Relazione Progetto Java

Prima prova intermedia A.A. 2020/2021

Autore: Marco Marinò

Matricola: 596440

Obiettivo:

Sviluppo di una componente software di supporto alla gestione e l'analisi di una rete sociale (Social Network)

Indice:

Come procedere nella lettura della documentazione presente nel codice

---

***Premessa**: nella specifica viene utilizzata la sintassi LPP*

INTERFACCE

Nelle interfacce si trovano elementi per definire l'***Abstract Data Type***:

```
- OVERVIEW (Descrizione del tipo di dato)

- TYPICAL ELEMENT (Elemento che implementa l'interfaccia)
```

CLASSI

Nelle classi si trovano:

```
- ABSTRACTION FUNCTION

- REPRESENTATION INVARIANT
```

METODI

I metodi vengono specificati attraverso la descrizione di quattro diverse clausole e una descrizione globale di ciò che fa il metodo considerato. La specifica la si può trovare al di sopra della firma del metodo preso in considerazione.

```
- Descrizione globale del metodo

- REQUIRES => pre-condizioni

- THROWS => eccezioni lanciate nel caso una delle pre-condizioni in
REQUIRES viene violata

- MODIFIES => cosa viene modificato nello stato interno dell'Abstract Data
Type
```

L'Idea

Premessa: *All'interno del progetto il riferimento a "seguire/follow un post" corrisponderà a "mettere like a un post"*

La rete sociale esposta dal problema consiste in una `Map<String, Set<String>>` denominata `socialNetwork` le cui chiavi rappresentano utenti della rete e il set le persone seguite dal generico utente

appartenente alla rete sociale e, logicamente, ogni persona seguita doveva appartenere alla Map stessa. Per poter sviluppare tutti i punti richiesti (in particolare la Parte 2) ho deciso sviluppare altre due strutture dati di supporto, molto simili alla Map vista precedentemente (in seguito vi sarà una spiegazione più approfondita e dettagliata):

- La prima `Map<Post, List<SortedSet<String>>` denominata `postFollowersTags` rappresenta una Map in cui a ogni Post (chiave) (il cui autore appartiene alla rete sociale) associo una lista composta da due `SortedSet` (più avanti viene spiegato come viene mantenuta l'invariante di avere due soli `SortedSet`):
    - il primo `SortedSet` sarà un insieme ordinato di persone (appartenenti alla rete sociale) che hanno messo like al Post in chiave;
    - il secondo `SortedSet` sarà un insieme ordinato di persone (appartenenti alla rete sociale) che sono state menzionate nel Post indicato in chiave;
- La seconda `Map<String, Integer>` denominata `topInfluencers` rappresenta una Map in cui a ogni stringa (utente appartenente alla rete sociale)

In questa maniera riesco a gestire ogni metodo richiesto dalla Parte 2, per la Parte 3 verrà implementato una estensione della classe SocialNetowrk che gestirà le segnalazioni di post dal contenuto offensivo.

Altra premessa: *spesso si farà riferimento alla parola "utente" e "post" essi sono da considerare appartenenti alla rete sociale (rispettivamente socialNetwork e postFollowersTags), in caso contrario verranno definiti come non appartenenti. Inoltre, quando ci si riferisce alla frase "appartenenti alla rete sociale" che sia di un utente o di un post, si intende che utente appartiene all'insieme delle chiavi di* `socialNetwork` *e post appartiene all'insieme delle chiavi di* `postFollowersTags`

Parte 1 - Post

OVERVIEW: Tipo di dato immutabile e identificatbile attraverso un codice 128 bit universale e univoco

Post è un tipo di dato che viene rappresentato da quattro attributi:

- **id:** identificatore univoco del post e nella mia scelta d'implementazione ho deciso di rappresentarlo con il tipo di dato UUID (identificatore pseudo-randomico) (*A class that represents an immutable universally unique identifier (UUID). A UUID represents a 128-bit value.* **Dalla Documentazione Ufficiale di Java SE 15**) (https://towardsdatascience.com/are-uuids-really-unique-57eb80fc2a87);
- **author:** utente della rete sociale che ha scritto il post. Tipo di dato String;
- **text:** testo del post (il vincolo dei 140 caratteri viene controllato nel processo di aggiunta del post vista in un metodo dedicato presente nella classe SocialNetwork). Tipo di dato String.

