PROGETTO LABORATORIO B CLIMATE MONITORING

SVILUPPATO DA: DUSHI GIULIO,
ROHIT KABOTRA,
AZUOKWU GOLDEN.
07/02/2024

INDICE DEL MANUALE:

Introduzione del progetto

- **"Climate Monitoring" è un progetto sviluppato in ambito accademico durante il LAB A' ed il 'LAB B. Il progetto presenta un sistema di monitoraggio e salvataggio di parametri climatici fornito da centri di monitoraggio, gestiti da Operatori abilitati e in grado di essere mostrati anche a degli utenti comuni.**

Il progetto è stato sviluppato in Java 21.0.1, utilizzando l'interfaccia grafica Swing, una libreria inclusa nelle API base di Java.
Per lo sviluppo è stato scelto l'IDE NetBeans 17, e il progetto è stato configurato con il sistema di build Maven, che garantisce una gestione efficiente delle dipendenze e della configurazione del progetto.
Il database utilizzato è PostgreSQL (v.16.x o superiore), garantendo stabilità, robustezza e

scalabilità.
Il progetto è stato strutturato per essere multipiattaforma, con test effettuati su:

Windows 10 Pro e Windows 11 (x64).
MacOS (x64 e ARM).

Questo progetto è suddiviso in package, ciascuno con una specifica responsabilità:

1. **client: Gestisce la connessione al database remoto tramite RMI.**
2. **common: Contiene classi e interfacce condivise (come Cittadino, Coordinate, Operatore).**
3. **database: Gestisce la persistenza, con classi per il database e l'esecuzione delle query.**
4. **layoutcittadino e layoutplot: Forniscono layout grafici per la gestione dei cittadini e dei grafici.**
5. **operatore: Contiene tutte le classi relative alla gestione e alla logica degli operatori (es. Login, Home, Config).**

La classe Client, contenuta nel pacchetto Client, ha il compito principale di stabilire una connessione remota al database tramite il protocollo RMI (Remote Method Invocation). Fornisce un metodo per ottenere un'istanza remota dell'oggetto Database.

1. **Importazioni principali:**
    - **java.rmi.RemoteException: Utilizzato per gestire eccezioni relative alle operazioni RMI.**
    - **java.rmi.registry.LocateRegistry: Permette di localizzare il registro RMI.**
    - **java.rmi.registry.Registry: Rappresenta il registro in cui gli oggetti remoti sono registrati.**
    - **LAB_B.Common.Config: Classe che probabilmente contiene le configurazioni necessarie per l'accesso remoto, come host, porta e nome dell'oggetto remoto.**
    - **LAB_B.Database.Database: Interfaccia remota che rappresenta l'oggetto del database a cui connettersi.**

Metodi

1. getDb()

- **Descrizione: Metodo statico che fornisce un'istanza dell'oggetto remoto Database tramite RMI.**
- **Dettagli Implementativi:**
  - **Caricamento della Configurazione:**
    - **Crea un'istanza di Config per leggere i parametri necessari per la connessione (es. host, porta e nome dell'oggetto remoto RMI).**
  - **Connessione al Registro RMI:**
    - **Usa LocateRegistry.getRegistry(cf.getHost(), cf.getRmiPort()) per ottenere il registro RMI configurato nell'host e porta specificati.**
  - **Lookup dell'Oggetto Remoto:**
    - **Utilizza registry.lookup(cf.getRmiDbName()) per cercare l'oggetto remoto (un'istanza del database) registrato nel registro con il nome specificato.**
  - **Gestione delle Eccezioni:**
    - **Se l'oggetto remoto non è trovato, lancia una RemoteException.**

        - **Gestisce altre eccezioni (es. problemi di connessione) rilanciandole con un messaggio personalizzato.**
- **Eccezioni Gestite:**
    - **NotBoundException: Se l'oggetto remoto non è trovato nel registro.**
    - **RemoteException: Per problemi di connessione o altri errori RMI generici.**
    - **Exception: Gestisce eventuali errori imprevisti.**
- **Ritorno:**

Restituisce un'istanza dello stub remoto Database, che può essere utilizzata per chiamare metodi remoti.

La classe LayoutCittadino è una componente dell'interfaccia grafica, progettata per consentire agli utenti di visualizzare città e posizioni geografiche su una mappa interattiva. Utilizza **JMapViewer**, una libreria per lavorare con mappe OpenStreetMap, e fornisce funzionalità per caricare città, cercarle, posizionarle sulla mappa e mostrare grafici correlati.

Dettaglio delle Funzionalità e Metodi

Attributi Principali

1. **Componenti dell'Interfaccia Grafica**:
    - mapViewer: Mappa interattiva dove vengono mostrati i marker.
    - comboBox: Dropdown che elenca le città disponibili.
    - caricaCitta: Pulsante per caricare le città vicine alla posizione attuale della mappa.
    - posiziona: Pulsante per aggiungere un marker alla mappa per la città selezionata.
    - search: Pulsante per cercare città tramite nome.
    - mostra: Pulsante per aprire una finestra con grafici relativi alla città selezionata.
2. **Dati**:
    - coordinateM: Lista delle coordinate recuperate dal database.
    - newDot: Marker posizionato sulla mappa per la città selezionata.
3. **Città Predefinita**:
    - La mappa è inizializzata su **Varese** con coordinate geografiche preimpostate.

Metodi Principali

Costruttore: LayoutCittadino()

- **Funzione**:
  - Configura l'interfaccia grafica.
  - Inizializza i componenti come mapViewer, comboBox e pulsanti.
  - Imposta la mappa su una posizione iniziale (Varese).
  - Aggiunge listener ai pulsanti per gestire le interazioni dell'utente.

---

setMarker()

- **Funzione**:
  - Recupera tutte le coordinate disponibili dal database remoto.
  - Aggiunge marker sulla mappa per ogni coordinata recuperata.
- **Utilizzo**:
  - Chiamato durante l'inizializzazione per mostrare tutte le città sulla mappa.

---

setComboBox()

- **Funzione**:

    - Riempie la comboBox con le città vicine alla posizione attuale della mappa.
    - Utilizza il livello di zoom della mappa per determinare la tolleranza geografica (più è alto lo zoom, più precise sono le città).

setComboBox(String search)

- **Funzione**:
    - Riempie la comboBox con i risultati della ricerca basata su un nome specificato dall'utente.
    - Gestisce errori di caricamento o dati non trovati.
- **Utilizzo**:
    - Chiamato quando l'utente inserisce il nome di una città da cercare nel campo di testo della comboBox.

---

setPosition(Coordinate pos)

- **Funzione**:
    - Posiziona la mappa su una specifica coordinata con uno zoom predefinito (livello 10).
- **Utilizzo**:

- Chiamato quando un utente seleziona una città dalla comboBox e preme il pulsante posiziona.

Listener dei Pulsanti

1. **caricaCitta**:
   - Richiama setComboBox() per aggiornare la lista delle città vicine.
   - Disabilita temporaneamente la comboBox durante il caricamento.
2. **posiziona**:
   - Recupera la città selezionata nella comboBox.
   - Posiziona un marker (MapMarkerDot) sulla mappa alla posizione della città.
3. **search**:
   - Recupera il nome della città dal campo di testo della comboBox.
   - Aggiorna la lista della comboBox con i risultati della ricerca.
4. **mostra**:
   - Controlla che una città sia stata selezionata.
   - Apre una nuova finestra con i grafici relativi alla città selezionata (utilizzando la classe LayoutPlot).

La classe LayoutPlot, appartenente al pacchetto LAB_B.Common.Cittadino, è una finestra grafica basata su **JFrame** che consente di visualizzare grafici climatici relativi a una specifica città. La classe utilizza un approccio basato su schede per presentare diverse visualizzazioni (plot) e offre un menu a tendina per cambiare il tipo di grafico.

