Relazione Primo Progetto Intermedio Programmazione II

Tiziano Calvani 579839

Introduzione

Viene richiesto di implementare un Social Network, dove l'utente dopo essersi registrato può creare Post e può seguire (mettendo "like") i Post degli altri Utenti, seguendoli di conseguenza.

Quindi:

1) Se un Utente segue un Post di un altro Utente esso seguirà automaticamente l'Utente che ha Postato il Post
2) I Post possono essere di lunghezza variabile ma non possono essere di lunghezza maggiore di 140 caratteri
3) Un Utente non può seguirsi da solo quindi mettersi "like" da solo
4) I post possono essere creati ma non modificati. L'unica modifica permessa ad ogni post è l'aggiunta di like.
5) Ogni Utente ha un nome diverso l'interno del Social Network

1. Classe Post

Per la classe Post, sono state usate le variabili richieste nella specifica dell'oggetto

- postID : ID del Post
- author : Stringa contenente l'autore del post
- text : Stringa di testo, di dimensione sia non nulla che minore di 140 caratteri
- timestamp : Oggetto Timestamp che riporta la data e l'ora di creazione del post
- likes : Array di Stringhe contenente gli Username che hanno messo like al Post

```
private final int postID;
private final String author;
private final String text;
private final Timestamp timestamp;
private ArrayList<String> likes;
```

Per la classe Post ho implementato i metodi getter per tutte le variabili così da poter ottenere le informazioni degli attributi dell'oggetto senza dover usare le variabili. In questo modo ho potuto impostare tutte le variabili con **private** e **final** non avendo necessità di modificarle. Ovviamente l'unico attributo senza **final** è la lista contenente i like, la quale può essere modificata: aggiungendo like.

Le eccezioni che possono essere lanciate dai metodi sono:

TextTooLongException: Eccezione lanciata dal metodo **Costruttore** quando il testo supera i 140 caratteri

UserCannotSelfLIkeException: Eccezione lanciata e gestita dal metodo **addLike** quando un Utente prova ad aggiungere "like" ad un suo stesso Post

UserCannotPutLikeTwiceException: Eccezione lanciata e gestita dal metodo **addLike :** un Utente prova a mettere due volte like ad un Post

2. Classe SocialNetwork

Per il Tipo di dato SocialNetwork si richiedeva di utilizzare la struttura di implementazione **Map<String, Set<String>>** quindi essendo Map un'interfaccia del tipo HashMap (e ugualmente Set è un'interfaccia per il tipo di dato ArrayList) ho deciso di rappresentare la struttura con una **HashMap<String, ArrayList<String>>.**

Ad ogni chiave, quindi, corrisponde un ArrayList di Stringhe contente tutte le persone seguite dall'Utente (l'Username dell'Utente è chiave dell'HashMap)

Oltre alla struttura dati relativa ai follow ho deciso di creare all'interno del tipo di dato SocialNetwork anche un'**HashMap<Integer, Post>** contenente tutti i post all'interno della rete sociale. L'HashMap è costituita dagli ID dei Post (chiavi dell'HashMap) e del relativo Post. Posso supporre che l'ID del post sia una chiave univoca dell'HashMap poiché esso stesso è gestito dalla Classe SocialNetwork che assegna un valore incrementale ad ogni Post che viene creato, nel **metodo addPost**.

2.1. Metodi Richiesti

```
public HashMap<String, ArrayList<String>> guessFollowers(List<Post> ps)
```

1) Il metodo restituisce la rete sociale derivata dalla lista di post (parametro del metodo).

   Creo un HashMap di supporto per memorizzare i dati (socialDerivateFromPost).

   Itero per ogni Post contenuto nella lista, e per ogni Post itero per ogni persona inserita nella lista dei seguiti.

   Se il seguito è stato inserito già nell'HashMap come chiave della stessa, controllo che nella lista di trabocco di quella chiave non sia stata inserito il nome l'autore del post preso in considerazione. Se l'autore del post non è stato inserito nella lista di trabocco dei seguiti dell'Utente lo inserisco. Se l'Utente non è presente inserisco una chiave dell'HashMap con il suo Username, inizializzo la lista e successivamente

aggiungo l'autore del Post preso in esame nella lista. (I due IF utilizzati servono per non avere ripetizioni di valori all'interno dell'HashMap).