- **timestamp:** data e ora di invio del post. Tipo di dato LocalDateTime (*LocalDateTime is an immutable date-time object that represents a date-time, often viewed as year-month-day-hour-minute-second.* **Dalla Documentazione Ufficiale di Java SE 15**)

Poiché Post è un tipo di dato immutabile, devo garantire che nessun metodo del Post consenta la modifica dello stato interno e ovviamente che lo stato interno sia privato. Dunque, tutte gli attributi sono private per cui non è consentito l'accesso dall'esterno, l'unico modo per rendere visibili è quello di usare metodi Getter come `getID(), getAuthor(), getText(), ...` Gli unici controlli che vengono fatti durante la creazione dell'istanza `Post post = new Post("nomeUtente)` sono sui parametri `String user` e `String textMessage` che non devono essere null. Eventuali controlli sulla validità del post (Lunghezza del testo compresa tra 1 e 140, verrà controllato al momento dell'inserimento del post; Nome utente - con numero caratteri compreso tra 6 e 30 e avente caratteri solo alfa numerici - sarà garantito valido durante la pubblicazione del post perché c'è un metodo che aggiunge l'utente (`addUser` in SocialNetowrk.java) che controlla proprio la validità del nome utente. È presente, inoltre, un metodo toString() per rappresentare in String tutte le informazioni del post.

Dunque il costruttore di Post, costruisce un'istanza di tipo Post che avrà dunque un identificatore, un autore, un testo, data e ora in formato HH:mm:ss - dd-MM-yyyy di creazione del post stesso.

```
public UUID getID();

public String getAuthor();

public String getText();

public String getDateTimeToString();

public String toString();
```

Parte 2 - SocialNetwork

OVERVIEW: Social Network è un tipo di dato mutabile che caratterizza una rete sociale

Le strutture dati che mantengono la classe SocialNetwork sono:

- **socialNetwork:** come precedentemente visto, si tratta di una `Map<String, Set<String>>` avente chiavi che rappresentano gli utenti appartenenti alla rete sociale e che mappano valori i quali sono insiemi di `String` che rappresentano utenti che segue l'utente specificato in chiave. Dunque, nei valori di questa variabile si trovano **esclusivamente** utenti che sono presenti come chiave nella variabile stessa (Invariante). Si è voluto implementare `socialNetwork` con una TreeMap per garantire l'ordine alfabetico delle chiavi. Secondo la documentazione `TreeMap` non accetta chiavi nulle e le chiavi devono essere `Comparable` quindi va bene per il nostro scopo poiché `String` è `Comparable`. In questo modo facciamo un accesso di costo `O(log n)`, più costoso di un accesso usando le `HashMap` (ausilio delle tabelle hash `O(1 + load factor)`, ma `TreeMap` ci consente di risparmiare memoria poiché non allochiamo lo spazio per la tabella hash, quindi, per il nostro obiettivo

può risultare utile risparmiare memoria in quanto ci possono essere molti utenti che usano la piattaforma.

```
protected Map<String, Set<String>> socialNetwork;

...

socialNetwork = new TreeMap<String, Set<String>>();
```

- **postFollowersTags:** questa è una `Map<Post, List<SortedSet<Post>>>` le cui chiavi di tipo Post sono mappate in valori di tipo `List<SortedSet<String>>>`. Prima di tutto vediamo che in questo caso è stato scelto d'implementare una `HashMap` piuttosto che una `TreeMap` perché un `Post`, per scelta d'implementazione, non è `Comparable`. L'invariante per questo attributo è che ogni `Post` in chiave è un post il cui autore è una chiave di `socialNetwork`. I valori mappati dalle chiavi sono di tipo `List<SortedMap<String>>>` e queste liste hanno dimensione fissata 2:
  - il primo elemento rappresenta un insieme ordinato di utenti (`SortedSet<String>`) che hanno messo like/seguono il post indicato in chiave.
  - il secondo elemento rappresenta un insieme ordinato di utenti (`SortedSet<String>`) che sono stati menzionati nel post indicato in chiave. Ci sono dei vincoli, come dichiarato nella specifica, tali per cui in un post non ci possono essere tra i like e tra i tags(menzionati) l'autore del post stesso, di fatto, vengono lanciate due eccezioni a runtime `IllegalFollowException` e `IllegalTagException`.