Caratteristiche Principali

- **Obiettivo**: Fornire un'interfaccia grafica per esplorare i dati climatici di una città.
- **Funzionalità**:
  - Menu a tendina (ComboBox) per selezionare i tipi di grafico.
  - Visualizzazione dinamica dei grafici tramite un CardLayout.
  - Integrazione con i dati forniti dal database remoto.

Attributi

1. **db**:
   - Tipo: Database

    - Descrizione: Riferimento all'oggetto del database remoto, utilizzato per recuperare i dati climatici.
2. **body**:
    - Tipo: Container
    - Descrizione: Contenitore principale della finestra.
3. **cards**:
    - Tipo: CardLayout
    - Descrizione: Layout che permette di alternare tra i diversi pannelli dei grafici.
4. **center**:
    - Tipo: JPanel
    - Descrizione: Pannello centrale che contiene le schede dei grafici.

Metodi Principali

1. Costruttore: LayoutPlot(Coordinate tempCity, Database db)

- **Parametri**:
    - tempCity: Coordinata della città di cui visualizzare i dati climatici.

    - db: Riferimento all'istanza del database remoto.
- **Funzione**:
    - Inizializza la finestra e configura il layout.
    - Chiama i metodi setNorth() e setCenter() per costruire rispettivamente il pannello superiore (menu) e quello centrale (grafici).

---

2. setNorth()

- **Funzione**:
    - Crea il pannello superiore con un menu a tendina per selezionare i tipi di grafico.
    - Associa un listener al menu per cambiare grafico quando l'utente seleziona un'opzione.
- **Dettagli**:
    - Recupera i tipi di grafico dall'enum TipiPlot.
    - Utilizza CardLayout per alternare i pannelli corrispondenti ai tipi di grafico.

---

3. setCenter(Coordinate city)

- **Funzione**:
    - Crea il pannello centrale che contiene i grafici.

    - Per ogni tipo di grafico (da TipiPlot), crea un'istanza di Plot1 e la aggiunge come scheda al CardLayout.
- **Dettagli**:
    - Utilizza le informazioni sulla città (Coordinate) per popolare i dati dei grafici.

Flusso Operativo

1. **Inizializzazione**:
    - L'utente seleziona una città tramite un'interfaccia (es. LayoutCittadino).
    - Viene aperta una finestra di LayoutPlot con i grafici relativi alla città selezionata.
2. **Visualizzazione Grafici**:
    - Il menu a tendina nella parte superiore consente di selezionare un tipo di grafico.
    - Il CardLayout mostra il grafico corrispondente al tipo selezionato.
3. **Interazione con il Database**:
    - Ogni pannello grafico (Plot1) richiama il database per ottenere i dati climatici della città.

Struttura dei Grafici

- Ogni tipo di grafico è rappresentato da una scheda distinta.
- I tipi di grafico sono definiti dall'enum TipiPlot (es. WIND, HUMIDITY).

La classe Coordinate implementa l'interfaccia ICoordinate fornita dalla libreria **JMapViewer** e rappresenta una posizione geografica associata a una città (Citta). Questa classe consente di lavorare facilmente con le coordinate di una città (latitudine e longitudine) e integra la serializzazione per supportare la persistenza e il trasferimento remoto.

Metodi della Classe

1. Costruttore: Coordinate(Citta citta)

- **Descrizione**:
  - Inizializza un'istanza della classe Coordinate utilizzando un oggetto Citta.
  - Imposta automaticamente la latitudine e la longitudine della città.
- **Parametri**:
  - Citta citta: L'oggetto Citta che contiene i dati geografici (latitudine e longitudine).

---

2. double getLat()

- **Descrizione**:
  - Ritorna la latitudine della coordinata.
- **Implementazione**:
  - Metodo dell'interfaccia ICoordinate di **JMapViewer**.

---

3. void setLat(double lat)

- **Descrizione**:
  - Imposta il valore della latitudine.
- **Parametri**:
  - double lat: Nuova latitudine da assegnare.

---

4. double getLon()

- **Descrizione**:
  - Ritorna la longitudine della coordinata.
- **Implementazione**:
  - Metodo dell'interfaccia ICoordinate.

---

5. void setLon(double lon)

- **Descrizione**:
  - Imposta il valore della longitudine.

- **Parametri**:
  - double lon: Nuova longitudine da assegnare.

---

6. Citta getCitta()

- **Descrizione**:
  - Restituisce l'oggetto Citta associato a questa coordinata.
- **Ritorno**:
  - Un'istanza di Citta.

---

7. String toString()

- **Descrizione**:
  - Fornisce una rappresentazione testuale della coordinata, utilizzando il metodo toString() della classe Citta.
- **Ritorno**:
  - Una stringa che rappresenta i dati della città associata.

Funzionalità Principali

1. **Associazione con una città**:

    - Ogni oggetto Coordinate è direttamente collegato a un'istanza di Citta, semplificando l'accesso alle informazioni anagrafiche e geografiche.
2. **Compatibilità con JMapViewer**:
    - Implementa i metodi richiesti dall'interfaccia ICoordinate, permettendo di lavorare con le coordinate all'interno di mappe interattive.
3. **Persistenza**:
    - Essendo Serializable, l'oggetto può essere salvato su disco o trasmesso su una rete.

La classe Coordinate implementa l'interfaccia ICoordinate fornita dalla libreria **JMapViewer** e rappresenta una posizione geografica associata a una città (Citta). Questa classe consente di lavorare facilmente con le coordinate di una città (latitudine e longitudine) e integra la serializzazione per supportare la persistenza e il trasferimento remoto.

Metodi della Classe

1. Costruttore: Coordinate(Citta citta)

- **Descrizione**:
    - Inizializza un'istanza della classe Coordinate utilizzando un oggetto Citta.

    - Imposta automaticamente la latitudine e la longitudine della città.
- **Parametri**:
    - Citta citta: L'oggetto Citta che contiene i dati geografici (latitudine e longitudine).

---

2. double getLat()

- **Descrizione**:
    - Ritorna la latitudine della coordinata.
- **Implementazione**:
    - Metodo dell'interfaccia ICoordinate di **JMapViewer**.

---

3. void setLat(double lat)

- **Descrizione**:
    - Imposta il valore della latitudine.
- **Parametri**:
    - double lat: Nuova latitudine da assegnare.

---

4. double getLon()

- **Descrizione**:
    - Ritorna la longitudine della coordinata.

- **Implementazione**:
  - Metodo dell'interfaccia ICoordinate.

---

5. void setLon(double lon)

- **Descrizione**:
  - Imposta il valore della longitudine.
- **Parametri**:
  - double lon: Nuova longitudine da assegnare.

---

6. Citta getCitta()

- **Descrizione**:
  - Restituisce l'oggetto Citta associato a questa coordinata.
- **Ritorno**:
  - Un'istanza di Citta.

---

7. String toString()

- **Descrizione**:
  - Fornisce una rappresentazione testuale della coordinata, utilizzando il metodo toString() della classe Citta.
- **Ritorno**:

- Una stringa che rappresenta i dati della città associata.

Funzionalità Principali

1. **Associazione con una città**:
   - Ogni oggetto Coordinate è direttamente collegato a un'istanza di Citta, semplificando l'accesso alle informazioni anagrafiche e geografiche.
2. **Compatibilità con JMapViewer**:
   - Implementa i metodi richiesti dall'interfaccia ICoordinate, permettendo di lavorare con le coordinate all'interno di mappe interattive.
3. **Persistenza**:
   - Essendo Serializable, l'oggetto può essere salvato su disco o trasmesso su una rete.

