SIMONE PAGLINO - 736763 **STEFANO CASCAVILLA - 736820**

MANUALE TECNICO

Nel seguente manuale verranno elencate tutte le classi utilizzate per realizzare il progetto e per ognuna di esse verranno discussi i seguenti punti :

- Campi della classe ;
- Costruttori ;
- Metodi ;
- Eventuali considerazioni sul perché di tale scelta di realizzazione di tali metodi ;

N.B. => Per fare riferimento al file di testo contenete i comandi che il programma dovrà eseguire o il file di testo che conterrà gli eventi su cui produrre la tabella finale, verranno utilizzati i seguenti nomi: “comandi.txt” e “eventi.txt” (per ulteriori informazioni guardare il MANUALE UTENTE).

Command :

Classe che rappresenta una singola linea di testo letta dal file “comandi.txt”.

Questa classe è stata creata con il fine di semplificare il recupero dati di ogni comando in

tutte le altre classi, tenendo delle informazioni di tale comando all’interno dei suoi campi.

– Campi :

- *name_command* : Questo campo contiene al suo interno il nome del metodo del comando. (Es: comando = “import(eventi.txt)” => name_command = “

- *parameter* : Questo campo contiene al suo interno il valore dell’argomento del comando. (Es: comando = “import(eventi.txt)” => parameter = “eventi.txt”) .

Tutti i campi sono String objects dato che sono possono essere stringhe di testo

– Costruttori :

- *Command (String name_command, String parameter)* : Questo costruttore crea una nuova istanza della classe Command. Gli argomenti che vengono ricevuti in ingresso sono il nome del comando che sarà eseguito (name_command) e il valore dell’argomento di tale comando (parameter).

- Metodi :
  - *getNameCommand* () : Questo metodo restituisce il valore del campo "name_command" come String object.
    COMPLESSITA' : O(1) - cioè costante .
  - *getParameter* () : Questo metodo restituisce il valore del campo "parameter" come String object.
    COMPLESSITA' : O(1) - cioè costante .
  - *convertStringToCommand (String command*) : Questo metodo riceve come parametro la linea di testo letta da file (rappresentante un comando che andrà eseguito). Dopo aver verificato che la linea di testo letta sia un vero comando (richiamando il metodo della classe Validation specifico per questo compito), preleva le varie informazioni dalla linea di testo tramite opportune split, ottenendo il nome del metodo e il valore del suo argomento. Infine viene tornato un oggetto Command rappresentante la linea passata come parametro. Nel caso la validazione fallisse, verrà tornato un valore null.
    COMPLESSITA' : O(1) - cioè costante .

Event :

Classe che rappresenta una singola linea di testo letta dal file "eventi.txt".

Questa classe è stata creata con il fine di semplificare il recupero dati di ogni evento in tutte le altre classi, memorizzando le informazioni di tale evento all'interno dei suoi campi.

- Campi :
  - *(boolean) signup* : Questo campo vale true se l'evento indica che l'utente era registrato
    quando tale evento è stato creato (altrimenti false).
  - *(boolean) login* : Questo campo vale true se l'evento indica che l'utente era attivo quando tale evento è stato creato (altrimenti false).
  - *(Date) date* : Questo campo di tipo Date da riferimento ad un'istanza della classe Date rappresentante il timestamp della registrazione dell'evento in considerazione .
  - *(String) id* : Questo campo contiene l'id utente (codice alfanumerico di 5 caratteri) .
  - *(Coordinate) coord* : Questo campo di tipo Coordinate fa riferitento ad un'istanza della classe Coordinate rappresentante il la latitudine e longitudine della posizione dell'utente al momento della registrazione dell'evento in considerazione .
  - (*char) emot* : Questo campo è composto da un singolo carattere che può assumere solo predefiniti valori e per ognuno di essi indica lo stato d'animo che l'utente ha registrato per l'evento in considerazione (A = Arrabbiato, F = Felice, S = Sorpreso, T = Triste, N = Neutro). I caratteri predefiniti possono solo essere questi e maiuscoli.
    - Costruttore :

- Costruttori:
  - *Event(boolean signup, boolean login, Date date, String id, double lat,*

double lng, char emot) : Questo costruttore crea una nuova istanza della classe Event. I parametri ricevuti in ingresso sono medesimi a quelli descritti in precedenza nei campi (l'unica differenza è che vengono dati ricevuti in ingresso : latitudine e longitudine di tipo double per creare un'istanza di un oggetto Coordinate).

