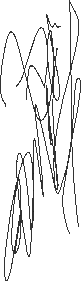
**COMANDO**

Classe dove vengono salvare in modo astratto le informazioni da richiamare e presenta i comandi fondamentali di **EmotionalMaps** che sono rispettivamente **TipoComando tipo** con il metodo **getTipo(),** e lo **String argomento** con il metodo **getArgomento().**

**PUNTO\_DI\_INTERESSE**

In questa classe vengono istanziate le coordinate dei punti di interesse con le emozioni totali e le emozioni di utenti attivi.

Viene generato un nuovo punto di interesse, con relativo nome e coordinata, e vengono istanziati a 0 i valori delle emozioni dei punti di interesse.

* Metodo **getEventiTotali()**: restituisce il totale di tutti gli eventi.
* Metodo **getPuntoDiInteresse()**: restituisce i vari punti di interesse.
* Metodo **resetStatistiche()**:resetta le statistiche dei punti di interesse.
* Metodo **statisticheAttivi()**: crea una stringa che contiene una percentuale di tutte le emozioni dei soli utenti attivi.
* Metodo **statisticheTotali()**: fa la stessa cosa del metodo precedente, ma per tutti gli utenti, qualunque sia lo stato.
* Metodo **aggiungiEvento()**: preleva l’evento e incrementa la variabile **eventiTotali**. Se il tipo di accesso è uguale a **login**, si aggiorna **eventiTotaliAttivi** quindi viene utilizzato uno **switch** per i vari stati emotivi. In base allo stato emotivo viene incrementato il rispettivo contatore totale e viene eseguito un controllo per verificare lo stato di accesso, se è nuovamente **login,** viene incrementato il contatore per lo stato emotivo di utenti attivi. Se lo stato emotivo non è tra quelli presenti viene stampato un messaggio di AVVERTIMENTO che specifica l’ID dell’evento.
* Metodo **calcolaDistanza()**: controlla la distanza assoluta, successivamente viene richiamato il metodo **distanza()** della classe UTILITA’ che restituisce la distanza assoluta tra il punto di interesse e le coordinate passate.
* Metodo **aPercentualeTotale()**: nel caso in cui EventiTotali è diverso da 0 restituisce un valore **intero** (arrotondato), altrimenti restituisce 0.
* Metodo **aPercentualeAttivi()**: nel caso in cui EventiTotaliAttivi è diverso da 0 restituisce un valore **intero** (arrotondato), altrimenti restituisce 0.

**UTILITÀ**

È la classe in cui vengono implementati i metodi di uso comune nel programma.

Per prima cosa viene passato il file: se non ci sono impedimenti nell’accesso viene caricata la stringa nel BufferedReader e successivamente viene aggiunta nell’ArrayList, altrimenti viene generato un AVVERTIMENTO. Nel caso in cui l’ArrayList risulta vuoto viene generato un AVVERTIMENTO.

* Metodo **boolean bisestile()**: restituisce TRUE se l’anno è bisestile.
* Metodo **controllaArrayStringhe():** riceve in input l’array di tipo String, e controlla se è null o se ha lunghezza 0. Tramite un ciclo FOR controlla se al suo interno presenta valori null oppure se è vuoto.
* Metodo **suddividiStringhe()**: riceve in input un array di String chiamato lista che viene controllato tramite il metodo **controllaArrayStringhe(). Successivamente si** crea una matrice dove i risultati vengono salvati al suo interno fin quando non viene riscontrata una virgola.
* Metodo **aggiungiEstensione()**: riceve in input nomefile e controlla se l’estensione è corretta. Nel caso in cui non sia presente viene aggiunto automaticamente **“.txt”**.
* Metodo **distanza():** vengono passate le coordinate di due punti, controllando se la variabile x del primo punto è maggiore di quella del secondo, successivamente viene creata una variabile temporanea con il valore di x1. Al valore di x1 viene assegnato il valore di x2 e successivamente al valore di x2 assegnato il valore di tmp. Lo stesso controllo viene effettuato con le coordinate y. Infine viene calcolata la distanza assoluta.

**MANAGER\_PDI (in questa classe tutti i metodi implementati sono di tipo void)**

Viene creta una lista di tipo **<PuntoDiInteresse>**.

