EMOTIONAL MAPS

Il progetto qui descritto ha lo scopo di calcolare la *mappa emotiva* relativa ad una determinata città (nel nostro caso Milano).
La mappa è computata sulla base di feedback emotivi, ricevuti dagli utenti, relativi a determinate zone di interesse prestabilite (POI).
Gli utenti, oltre a fornire i vari feedback possono compiere azioni secondarie come ad esempio: effettuare login/logout o iscriversi/disiscriversi dall'app.

Indice

Entità:

Vengono di seguito presentate le entità usate in questo progetto:

- **Coordinate**: Entità utilizzata per conservare le coordinate di POI e utenti. È stata pensata con lo scopo di standardizzare l'acquisizione e l'uso delle coordinate da parte dell'applicativo.
  Coordinate è caratterizzato da:
  - latitudine
  - longitudine

  All'interno dei package di java è presente una classe chiamata Point, dentro `java.awt.Point`, che sfortunatamente non fa al caso nostro per lo svolgimento del progetto.

  L'oggetto è accompagnato da una classe denominata **CoordinateException**, per la comunicazione di eccezioni in caso di Coordinate errate o anomalie non gestibili.

- **Dati**: Rappresenta i dati relativi ad un utente, queste informazioni sono di accompagnamento ad ogni emozione inviata dagli utenti.
  Dati è composta da:
  - data, utilizzata per identificate l'utilizzo dell'applicativo da parte di un utente
  - coordinate, utilizzate per identificare la locazione precisa dell'utente
  - umore, utilizzato per indicare lo stato d'animo dell'utente
  - login/logout, utilizzato per identificare lo stato di accesso
  - in/out, utilizzato per identificare lo stato di iscrizione

- **POI**: Entità usata per rappresentare i POI di nostro interesse, ogni istanza di questa entità è caratterizzata da:
  - nome, proprio del punto di interesse
  - coordinate, utilizzate per identificare univocamente la sua posizione

  L'oggetto verrà accompagnato con una classe denominata **POIException**, per la comunicazione di eccezioni in caso di anomalie non gestibili.

- **Umore**: Questa entità ha lo scopo di rappresentare gli umori che gli utenti possono attribuire ad ogni POI. Naturalmente per ogni umore si avrà una ed una sola istanza di tale entità.
  L'oggetto è di tipo *Enum* poiché è possibile rappresentare ogni emozione con un valore univoco e rendere la gestione molto più fluida e rapida.

Unità:

Le classi presentate in questa sezione sono sprovviste di commenti e documentazione; sono da considerarsi come prototipi di classi.
Le eccezioni non verranno trattate all'interno del seguente documento, ma saranno solamente citate.

```
class Coordinate{
     private double longitudine;
     private double latitudine;

     public Coordinate(double _latitudine ,double _longitudine);
     public Coordinate(String _coordinate);
     public void setLatitudine(double _latitudine);
     public double getLatitudine();
     public void setLongitudine(double _longitudine);
     public double getLongitudine();
     public double getDistanza(Coordinate _c);
     public String toString();
}

class POI{
     public Coordinate coord;
     String nome;

     public POI(String _nome, Coordinate _coord);
     public POI(String _info);
     public void setNome(String _nome);
     public String getNome();
     public Coordinate getCoordinate();
     public static LinkedList<POI> caricaPOI(String _file);
     public String toString();
}

public enum Umore {
     A ("ARRABBIATO"),
     F ("FELICE"),
     S ("SORPRESO"),
     T ("TRISTE"),
     N ("NEUTRO");
}

class Dati{
     private boolean inout;
     private boolean loginlogout;
     private Date data;
     public Coordinate c;
     public Umore u;
```

```
     public Dati(String inout, String loginlogout, String data, Umore
u, Coordinate c);
     public void setINOUT(String inout)
     public boolean getINOUT();
public void setLOGINLOGOUT(String loginlogout);
     public boolean getLOGINLOGOUT();
     public static boolean verificaData(int year, int mounth, int
day);
     public static Date predisponiData(String s);
     public void setData(String data);
     public Date getData();
     public Coordinate getCoordinate();
     public Umore getUmore();
     public String toString();
}
```

