Laboratorio interdisciplinare A

Climate Monitoring

Un sistema di monitoraggio di parametri climatici

CATTANEO LUCA

FICARA PAOLO  
MAURI ANDREA

Sommario

[Introduzione 2](#_Toc136376529)

[Librerie esterne utilizzate 2](#_Toc136376530)

[ codice-fiscale-java-master 2](#_Toc136376531)

[ Java.nio 2](#_Toc136376532)

[Struttura generale del sistema di classi 2](#_Toc136376533)

[Gestione dei dati e rappresentazioni oggetti 2](#_Toc136376534)

[Gestione dell’interfaccia grafica 2](#_Toc136376535)

[Classi per la gestione dei dati 3](#_Toc136376536)

[InterestingAreas 3](#_Toc136376537)

[Metodi principali della classe: 3](#_Toc136376538)

[MonitoringStation 3](#_Toc136376539)

[User 3](#_Toc136376540)

[Metodi principali della classe: 4](#_Toc136376541)

[Forecast 4](#_Toc136376542)

[Metodi della classe: 4](#_Toc136376543)

[FileManager 5](#_Toc136376544)

[Metodi principali della classe: 5](#_Toc136376545)

[DatiCondivisi 6](#_Toc136376546)

[Metodi principali della classe: 6](#_Toc136376547)

# Introduzione

Climate monitoring è un progetto sviluppato nell’ambito del progetto di Laboratorio A 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 in NetBeans apposita per lo sviluppo del design delle applicazioni desktop, ed è stato sviluppato e testato su Windows 11.

# Librerie esterne utilizzate

Durante lo sviluppo di questo progetto è stata utilizzata una libreria di terze parti in aggiunta,

* codice-fiscale-java-master  
  Il codice sorgente è stato reperito in rete ed è stato adattato e trasformato in libreria. È incluso nel progetto ai fini di denominare il codice fiscale avendo tutte le generalità necessarie fornite dall’utente in fase di registrazione.
* Java.nio  
  (NIO sta per New Input/Output) è una raccolta di API del linguaggio di programmazione Java che offre funzionalità per operazioni di I/O intensive. È stato introdotto con la versione J2SE 1.4 di Java da Sun Microsystems per integrare un I/O standard esistente. Java NIO consente di eseguire IO non bloccanti. Ad esempio, un thread può chiedere a un canale di leggere i dati in un buffer. Mentre il canale legge i dati nel buffer, il thread può fare qualcos'altro. Una volta che i dati vengono letti nel buffer, il thread può continuare a elaborarli. Lo stesso vale per la scrittura dei dati nei canali. Questa libreria è stata aggiunta anche in ottica futura riguardo la programmazione concorrente.

# Struttura generale del sistema di classi

Nel progetto si possono suddividere le classi in due macro-rami: classi adibite per la gestione dei dati e la rappresentazione degli oggetti in oggetto e classi impiegate alla gestione dell’interfaccia grafica.

## Gestione dei dati e rappresentazioni oggetti

* DatiCondivisi
* FileManager
* Forecast
* InterestingAreas
* MonitoringStation
* User

## Gestione dell’interfaccia grafica

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

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

## Metodi principali della classe:

1. Metodo formCSV(String info)
   * Descrizione: Crea un oggetto User a partire da una stringa CSV contenente le informazioni.
   * Parametri:
     + info: la stringa CSV contenente le informazioni dell'utente
   * Ritorno: un nuovo oggetto User con le informazioni specificate
   * Complessità: O(1)

Nota: Tutti i metodi hanno complessità O(1), poiché eseguono operazioni dirette sugli attributi dell'oggetto, senza iterazioni o algoritmi complessi. Il metodo formCSV ha anche complessità O(1), in quanto suddivide la stringa CSV in un array e crea un nuovo oggetto User con i valori estratti dall'array.

# Forecast

Forecast è la classe che rappresenta una previsione metereologica fatta da un operatore.