Una precisazione: c'è un'invariante che garantisce che ogni elemento di lista e/o di set viene inizializzato correttamente, in quando ogni volta che si aggiunge un post, o un utente, viene prima allocato lo spazio correttamente. Quest'ultimo è ben definito nel codice.

```
protected Map<Post, List<SortedSet<String>>> postFollowersTags;

...

postFollowersTags = new HashMap<Post, List<SortedSet<String>>>();
```

- **topInfluencers:** struttura dati di tipo `Map<String, Integer>` che mappa utenti presenti in `socialNetwork.keySet()` in un intero che rappresenta il numero di utenti che seguono l'utente in chiave. Il metodo `addFollower` garantisce che se il parametro `username` è la prima volta che mette like a un post del `postAuthor` (si veda nella specifica) allora automaticamente viene incrementato il numero di seguaci di `postAuthor` presente in `topInfluencers`. Ma come viene garantito il fatto che in `topInfluencers` ci sia nell'insieme delle chiavi `postAuthor`? Durante l'inserimento di un utente attraverso il metodo `addUser` viene allocato e inserito `postAuthor`, inizialmente, con 0 seguaci, poi ogni volta che il metodo `addFollower` aggiunge un like a un post allora viene attivato il meccanismo precedentemente spiegato.

```
protected Map<String, Integer> topInfluencers;

...

topInfluencers = new TreeMap<String, Integer>();
```

Descrizione meccanismo d'inserimento/rimozione like ad un post

In questo paragrafo discuto di come sia possibile inserire e/o rimuovere un like da un post e gli effetti che questa azione ha sulle strutture dati.

È consigliabile visionare la classe TestClass.java in cui è presente la batteria di test per capire meglio.

Il metodo `addFollower` prende in ingresso `String username, UUID postID` per indicare che l'utente (c'è un controllo che verifica tale condizione) `username` vuole mettere like al post identificato con `postID` (c'è un controllo che verifica se il post è presente nella rete sociale (quindi se è presente come chiave in `postFollowersTags`)). Qualora tutte le condizioni indicate nella specifica in REQUIRES siano soddisfatte allora va a cercare in `postFollowersTags` il post, entra nella lista bidimensionale, in particolare entra nel primo `SortedSet ... .get(0)` e aggiunge `username` se non presente, qualora fosse presente, semplicemente non aggiunge. Quello che fa questo metodo è non solo aggiungere il like, ma anche inserire in `socialNetwork.get(username)` l'autore del post, qualora fosse già presente allora non aggiunge (definizione di Set). Un'altra cosa che fa questo metodo è quello detto in precedenza, ovvero, qualora fosse la prima volta che `username` mette like all'autore del post allora incrementa `topInfluencers.get(postAuthor)`. Il meccanismo di rimozione di un utente è gestito dal metodo `removeFollowerFromPost` che è molto simile a quello di `addFollower`, ma funziona in maniera inversa.

Per l'inserimento dei tag, si fa quasi la stessa cosa dell'aggiunta dei likes, ma invece viene preso il secondo set `... .get(1)` e viene aggiunto se assente il menzionato. Ricordiamo che viene seguita una politica del tipo: l'autore del post non può nè mettere like a un suo post nè menzionarsi in un suo post.

Ulteriori dettagli implementativi sono descritti nella specifica del codice. (Consigliato).

Nella classe `SocialNetwork` vi sono presenti, inoltre, tutti i metodi richiesti dalla traccia che sono stati resi funzionanti grazie all'implementazione dei metodi precedentemente analizzati.

Parte 3 - ProtectedSocialNetwork extension

OVERVIEW: ProtectedSocialNetwork è un tipo di dato mutabile che rappresenta una rete sociale a partire dai metodi e attributi ereditati dalla classe SocialNetwork. In questa estensione si può controllare che il conenuto di un testo di un post durante la sua pubblicazione (addPost) sia privo di parole cattive (che si possono reperire nel not_allowed_words.json). Inoltre, si possono segnalare contenuti offensivi e ogni segnalazione viene memorizzata in posts_to_control.json.