String getGeoname()

- **Descrizione**:
  - Restituisce l'identificativo unico della città (geoname_id).
- **Ritorno**:
  - Una stringa che rappresenta l'ID geografico della città.

---

3. String getName()

- **Descrizione**:
  - Restituisce il nome completo della città.
- **Ritorno**:
  - Una stringa con il nome della città.

4. String getAscii_name()

- **Descrizione**:
  - Restituisce il nome della città in formato ASCII.
- **Ritorno**:
  - Una stringa con il nome della città in ASCII.

---

5. String getCountry_code()

- **Descrizione**:
  - Restituisce il codice del paese associato alla città (es. IT per Italia).
- **Ritorno**:
  - Una stringa con il codice del paese.

---

6. String getCountry_name()

- **Descrizione**:

    - Restituisce il nome completo del paese associato alla città.
- **Ritorno**:
    - Una stringa con il nome del paese.

---

7. double getLongitude()

- **Descrizione**:
    - Restituisce la longitudine della città.
- **Ritorno**:
    - Un valore in formato double che rappresenta la longitudine.

---

8. double getLatitide()

- **Descrizione**:
    - Restituisce la latitudine della città.
- **Ritorno**:
    - Un valore in formato double che rappresenta la latitudine.
- **Nota**:
    - Nel metodo, il nome è stato scritto come getLatitide (probabilmente un errore di battitura, dovrebbe essere getLatitude).

---

9. String toString()

- **Descrizione**:
  - Fornisce una rappresentazione testuale della città.
  - Combinazione di:
    - Nome ASCII (ascii_name).
    - Codice del paese (country_code).
    - Nome del paese (country_name).
- **Ritorno**:
  - Una stringa nel formato: <ascii_name>, <country_code>, <country_name>.
-

Classe Operatore

Descrizione Generale

La classe Operatore, contenuta nel pacchetto LAB_B.Common.Interface, rappresenta un utente operatore del sistema. È progettata per gestire e validare i dati di un operatore, come nome, cognome, codice fiscale, email e password. Inoltre, offre funzionalità per generare uno username unico tramite un'interazione con il database.

Attributi

1. **nome, cognome**:
    - Contengono rispettivamente il nome e il cognome dell'operatore.
    - Devono avere una lunghezza massima di 30 caratteri.
2. **codFiscale**:
    - Codice fiscale dell'operatore, deve essere alfanumerico e lungo esattamente 16 caratteri.
3. **email**:
    - Indirizzo email dell'operatore, con validazione del formato.
4. **password, confermaPassword**:
    - La password deve rispettare requisiti di sicurezza (lunghezza minima di 8 caratteri, presenza di maiuscole, minuscole, numeri e simboli).
    - confermaPassword deve coincidere con password.
5. **username**:
    - Username generato o fornito per l'operatore, unico per ogni utente.

6. **err**:
    - Oggetto StringBuilder utilizzato per accumulare messaggi di errore durante la validazione dei dati.

Metodi

1. Costruttore: Operatore(String nome, String cognome, String codFiscale, String email, String password, String confermaPassword, String username)

- **Funzione**:
    - Inizializza l'istanza con i dati forniti.
    - Assicura che il codice fiscale (codFiscale) sia sempre in maiuscolo.

---

2. String getNome()

- **Descrizione**:
    - Restituisce il nome dell'operatore.

---

3. String getCognome()

- **Descrizione**:
    - Restituisce il cognome dell'operatore.

---

4. String getCodFiscale()

- **Descrizione**:
  - Restituisce il codice fiscale dell'operatore.

---

5. String getEmail()

- **Descrizione**:
  - Restituisce l'indirizzo email dell'operatore.

---

6. String getPassword()

- **Descrizione**:
  - Restituisce la password dell'operatore.

---

7. String getUsername()

- **Descrizione**:
  - Restituisce lo username generato o fornito.

---

8. boolean validate()

- **Descrizione**:
  - Valida i dati dell'operatore secondo regole specifiche:
    - Nome e cognome devono essere non vuoti e con un massimo di 30 caratteri.

        - Il codice fiscale deve essere alfanumerico e lungo 16 caratteri.
        - L'email deve avere un formato valido.
        - La password deve rispettare criteri di sicurezza.
        - Le password devono coincidere.
    - Se username non è specificato, viene generato tramite il database.
- **Ritorno**:
    - true se tutti i dati sono validi; false altrimenti.

---

9. String getErrorMessages()

- **Descrizione**:
    - Restituisce tutti i messaggi di errore accumulati durante la validazione.
- **Ritorno**:
    - Una stringa contenente tutti gli errori rilevati.

---

10. boolean isValidEmail(String email)

- **Descrizione**:
    - Controlla se l'email fornita è valida.

    - Utilizza una regex per verificare la presenza di un formato corretto (es. example@domain.com).
- **Ritorno**:
    - true se l'email è valida; false altrimenti.

---

11. boolean isValidPassword(String password)

- **Descrizione**:
    - Controlla se la password soddisfa i seguenti criteri:
        - Almeno una lettera maiuscola.
        - Almeno una lettera minuscola.
        - Almeno un numero.
        - Almeno un carattere speciale.
- **Ritorno**:
    - true se la password è valida; false altrimenti.

Classe TipiPlot

Descrizione Generale

TipiPlot è un'**enum** che rappresenta i diversi tipi di grafici climatici supportati dal sistema. Ogni elemento dell'enumerazione ha un codice univoco e un nome che lo identifica. La classe è progettata per fornire una

gestione organizzata e chiara dei tipi di grafici utilizzati nell'applicazione.

Valori dell'Enumerazione

1. **WIND**: Grafico per la velocità del vento.
2. **HUMIDITY**: Grafico per l'umidità.
3. **PRESSURE**: Grafico per la pressione atmosferica.
4. **TEMPERATURE**: Grafico per la temperatura.
5. **PRECIPITATION**: Grafico per le precipitazioni.
6. **GALCIER_ALTITUDE**: Grafico per l'altitudine dei ghiacciai (con possibile errore di battitura nel nome).
7. **GLACIER_MASS**: Grafico per la massa dei ghiacciai.

Attributi

1. **codice**:
    - Tipo: int
    - Descrizione: Identificativo numerico unico associato a ciascun tipo di grafico.
2. **nome**:
    - Tipo: String

- Descrizione: Nome descrittivo del tipo di grafico.

Metodi

1. Costruttore: TipiPlot(int codice, String name)

- **Descrizione**:
  - Inizializza un valore dell'enum con un codice e un nome.
- **Parametri**:
  - codice: Codice numerico univoco.
  - name: Nome del tipo di grafico.

---

2. int getCodice()

- **Descrizione**:
  - Restituisce il codice numerico associato al tipo di grafico.
- **Ritorno**:
  - Valore int del codice.

---

3. String getName()

- **Descrizione**:
  - Restituisce il nome del tipo di grafico.

- **Ritorno**:
  - Valore String del nome.

Classe LayoutOperatore

Il codice presentato è una classe Java chiamata LayoutOperatore, che estende una classe padre LayoutStandard. Questa classe è responsabile della creazione e gestione dell'interfaccia grafica per un operatore, utilizzando il framework Swing. Vediamo i dettagli delle principali funzionalità implementate.