L’oggetto viene identificato con i seguenti attributi:

* idCitta: l'ID della città rappresentato dal tipo String
* NomeStazione: il nome della stazione rappresentato dal tipo String
* data: la data della previsione rappresentato dal tipo Date
* ora: l'ora della previsione rappresentato dal tipo Date
* vento: i dati relativi al vento rappresentato da un vettore di String
* umidita: i dati relativi all'umidità rappresentato da un vettore di String
* pressione: i dati relativi alla pressione rappresentato da un vettore di String
* temperatura: i dati relativi alla temperatura rappresentato da un vettore di String
* precipitazioni: i dati relativi alle precipitazioni rappresentato da un vettore di String
* altitudine: i dati relativi all’latitudine dei ghiacciai rappresentato da un vettore di String
* massa: i dati relativi alla massa dei ghiacciai rappresentato da un vettore di String

## Metodi della classe:

1. Metodo formCSV(String info)
   * Descrizione: Crea un oggetto User a partire da una stringa CSV contenente le informazioni.
   * Parametri:
     + info: la stringa CSV contenente le informazioni dell'utente
   * Ritorno: un nuovo oggetto User con le informazioni specificate
   * Complessità: O(1)

Nota: Tutti i metodi hanno complessità O(1), poiché eseguono operazioni dirette sugli attributi dell'oggetto, senza iterazioni o algoritmi complessi. Il metodo formCSV ha anche complessità O(1), in quanto suddivide la stringa CSV in un array e crea un nuovo oggetto User con i valori estratti dall'array.

# FileManager

FileManager è una classe prettamente di metodi statici ed è una delle classi fulcro del progetto in quanto garantisce la lettura e scrittura su file. Sfrutta la classe “java.nio”, un set di API per la lettura e scrittura su file, in quanto fornisce metodi con prestazioni molto efficienti per appunto le due funzioni principali delle operazioni su file soprattutto per file di notevoli dimensioni. Le operazioni di lettura e scrittura da parte di questa classe sono solitamente eseguite utilizzando un buffer per memorizzare temporaneamente i dati da leggere o scrivere.

## Metodi principali della classe:

1. Metodo readUser(Path path)
   * Descrizione: Legge il file degli utenti e restituisce una lista di oggetti User.
   * Parametri:
     + path: il percorso del file da leggere
   * Ritorno: una lista di oggetti User letti dal file
   * Eccezioni:
     + IOException: se si verifica un errore di input/output durante la lettura del file
   * Complessità: O(n), dove n è il numero di righe nel file degli utenti
2. Metodo readAreas(Path path)
   * Descrizione: Legge il file delle aree interessanti e restituisce una lista di oggetti InterestingAreas.
   * Parametri:
     + path: il percorso del file da leggere
   * Ritorno: una lista di oggetti InterestingAreas letti dal file
   * Eccezioni:
     + IOException: se si verifica un errore di input/output durante la lettura del file
   * Complessità: O(n), dove n è il numero di righe nel file delle aree interessanti
3. Metodo readStation(Path path)
   * Descrizione: Legge il file delle stazioni di monitoraggio e restituisce una lista di oggetti MonitoringStation.
   * Parametri:
     + path: il percorso del file da leggere
   * Ritorno: una lista di oggetti MonitoringStation letti dal file
   * Eccezioni:
     + IOException: se si verifica un errore di input/output durante la lettura del file
   * Complessità: O(n), dove n è il numero di righe nel file delle stazioni di monitoraggio
4. Metodo readForecast(Path path)
   * Descrizione: Legge il file delle previsioni meteo e restituisce una lista di oggetti Forecast.
   * Parametri:
     + path: il percorso del file da leggere
   * Ritorno: una lista di oggetti Forecast letti dal file
   * Eccezioni:
     + IOException: se si verifica un errore di input/output durante la lettura del file
     + ParseException: se si verifica un errore durante il parsing delle date
   * Complessità: O(n), dove n è il numero di righe nel file delle previsioni meteo
5. Metodo write(String content, Path path)
   * Descrizione: Scrive il contenuto specificato nel file specificato.
   * Parametri:
     + content: il contenuto da scrivere nel file
     + path: il percorso del file in cui scrivere
   * Eccezioni:
     + IOException: se si verifica un errore di input/output durante la scrittura del file
   * Complessità: O(1)

Nota: I metodi readUser, readAreas, readStation e readForecast hanno complessità O(n), dove n è il numero di righe nel file corrispondente. Questo perché ogni riga viene letta e suddivisa in campi separati da ";", quindi viene creato un nuovo oggetto corrispondente e aggiunto alla lista. Il metodo write ha complessità O(1), poiché scrive direttamente il contenuto nel file senza iterazioni o algoritmi complessi.