Infine, restituisco l'HashMap contenente tutti gli utenti derivati dalla lista di Post e i loro rispettivi seguiti.

```
public List<String> influencers(HashMap<String, ArrayList<String>> followers)
```

2) Restituisce gli utenti più influenti della rete sociale (parametro del metodo), ovvero quelli che hanno un numero maggiore di "follower".

   Creo un HashMap di supporto che mi consente di memorizzare Utente e persone che seguono l'Utente.

   Successivamente creo un'altra HashMap di supporto che memorizza gli Utenti e il numero di quante persone seguono l'Utente.

   Poi, creo un Set e una List che mi consentiranno di ordinare l'HashMap per valori crescenti e restituire le chiavi.

   Visto che l'HashMap ordinata secondo valori crescenti parto dall'ultima posizione e copio tutte le chiavi dell'HashMap.

   Restituendo infine List avrò una Lista di elementi ordinati in ordine descrescenti di Followers.

```
public Set<String> getMentionedUsers()
```

3) Restituisce l'insieme degli utenti menzionati (inclusi) nella rete sociale. Quindi restituisce tutte le chiavi della mia HashMap retesociale.

```
public Set<String> getMentionedUsers(List<Post> ps)
```

4) Restituisce l'insieme degli utenti menzionati (inclusi) nella lista parametro del metodo. Quindi restituisce tutti gli autori dei Post contenuti nella lista passata come parametro senza ripetizioni.

```
public List<Post> writtenBy(String username)
```

5) Restituisce una lista di Post contenente tutti i Post scritti da quella persona senza ripetizioni all'interno della Lista. Per questo scopo prendo in considerazione postMap, ovvero dove ho memorizzato tutti i Post della retesociale, e tramite un ciclo FOR controllo tutti i Post e li aggiungo nella mia lista di Post, valore che successivamente restituirò, se e solo se l'autore del Post è uguale al nome Utente del quale voglio trovare i Post.

   Non ho problemi con i duplicati poiché la lista postMap non contiene duplicati.

```
public List<Post> writtenBy(List<Post> ps, String username)
```

6) Restituisce una lista di Post contenente tutti i Post, contenuti dentro la lista di Post passata come parametro, scritti da quella persona senza ripetizioni all'interno della Lista. Come nel precedente metodo controllo tutti i Post che hanno campo autore uguale alla stringa Username passata per parametro e li aggiungo in una lista che poi restituirò.

L'inserimento nella lista avviene solo se la lista non contiene il Post così da evitare duplicati.

```
public List<Post> containing(List<String> words)
```

7) Restituisce una lista di Post contenente tutti i Post che contengono all'interno del parametro text almeno una delle parole contenute nella lista di Stringhe passate come parametro, senza ripetizioni.

   Ho scelto di implementare questo metodo attraverso due FOR annidati così da poter scorrere tutti i post della rete sociale e la Lista di parole passata come parametro.

2.2. Metodi Aggiuntivi

```
public HashMap<String, ArrayList<String>> guessFollows(List<Post> ps)
```

1) Il metodo restituisce la rete sociale derivata dalla lista di post costruita come Utente (String e Chiave) persone che seguono l'utente.

   Creo un HashMap di supporto per memorizzare i dati (socialDerivateFromPost).

   Itero per ogni post contenuto nella lista e per ogni post controllo se l'autore è già stato immesso come chiave dell'HashMap. Se non è stato inserito nell'HashMap lo aggiungo inizializzando una Lista nella quale andrò a mettere gli Utenti che segue. Se è già presente come chiave aggiungo solo i nomi degli Utenti che lo seguono.

   Infine, restituisco un'HashMap contente Utente e persone che seguono l'Utente.

```
public void addUser(String name)
```

2) Il metodo aggiunge un Utente nella retesociale. Controllando se ci sono duplicati e se il nome immesso è corretto (non è nullo oppure una stringa vuota).

```
public void addPost(String author, String text)
      throws ExplicitLanguageException
```

3) Il metodo aggiunge un Post nella mappa dei post (postMap) controllando che l'autore e il testo non siano vuoti e gestendo l'eccezione testo troppo lungo (TextTooLongException). Oltre ad aggiungere il Post incrementa l'ID generale dei Post in modo tale che i Post siano identificati da un ID univoco.

```
public void addLike(String follower, int IDPost)
```

4) Il metodo cerca all'interno della mappa dei post il post con lo stesso ID passato come parametro e aggiunge un Follower nella lista del Post.