Struttura e Scopo

1. **Estensione di LayoutStandard:**
   - **La classe LayoutOperatore eredita da LayoutStandard, presumibilmente una classe che fornisce funzionalità base comuni per la struttura delle interfacce.**
   - **Personalizza l'interfaccia per il contesto dell'operatore, fornendo funzionalità specifiche come:**
     - **Visualizzazione dei centri associati.**
     - **Aggiunta di dati climatici.**
     - **Creazione di nuovi centri.**

2. **Costruttore:**
    - **Inizializza i componenti dell'interfaccia e struttura il layout grafico.**
    - **Imposta il titolo con un messaggio di benvenuto che include il nome dell'utente.**
    - **Configura pannelli e bottoni per interazioni specifiche.**

Principali Componenti e Metodi

1. Costruzione del Layout

- **Etichetta di Benvenuto:**
    - **Mostra un messaggio personalizzato per l'operatore utilizzando il nome utente in maiuscolo.**
- **Lista dei Centri:**
    - **Usa una JList associata a un modello DefaultListModel per visualizzare i centri associati all'operatore.**
    - **Popola i dati con il metodo caricaCentri().**
- **Dropdown delle Città:**
    - **Usa un JComboBox per elencare le città recuperate dal database tramite caricaCitta().**

- **Bottoni Interattivi:**
  - **Crea Centro Monitoraggio:**
    - **Permette di aggiungere nuovi centri.**
    - **Apre una finestra dedicata tramite il metodo apriFinestraCreaCentro.**
  - **Aggiungi Dati Climatici:**
    - **Permette di inserire dati climatici in un centro selezionato.**
    - **Apre una finestra dedicata tramite il metodo apriFinestraDatiClimatici.**

---

2. Metodi di Supporto

caricaCentri()

- **Recupera dal database i centri associati all'utente.**
- **Mostra un messaggio appropriato se non ci sono centri disponibili.**

caricaCitta()

- **Popola il dropdown delle città recuperandole dal database.**
- **Gestisce eventuali errori remoti con un'eccezione.**

---

3. Gestione Eventi

- **Usa un listener personalizzato Gestore per associare azioni ai bottoni.**
  - **Azione su creaCentroButton:**
    - **Apre la finestra per creare un nuovo centro.**
  - **Azione su aggiungiDatiClimatici:**
    - **Apre la finestra per inserire dati climatici.**

---

4. Finestra per Dati Climatici

- **Campi Dinamici:**
  - **Crea dinamicamente campi per inserire valori climatici come temperatura, pressione, ecc.**
- **Validazione Input:**
  - **Usa filtri per garantire che i valori inseriti siano numerici e rispetti requisiti di lunghezza per i commenti.**
- **Salvataggio Dati:**
  - **Recupera i valori dai campi e li invia al database tramite il metodo salvaDatiClimatici.**

---

5. Finestra per Creazione Centro

- **Campi Obbligatori:**
  - **Richiede l'inserimento del nome e indirizzo del centro.**
- **Interazione col Database:**
  - **Salva i dati nel database e aggiorna la lista dei centri se l'operazione ha successo.**

Classe Login

Descrizione Generale

La classe Login è un'interfaccia grafica basata su **Swing** che gestisce l'autenticazione degli operatori. Gli utenti possono inserire il loro username (o codice fiscale) e password, effettuare il login o registrarsi tramite una nuova finestra. La classe implementa funzionalità per limitare i tentativi di accesso, navigare tra schermate e validare i dati.

Componenti Principali

Attributi

1. **MAX_RETRIES**:

    - Valore costante (3) che rappresenta il numero massimo di tentativi di login permessi.
2. **usernameField**:
    - Campo di testo per l'inserimento dello username o del codice fiscale.
3. **passwordField**:
    - Campo di tipo password per l'inserimento della password.
4. **loginButton**:
    - Pulsante per avviare la procedura di login.
5. **registerButton**:
    - Pulsante per navigare alla finestra di registrazione.
6. **loginAttempts**:
    - Contatore dei tentativi di login effettuati dall'utente.

Metodi

1. Costruttore: Login()

- **Descrizione**:
    - Inizializza l'interfaccia utente.
    - Configura il layout e i componenti, tra cui i campi di input, i pulsanti e i listener.

- **Esempio**:
  - Chiama i metodi:
    - setupUI() per creare i componenti grafici.
    - configureButtons() per assegnare azioni ai pulsanti.

---

2. setupUI()

- **Descrizione**:
  - Configura i componenti grafici principali.
  - Crea un pannello centrale (centerPanel) che contiene i campi di input e i pulsanti.
  - Aggiunge il pannello alla finestra principale.

---

3. createTextField(String title)

- **Descrizione**:
  - Crea un campo di testo generico con un titolo.
- **Parametri**:
  - title: Titolo visualizzato nel bordo del campo.
- **Ritorno**:
  - Un'istanza di JTextField.

---

4. createPasswordField(String title)

- **Descrizione**:
  - Crea un campo di input per password con un titolo.
- **Parametri**:
  - title: Titolo visualizzato nel bordo del campo.
- **Ritorno**:
  - Un'istanza di JPasswordField.

---

5. createButton(String text, Color color)

- **Descrizione**:
  - Crea un pulsante con il testo e il colore specificati.
- **Parametri**:
  - text: Testo visualizzato sul pulsante.
  - color: Colore di sfondo del pulsante.
- **Ritorno**:
  - Un'istanza di JButton.

---

6. configureButtons()

- **Descrizione**:
  - Configura i listener per i pulsanti:

        - loginButton: Associa il metodo handleLogin().
        - registerButton: Associa il metodo handleRegistration().
        - home: Torna alla finestra principale.

---

7. handleLogin()

- **Descrizione**:
    - Recupera i valori inseriti dall'utente nei campi di input.
    - Esegue controlli preliminari sulla presenza di username e password.
    - Chiama performLogin() per effettuare il login.

---

8. performLogin(String username, String password)

- **Descrizione**:
    - Esegue il login dell'utente interagendo con il database.
    - Se l'autenticazione riesce:
        - Mostra un messaggio di successo.
        - Apre l'interfaccia dell'operatore (LayoutOperatore).
    - Se fallisce:

        - Incrementa il contatore dei tentativi.
        - Mostra un messaggio di errore.
        - Disabilita il pulsante di login al superamento dei tentativi massimi.

---

9. handleRegistration()

- **Descrizione**:
    - Apre la finestra di registrazione (SignUp).
    - Chiude la finestra di login.

---

10. navigateToHome()

- **Descrizione**:
    - Torna alla schermata principale (Home).

---

11. showErrorMessage(String message)

- **Descrizione**:
    - Mostra un messaggio di errore all'utente tramite un pop-up (JOptionPane).

---

12. showSuccessMessage(String message)

- **Descrizione**:

- Mostra un messaggio di successo all'utente tramite un pop-up (JOptionPane).

Flusso Operativo

1. **Fase di Login**:
   - L'utente inserisce lo username/codice fiscale e la password.
   - Premendo "Login", il sistema verifica i dati:
     - Effettua controlli preliminari (campi vuoti).
     - Autentica l'utente tramite il database.
   - Se il login riesce:
     - Apre l'interfaccia dedicata all'operatore.
   - Se fallisce:
     - Incrementa il contatore e gestisce il blocco dell'utente dopo 3 tentativi.
2. **Navigazione**:
   - L'utente può:
     - Registrarsi tramite il pulsante "Register".
     - Tornare alla schermata principale tramite il pulsante "Home".

Classe SignUp

Descrizione Generale

La classe SignUp è un'estensione della classe LayoutStandard e rappresenta l'interfaccia grafica per la registrazione di un nuovo operatore nel sistema. Utilizza Swing per la creazione dei componenti e fornisce campi di input per i dettagli dell'operatore, controlli di validazione, e interazioni con il database per memorizzare i dati.

Componenti Principali

Attributi

1. **Campi di input per la registrazione**:
    - **nomeField**: Campo per l'inserimento del nome.
    - **cognomeField**: Campo per l'inserimento del cognome.
    - **codiceFiscaleField**: Campo per l'inserimento del codice fiscale.
    - **emailField**: Campo per l'inserimento dell'email.
    - **passwordField**: Campo per l'inserimento della password.
    - **confermaPasswordField**: Campo per la conferma della password.
2. **Pulsanti**:

- **helpButton**: Pulsante per mostrare una guida alla compilazione dei campi.
- **saveButton**: Pulsante per salvare i dati di registrazione.
- **backButton**: Pulsante per tornare alla schermata di login.

