Relazione a proposito del progetto intermedio riguardante la programmazione orientata agli oggetti

Programmazione II – A.A. 2020/21

Corso A

Indice

1: Terminologia

- "MicroBlog", "Progetto": il sistema software presentato in questa relazione.
- "Iscritto": entità astratta che si registra al MicroBlog chiamando il metodo registerUser(String). Un iscritto è sostanzialmente una stringa.
- "Utente": entità concreta che utilizza il codice del progetto.
- "Applicazione esterna", "rete sociale": l'ipotetico sistema software che utilizza il MicroBlog come strumento di analisi dei dati.

2: Introduzione e note generali

Il progetto è stato principalmente inteso come uno strumento volto all'analisi dei dati relativi a una rete sociale del tipo di quella descritta nella traccia del progetto. A tal fine si è supposto che, per funzionare correttamente, tale strumento dovesse essere in qualche

modo sincronizzato con un'applicazione simile a un moderno social network, in modo da ottenere da esso dati coerenti con la situazione attuale della rete sociale.
Funzioni quali "like" o "registerUser" sono quindi state pensate come metodi potenzialmente chiamati da un'applicazione esterna (il social network concreto) al fine di avere a disposizione uno strumento aggiornato di analisi dei dati.

In questa relazione si discuteranno i principali dettagli implementativi del progetto, e si eviterà pertanto di discutere (o di discutere troppo a lungo) implementazioni di specifiche di interpretazione ovvia.

2.1 Commenti

Per la stesura dei commenti si è optato per una via di mezzo tra Javadoc e lo stile di commenti affrontato a lezione. Per questo motivo ogni tipo diverso di clausola (requires, param, effects, modifies, return e throws) è preceduto dal carattere '@'.

2.2 Guida all'esecuzione del codice

Il progetto è stato realizzato con l'ambiente di sviluppo IntelliJ Idea Community Edition 2020.2.2, pertanto è possibile aprire l'intero progetto ed eseguirlo tramite il suddetto programma, se si dispone di esso.
Alternativamente, su Windows:

- Estrarre dal file .zip la cartella del progetto
- Aprire una powershell e posizionarsi nella cartella "Nicola Palomba"
- Eseguire il seguente comando per compilare:

```
javac -d out\ ^
.\src\*.java ^
.\src\Exceptions\*.java ^
.\src\Exceptions\Post\*.java ^
.\src\Exceptions\Reporting\*.java ^
.\src\Exceptions\User\*.java
```

- Eseguire il codice utilizzando i seguenti comandi:

```
cd out\
java Main
```

Su Linux:

- Estrarre dal file .zip la cartella del progetto
- Aprire una shell e posizionarsi nella cartella "Nicola Palomba"
- Creare una cartella chiamata "out"
- Eseguire il seguente comando per compilare:

```
javac -d out/ \
./src/*.java \
./src/Exceptions/*.java \
./src/Exceptions/Post\*.java \
./src/Exceptions/Reporting\*.java \
```

./src/Exceptions/User/*.java

- Eseguire il codice utilizzando i seguenti comandi:
  cd out
  java Main

3: Scelte progettuali

3.1 Parte 1: Post

Post è un tipo di dato immutabile contenente gli attributi descritti nella traccia.
L'immutabilità di questo tipo di dato astratto è diretta conseguenza dell'implementazione dei like (descritta più accuratamente nella sezione SocialNetwork).
La prima idea è stata quella di fornire ogni Post di una lista di iscritti a cui il suddetto Post fosse piaciuto. Ciò, però, avrebbe reso il dato Post modificabile da qualsiasi utente del progetto, senza che fossero effettuati i dovuti controlli in relazione all'intero MicroBlog (sarebbe stato possibile aggiungere un like da parte di un iscritto senza controllare che fosse effettivamente registrato nel MicroBlog). Si sarebbe potuto anche implementare Cloneable nell'implementazione di Post al fine di rendere l'oggetto clonabile, ma si è ritenuto che il consumo di risorse per copiare ogni volta i Post fosse tranquillamente evitabile. Inoltre il codice risulta più pulito senza le innumerevoli chiamate a PostImpl.clone() che sarebbero state altrimenti necessarie.