# DatiCondivisi

DatiCondivisi è una classe Singleton che memorizza i vari contenuti dei file utili del progetto in delle liste apposite. Di per certo è la classe più importante del progetto in quanto offre i metodi principali per i vari calcoli delle distanze tra le città ed inoltre fornisce tutti i dati utili.

Possiede sei attributi:

* instance: identifica l’istanza della classe
* monitoringStations: una lista di MonitoringStation che identifica le stazioni di monitoraggio dove fare le rilevazioni
* users: una lista di User che identifica gli operatori registrati
* areas: una lista di InterestingAreas che identifica tutte le aree di interesse disponibili
* forecasts: una lista di Forecast che identifica tutte le rilevazioni fatte fino a quel momento
* operatore: l’operatore attualmente in sessione

## Metodi principali della classe:

1. Metodo getInstance()
   * Descrizione: Restituisce l'istanza unica della classe DatiCondivisi utilizzando il pattern Singleton.
   * Ritorno: l'istanza della classe DatiCondivisi.
   * Eccezioni:
     + IOException: se si verifica un errore di input/output durante la lettura dei file
     + ParseException: se si verifica un errore durante il parsing delle date
   * Complessità: O(1)
2. Metodo refresh()
   * Descrizione: Ricrea l'istanza della classe DatiCondivisi, aggiornando i dati leggendo nuovamente i file corrispondenti.
   * Eccezioni:
     + IOException: se si verifica un errore di input/output durante la lettura dei file
     + ParseException: se si verifica un errore durante il parsing delle date
   * Complessità: O(n), dove n è il numero di righe nei file delle stazioni di monitoraggio, degli utenti, delle aree di interesse e delle previsioni meteorologiche.
3. Metodo cercaAreaGeografica(String s)
   * Descrizione: Cerca l'area geografica corrispondente alla stringa specificata, che può essere un nome di città o una coppia di coordinate latitudine e longitudine.
   * Parametro:
     + s: una stringa che rappresenta l'area da cercare.
   * Ritorno: un array di stringhe che memorizza le città nell'area di interesse.
   * Complessità: O(n), dove n è il numero di aree di interesse.
4. Metodo cercaLimitrofo(double lat1, double lon1)
   * Descrizione: Cerca le città aree di interesse vicine a una determinata posizione geografica specificata dalle coordinate latitudine e longitudine.
   * Parametri:
     + lat1: la latitudine del punto di riferimento.
     + lon1: la longitudine del punto di riferimento.
   * Ritorno: una lista di stringhe che contiene le aree di interesse vicine.
   * Complessità: O(n), dove n è il numero di aree di interesse.
5. Metodo existForecast(String area)
   * Descrizione: Verifica se una determinata area di interesse ha delle previsioni meteorologiche.
   * Parametro:
     + area: l'area di interesse da verificare.
   * Ritorno: true se l'area ha delle previsioni, false altrimenti.
   * Complessità: O(n), dove n è il numero di previsioni meteorologiche.
6. Metodo getForecasts(String name)
   * Descrizione: Restituisce le previsioni meteorologiche per una determinata città.
   * Parametro:
     + name: il nome della città.
   * Ritorno: una lista di oggetti Forecast corrispondenti alle previsioni per la città specificata.
   * Complessità: O(n), dove n è il numero di previsioni meteorologiche.
7. Metodo sortAreas()
   * Descrizione: Ordina in modo crescente la lista delle aree di interesse in base al nome.
   * Complessità: O(n log n), dove n è il numero di aree di interesse. La complessità è data dal fatto che il metodo è costituito da un override di un quicksort