- Metodi:
  - *public boolean getSignup ()* : Questo metodo ritorna true se l'utente era registrato quando l'evento è stato creato (altrimenti false).
    COMPLESSITA' : O(1) - cioè costante .
  - *public boolean getLogin ()* : Questo metodo ritorna true se l'utente era attivo quando l'evento è stato creato (altrimenti false).
    COMPLESSITA' : O(1) - cioè costante .
  - *public Date getDate ()* : Questo metodo ritorna un oggetto Date rappresentante la data della creazione dell'evento.
    COMPLESSITA' : O(1) - cioè costante .
  - *public String getId ()* : Questo metodo ritorna l'id utente.
    COMPLESSITA' : O(1) - cioè costante .
  - *public Coordinate getCoord()* : Questo metodo ritorna un oggetto Coordinate rappresentante le coordinate relative alla posizione dell'utente quando l'evento è stato creato.
    COMPLESSITA' : O(1) - cioè costante .
  - *public char getEmotion ()* : Ritorna una lettera maiuscola predefinita per le emozioni.
    COMPLESSITA' : O(1) - cioè costante .

Read :

Classe che possiede diversi metodi per la lettura da file con uno specifico contenuto, dando la possibilità di ottenere una rappresentazione del contenuto del file letto più semplificato e comprensibile per il programma.

- Metodi (tutti statici) :
  - *public static ArrayList<Command> readCommands (String name_file)* : Questo metodo dopo aver ricevuto in ingresso come parametro il nome del file da cui leggere, ne legge il contenuto riga per riga. Ad ogni riga letta viene chiamato un metodo della classe Validation che permette di verificare che la linea di testo letta (rappresentante un comando), sia valida. In caso positivo, viene generato un oggetto Command e inserito in un ArrayList con tipo parametrizzato Command, altrimenti viene ignorato. Infine il metodo ritorna un ArrayList<Command> contenente tutti i comandi letti da file che verranno poi eseguiti nel metodo main della classe EmotionalMaps (altrimenti una lista vuota).
    COMPLESSITA' : θ(n) - dove n è il numero di righe del file letto.

    Considerazioni : La struttura dati utilizzata è una ArrayList<E> poiché non c'era nessuna particolare necessità di memorizzare i dati in un modo specifico o per altre ragioni. Per tanto dato che l'obiettivo era quello di memorizzare i dati in una struttura che non avesse una lunghezza prefissata e che fosse dinamica oltre che semplice, è stato scelto di utilizzare l'ArrayList<E>.

    Per ulteriori informazioni consultare javadoc su ArrayList .

N.B. => Questo metodo può essere utilizzato solamente per la lettura di file di testo contenenti uno o più comandi (comandi.txt). Inoltre nel caso si verifichino alcune eccezioni come per la lettura da file, il programma le gestirà stampando a video un relativo messaggio di avviso per l'utente.

- *public static ArrayList<Event> readEvents (String name_file) :* Questo metodo dopo aver ricevuto in ingresso come parametro il nome del file da cui leggere, ne legge il contenuto riga per riga. Ad ogni riga letta vene chiamato un metodo della classe Validation che permette di verificare che la linea di testo letta (rappresentante un evento), sia valida. In caso positivo, viene generato un oggetto Event e inserito in un ArrayList con tipo parametrizzato Event, altrimenti viene ignorato. Infine il metodo ritorna un ArrayList<Event> contenente tutti gli eventi letti (altrimenti una lista vuota).
  COMPLESSITA' : θ(n) - dove n è il numero di righe del file letto.

  Considerazioni : La struttura dati utilizzata è una ArrayList<E> poiché non c'era nessuna particolare necessità di memorizzare i dati in un modo specifico o per altre ragioni. Per tanto dato che l'obiettivo era quello di memorizzare i dati in una struttura che non avesse una lunghezza prefissata e che fosse dinamica oltre che semplice, è stato scelto di utilizzare l'ArrayList<E>.

  Per ulteriori informazioni consultare javadoc su ArrayList .

N.B. => Questo metodo può essere utilizzato solamente per la lettura di file di testo contenenti uno o più eventi (eventi.txt). Inoltre nel caso si verifichino alcune eccezioni come per la lettura da file, il programma le gestirà stampando a video un relativo messaggio di avviso per l'utente.