Scelta d'implementazione

Per gestire la Parte 3 del progetto ho deciso di realizzare un meccanismo di segnalazione (report) di post con contenuto offensivo o non adeguato secondo il regolamento della rete sociale, qui ne viene spiegato il funzionamento.

La classe `ProtectedSocialNetwork` che estende `SocialNetwork` e implementa l'interfaccia `ProtectedSocialNetworkInterface`, si occupa non solo di gestire tutti le funzionalità descritte nella Parte 2, ma anche di aggiungere:

- un controllo automatico che si attiva quando si tenta di pubblicare un post con `addPost` il quale rileva se nel testo del post preso in parametro ci sono delle parole non accettabili presenti nel `Set<String>` (più avanti viene descritto come viene riempito questo `Set`); qualora ci fossero parole non accettabili allora lancia `BadWordsException`.
- un metodo `addDangerousReport` che raccoglie una segnalazione con parametri `String username, UUID idToReport, Integer dangerDegree, String motivation`. Questo metodo, qualora vengano soddisfatti tutti i REQUIRES, aggiunge in un file JSON (*posts_to_control.json*) il report (**in questo modo si potrebbe rappresentare una soluzione reale, in cui il file JSON viene caricato nel server e controllato da moderatori o algoritmi di Machine Learning che capiscono se il contenuto è davvero offensivo**)

Struttura Dati

La struttura dati di questa classe è la stessa della sua classe padre (`SocialNetwork`) in aggiunta vi è un `Set<String> badWords` che come detto prima, rappresenta un insieme di parole non ammesse in un post della rete sociale.

```
private Set<String> badWords;

...

badWords = new TreeSet<String>();
```

Per ulteriori dettagli sull'implementazione, si faccia riferimento alla Specifica riportata nel progetto.

Nota sulle eccezioni

Per poter garantire maggiore protezione ho voluto rappresentare le eccezioni del programma come estensione gerarchica di `Exception`, quindi eccezioni checked.

- `BadWordsException` viene lanciata qualora venga rilevata una parola non ammessa nel post
- `EmptyNetworkException` viene lanciata nel momento in cui `socialNetwork` è vuota
- `EmptyPostException` viene lanciata quando, durante la pubblicazione, il post ha un messaggio vuoto
- `EmptyPostsException` viene lanciata quando `postFollowers` è vuota
- `IllegalFollowException` viene lanciata quando si vuole aggiungere un like e l'utente che vuole mettere like, mette like a un post il cui autore è l'utente stesso
- `IllegalTagException` viene lanciata quando si vuole aggiungere un tag/menzione e l'utente che si vuole taggare nel post è l'autore del post stesso
- `NoFollowerInPostException` viene lanciata quando si tenta di compiere un'azione su un follower in un post, ma quel follower non è presente nella lista di chi ha messo like a quel post
- `NoPostInNetworkException` viene lanciata quando si tenta di compiere un'azione su un post non presente nella rete sociale (`postFollowersTags`)
- `PostException` eccezione generica per identificare un errore durante la pubblicazione o l'azione di un post nella rete sociale
- `TextTooLongException` viene lanciata quando, durante la pubblicazione, il testo di un post supera i 140 caratteri
- `UserAlreadyExistsException` viene lanciata quando si tenta di aggiungere un utente nella rete sociale, ma questo utente esiste già
- `UserException` eccezione generica per identificare un errore durante la aggiunta o l'azione di un utente nella rete sociale
- `UsernameNotAllowedException` viene lanciato quando, durante l'aggiunta di un utente nella rete sociale, l'username dell'utente non ha dei caratteri validi
- `UserNotInSocialNetworkException` viene lanciato quando si tenta di eseguire un'azione su un utente che non esiste nella rete sociale

Conclusioni

Questo progetto, è servito per applicare tutti i concetti di OOP visti a lezione quali Encapsulation, Information Hiding, Inheritance e Polymorphism. Questo progetto potrà essere modificato nel corso del tempo per supportare altre funzionalità, lascio il link a GitHub.

https://github.com/marcomarinodev