Viene creato il metodo **ManagerPDI()** dove la lista viene istanziata come ArrayList di tipo **<PuntoDiInteresse>**. Vengono inseriti nella lista 3 punti di interesse, POI1, POI2, POI3 con le loro relative coordinate mediante il metodo **aggiungiPDI()** il quale inserisce all’interno dell’ArrayList il punto di interesse.

* Metodo **elabora()**: riceve in input l**’Evento e**

All’interno vengono istanziati:

* Indice
* IndicePDI
* Distanza minima e Distanza calcolata

Tramite una scansione dell’ArrayList vengono assegnate le coordinate del punto **e** al punto di interesse più vicino e poi viene aggiunto nell’ArrayList.

Nel caso in cui l’indice sia uguale a **-1** viene stampato un AVVERTIMENTO nell’elaborazione dell’evento.

* Metodo **resetStatistiche()**: resetta le statistiche di tutti i punti di interesse .
* Metodo **stampaStatisticheTotali()**: stampa la percentuale delle statistiche totali.
* Metodo **stampaStatisticheAttivi()**:stampa la percentuale delle statistiche dei soli utenti attivi.

**EVENTO**

È la classe che gestisce nella maniera più semplice possibile i dati che riguardano un evento.

* Metodo **evento()**: crea l’evento ricevendo in input i dati dei file che viene importato.
* Metodo **getAccesso()**: fa riferimento alla classe **enumerativa** StatoUtente.
* Metodo **getID():** restituisce l'id dell'utente.
* Metodi **getX()** e **getY()**: restituiscono le relative coordinate di X e Y.
* Metodo **getData()**: restituisce la data dell’evento.
* Metodo **getStatoEmotivo()**: fa riferimento alla classe **enumerativa StatoEmotivo.**

**MANAGER\_EVENTI**

In questa classe viene creata una HashMap*(presente nel pacchetto java.util messo a disposizione da Java),* che memorizza coppie formate da chiave e valore.

Nel nostro caso come *chiave* vengono utilizzate stringhe mentre come *valore* viene utilizzato un ArrayList chiamato <**listaEventi**>.

* Metodo **aggiungiEvento()**: riceve un input la stringa **<nomefile>**. Nel caso in cui venga riscontrato un problema con il file viene sollevata l’eccezione di tipo IOException e viene stampato un AVVERTIMENTO.

Come prima cosa vengono inizializzati come null sia l’ArrayList **<listaCorrente>** che la stringa **<dataListaCorrente>**. Successivamente viene creato un array di stringhe in cui, tramite una chiamata al metodo **caricaStringhe(),** viene caricata la stringa contenuta nel file. Viene verificato se l’array di stringhe non è nullo e, tramite il metodo **controllaArrayStringhe()**, che l’array e il suo contenuto siano formalmente corretti.

* Metodo **converti()**: questa funzione prende in input un array di String di nome **<datiGrezzi>** e restituisce un array di tipo Evento contenente tutti gli eventi che è riuscito a convertire. La funzione prevede un controllo iniziale della stringa che sta convertendo per verificare che i dati siano nel formato corretto. Se i dati sono nel formato corretto li converte e li archivia temporaneamente nell’ArrayList convertiti, invece se non risulta corretto avverte l’utente tramite un AVVERTIMENTO e passa alla prossima stringa. In fine converte l’ArrayList in array e restituisce la lista degli eventi.
* Metodo **ricerca()**: riceve in input **data1** e **data2**, dichiara una stringa data corrente e inizializza un ArrayList eventi di tipo Evento. Successivamente viene controllato se le date sono nel formato corretto. Dopo di che controlla se **data1 < data2**, se non lo è la ricerca viene saltata e ritorna null. Vengono recuperati gli eventi compresi tra le date e vengono salvati nell’ArrayList **<lista>**. Se lista è diversa da null vengono aggiunti ad **<eventi>** tutti gli eventi contenuti nell’HashMap che hanno quella data e restituisce gli eventi trovati.
* Metodo **contieneEventoId()**: riceve come argomento l’**ID** e una lista. Questo metodo verifica che nella lista sia presente l’**ID** dell’evento.
* Metodo **controllaData()**: viene passato come argomento la data e nel caso in cui la lunghezza sia diversa da 8, o il formato della data non dovesse essere di soli numeri, o l’anno di inizio sia inferiore al 1970, o il formato della data non dovesse essere valido, il metodo restituisce FALSE, altrimenti TRUE.
* Metodo **primaDiData()**: controlla che **data1** preceda **data2**.
* Metodo **stessaData()**: controlla che **data1** e **data2** facciano riferimento alla stessa data.
* Metodo **dopoDiData()**: controlla che **data1** sia successiva alla **data2**.

