[Corso A] Primo Progetto Intermedio

sre990

June 23, 2021

Abstract

Il progetto ha l'obiettivo di applicare i concetti e le tecniche di programmazione Object-Oriented esaminate durante il corso. Lo scopo del progetto e' lo sviluppo di una componente software di supporto alla gestione e l'analisi di una rete sociale (Social Network) denominata MicroBlog.

La rete sociale consente agli utenti di inviare messaggi di testo di breve lunghezza, con un massimo di 140 caratteri, chiamati post. Gli utenti possono mettere "mi piace" ai post di altri utenti oppure "seguirli".

Contents

1 La classe Post **2**

2 La classe SocialNetwork **2**
2.1 Strutture dati 2
2.2 Metodi richiesti 3

3 La classe ModeratedSocial, estensione di SocialNetwork **3**
3.1 Strutture dati 4
3.2 Sistemi per la segnalazione 4

4 Metodi aggiuntivi **4**

5 Eccezioni Custom **4**

6 Testing **5**

1 La classe Post

Era richiesto di realizzare il tipo di dato Post per rappresentare un post all'interno della rete sociale.

Ho pensato a un post come a una classe **mutabile** che contiene - come richiesto dalla specifica - un id **univoco**, un autore, il testo del messaggio e un timestamp.

Questa classe **implementa** l'interfaccia *Comparable*, cio' e' cruciale al fine di tenere tutti i post ordinati per id in un *TreeSet* (cfr. sezione 2.1 per approfondire). Ai fini di testing e di simulazione di una corretta esecuzione del codice, ho introdotto un ritardo di 200ms nella generazione di ogni timestamp.

Per ciascuna variabile di cui sopra, sono presenti dei getter, mentre i setter sono riservati solo all'autore e al testo di un post. Inoltre a questi ultimi e' proibito prendere in input valori **null** e il testo deve avere una lunghezza tra 1 e 140 caratteri.

Infine, abbiamo un metodo *toString()* che ci permette di stampare un post nel seguente formato:

```
post[id=X, by=AUTHOR, at=TIME, text=TEXT]
```

2 La classe SocialNetwork

SocialNetwork e' la classe indispensabile per operare sulla rete sociale MicroBlog. Questa classe implementa tutti i metodi richiesti dalla specifica e alcune funzionalita' extra (cfr. sezione 4 per approfondire).

E' stata operata una distizione tra utenti **ospiti** e utenti registrati. Ogni utente registrato puo' seguirne un altro inserendo nel testo del suo messaggio la parola chiave "*follow:*", seguita dal nome dell'utente che vorrebbe seguire. E' possibile inoltre, sempre per gli utenti registrati, mettere "mi piace" al post di un utente e - al contempo - iniziare a seguirlo. Cio' e' possibile inserendo nel testo del messaggio la parola chiave "*like:*", seguita dall'id del post a cui si vorrebbe mettere "mi piace".

Vi sono pero' alcuni vincoli: ad esempio un utente non puo' seguire se stesso o un utente non registrato e non e' possibile mettere "mi piace" a un post che non esiste.

2.1 Strutture dati

Oltre alla *HashMap* richiesta dalla specifica, che ho chiamato "*social*", per tenere traccia delle persone seguite nella rete sociale da ciascun utente, ho introdotto un'ulteriore *HashMap* denominata "*followers*".

"*Followers*" e' utile per ricordare, per ciascun utente registrato, la lista delle persone che lo stanno seguendo. Ho scelto di aggiungere questa ulteriore struttura per ridurre la complessita' in tempo del programma, a scapito della complessita' nello spazio.

Infine, ho scelto di rappresentare i post nella rete sociale servendomi di un *TreeSet*; poiche' avevo bisogno di una struttura dati che non contenesse duplicati (ogni post ha un id univoco), sulla quale eseguire operazioni di aggiunta, rimozione e visita, e che rimanesse ordinata per id.

2.2 Metodi richiesti

Per quanto riguarda il metodo *guessFollowers(ps)* ho usato il seguente algoritmo:

- Ho dichiarato tre strutture dati (*HashMap*): ciascuna contenente la coppia (id, post) dei post normali, di quelli il cui testo contiene un "*follow*" e di quelli il cui testo contiene un "*like*"
- Scorro la lista dei post e li distribuisco tra le tre *HashMap*, post normali, like e follow
- Ispeziono l'hashmap dei follow: controllo che l'utente sia registrato e aggiorno opportunamente "*social*" e "*followers*"
- Ispeziono l'hashmap dei like: controllo che l'id esista e, se esiste, mi recupero l'autore del post e aggiorno opportunamente "*social*" e "*followers*"
- Restituisco, infine, l'hashmap "*social*"

Per quanto riguarda il metodo *influencers()*:

- Dichiaro una *HashMap* contentente tutte le coppie di utenti e numero di follower ricavate dalla rete sociale
- Mi servo di una lista di appoggio per ordinare l'hashmap ricavata per ordine decrescente di follower
- Scorro la lista ricavata e aggiungo i nomi degli influencer in una ulteriore lista, che infine restituisco

Per quanto riguarda i metodi *getMentionedUsers()* e *getMentionedUsers(ps)*, li ho intesi e implementati in due modi distinti.

Nel primo caso, restituisco tutti i campi *String* della rete sociale "*social*". In parole povere, ho pensato a questo metodo come quello che restituisce gli utenti registrati.