- *public static ArrayList<PointOfInterest> readPOI (String name_file) :* Questo metodo dopo aver ricevuto in ingresso come parametro il nome del file da cui leggere, ne legge il contenuto riga per riga (il nome del file viene inserito direttamente nel codice usando hard coding). Adogni riga letta viene chiamato un metodo della classe Validation che permette di verificare che la linea di testo letta (rappresentante un punto di interesse), sia valida. In caso positivo, viene generato un oggetto PointOfInterest e inserito in un ArrayList con tipo parametrizzato PointOfInterest, altrimenti viene ignorato. Infine il metodo ritorna un ArrayList< PointOfInterest > contenente tutti i punti di interesse letti (altrimenti una lista vuota).

  COMPLESSITA' : θ(n) - dove n è il numero di righe del file letto.

N.B. => Questo metodo può essere utilizzato solamente per la lettura del file di testo (poi.txt) contenenti uno o più punti di interesse validi. Inoltre nel caso si verifichino alcune eccezioni come per la letturada file, il programma le gestirà stampando a video un relativo messaggio di avviso per l'utente. Pur essendoci alcuni controlli basilari, si richiede che almeno i punti di interesse predefiniti siano validi e corretti.

- Considerazioni : La struttura dati utilizzata è una ArrayList<E> poiché non c'era nessuna particolare necessità di memorizzare i dati in un modo specifico o per altre ragioni. Per tanto dato che l'obiettivo era quello di memorizzare i dati in una struttura che non avesse una lunghezza prefissata e che fosse dinamica

oltre che semplice, è stato scelto di utilizzare l'ArrayList<E>.

Per ulteriori informazioni consultare javadoc su ArrayList .

EmotionalMaps:

Classe che possiede i metodi necessari all'esecuzione dell'intero programma.

- Metodi :
  - *public static void main(String[] args) :* Questo è il metodo principale per l'esecuzione del programma. Vengono inizializzati tre ArrayList (PointOfInterest, Command, Event). In seguito viene letto e memorizzato il contenuto del file "poi.txt". Successivamente viene recuperato il nome del file contenente i comandi che poi dovranno essere eseguiti e li memorizza nel secondo ArrayList. Infine il programma esegue ogni comando letto da file ieraticamente chiamando il metodo *callMethod*.

    COMPLESSITA' : O(n*m*w) - dove n è il numero totale di POI, m è il numero totale di eventi validi, w è il numero numero totale di comandi validi.
  - *public static void callMethod (ArrayList<Event> listEvent, ArrayList<PointOfInterest> listPoi, Command comm) :* Questo metodo esegue il comando ricevuto come parametro (comm). Se il nome del metodo contenuto nell'oggetto Command è "import", il metodo legge dal file (il cui nome è contenuto all'interno dell'oggetto Command passato come parametro), tutti gli eventi e li memorizza nell'ArrayList passato come parametro dal metodo main (al fine di tenere traccia di tutti quanti gli eventi). Se il nome del metodo contenuto nell'oggetto Command è "create_map", il metodo chiama "generateFilteredEmotionalMaps" poi "generateFullEmotionalMaps" (della classe "GenerateEmotionalMaps"), i quali stampano a video due tabelle (le cui descrizione viene riportata nella descrizione della classe GenerateEmotionalMaps).

    COMPLESSITA' : O(n*m) - dove n è il numero totale di POI, m è il numero totale di eventi validi.
- Considerazioni : La struttura dati utilizzata è una ArrayList<E> poiché non c'era nessuna particolare necessità di memorizzare i dati in un modo specifico o per altre ragioni. Per tanto dato che l'obiettivo era quello di memorizzare i dati in una struttura che non avesse una lunghezza prefissata e che fosse dinamica oltre che semplice, è stato scelto di utilizzare l'ArrayList<E>.

Date :

Classe che rappresenta una data (composta da giorno, mese e anno).

- Campi :
  - *(int) gg* : Questo campo contiene il valore numerico del giorno del mese.
  - *(int) mm* : Questo campo contiene il valore numerico del mese.
  - *(int) yyyy* : Questo campo contiene il valore numerico dell'anno.

- Costruttori:

  Date(int gg, int mm, int yyyy) : Questo costruttore crea una nuova istanza della classe Date. I parametri ricevuti in ingresso sono medesimi a quelli descritti in precedenza nei campi.