Metodi Principali

1. Costruttore: SignUp()

- **Funzione**:
  - Inizializza la finestra di registrazione.
  - Configura i layout, i campi di input e i pulsanti.
  - Mostra la finestra con un'interfaccia grafica non ridimensionabile.

---

2. initializeUI()

- **Funzione**:
  - Configura l'interfaccia utente, suddividendola in tre sezioni:
    - **Titolo**: Visualizza il titolo "Registra un nuovo Operatore" e un pulsante di aiuto.
    - **Input**: Crea campi di input per i dati dell'operatore con etichette associate.

        - **Pulsanti di azione**: Aggiunge pulsanti per salvare o tornare alla schermata di login.
- **Comportamento**:
    - Utilizza layout come BorderLayout e GridLayout per organizzare i componenti.

---

3. addField(JPanel panel, String label, JTextField field)

- **Funzione**:
    - Aggiunge un campo di input al pannello con una relativa etichetta.
- **Parametri**:
    - panel: Il pannello a cui aggiungere il campo.
    - label: Il testo dell'etichetta.
    - field: Il campo di input associato.

---

4. createHelpPanel()

- **Funzione**:
    - Crea un pannello che contiene il pulsante di aiuto.
    - Assegna un listener al pulsante per aprire una finestra di guida.

---

5. showHelpDialog()

- **Funzione**:
  - Mostra un messaggio di guida con suggerimenti per la compilazione dei campi, ad esempio:
    - Formato del nome e cognome.
    - Validità del codice fiscale e dell'email.
    - Requisiti per la password.
- **Comportamento**:
  - Utilizza JOptionPane per visualizzare il messaggio.

---

6. handleRegistration()

- **Funzione**:
  - Gestisce la registrazione di un nuovo operatore.
  - Esegue i seguenti passaggi:
    . Recupera i dati inseriti dall'utente.
    . Valida i dati utilizzando la classe Operatore.
    . Salva i dati nel database se validi.
    . Mostra messaggi di successo o errore a seconda dell'esito.
- **Eccezioni Gestite**:

- RemoteException per errori di connessione al server.
- Messaggi di errore specifici per dati non validi.

Flusso Operativo

1. **Interfaccia Utente**:
   - L'utente apre la schermata di registrazione.
   - Compila i campi richiesti:
     - Nome, cognome, codice fiscale, email, password e conferma password.
   - Può consultare la guida tramite il pulsante "?".
2. **Registrazione**:
   - L'utente clicca su "Salva".
   - Il sistema valida i dati:
     - Nome e cognome con massimo 30 caratteri.
     - Codice fiscale esattamente di 16 caratteri.
     - Email valida.
     - Password conforme ai requisiti di sicurezza.
     - Le password devono coincidere.

    - Se i dati sono validi, l'operatore viene registrato nel database.
3. **Navigazione**:
    - L'utente può tornare alla schermata di login tramite il pulsante "Torna indietro".

Classe Config

Descrizione Generale

La classe Config è responsabile della gestione della configurazione del sistema. Legge le proprietà da un file di configurazione config.properties e fornisce metodi per accedere ai valori di configurazione necessari per connettersi al database e al sistema RMI.

Funzionalità

1. **Caricamento Configurazione**:
    - Legge i parametri dal file config.properties utilizzando la classe Properties.
    - Permette di centralizzare la gestione delle configurazioni, rendendo il sistema più flessibile e facilmente configurabile.
2. **Gestione Connessione Database**:

- Fornisce metodi per ottenere le informazioni necessarie per stabilire una connessione al database.

3. **Gestione RMI (Remote Method Invocation)**:
   - Recupera i dettagli per configurare il sistema RMI, come porta e nome del database remoto.

Attributi

1. **fileConfigPath**:
   - **Tipo**: String
   - **Descrizione**: Percorso relativo al file di configurazione config.properties.
2. **Campi di configurazione**:
   - **host**: Indirizzo host del database.
   - **port**: Porta per la connessione al database.
   - **db_username**: Nome utente per la connessione al database.
   - **db_password**: Password per la connessione al database.
   - **db_name**: Nome del database.
   - **rmi_port**: Porta per la connessione RMI.
   - **rmi_db_name**: Nome del database remoto.

Metodi Principali

1. Costruttore: Config()

- **Descrizione**:
  - Carica il file di configurazione config.properties.
  - Se il file non viene trovato o non è accessibile, stampa un messaggio di errore nel log.
- **Esempio di Caricamento**:

properties

CopiaModifica

host=localhost

port=5432

db.username=admin

db.password=password123

db.name=my_database

rmi.port=1099

rmi.db.name=rmi_database

---

2. getHost()

- **Descrizione**:
  - Restituisce l'indirizzo host del database.
- **Ritorno**:

- String con l'host.

---

3. getPort()

- **Descrizione**:
  - Restituisce la porta del database.
- **Ritorno**:
  - int con il numero di porta.

---

4. getDbUsername()

- **Descrizione**:
  - Restituisce il nome utente per il database.
- **Ritorno**:
  - String con lo username.

---

5. getDbPassword()

- **Descrizione**:
  - Restituisce la password per il database.
- **Ritorno**:
  - String con la password.

---

6. getDbName()

- **Descrizione**:
  - Restituisce il nome del database.
- **Ritorno**:
  - String con il nome del database.

---

7. getRmiPort()

- **Descrizione**:
  - Restituisce la porta utilizzata per la connessione RMI.
- **Ritorno**:
  - int con la porta RMI.

---

8. getRmiDbName()

- **Descrizione**:
  - Restituisce il nome del database RMI.
- **Ritorno**:
  - String con il nome del database remoto.

---

9. getUrlDb()

- **Descrizione**:
  - Costruisce l'URL completo per la connessione al database.

- **Ritorno**:
    - Una stringa nel formato: jdbc:postgresql://<host>:<port>/<db_name>.

Flusso Operativo

1. **Caricamento Configurazione**:
    - Quando viene creata un'istanza di Config, la classe cerca di caricare il file config.properties.
2. **Accesso ai Parametri**:
    - I metodi getter forniscono accesso ai parametri caricati.
3. **Costruzione dell'URL**:
    - getUrlDb() combina i valori di configurazione per costruire un URL compatibile con PostgreSQL.

Classe Home

Descrizione Generale

La classe Home è la finestra principale dell'applicazione e funge da punto di accesso iniziale. Gli utenti possono scegliere tra due modalità:

1. **Cittadino**: Apre un'interfaccia per visualizzare informazioni climatiche.

2. **Operatore**: Consente agli operatori di autenticarsi tramite la schermata di login.

La classe utilizza Swing per l'interfaccia grafica e implementa un layout centrato per la disposizione dei componenti.

Componenti Principali

1. Attributi

- **bottoneCittadino**:
  - Bottone che avvia l'interfaccia per i cittadini.
  - Configurato con un'icona e uno stile specifico.
- **bottoneOperatore**:
  - Bottone che avvia la schermata di login per gli operatori.
  - Anch'esso configurato con un'icona e uno stile.

Metodi

1. Costruttore: Home()

- **Descrizione**:
  - Inizializza la finestra principale con titolo e layout.
  - Configura due bottoni principali per la navigazione e un'etichetta di benvenuto.

- **Comportamento**:
  - Imposta il layout con GridBagLayout per un posizionamento flessibile.
  - Configura l'aspetto e l'azione dei bottoni tramite la classe interna Gestore.