Considerazioni:

- Per facilitare la ricerca delle informazioni e ottimizzare i tempi ci siamo serviti di una specifica classe java: **TreeMap.**
  Grazie a questa implementazione è possibile ottenere un accesso in tempo logaritmico alle informazioni rendendo il programma molto efficiente.
  Le TreeMap assegnano ad un tipo riferimento, detto chiave (K), un secondo tipo riferimento, detto valore (V).
  Poiché l'utente è identificato tramite un idUtente (che non è altro che una stringa di caratteri) e una serie di informazioni la TreeMap per mantenere le informazioni avrà la seguente struttura: TreeMap<IdUtente, Dati> che verrà poi tradotta come:

  ```
  TreeMap<String, LinkedList<Dati>>
  ```

- Adottando una **LinkedList**, anch'essa integrata in java, avremo un accesso complessivo di *nlog(n)*. Questo valore lo rende più efficiente di un ipotetico valore n^2, ma meno efficiente di *log(n)*. Facendo vari test si è dimostrata una complessità accettabile.
  L'utilizzo di una LinkedList come valore per la TreeMap è reso inoltre necessario dal fatto che non è possibile adottare la data del caricamento come chiave, poiché un utente potrebbe caricare più foto in una sola giornata. Se fosse stato possibile avremmo potuto adottare la seguente struttura: TreeMap<IdUtente, TreeMap<Data, Dati>> che verrebbe tradotta come:

  ```
  TreeMap<String, TreeMap<Data, Dati>>
  ```

- Il programma, così strutturato, rende facile l'aggiunta di POI in modo automatico attraverso la lettura da file (`public static LinkedList<POI> caricaPOI(String _file)`).

- Rende facile anche l'aggiunta di nuovi stati emotivi, attraverso la classe enum Umore, che casta in modo semi-automatico le informazioni che codificano l'umore.

- I metodi `toString()` delle classi, a parte della classe Umore, sono strutturati in modo da restituire le informazioni adottando lo stesso formato di input.

Le entità nel dettaglio:

Coordinate:

Entità utilizzata per indentificare la posizione dell'utente all'atto del caricamento di una foto.
La classe è formata da due attributi numerici: **latitudine** e **longitudine**, i quali hanno un dominio che spazia dai valori negativi a quelli positivi.
Il tipo che meglio ci permette di rappresentare queste informazioni è il tipo *double* poiché ci permette di rappresentare valori numerici sia positivi che negativi senza perdere precisione con valori numerici razionali molto grandi, ma anche molto piccoli (a differenza del tipo *float*).
Per motivi di scalabilità e sicurezza gli attributi sono dichiarati privati e, pertanto, per lavorarci è necessario utilizzare i seguenti metodi:

- `public double getLatitudine()` e `public double getLongitudine()`
  Questi metodi permettono di leggere i valori degli attributi.

- `public void setLatitudine(double _latitudine)` e `public void setLongitudine(double _longitudine)`
  Questi metodi permettono di assegnare i valori agli attributi.

- `public String ToString()`
  Questo metodo permette di restituire le Coordinate in formato stringa, con il seguente ordine: latitudine, longitudine.

- `public double getDistanza(Coordinate _c)`
  Questo metodo permette di calcolare la distanza fra due Coordinate, restituendo un valore *double* ottenuto per mezzo del teorema di Pitagora.