- Metodi:
  - *public int getDay ()* : Questo metodo ritorna il valore numerico del giorno del mese dell'oggetto Date.
    COMPLESSITA' : O(1) - cioè costante .
  - *public int getMonth ()* : Questo metodo ritorna il valore numerico del mese dell'oggetto Date.
    COMPLESSITA' : O(1) - cioè costante .
  - *public int getYear ()* : Questo metodo ritorna il valore numerico dell'anno dell'oggetto Date.
    COMPLESSITA' : O(1) - cioè costante .
  - *public static boolean isFormatDate(String timestamp)* : Questo metodo ritorna true se il timestamp di tipo String è valido (altrimenti false).
    Prima verifica che la sua lunghezza sia di 8 caratteri. In seguito viene verificato che ci siano solo valori numerici all'interno del timestamp. Infine viene richiamato il metodo "isDate" per verificare che il timestamp rappresenti una data vera (controlla la coerenza del valore numerico del valore del giorno del mese, del mese e anno).
    COMPLESSITA' : O(1) - cioè costante .
  - *public static boolean isDate(int d, int m, int y)* : Questo metodo ritorna true se i valori dei parametri passati rappresentano un valido oggetto Date, ovvero una data valida (altrimenti false). Prima controlla che il valore dell'anno sia compreso fra il 1900 e l'anno corrente (la data corrente viene prelevata tramite il metodo "getCurrentDate()"). In seguito confronta che il valore numerico del numero del giorno del mese sia compreso tra 1 e 31 e che sia coerente con il valore del mese. Per il mese di febbraio viene verificato che l'anno sia bisestile o meno.
    COMPLESSITA' : O(1) - cioè costante .
  - *public static Date getCurrentDate ()* : Questo metodo ritorna un oggetto Date della data del giorno corrente.
    COMPLESSITA' : O(1) - cioè costante .
  - *public static boolean isBisestile(int y) :* Questo metodo ritorna true se l'anno ricevuto come parametro è bisestile (altrimenti false). Il procedimento per verificare che l'anno sia bisestile viene riportato in modo esaustivo nei commenti all'interno della classe e a questo link (https:// support.microsoft.com/en-au/help/214019/method-to- determine-whether-a-year-is-a-leap-year).
    COMPLESSITA' : O(1) - cioè costante .
  - *public static int compareTo(Date d1, Date d2) :* Questo metodo ritorna 1 se la prima data è maggiore della seconda. Nel caso opposto -1 e nel caso siano uguali 0. Il metodo confronta dapprima il valore numerico dell'anno della prima e seconda data. In caso la prima data abbia il valore dell'anno maggiore della seconda viene ritornato 1. Invece se l'anno di entrambe le date risulta essere uguale, allora si confrontano il valore del mese della prima data con quello della seconda. Se il mese della prima data è maggiore, allora vuol dire che la data è maggiore e viene tornato 1. In seguito se il giorno della prima data è maggiore della seconda, allora vuol dire che la prima data è maggiore e viene tornato
    1. Se le date hanno giorno uguale, allora sono identiche e viene tornato 0. In ogni altro caso viene tornato -1.
    COMPLESSITA' : O(1) - cioè costante .
  - *public static Date getDateObj(String date) :* Questo metodo ritorna un oggetto Date rappresentante il timestamp ricevuto come parametro (String date). Il metodo ottiene il valore numerico del giorno del mese recuperando

le prime due cifre dal timestamp, per il valore mese preleva le due successive e per l'anno le restanti. Infine crea e ritorna una nuova istanza di un oggetto Date con i dati appena ottenuti. (Dato che ogni valore prelevato è di tipo String, si utilizza il metodo parseInt della classe Integer per convertirli nel tipo primitivo int).
COMPLESSITA' : O(1) - cioè costante .

Validation :

Classe che fornisce una serie di metodi per la validazione di oggetti come Command, Event, PointOfInterest.