---

2. Classe Interna Gestore

- **Descrizione**:
  - Implementa l'interfaccia ActionListener per gestire i clic sui bottoni.
- **Metodi**:
  - **actionPerformed(ActionEvent e)**:
    - Se premuto il bottone **Cittadino**:
      - Avvia l'interfaccia LayoutCittadino.
      - Chiude la finestra principale.
    - Se premuto il bottone **Operatore**:
      - Avvia la schermata di login tramite la classe Login.
      - Chiude la finestra principale.

---

3. main(String[] args)

- **Descrizione**:

- Metodo di avvio dell'applicazione.
- Crea e mostra un'istanza della classe Home.

Funzionalità Offerte

1. **Accesso alla Modalità Cittadino**:
   - Permette agli utenti di esplorare i dati climatici senza autenticazione.
2. **Accesso alla Modalità Operatore**:
   - Richiede un'autenticazione per accedere alle funzionalità riservate agli operatori.
3. **Interfaccia Utente**:
   - Presenta un design semplice e intuitivo con pulsanti centrati e stilizzati.

Dettagli Tecnici

- **Layout**:
  - Utilizza GridBagLayout per una disposizione flessibile dei componenti.
- **Personalizzazione**:
  - Colore di sfondo dei bottoni, bordi e font sono configurati per migliorare l'esperienza utente.
- **Eventi**:
  - Implementa ActionListener per rilevare clic sui bottoni e avviare le rispettive finestre.

Classe LayoutStandard

Descrizione Generale

La classe LayoutStandard è una classe astratta che fornisce un layout di base per altre finestre dell'applicazione. Include un bottone "Home" per tornare alla schermata principale e un contenitore principale (body) per gestire i componenti dell'interfaccia. È progettata per essere estesa da altre classi che necessitano di una struttura grafica uniforme.

Attributi

1. **home**:
    - **Tipo**: JButton
    - **Descrizione**: Pulsante che consente di tornare alla finestra "Home".
2. **body**:
    - **Tipo**: Container
    - **Descrizione**: Contenitore principale della finestra, configurato con un BorderLayout.
3. **gestore**:
    - **Tipo**: Gestore

    - **Descrizione**: Classe interna che gestisce gli eventi dei pulsanti (implementa ActionListener).
4. **db**:
    - **Tipo**: Database
    - **Descrizione**: Istanza del database ottenuta tramite il client.

Metodi

1. Costruttore: LayoutStandard()

- **Descrizione**:
    - Inizializza la finestra con titolo e layout di base.
    - Configura il look and feel (utilizzando Nimbus, se disponibile).
    - Crea e aggiunge il pulsante "Home".
    - Recupera un'istanza del database tramite Client.getDb().
    - Imposta proprietà di base della finestra tramite setDefaultProperties().

---

2. setDefaultProperties()

- **Descrizione**:
    - Configura proprietà comuni della finestra:

        - Dimensioni fisse (800x800).
        - Chiusura dell'applicazione alla chiusura della finestra.
        - Posizionamento centrato.
        - Disabilita il ridimensionamento.

---

3. setDefaultProperties(JFrame temp)

- **Descrizione**:
    - Configura proprietà per finestre secondarie.
    - Simile al metodo precedente, ma con opzioni più flessibili (es. finestra ridimensionabile e chiusura specifica con DISPOSE_ON_CLOSE).

---

4. getBody()

- **Descrizione**:
    - Restituisce il contenitore principale della finestra.
- **Ritorno**:
    - Un'istanza di Container.

---

5. getGestore()

- **Descrizione**:

    - Restituisce il gestore degli eventi.
- **Ritorno**:
    - Un'istanza della classe interna Gestore.

Classe Interna Gestore

Funzione:

- Implementa ActionListener per gestire eventi generati dai pulsanti.

Metodi

actionPerformed(ActionEvent e)

- **Descrizione**:
    - Gestisce il clic sul pulsante "Home".
    - Azioni eseguite:
        . Crea un'istanza della classe Home.
        . Rende visibile la finestra "Home".
        . Chiude la finestra corrente.

Flusso Operativo

1. **Creazione della Finestra**:
    - Quando viene creata un'istanza di una classe che estende LayoutStandard, viene configurata con un layout uniforme.

2. **Navigazione con il Bottone "Home"**:
    - Il pulsante "Home", gestito dalla classe Gestore, consente di tornare alla schermata principale.
3. **Configurazione del Database**:
    - La classe recupera un'istanza del database tramite Client.getDb().

Classe Database

Il codice fornito rappresenta un'interfaccia Java chiamata Database che definisce vari metodi per la gestione di un database remoto tramite RMI (Remote Method Invocation). L'interfaccia permette operazioni relative a login, registrazione, gestione di dati climatici e centri di monitoraggio, nonché l'accesso a informazioni sulle coordinate e ai parametri per la visualizzazione di grafici.

---

1. Pacchetto e importazioni

- **package LAB_B.Database: Definisce il pacchetto in cui si trova l'interfaccia Database.**
- **Importa diverse classi necessarie:**

- LAB_B.Common.Interface: Contiene interfacce come Coordinate, Operatore, e TipiPlot utilizzate in questa interfaccia.
- org.jfree.data.category.DefaultCategoryDataset: Per creare dataset utili per grafici.
- Classi di base per RMI (Remote, RemoteException) e gestione SQL (SQLException).
- Strutture dati come ArrayList e List.

---

2. Dichiarazione dell'interfaccia

- L'interfaccia Database estende Remote, il che la rende un'interfaccia remota per l'utilizzo di RMI.
- Ogni metodo dichiara l'eccezione RemoteException per indicare che potrebbe verificarsi un errore durante una chiamata remota.

---

3. Metodi definiti nell'interfaccia

a. Autenticazione

1. login(String codiceFiscale, String password)
   - Consente di verificare le credenziali di un operatore.

- **Parametri:**
  - **codiceFiscale: Il codice fiscale dell'operatore.**
  - **password: La password dell'operatore.**
- **Ritorna un boolean che indica se il login è andato a buon fine.**
- **Può sollevare RemoteException o SQLException.**

2. **registrazione(Operatore operatore)**
   - **Consente di registrare un nuovo operatore.**
   - **Parametro:**
     - **operatore: Un oggetto di tipo Operatore che contiene i dati dell'operatore da registrare.**
   - **Ritorna un boolean che indica se la registrazione è avvenuta con successo.**

---

b. Gestione delle coordinate

3. **getCoordinaResultSet()**
   - **Restituisce una lista di oggetti Coordinate.**
   - **Utilizzato per recuperare tutte le coordinate registrate.**
4. **getCoordinaResultSet(String name)**

- Restituisce una lista di coordinate che corrispondono al nome fornito.

5. getCoordinaResultSet(double latitude, double longitude, double tolerance)
    - Restituisce una lista di coordinate che si trovano entro una tolleranza specificata attorno alla latitudine e longitudine fornite.

---

c. Salvataggio dati

6. salvaDatiClimatici(String[] parametro, ArrayList<String> valori, ArrayList<String> commenti, ArrayList<Integer> punteggi, String username, long timestamp, Coordinate citta, String centro)
    - Permette di salvare dati climatici associati a una città e a un centro di monitoraggio.
    - Parametri principali:
        - parametro: Array di parametri climatici (ad esempio, temperatura, umidità).
        - valori: Lista dei valori per ogni parametro.
        - commenti: Lista di commenti relativi ai dati.
        - punteggi: Lista dei punteggi associati ai dati.

