## Attributi della classe

* PORT: Costante che rappresenta la porta su cui il client si connette al server. Tipo: int.
* stub: Riferimento all’interfaccia del server per l’invocazione dei metodi remoti. Tipo: ServerInterface.
* instance: Singleton che rappresenta l’istanza unica della classe. Tipo: ClientHandler.

## Metodi Principali della Classe

1. ClientHandler(): Costruttore della classe. Crea una connessione remota con il server. Complessità: O(1).
2. getStub(): Restituisce il riferimento all’interfaccia del server. Complessità: O(1).
3. getInstance(): Restituisce l’unica istanza della classe ClientHandler, creando l’istanza se non esiste. Complessità: O(1).

**Nota** La complessità degli accessi e delle modifiche agli attributi è O(1), in quanto si tratta di operazioni dirette sugli attributi dell’oggetto. Le eccezioni gestite includono RemoteException, NotBoundException e InterruptedException.