- Metodi :
  - *public static boolean isThereCharacter(String str) :* Questo metodo ritorna true se la stringa passata come parametro contiene almeno un carattere (altrimenti false).
    COMPLESSITA' : θ(n) - dove n è il numero di caratteri che compongono la stringa passata per parametro.
  - *public static boolean validatePOI(String poi) :* Questo metodoritorna true se la stringa passata come parametro (letta dal file poi.txt) rappresenta un punto di interesse valido (altrimenti false). Inizialmente il metodo separa il nome del punto di interesse dalle sue coordinate (mediante una split sul carattere "-"). In seguito controlla che la latitudine e longitudine siano composte solo da numeri.
    Successivamente vengono ottenuti i valori singoli di latitudine e longitudine e li converte in double. Infine se le due coordinate risultano essere double (non generando eccezioni particolari), la validazione è terminata con successo.
    COMPLESSITA' : θ(n) - dove n è il numero di caratteri che compongono la stringa "name".
  - *public static boolean validateCommand(String comm) :* Questo metodo ritorna true se la stringa passata come parametro (letta dal file comandi.txt) rappresenta un comando valido (altrimenti false). Il metodo separa il nome del comando letto dal valore del suo parametro. In seguito viene chiamato il metodo *validateCommandPossibleScenario* per verificare che il nome del comando letto e i valore del suo parametro siano validi. In caso affermativo viene tornato true, altrimenti false.
    COMPLESSITA' : O(1) - cioè costante .
  - *public static boolean validateCommandPossibleScenario(String comm, String str_method, String str_param) :* Questo metodo ritorna true se la stringa passata come parametro (letta dal file comandi.txt), il nome del comando e il valore del suo parametro sono validi. Il metodo distingue due casi in base al nome dei due metodi e per ognuno di essi viene chiamato un metodo di validazione differente. In caso la validazione sia un successo il valore ritornato sarà true.
    COMPLESSITA' : O(1) - cioè costante .
  - *public static boolean validateImport(String param) :* Questo metodo ritorna true se il valore del parametro del comando import (passato al seguente metodo), è valido (altrimenti false). Il metodo controlla che nel valore del parametro sia contenuta la stringa ".txt" e che il prima carattere sia diverso da un punto ".". Un valore valido per "param" sarebbe: "eventi.txt".
    COMPLESSITA' : O(1) - cioè costante .
  - *public static boolean validateCreateMap(String param) :*Questo metodo ritorna true se il valore del parametro del comando create_map

(passato al seguente metodo), è valido (altrimenti false). Il metodo effettua dei controlli per i due timestamp passati come singolo parametro richiamando due volte il metodo *isFormateDate* della classe Date.
COMPLESSITA' : O(1) - cioè costante .

- *public static boolean validateElementsLine(String elem_line)* : Questo metodo ritorna true se l'evento letto da file e passato come parametro è valido oppure meno (altrimenti false). Il metodo esegue una split sugli spazi per ottenere le singole informazioni che compongono un evento. In seguito per ognuno di essi in modo iterativo, chiama il chiama il metodo *listEventPossibleScenarios* che valida la singola informazione dell'evento tornando true in caso positivo o false in caso negativo.
  COMPLESSITA' : O(1) - cioè costante .
- *public static boolean validateUserId(String id) :* Questo metodo ritorna true se il valore dell'id utente è valido (altrimenti false). La lunghezza dell'id dev'essere di 5 caratteri alfanumerici (non sono inclusi caratteri oltre ai seguenti a-z, A-Z, 0-9).
  COMPLESSITA' : O(1) - cioè costante .
- *public static boolean validateCoordinates(String coord) :* Questo metodo ritorna true se il valore delle coordinate passate come parametro è valido (altrimenti false). Il metodo separa la latitudine e longitudine basandosi sulla ",". L'array ottenuto dev'essere formato da due informazioni, ovvero latitudine e longitudine. Se la latitudine è compresa tra -90 e 90 e se la longitudine è compresa tra -180 e 180, e se entrambi i valori possono essere convertiti in double, allora viene restituito come valore true (altrimenti false).
  COMPLESSITA' : O(1) - cioè costante .
- *public static boolean validateEmotion(char em) :* Questo metodo ritorna true se il carattere rappresentante l'emozione provata quando l'evento è stato creato è compresa nei valori predefiniti (A, F, S, T, N). Sono accettati solo valori maiuscoli (uppercase).
  COMPLESSITA' : O(1) - cioè costante .
- *public static boolean listEventPossibleScenarios(String el, int counter) :* Questo metodo ritorna true se il valore (*el*) contenuto all'interno di un evento è valido (altrimenti false). Il metodo tramite uno switch riesce a differenziare i vari casi, identificando grazie ad un indice *counter* la tipologia di dato da validare. Se nessuno dei casi si verifica (ovvero se un evento è formato da 7 elementi invece che da 6), allora l'evento non è valido.
  COMPLESSITA' : O(1) - cioè costante .

Emotions:

Classe che identifica le emozioni (A, F, S, T, N) e tiene conto del numero di occorrenze per ognuna di esse.

- Campi:
  - countA: Questo campo identifica il numero di occorrenze per l'emozione A (Arrabbiato).
  - countF: Questo campo identifica il numero di occorrenze per l'emozione F (Felice).
  - countS: Questo campo identifica il numero di occorrenze per l'emozione S (Sorpreso).
  - countT: Questo campo identifica il numero di occorrenze per

l'emozione T (Triste).
  - countN: Questo campo identifica il numero di occorrenze per l'emozione N (Neutro).