Il secondo metodo, invece, restituisce gli utenti menzionati (inclusi) nella lista dei post in input. Dunque si comporta come facesse la ricerca di una sorta di "*tag*" all'interno del testo del post. Ho stabilito che le menzioni debbano essere del formato "*@nomeutente*".

Anche nel caso di *writtenBy(user)* e *writtenBy(ps, user)* ho fatto delle distinzioni.

Il primo metodo, infatti, prende in input uno username e ispeziona la rete sociale per trovare i post di quell'utente.

Invece, nel caso del secondo metodo, vado a ricercare i post di quel determinato utente registrato nella lista che prendo in input.

Il metodo *containing(words)* fa una ricerca **case insensitive** tra i post della rete sociale delle parole prese in input. E' sufficiente che una sola delle parole sia presente nel testo del post affinche' venga inserita nella lista da restituire.

3 La classe ModeratedSocial, estensione di SocialNetwork

La classe ModeratedSocial e' un'estensione di SocialNetwork, con l'aggiunta di un sistema di moderazione, segnalazione dei post e censura degli stessi.

Un utente moderatore e' in grado di cancellare altri utenti e post. Inoltre, ogni utente registrato puo' segnalare un post tramite il suo id; il post segnalato, se presente nella rete sociale, verra' aggiunto nella lista dei post segnalati.

E' inoltre possibile inserire delle parole nella lista delle parole proibite e censurare i post che le contengono sostituendole con un *, oppure rimuovere l'intero post dalla lista dei post della rete sociale.

3.1 Strutture dati

In questo caso abbiamo, oltre alle strutture dati ereditate da SocialNetwork, anche:

- Un *HashSet* contentente i moderatori della rete sociale denominato "*mods*"
- Un *HashSet* contentente le parole proibite denominato "*blacklist*"
- Una *HashMap* contenente le coppie (id, utente) dei post segnalati, denominata "*reported*"

3.2 Sistemi per la segnalazione

Ho previsto sia un sistema - un po' rudimentale - di moderazione che di segnalazione dei post, e due metodi per la censura dei post.

Posso aggiungere e rimuovere dal set dei moderatori degli utenti registrati. Ciascun moderatore puo' eliminare post di cui non e' l'autore e puo' eliminare utenti registrati (una sorta di "*ban*").

Un utente registrato puo' segnalare un post, il quale verra' aggiunto alla lista *HashMap* contenente le coppie (id, utente) dei post segnalati, denominata "*reported*". L'utente puo' fare una segnalazione inserendo nel testo del suo messaggio la parola chiave "*report:*", seguita dall'id del messaggio che vuole segnalare.

Per quanto riguarda la censura invece ho previsto due metodi, "*filterPosts()*" e "*censorPosts()*".

"*filterPosts()*" usa un algoritmo molto simile a quello del metodo "*containing()*", controlla se nei post della rete sociale ci sono parole presenti nella "*blacklist*". In caso affermativo, il post viene totalmente rimosso dalla rete sociale. "*censorPosts()*" invece controlla se nei post della rete sociale ci sono parole presenti nella "*blacklist*". In caso affermativo rimpiazza quelle parole con un asterisco.

4 Metodi aggiuntivi

Oltre a vari metodi di printing, trascurabili se non ai fini di testing, ho predisposto una serie di altri metodi e funzionalita'.

Ho introdotto il concetto di "**guest user**", ovvero un utente, non presente nella rete sociale, che e' in grado soltanto di scrivere post: non puo' segnalare, mettere "mi piace" o "seguire" altri utenti.

Ogni volta che viene aggiunto un post nella rete sociale viene ispezionato il campo autore: se l'autore non e' presente nella rete social, il post apparira' come se fosse stato scritto da un utente "*ospite*".

Vi sono inoltre metodi per creare/cancellare un utente e per creare/cancellare un post. Queste funzionalita' cambiano tra *SocialNetwork* e *ModeratedSocial*.

Infatti, in *SocialNetwork* un utente puo' cancellare solo il proprio "account" e un proprio post, mentre in *ModeratedSocial* un utente moderatore puo' bannare altri utenti e cancellare i post suoi e degli altri.

5 Eccezioni Custom

Ho pensato a quattro diversi tipi di eccezioni:

- **DuplicateException** viene lanciata quando un elemento che vogliamo inserire e' gia' presente nella rete sociale
- **ForbiddenActionException** viene lanciata quando stiamo provando a eseguire un'azione che non e' permessa/prevista
- **PermissionDeniedException** viene lanciata quando stiamo provando ad eseguire un'azione per cui non abbiamo i permessi necessari

- **TextFormatException** viene lanciata quando un campo *String* non e' della giusta lunghezza o contiene caratteri proibiti

  Un piccolo esempio per illustrare la differenza tra *ForbiddenActionException* e *PermissionDeniedException*: non e' **possibile** smettere di seguire un utente che non stavamo gia' seguendo prima (caso *ForbiddenActionException*). Allo stesso tempo ai *guest* non e' **permesso** seguire altri utenti, perche' non sono utenti registrati nella rete sociale (caso *PermissionDeniedException*).

6 Testing

Per quanto riguarda il testing, ho previsto due classi.

La classe *Main* e' la classe che simula una esecuzione "ideale" di MicroBlog: non ci aspettiamo alcun tipo di errore o comportamenti che potrebbero generare eccezioni.

La classe *BatteryTest* e' invece la classe in cui vengono testate tutte le eccezioni.