- **username: Operatore che salva i dati.**
- **timestamp: Momento in cui i dati sono stati raccolti.**
- **citta: Coordinate della città.**
- **centro: Nome del centro di monitoraggio.**

- **Può sollevare eccezioni generiche (Exception).**

7. **salvaCentroMonitoraggio(String nomeCentro, String descrizione, String username)**
   - **Permette di salvare un nuovo centro di monitoraggio.**
   - **Parametri:**
     - **nomeCentro: Nome del centro.**
     - **descrizione: Descrizione del centro.**
     - **username: Operatore responsabile.**

---

d. Recupero dei centri

8. **getCentriPerOperatore(String username)**
   - **Restituisce una lista di centri di monitoraggio associati a uno specifico operatore.**

---

e. Parametri per i grafici

9. **getParametri(Coordinate city, TipiPlot type)**
    - **Restituisce un oggetto DefaultCategoryDataset che rappresenta i parametri climatici di una città in base al tipo di grafico specificato.**
    - **Parametri:**
        - **city: Coordinate della città.**
        - **type: Tipo di grafico richiesto (ad esempio, istogramma, linea).**

---

4. Eccezioni

- **Ogni metodo può sollevare una RemoteException, indicando problemi di connessione o altre eccezioni legate alla comunicazione remota.**
- **Alcuni metodi includono altre eccezioni come SQLException o Exception generica.**

Classe DatabaseImpl

Il codice fornito definisce una classe Java chiamata DatabaseImpl, che implementa l'interfaccia Database. È progettata per interagire con un database remoto e utilizza tecnologie come RMI

(Remote Method Invocation) per comunicare tra client e server.

Ecco una descrizione dettagliata delle principali funzionalità:

---

1. Attributi

- **connection: Un oggetto statico Connection per gestire la connessione al database.**
- **queryExecutorImpl: Istanza della classe QueryExecutorImpl utilizzata per eseguire operazioni specifiche sul database, come login, registrazione, e altre query.**

---

2. Blocco statico

Il blocco static viene eseguito una sola volta al caricamento della classe. In questo caso, viene utilizzato per inizializzare la connessione al database attraverso il metodo initializeConnection.

---

3. Costruttore

Il costruttore DatabaseImpl:

- **Estende UnicastRemoteObject per abilitare l'accesso remoto tramite RMI.**

- **Controlla se la connessione al database è già aperta. In caso contrario, la inizializza.**
- **Inizializza l'oggetto QueryExecutorImpl.**

---

4. Metodi principali

4.1. Gestione della connessione al database

- **initializeConnection():**
  - **Configura e apre la connessione al database utilizzando i parametri recuperati da un oggetto Config.**
  - **Disabilita l'autocommit per abilitare transazioni manuali.**
- **getConnection():**
  - **Controlla lo stato della connessione e, se necessario, la riavvia.**
- **closeConnection():**
  - **Chiude la connessione se è aperta.**

4.2. Operazioni sul database

Questi metodi utilizzano l'istanza queryExecutorImpl per eseguire operazioni specifiche sul database.

- **Login**

java

CopiaModifica

public boolean login(String codiceFiscale, String password)

Controlla le credenziali dell'utente tramite il metodo login della classe QueryExecutorImpl.

- **Registrazione**

java

CopiaModifica

public boolean registrazione(Operatore operatore)

Registra un nuovo operatore nel database utilizzando il metodo salvaOperatore.

- **Gestione delle coordinate**
    - **Vari metodi per ottenere liste di coordinate geografiche:**
        - **In base a latitudine, longitudine e tolleranza:**

java

CopiaModifica

public List<Coordinate> getCoordinaResultSet(double latitude, double longitude, double tollerance)

        - **In base al nome:**

java

CopiaModifica

public List<Coordinate> getCoordinaResultSet(String name)

- **Tutte le coordinate:**

java

CopiaModifica

public List<Coordinate> getCoordinaResultSet()

- **Gestione dei centri di monitoraggio**
  - **Salvataggio di un centro:**

java

CopiaModifica

public boolean salvaCentroMonitoraggio(String nomeCentro, String descrizione, String username)

Registra un nuovo centro di monitoraggio nel database.

- **Salvataggio dati climatici**

java

CopiaModifica

public boolean salvaDatiClimatici(String[] parametri, ArrayList<String> valori, ArrayList<String> commenti, ArrayList<Integer> punteggi, String username, long timestamp, Coordinate citta, String centro)

Salva dati climatici relativi a un centro specifico.

- **Recupero parametri climatici**

java

CopiaModifica

public DefaultCategoryDataset getParametri(Coordinate city, TipiPlot type)

Ottiene i dati climatici di una città in un formato grafico (DefaultCategoryDataset).

- **Centri per operatore**

java

CopiaModifica

public List<String> getCentriPerOperatore(String username)

Recupera una lista di centri monitorati da un operatore specifico.

---

5. Errori e gestione delle eccezioni

- **I metodi catturano e gestiscono diverse eccezioni, come SQLException e IOException, stampando informazioni dettagliate in caso di errore.**
- **Gli errori durante l'inizializzazione della connessione o l'esecuzione delle query sono segnalati con messaggi utili.**

Classe QueryExecutorImpl

Il codice definisce una classe QueryExecutorImpl per l'interazione con un database. Questa classe gestisce la connessione al database e offre metodi per eseguire operazioni di lettura e scrittura. Di seguito, analizziamo i componenti principali.

---

1. Connessione al database

La classe utilizza un attributo conn di tipo Connection per gestire la connessione al database.

- **Costruttore:**

java

CopiaModifica

```
public QueryExecutorImpl() {
   this.conn = DatabaseImpl.getConnection();
}
```

Il costruttore inizializza la connessione tramite un metodo statico DatabaseImpl.getConnection().

- **Verifica della connessione:**

java

CopiaModifica

```
public void ensureConnection() throws
SQLException {
   if (conn == null || conn.isClosed()) {
      conn = DatabaseImpl.getConnection();
      if (conn == null || conn.isClosed()) {
         throw new SQLException("Impossibile stabilire
la connessione al database.");
      }
   }
}
```

Questo metodo assicura che la connessione sia aperta. Se la connessione è chiusa o nulla, tenta di ristabilirla.

---

2. Metodi principali

a. Login

java

CopiaModifica

```
public boolean login(String username, String
password)
```

Verifica le credenziali di accesso confrontandole con i dati presenti nel database. Se una corrispondenza esiste, restituisce true.

---

b. Validazione dei dati

- **Email: Metodo privato per verificare se un'email è valida tramite regex.**

java

CopiaModifica

private boolean isValidEmail(String email)

- **Codice fiscale: Controlla che il codice fiscale abbia 16 caratteri alfanumerici.**

java

CopiaModifica

private boolean isValidCodiceFiscale(String codiceFiscale)

---

c. Controllo unicità

- **Email esistente:**

java

CopiaModifica

public boolean emailEsistente(String email) throws SQLException

Usa una query per verificare se l'email esiste già.

- **Codice fiscale esistente:**

java

CopiaModifica

public boolean codiceFiscaleEsistente(String codiceFiscale) throws SQLException

Effettua una query per controllare la presenza del codice fiscale.

---

d. Salvataggio di un operatore

java

CopiaModifica

public boolean salvaOperatore(Operatore operatore) throws SQLException

Inserisce un nuovo operatore nel database. Include:

1. **Validazione dei campi (nome, cognome, codice fiscale, ecc.).**
2. **Verifica dell'unicità di email e codice fiscale.**
3. **Generazione dello username se necessario.**
4. **Esecuzione della query di inserimento.**

---

e. Generazione dello username

java

CopiaModifica

public String generateUsername(String nome, String cognome, String codFiscale) throws SQLException

Crea uno username unico basato sui primi caratteri del nome, cognome e una porzione del codice fiscale. Se lo username generato esiste già, aggiunge un contatore incrementale.

---

f. Salvataggio di centri monitoraggio

java

CopiaModifica

public boolean salvaCentroMonitoraggio(String nomeCentro, String indirizzo, String currentUsername)

Inserisce un nuovo centro di monitoraggio nel database e associa l'operatore corrente (identificato tramite username) al centro.