- Considerazioni: Tutti questi campi sono di tipo int dato che identificano un contatore. Avrei potuto utilizzare tipi che occupassero meno di 32 bit in memoria ma così facendo, se il numero di eventi ed emozioni relative fosse molto alto, andrei a inserire potenziali rischi per quanto riguarda i valori rappresentabili. Con 32 bit è possibile rappresentare 2^32, il che è un numero molto alto.

- Costanti: Per evitare l'utilizzo di hard-coding nel metodo increase(), è stato deciso di creare queste costanti di tipo char che verranno usate nel metodo increase() come sorgente di confronto.

- Costruttori:
  - Emotions(): Questo costruttore inizializza tutti e 5 i campi count* a 0. Questo perché quando sarà necessario incrementare i valori di un qualsiasi campo, questo deve essere inizializzato ad un valore iniziale (in questo caso 0).

- Metodi:
  - numeroOccorrenze(): Questo metodo inizializza una variabile di tipo HashMap dove verranno inseriti item del tipo key: Character, value: Integer. Nell'HashMap vengono inseriti 5 item, uno per ogni emozione valida. Le keys sono A, F, S, T, N e i valori saranno le relative occorrenze. Il metodo restituisce l'HashMap popolato.
    COMPLESSITA' : O(1) - cioè costante .
  - increase(char emotion): Questo metodo accetta come parametro una variabile emotion di tipo char. La variabile rappresenta l'emozione di cui si vuole incrementare il valore. Per evitare problemi legati al case sensitive, rendo il carattere uppercase per poi confrontarlo con i caratteri validi (le 5 emozioni valide). Se combacia con uno di questi caratteri validi, il metodo incrementa il numero di occorrenze dell'emozione identificata dal parametro emotion.
    COMPLESSITA' : O(1) - cioè costante .

    - Considerazioni: Abbiamo deciso di utilizzare la struttura dati HashMap perché è quella che meglio rappresenta una struttura key-value. In una struttura key-vale ogni elemento ha due componenti: una chiave (key) che serve per identificare l'elemento e un valore (value). Per ulteriori informazioni consultare javadoc HashMap

Coordinate:

Classe che identifica le coordinate longitudine e latitudine di un qualsiasi punto sulla terra

- Campi:
  - longitude: Questo campo identifica la longitudine.
  - latitude: Questo campo identifica la latitudine.

- Considerazioni: Entrambi i campi sono di tipo BigDecimal e non double o float perché, dato che le coordinate long e lat sono numeri decimali in alcuni casi con molte cifre, il tipo double e float risulta essere carente nel momento in cui la precisione di rappresentazione è alta. Quindi per evitare perdita di informazione, abbiamo optato per il tipo BigDecimal.

- Costruttori:
  - Coordinate(double longitude, double latitude): Questo costruttore inizializza i due campi longitude e latitude con i relativi valori double passati come parametro. In fase di assegnamento, i valori double verrano wrappati all'interno delle variabili di tipo BigDecimal.

- Metodi:
  - getLongitude(): Questo metodo restituisce il campo longitude associato al punto in questione.
    COMPLESSITA' : O(1) - cioè costante .
  - getLatitude(): Questo metodo restituisce il campo latitude associato al punto in questione.
    COMPLESSITA' : O(1) - cioè costante .
  - getDistanceBetweenTwoPoints(Coordinate point): Questo metodo calcola la distanza tra il punto in questione e un altro punto di tipo Coordinate passato come parametro con il nome point. Per prima cosa si calcola p1, cioè la differenza di latitudine tra il punto in questione e point (la latitudine è sull'asse orizzontale quindi è come se stessimo calcolando la differenza di ascisse); poi si procede con il calcolo di p2, cioè la differenza di longitudine tra il punto in questione e point (la longitudine è sull'asse verticale quindi è come se stessimo calcolando la differenza di ordinate). A questo punto, si può calcolare la distanza tra i due punti facendo ricorso alla formula di geometria analitica per il calcolo della distanza fra due punti: sqrt(p1^2 + p2^2). Questo risultato è il valore restituito al chiamante dal metodo.
    COMPLESSITA' : O(1) - cioè costante .