3.2 Parte 2: SocialNetwork

3.2.1: Iscritti

Si è scelto di implementare un semplice sistema di registrazione di uno username, in modo che si possa verificare se un iscritto è presente all'interno del MicroBlog o no.
Dal momento che è possibile registrarsi (metodo registerUser(String user)), si è ritenuto sensato implementare anche la rimozione di un iscritto (metodo removeUser(String user)).

3.2.2: Like e followers

Il concetto di like è stato modellato per analogia con quelli già esistenti nei social network moderni (in particolare Twitter). Un iscritto può mettere like a un post di un altro iscritto, cominciando a seguirlo ed entrando quindi a far parte della sua rete di seguaci. Se un iscritto rimuove tutti i like messi a un certo utente, smette di seguirlo ed esce quindi dalla sua rete di seguaci. Il concetto di follower è quindi derivato da quello di like.
Dal momento che è possibile mettere un like (metodo like(Post toLike, String follower)), si è ritenuto sensato anche implementare la possibilità di rimuoverlo (metodo unLike(Post toUnlike, String follower)).

3.2.3: Menzioni

E' stata proposta un'interpretazione di "menzione" simile a quella implementata da Twitter: una menzione inizia con il carattere '@', seguito dal nome utente dell'iscritto che si intende menzionare. Questo serve per distinguere il contesto di una citazione di un iscritto: utilizzare il nome "mario" in un post può avere un significato diverso da menzionare consapevolmente "@mario", magari con l'intento di instaurare una discussione con quel particolare iscritto. Se un iscritto ne menziona un altro, ma quest'ultimo viene rimosso dal SocialNetwork, la menzione non viene più considerata come valida.

3.2.4: Hashtags e getTrending()

Con "hashtag" si intende una stringa alfanumerica (che eventualmente contiene underscores) che comincia con il carattere '#'. La funzione aggiuntiva getTrending() permette di ottenere tutti gli hashtags usati all'interno del SocialNetwork in ordine non crescente di utilizzo.

3.2.5: Metodi

guessFollowers(List<Post> ps)
La specifica fornita è stata interpretata come la restituzione di una Map che collega ogni autore presente nella lista di post passata come parametro ai suoi followers.

influencers()

L'implementazione proposta fornisce la lista di tutti gli iscritti al MicroBlog in ordine non crescente di numero di followers.

publishPost(Post toPublish)

Dato che un oggetto che implementa Post può esistere anche al di fuori del contesto del SocialNetwork (come si può notare dal fatto che diversi metodi di SocialNetwork restituiscono collezioni di post), si è deciso di fornire SocialNetwork di un metodo publishPost, che aggiunge il post passato come parametro al MicroBlog. Questo, sostanzialmente, per distinguere tra un post a sé stante e un post legato al contesto del SocialNetwork.
Dato che è possibile aggiungere un post al SocialNetwork, è anche possibile rimuoverlo (metodo deletePost(Post toDelete)).

unLike(Post toUnlike, String follower)
Se l'utente A rimuove correttamente il like a un post di B e se tale like era l'unico da parte di A ai post di B, allora A smette di seguire B.

removePost(Post toRemove)

Se il post viene rimosso correttamente, allora vengono rimossi anche tutti i like associati a quel post (con le dovute conseguenze descritte in unLike).

removeUser(String user)

Se l'iscritto viene rimosso correttamente, allora tutti i post dell'iscritto vengono cancellati (con le dovute conseguenze descritte in removePost).

3.2.6: SocialNetworkImpl

SocialNetworkImpl implementa l'interfaccia SocialNetwork. Per consentire una corretta implementazione della rete sociale descritta nella traccia del progetto, si è fatto uso di tre TreeMap (che implementano Map e incluse nel package java.util). Ogni chiave della TreeMap è collegata a un TreeSet di valori: per le chiavi è stata scelta una TreeMap per mantenere un ordinamento e consentire una ricerca veloce, se necessario, per i valori sono stati scelti TreeSet per gli stessi motivi. Tali strutture dati, inoltre, impediscono elementi duplicati, caratteristica che agevola l'implementazione del SocialNetwork.