POI:

Con l'acronimo POI (**P**oint **O**f **I**nterest) si intende indicare un punto ben noto della propria mappa di riferimento preposto a catalizzare tutte le emozioni inviate nel suo intorno.
L'entità POI non necessita di molti attributi. È sufficiente conoscerne il **nome**, rappresentato da un attributo di tipo *String*, e le sue **coordinate**, contenute in un oggetto di tipo *Coordinate*.
Per motivi di scalabilità e sicurezza gli attributi sono dichiarati privati e, pertanto, per lavorarci è necessario utilizzare i seguenti metodi:

- `public String getNome()`
  Questo metodo permette di leggere il valore stringa del nome del POI.

- `public void setNome(Stirng _nome)`
  Questo metodo permette di assegnare un nuovo nome al POI.

- `public String ToString()`
  Questo metodo permette di restituire il valore del POI in formato stringa, con il seguente ordine: nome, coordinate.

- `public Coordinate GetCoordinate()`
  Questo metodo permette di restituire i valori delle coordinate.

- `public static LinkedList<POI> caricaPOI(String _file)`
  Questo metodo permette di eseguire il caricamento del *file* relativo ai POI*.

  *Questo metodo mette in luce una feature importante dell'applicativo.
  Il caricamento dei POI avviene attraverso l'ausilio di un file.
  Ciò consente al programma di cambiare ed aggiungere i POI a cui fa riferimento senza dover metter mano al codice.
  Tutti i POI sono contenuti in un apposito file di testo che viene caricato al lancio del programma.
  Si possono cambiare i POI a piacimento purché di rispettare il seguente formato:
  <Nome>-<Coordinate>
  Qualunque altro formato non concerne a quello precedentemente indicato verrà ignorato.

Umore:

L'entità umore è stata creata ad hoc per gestire gli umori associabili ad ogni foto. Gli umori totali disponibili sono 5:

1. A = ARRABBIATO
2. F = FELICE
3. S = SORPRESO
4. T = TRISTE
5. N = NEUTRO

La classe è di tipo *Enum*, questa scelta consente di castare in automatico i dati inseriti in tipo *enum* rendendo i valori scalabili, è infatti possibile aggiungere o rimuovere umori senza stravolgere il codice.
Il funzionamento della classe e molto semplice e consiste nel far corrispondere ad un certo carattere un umore vero e proprio comprensibile a chi legge la mappa.
L'unico attributo presente all'interno della classe è l'attributo nome, con cui si indica il nome del dato atttributo. Il passaggio da carattere a nome vero e proprio si trova nel costruttore della classe che, in automatico, prepara un'informazione leggibile.
Per motivi di scalabilità e sicurezza gli attributi sono dichiarati privati e, pertanto, per lavorarci è necessario utilizzare i seguenti metodi:

- `public String ToString()`
  Questo metodo permette di restituire il valore dell'umore in formato stringa.

Dati:

L'entità rappresenta un gruppo di informazioni che caratterizzano un'operazione effettuata da un determinato utente.
Per "dati" si intende l'insieme delle seguenti informazioni:

- iscrizione → rappresenta alternativamente lo stato “iscritto” o “disiscritto”.*
- login/logout → indica se l’utente ha effettuato operazioni di login o logout.*
- data → rappresenta la data in cui è stata effettuata l’operazione.**
- coordinate del caricamento → indica le coordinate a cui è stato effettuata l’operazione.
- umore rilevato → rappresenta l’umore rilevato all’atto del caricamento.

*Le informazioni relative allo stato dell’iscrizione sono codificate come booleani: assumono solo due valori: true (iscritto e login), false (disiscritto e logout).
**La data è codificata come tipo Date con relativo controllo di validità (il 31 Febbraio, ad esempio non è ammesso).

Le coordinate del caricamento e l’umore rilevato sono invece istanze delle classi viste precedentemente.

I metodi presenti nella classe sono i seguenti:

- `public void setINOUT(String inout)` e `public void setLOGINLOGOUT(String loginlogout)`
  Questi metodi permettono di assegnare i valori agli attributi *inOut* e *loginLogout*.

- `public boolean getINOUT()` e `public boolean getLOGINLOGOUT()`
  Questi metodi permettono di restituire i valori di di stato del login e dell’iscrizione.