PointOfInterest:
Classe che identifica un punto di interesse (POI)

- Campi:
  - pointCoordinate: Questo campo identifica le coordinate (latitudine e longitudine) associate al punto di interesse in questione.
  - relatedEmotions: Questo campo identifica le emozioni associate al punto di interesse in questione.
  - poiName: Questo campo identifica il nome del punto di interesse in questione.

- Costruttori:
  - PointOfInterest(double longitude, double latitude, String name):

Questo costruttore inizializza il campo pointCoordinate con un oggetto di tipo Coordinate al quale vengono passate longitudine e latitudine passate come parametro; inizializza il campo poiName con la stringa passata per parametro; inizializza il campo relatedEmotions con un oggetto di tipo Emotions.

- Metodi:
  - getCoordinatePOI(): Questo metodo restituisce il campo pointCoordinate associato al punto di interesse in questione.
    COMPLESSITA' : O(1) - cioè costante .
  - getPOIName(): Questo metodo restituisce il campo poiName associato al punto di interesse in questione.
    COMPLESSITA' : O(1) - cioè costante .
  - getRelatedEmotions(): Questo metodo restituisce il campo relatedEmotions associato al punto di interesse in questione.
    COMPLESSITA' : O(1) - cioè costante .
  - getTotalEvents(): Questo metodo, dopo aver preso la struttura dati HashMap che identifica le varie occorrenze delle emozioni associate al punto di interesse in questione, per ogni item presente nella struttura HashMap, prende il valore (value) e, dato che si tratta di un valore numerico intero, lo va a sommare ad una variabile ausiliaria nominata numeroOccorrenze. numeroOccorrenze, alla fine del ciclo for, verrà restituita al chiamante (questa variabile identifica tutti gli eventi associati al punto di interesse in questione dato che ad ogni evento è associata una e una sola emozione).
    COMPLESSITA' : O(1) - cioè costante .

GenerateEmotionalMaps

Classe che contiene i metodi necessari per la creazione delle due mappe emozionali richieste

- Metodi:
  - generateFullEmotionalMaps(ArrayList eventsList, ArrayList pointsOfInterest, Date firstLimit, Date secondLimit): Questo metodo inizializza un array di double (distances) che servirà per memorizzare le distanze di un evento da ogni POI. Poi il metodo inizializza una variabile (pointOfInterestIndex) che conterrà l'indice in cui si trova la distanza minore (nell'array distances). Esiste però una corrispondenza tra gli indici di distances e quelli di pointsOfInterest. es: distances[2] = distanza dell'evento dal POI presente in pointsOfInterest[2]. A questo punto il metodo inizializza un ArrayList di eventi in qui verranno filtrati tutti gli eventi presenti in eventsList e scelti solo quelli che sono stati fatti tra firstLimit e secondLimit (rappresentano due date). Viene utilizzato un ArrayList perché risulta la struttura dati migliore per gestire un array senza conoscerne la dimensione iniziale (questo array è dinamico). A questo punto parte un ciclo che scandisce tutti gli eventi validi (cioè quelli che hanno passato la filtrazione); per ognuno di questi c'è un altro ciclo che va a calcolare, per ogni POI, la distanza tra il luogo

in cui è stato registrato l'evento e il POI stesso. Queste distanze vengono messe nell'array distances. A questo punto viene memorizzato l'indice di distances in cui si trova il valore minore (distanza minima) e il metodo prende il POI che si trova in posizione pointOfInterestIndex per poi andare ad incrementare il contatore dell'emozione associata all'evento che nel seguente momento è in fase di valutazione nel ciclo. Alla fine del ciclo sugli eventi, viene eseguito un ultimo ciclo che, per ogni POI, chiama il metodo printSinglePOILine che stampa la riga con le relative informazioni sul POI.

COMPLESSITA' : O(n*m) dove n è il numero di eventi totali, mentre m è il numero totale di POI(Point Of Interests).