   Aggiunge anche la persona seguita nella retesociale in modo tale da avere un HashMap retesociale sempre aggiornata.

```
 public HashMap<Integer, Post> getPost()
```

5) Il metodo restituisce l'HashMap dei Post (postMap).

```
public HashMap<String, ArrayList<String>> getSocial()
```

6) Il metodo restituisce la rete sociale (retesociale).

```
public int getNextPostID()
```

7) Il metodo restituisce quale sarà il prossimo ID del prossimo Post (idPostCount).

3. Segnalazione Contenuti Offensivi

3.1. Introduzione

È stato richiesto di progettare un'estensione gerarchica del tipo di dato SocialNetwork in grado di gestire i contenuti espliciti.

Ho deciso di implementare quest'estensione gerarchica attraverso due meccanismi:

- Filtraggio dei contenuti immessi nel Social Network attraverso un controllo del testo dei Post.
- Un sistema per segnalare contenuti offensivi.

L'implementazione del secondo meccanismo risulta essere necessaria poiché molte volte i Post possono risultare offensivi anche senza utilizzare parole esplicite.

Dopo le segnalazioni può essere eseguito un metodo per cancellare i post segnalati più volte da Utenti diversi (in questa implementazione dopo due segnalazioni).

3.2. Implementazione

3.2.1. Controllo immissione Post nel Social Network

Il tipo di dato SocialNetworkReportOffensivPost contiene all'interno due attributi chiamati:

- blackList : lista contenente tutte le parole ritenute offensive
- offensivePostSegnalation : HashMap contenente i post ritenuti offensivi e i relativi segnalatori

```
public void addBlackList(String badWord)
```

- Questo metodo aggiunge una parola alla lista delle parole non consentite all'interno del Social. Ho trovato la necessità di avere una lista variabile di parole poiché nel tempo possono crearsi nuove parole che possono urtare la sensibilità degli utenti.

```
public void removeWordBlacklist(String badWord)
```

- Questo metodo rimuove una parola alla lista delle parole non consentite all'interno del Social.

```
public List<String> getBlackList()
```

- Il metodo restituisce tutte le parole non consentite all'interno del Social.

```
public void addPost(String author, String text)
      throws ExplicitLanguageException
```

- Il metodo addPost aggiunge un Post alla rete Sociale ma a differenza del metodo addPost di SocialNetwork esso esegue un controllo sul testo del Post per controllare la presenza di parole ritenute non consentite.

  Esegue un ciclo FOR nel quale controlla tutte le parole del testo e le confronta con le parole della blackList. Se dovesse essere presente nel testo una di queste parole il metodo lancia un'eccezione (ExplicitLanguageException) che viene gestita all'interno

del metodo stesso: il post non viene aggiunto nella postMap (post contenuti nella retesociale)

Infine, se il Post non contiene parole non consentite può essere immesso nel Social (invocando il metodo addPost della superClasse SocialNetwork).

3.2.2. Segnalazione contenuti offensivi

```
public void addReport(String reporter, int postID)
      throws DoubleSegnalationException
```

- Il metodo aggiunge il Post segnalato nell'HashMap offensivePostSegnalation con l'Username del segnalante. Esso gestisce due eccezioni all'interno del metodo:
    - DoubleSegnalationException : si verifica quando un Utente tenta di segnalare due volte uno stesso post
    - YouMustBeRegisteredToReportPostException : si verifica quando un Utente non registrato nel Social tenta di segnalare un Post che ritiene offensivo

```
public void controlReportMap()
```

- Questo metodo si occupa della rimozione dei Post ritenuti offensivi dagli Utenti. Esso controlla l'HashMap delle segnalazioni (offensivePostSegnalation) e se ci sono due o più utenti che hanno segnalato uno stesso Post il metodo rimuove il Post sia dai Post all'interno del Social sia dall'HashMap delle segnalazioni.