- `public static boolean verificaData(int year, int mounth, int day)`
  Questo metodo permette di verificare la validità di una data.

- `public static Date predisponiData(String s)`
  Questo metodo permette di trasformare le informazioni di una stringa in una data vera e propria.

- `public void setData(String data)`
  Questo metodo permette di settare la data, una volta che è stata verificata e validata.

- `public Date getData()` e `public Coordinate getCoordinate()` e `public Umore getUmore()`
  Restituisce il valore della data, delle coordinate e dell’umore.

- `public String toString()`
  Questo metodo permette di restituire il valore dell’umore in formato stringa.

EmotionalMaps:

La seguente classe contiene le principali funzionalità del programma; essa mette assieme tutti gli oggetti e presenta metodi che potranno essere implementati anche in veste grafica/altri tipi di utilizzi.
I metodi privati sono necessari al fine di un utilizzo da riga di comando, quindi non necessari per un utilizzo da applicazione in veste grafica.

Presenta i seguenti metodi:

- `public static void initFile()`
  Consente di inizializzare il file di log.

- `private static int getNearestPoi(Coordinate c)`
  Consente di Restituire la posizione del POI, presente nella lista dei POI, più vicino all'utente.

- `public static void scriviSuLog(Exception e)`
  Questo metodo si occupa di registrare eventuali errori sul file di log.
  Il file di log è fondamentale per verificare quali errori si sono verificati durante il corso del programma consentendo di registrarli senza interrompere l'esecuzione.

- `public static void distinguiImportExport(String file)`
  Consente di distinguere le operazioni che l'utente desidera eseguire, indirizzando il flusso del programma verso una particolare operazione.
  Le uniche operazioni riconosciute sono, ovviamente, quelle di **create map** e **import** di dati.
  createmap(<data_inizio>**-**<data_fine>**)**
  import(<nome_file>**)**

- `public static void importFF(String file)`
  Questo metodo si occupa di gestire il caricamento della *TreeMap* con i dati contenuti nel file allegato.

- `public static String createMap(Date inizio, Date fine)`
  Questo metodo permette di creare le due mappe emotive nell'intervallo temporale desiderato.

- `private static String CreateCompleteMap(date start, date end)`
  Questo metodo permette di creare la mappa emotiva nell'intervallo temporale desiderato, tenendo conto di tutti i dati provenienti anche dagli utenti non più attivi.

- `private static String activeUserMap(date start, date end)`
  Questo metodo permette di creare la mappa emotiva nell'intervallo temporale desiderato, tenendo conto dei dati provenienti dai soli utenti attivi.

- `private static void stampaAVideo(String s)`
  Questo metodo permette di visualizzare a video le mappe emozonali.
  Come precedentemente riferito il seguente metodo è necessario solo per l'utilizzo da riga di comando, dunque non potrà essere richiamato se un utente esterno utilizza la classe EmotionaMaps.
  Il metodo esiste per essere riscritto nel caso di un utilizzo in veste grafica.

- `private static void Conteggia(Dati d, HashMap<POI, int[]> calcolo)`
  Questo metodo consente di contare gli stati di umore riguardo il POI più vicino.

- `private static HashMap<POI, int[]> InizializzaCalcolo()`
  Inizializza la mappa che verrà adottata per eseguire il calcolo degli stati emotivi.

- `private static void createMapString(HashMap<POI, int[]> calcolo)`
  Questo metodo consente di costruire le stringhe relative alle mappe emozionali, secondo il formato richiesto.

- `public static void main(String[] args)`
  Avvia il programma.

Bibliografia:

Qui si raccolgono le fonti usate per la realizzazione di questo progetto.

https://stackoverflow.com/
https://docs.oracle.com/javase/8/docs/
https://www.7-zip.org/
https://www.html.it/
https://github.com/
https://www.sublimetext.com/
https://www.jetbrains.com/idea/
https://www.gitkraken.com/