---

g. Ottenere dati geografici

- **Recupera tutte le coordinate:**

java

CopiaModifica

public List<Coordinate> getCoordinate() throws SQLException

Restituisce una lista di oggetti Coordinate con le informazioni di tutte le città.

- **Con filtro di nome:**

java

CopiaModifica

public List<Coordinate> getCoordinate(String text) throws SQLException

Filtra le coordinate basandosi sul nome della città.

- **Con tolleranza geografica:**

java

CopiaModifica

public List<Coordinate> getCoordinate(double latitude, double longitude, double tolerance) throws SQLException

Ritorna le coordinate comprese in un'area definita dalla latitudine, longitudine e tolleranza specificate.

---

h. Salvataggio dati climatici

java

CopiaModifica

public boolean salvaDatiClimatici(String[] parametro, ArrayList<String> valori, ArrayList<String> commenti, ArrayList<Integer> punteggi, String

username, long timestamp, Coordinate citta, String centro)

1. **Inserisce dati climatici nella tabella Parametro.**
2. **Salva una rilevazione associata ai dati climatici nella tabella Rilevazione.**

---

3. Gestione degli errori

- **Ogni operazione include gestione degli errori tramite try-catch per gestire eccezioni SQL e problemi di connessione.**
- **Transazioni:**
  - **Usa conn.commit() per salvare i cambiamenti.**
  - **Usa conn.rollback() per annullare transazioni in caso di errore.**

---

4. Metodi di supporto

- **Verifica se un valore esiste nel database:**

java

CopiaModifica

private boolean existsInDatabase(String field, String value) throws SQLException

Verifica se un valore specifico esiste in una colonna del database.

- **Ottenere codice fiscale:**

java

CopiaModifica

private String getCF(String username)

Recupera il codice fiscale di un operatore dato il suo username.

---

5. Ottenere dataset per grafici

java

CopiaModifica

public DefaultCategoryDataset getParametri(Coordinate city, TipiPlot type)

Crea un dataset per visualizzare dati climatici in un grafico. Filtra i dati in base alla città e al tipo di parametro (es., temperatura, pressione).

Classe Server

Descrizione Generale

La classe Server è responsabile dell'avvio e della gestione di un server RMI (**Remote Method Invocation**). Questo server consente ai client di interagire con il

database attraverso oggetti remoti esposti. La classe si occupa di configurare il registro RMI, registrare i servizi e fornire metodi per avviare e fermare il server.

Componenti Principali

1. Attributi

- **registry**:
  - **Tipo**: Registry
  - **Descrizione**: Gestisce il registro RMI utilizzato per pubblicare oggetti remoti.

Metodi

1. Costruttore: Server()

- **Descrizione**:
  - Avvia un thread per eseguire il server.
  - Richiama il metodo start() che esegue il codice nel metodo run().

---

2. run()

- **Descrizione**:
  - Metodo principale che avvia il server RMI.
  - Azioni principali:

- **Recupera la configurazione**:
  - Utilizza la classe Config per ottenere i dettagli di configurazione, come porta RMI (cf.getRmiPort()) e nome del servizio RMI (cf.getRmiDbName()).
- **Crea il registro RMI**:
  - Utilizza LocateRegistry.createRegistry() per avviare un registro RMI sulla porta configurata.
- **Espone il servizio RMI**:
  - Crea un'istanza di DatabaseImpl e lo registra come servizio remoto tramite Naming.rebind().
- **Output dei Log**:
  - Stampa messaggi per indicare lo stato del server (es. "Server RMI avviato").
- **Eccezioni**:
  - Gestisce eventuali errori durante l'inizializzazione del server e li stampa tramite e.printStackTrace().

---

3. stopServer()

- **Descrizione**:

- Metodo per fermare il server RMI e liberare le risorse.
- Azioni principali:
  . Recupera la configurazione (Config).
  . Utilizza il metodo registry.unbind() per rimuovere il servizio registrato.
  . Stampa un messaggio di conferma ("Server RMI fermato").
- **Eccezioni**:
  . Gestisce eventuali errori durante l'unbinding del servizio.

Flusso Operativo

1. **Avvio del Server**:
   - Quando viene creata un'istanza della classe Server, il thread viene avviato e il metodo run() registra il servizio RMI.
   - L'oggetto remoto DatabaseImpl diventa disponibile per i client.
2. **Fermare il Server**:
   - Il metodo stopServer() consente di fermare il server e liberare le risorse.

Classe Tools

Il codice definisce una classe Java chiamata Tools, progettata per semplificare l'assegnazione di parametri a oggetti PreparedStatement. Questo è utile quando si eseguono query SQL con parametri dinamici in Java.

Descrizione generale

- **La classe contiene un unico metodo statico: setParametri.**
- **Questo metodo accetta un oggetto PreparedStatement e una lista variabile di parametri (Object... params).**
- **A seconda del tipo di ciascun parametro, il metodo assegna il valore corrispondente alla posizione corretta nel PreparedStatement.**

Dettaglio del metodo

Dichiarazione del metodo

java

CopiaModifica

public static PreparedStatement setParametri(PreparedStatement query, Object... params)

- **Parametri:**
  - **query: un oggetto PreparedStatement su cui impostare i valori.**
  - **params: una lista variabile di oggetti che rappresentano i valori dei parametri.**
- **Restituisce: lo stesso oggetto PreparedStatement, con i parametri impostati, o null in caso di errore.**

Logica del metodo

1. **Iterazione sui parametri**

java

CopiaModifica

for (var parametro : params)

Il metodo scorre ciascun parametro fornito e determina il tipo di dato.

2. **Controllo dei tipi**
   - **Stringhe:**

java

CopiaModifica

query.setString(++i, (String) parametro);

Il metodo assegna stringhe al parametro corrispondente usando setString.

- Numeri (Double, Long, Integer):

java

CopiaModifica

```
query.setDouble(++i, (Double) parametro);
query.setLong(++i, (Long) parametro);
query.setInt(++i, (Integer) parametro);
```

- Date:

java

CopiaModifica

```
query.setDate(++i, (Date) parametro);
```

- BigDecimal e BigInteger:

java

CopiaModifica

```
query.setBigDecimal(++i, (BigDecimal) parametro);
query.setLong(++i, b.longValue());
```

- ArrayList:

java

CopiaModifica

```
for(var t:((ArrayList)parametro)){
   if(t.getClass() == String.class){
      query.setString(++i, (String) t);
```

```
    }else if(t.getClass() == Integer.class){
        query.setInt(++i, (Integer) t);
    }
}
```

Se il parametro è una lista (ArrayList), il metodo itera sui suoi elementi e imposta ciascuno di essi nel PreparedStatement.

3. Gestione degli errori
    - Se un parametro è null, viene sollevata un'eccezione:

java

CopiaModifica

```
if (parametro == null) {
    throw new Exception("parametro nullo");
}
```

    - Se il tipo del parametro non è supportato, viene sollevata un'altra eccezione:

java

CopiaModifica

```
throw new Exception("tipo non ancora implementato");
```

4. **Catch delle eccezioni Se si verifica un'eccezione durante l'esecuzione, viene stampato un messaggio di errore e il metodo restituisce null:**

java

CopiaModifica

System.err.println(e.getMessage());

return null;

Funzionalità chiave

- **Compatibilità multipla: Supporta diversi tipi di dati (String, Double, Long, Integer, Date, BigDecimal, BigInteger) e liste (ArrayList).**
- **Gestione automatica degli indici: L'indice del parametro viene incrementato automaticamente tramite ++i.**
- **Estendibilità: Può essere facilmente esteso per supportare nuovi tipi di dati.**
- **Gestione degli errori: Impedisce il passaggio di parametri null o non implementati, migliorando la robustezza.**