- generateFilteredEmotionalMaps(ArrayList eventsList, ArrayList pointsOfInterest, Date firstLimit, Date secondLimit): Questo metodo inizializza un array di double (distances) che servirà per memorizzare le distanze di un evento da ogni POI. Poi il metodo inizializza una variabile (pointOfInterestIndex) che conterrà l'indice in cui si trova la distanza minore (nell'array distances). Esiste però una corrispondenza tra gli indici di distances e quelli di pointsOfInterest. es: distances[2] = distanza dell'evento dal POI presente in pointsOfInterest[2]. A questo punto il metodo inizializza un ArrayList di eventi in qui verranno filtrati tutti gli eventi presenti in eventsList e scelti solo quelli che sono stati fatti tra firstLimit e secondLimit (rappresentano due date) da utenti attivi. Viene utilizzato un ArrayList perché risulta la struttura dati migliore per gestire un array senza conoscerne la dimensione iniziale (questo array è dinamico). A questo punto parte un ciclo che scandisce tutti gli eventi validi (cioè quelli che hanno passato la filtrazione); per ognuno di questi c'è un altro ciclo che va a calcolare, per ogni POI, la distanza tra il luogo in cui è stato registrato l'evento e il POI stesso.
  Queste distanze vengono messe nell'array distances. A questo punto viene memorizzato l'indice di distances in cui si trova il valore minore (distanza minima) e il metodo prende il POI che si trova in posizione pointOfInterestIndex per poi andare ad incrementare il contatore dell'emozione associata all'evento che nel seguente momento è in fase di valutazione nel ciclo. Alla fine del ciclo sugli eventi, viene eseguito un ultimo ciclo che, per ogni POI, chiama il metodo printSinglePOILine che stampa la riga con le relative informazioni sulPOI.

  COMPLESSITA' : O(n*m) dove n è il numero di eventi totali, mentre m è il numero totale di POI(Point Of Interests).
- filterDates(ArrayList eventsList, Date firstLimit, Date secondLimit): Questo metodo inizializza un ArrayList nel quale verranno memorizzati solamente gli eventi che rispettano il filtro che il metodo applicherà successivamente a tutti gli eventi (eventsList). Viene utilizzato un ArrayList perché risulta la struttura dati migliore per gestire un array senza conoscerne la dimensione iniziale (questo array è dinamico). A questo punto viene eseguito un ciclo for su tutti gli eventi (eventsList) e, per ogni evento, si controllerà che questo sia stato eseguito tra firstLimit e secondLimit (due date). Quelli che rispettano questo filtro, verranno aggiunti all'array returnArray, gli altri no. Alla fine dell'iterazione l'array returnArray viene restituito al chiamate.

  COMPLESSITA' : θ(n) - dove n è il numero di elementi contenuti nell'array "eventsList".
- filterActiveUsers(ArrayList eventsList): Questo metodo inizializza un

ArrayList nel quale verranno memorizzati solamente gli eventi che rispettano il filtro che il metodo applicherà successivamente a tutti gli eventi (eventsList). Viene utilizzato un ArrayList perché risulta la struttura dati migliore per gestire un array senza conoscerne la dimensione iniziale (questo array è dinamico). A questo punto viene eseguito un ciclo for su tutti gli eventi (eventsList) e, per ogni evento, si controllerà che sia stato registrato da un utente attivo. Quelli che rispettano questo filtro, verranno aggiunti all'array returnArray, gli altri no. Alla fine dell'iterazione l'array returnArray viene restituito al chiamate.

COMPLESSITA' : θ(n) - dove n è il numero di elementi contenuti nell'array "eventsList".

- findSmallestIndex(double[] distances): Questo metodo inizializza una variabile int (index) a 0 che servirà a tenere traccia dell'indice dell'elemento minore nell'array distances. Poi viene inizializzato una variabile double che conterrà il valore minore nell'array distances. A questo punto viene eseguito un ciclo for su ogni elemento distances e, ad ogni iterazione, verrà fatto il confronto tra l'i-esimo elemento e il min. Se l'i-esimo risulta essere minore, index e min verranno aggiornate di conseguenza. Alla fine del ciclo viene restituito l'indice dell'elemento minore al chiamate.

  COMPLESSITA' : θ(n) - dove n è il numero di elementi contenuti nell'array "distances".
- printSinglePOILine(PointOfInterest singlePOI, int totalEvents): Questo metodo inizializza una variabile di tipo HashMap (emotions) che contiene le varie emozioni associate a singlePoi. Se totalEvents (numero totale di eventi associati a singlePoi) è uguale a 0, viene stampata la linea descritta nella documentazione del progetto con tutti i valori percentuali a 0. Viene fatto in modo manuale e senza alcun calcolo perché, seguendo la logica descritta nella seconda parte del metodo, il risultato prodotto sarebbe NaN(Not a Number) a causa di una divisione 0/0. Se invece totalEvents è diverso da 0, viene stampata la linea descritta nella documentazione del progetto con i valori percentuali calcolati tramite questa espressione: ((occorrenze emozione X)/(numero totali eventi))*100.

  COMPLESSITA' : O(1) - cioè costante.