**MANAGER\_COMANDI**

È la classe che gestisce i comandi di EmotionalMaps e in cui vengono dichiarate le variabili **mdpi** di tipo **ManagerPDI**, **me** di tipo ManagerEventi e l’array listaComandi di tipo Comando.

Viene creato un nuovo ManagerComandi inizializzando le variabili precedentemente citate.

Viene creato un nuovo metodo **ManagerComandi** inizializzando l’oggetto **ManagerEventi** con uno già esistente.

* Metodo **caricaComandi()**: carica i comandi contenuti nel file **<nomefile>**. Questa funzione riceve come input una stringa contenente il path al file da cui caricare i comandi per l’esecuzione del programma. Nel caso in cui, durante la lettura del file, si verifichi un errore viene stampato a schermo la **traccia dello stack dell’istanza System.exit** e il programma viene arrestato in maniera forzata.
* Metodo **esegui()**: esegue i comandi contenuti in **<listaComandi>**. Nel caso in cui non riconosca il comando, stampa a schermo un AVVERTIMENTO con l’indice del comando non riconosciuto. Se il tipo del comando è **IMPORTA**, viene aggiunto l’argomento all’interno di **me,** mentre se il tipo del comando è **CREA\_MAPPA**, viene richiamato il metodo **creaMappa()** per quell’argomento.
* Metodo **converti()**: restituisce un array di tipo **comando** ricevendo in input un array di tipo String chiamato dati. Se la stringa non può essere convertita verrà ignorata e la funzione continuerà a procedere. Nel caso in cui l’array in input dovesse essere null oppure se il metodo **controlloArrayStringhe()** dovesse restituire FALSE, viene stampato a schermo un ERRORE.
* Metodo **ottieniArgomento()**: restituisce l’argomento passato al comando contenuto nella variabile stringa ricevuta in input.
* Metodo **formatoComandoCorretto()**: controlla che la stringa passata in input sia conforme alle specifiche riguardanti i comandi accettati dal programma. Questo controllo viene effettuato tramite **Regex**, che controlla la struttura lessicale della stringa.
* Metodo **creaMappa()**: genera una mappa emozionale degli eventi caricati e avvenuti nell’intervallo di tempo contenuto nella variabile argomento.

**EMOTIONAL\_MAPS**

Per prima cosa gestisce il flusso di funzionamento del programma. Viene istanziato **mc** di tipo ManagerComandi come null. Nel main viene chiamato il metodo **controllaArgomenti()**, il quale controlla che i parametri forniti nella linea di comando siano corretti.

I controlli eseguiti sono:

* **Numero degli argomenti**: nel caso in cui non venga passato nessun argomento viene stampato a schermo un ERRORE, se invece vengono passati più argomenti del dovuto viene stampato un AVVERTIMENTO il quale avvisa che verrà utilizzato solo il primo argomento;
* **Estensione del file**: nel caso in cui il file non presenti un’estensione verrà richiamato il metodo **aggiungiEstensione()** della classe UTILITA che gli attribuirà l’estensione “**.txt**”;
* **Esistenza del file fornito**: se il file non esiste viene stampato a schermo un ERRORE.

Vengono richiamati due metodi:

* Metodo **importa()**: carica i comandi contenuti in **<nomefile>** nel **ManagerComandi** tramite il metodo **caricaComandi()** del ManagerComandi.
* Metodo **esegui()**: richiama il comando **esegui()**  del ManagerComandi.

**STATO\_EMOTIVO**

È una classe che descrive gli stati emotivi gestiti dal sistema che possono essere: NEUTRO, FELICE, TRISTE, ARRABBIATO, SORPRESO.

**STATO\_REGISTRAZIONE**

È classe che descrive quali sono gli stati di registrazione degli utenti gestiti dal sistema: IN e OUT.

**STATO\_UTENTE**

È classe che descrive quali sono gli stati di accesso degli utenti gestiti dal sistema: LOGIN e LOGOUT

**TIPO\_COMANDO**

È classe che descrive le tipologie di comando gestite dal sistema: IMPORTA e CREA\_MAPPA.