- Map<Post, Set<String>> postLikes: una Map che collega a ogni post l'insieme dei nomi utente degli iscritti che vi hanno messo likes. Poiché ogni Post, quando viene pubblicato tramite il metodo publishPost, viene aggiunto come chiave di un TreeSet vuoto all'interno di postLikes, quest'ultima Map è utilizzata per verificare se un Post passato come parametro è stato pubblicato o meno.

- Map<String, Set<Post>> userPublished : una Map che tiene traccia dell'insieme dei post pubblicati da ogni utente. Quando un utente viene registrato, viene aggiunto come chiave di un TreeSet vuoto all'intero di userPublished

- Map<String, Set<String>> userFollowing: una Map che collega a ogni iscritto del MicroBlog l'insieme degli utenti che segue (ovvero di cui ha messo mi piace ad almeno un post da loro pubblicato). Quando un utente viene registrato, viene aggiunto come chiave di un TreeSet vuoto all'intero di userFollowing, pertanto tale Map è usata per controllare che un nome utente sia registrato nel MicroBlog

3.3: Parte 3: Segnalazioni

3.3.1: Discussione di possibili soluzioni

La segnalazione dei contenuti in un social network può avvenire in due modi.

- Automaticamente: al momento della pubblicazione di un post, si controlla il suo testo al fine di cercare eventuali parole considerate offensive o inappropriate. Tale metodo è veloce e ha chiaramente il vantaggio di essere automatico, ma è poco preciso. A meno che non si decida che qualsiasi post contenente una certa parola è

considerato offensivo in qualsiasi situazione (il che, a mio parere, non è la soluzione migliore), il contesto in cui un termine viene usato gioca un ruolo fondamentale; un sistema automatico potrebbe inviare segnalazioni di post che, dopo un'analisi umana, potrebbero risultare invece accettabili. Al contrario, un utente può scrivere una parola in diversi modi (ad esempio usando i num3r1 al posto delle lettere): prevedere tutte le possibili opzioni diventa estremamente complesso.

- Tramite segnalazioni da parte degli iscritti: questi ultimi sarebbero in grado di inviare al sistema delle segnalazioni riguardanti i Post che considerano offensivi. Ciò apre però la strada a una vasta gamma di problemi, di seguito i più evidenti:
  - Il giudizio degli iscritti è soggettivo e un Post potrebbe non essere effettivamente offensivo, o al contrario, dei Post offensivi potrebbero sopravvivere a lungo all'interno della rete prima di essere segnalati.
  - Gli iscritti potrebbero segnalare dei Post solo perché contrari a un certo parere o per pura insofferenza verso l'autore di quel Post.
  - In generale, si verrebbe a creare la possibilità di abusare della funzionalità di segnalazione da parte degli iscritti.

La soluzione implementata è un sistema misto. Al momento della pubblicazione di un post, si esamina il suo testo: se contiene parole considerate offensive o inappropriate, si invia una segnalazione automatica.
Gli iscritti, inoltre, hanno la possibilità di segnalare (una sola volta per Post, così da limitare l'abuso della funzionalità) i Post che ritengono offensivi o inappropriati.
È stato inoltre implementato un sistema di gravità delle segnalazioni: quelle automatiche hanno meno peso, poiché ne viene inviata una per ogni parola offensiva e poiché si tiene conto del fatto che il contesto potrebbe consentire l'utilizzo di un tale termine, mentre quelle manuali hanno più peso poiché inviate da esseri umani in grado di discernere l'ambito di applicazione della parola.

Il sistema potrebbe essere sicuramente espanso e migliorato in più modi:

- Si potrebbero implementare delle categorie di segnalazione (ad esempio "Incitazione all'odio" o "Spam") e assegnare a ognuna di esse un peso diverso. Ad ogni modo, la questione di determinare la gravità di ogni categoria può essere facile nell'esempio riportato, ma può essere molto spinosa in altri.

- Si potrebbe implementare un sistema di reputazione degli iscritti, cosicché se si rileva che una persona è solita segnalare contenuti che, in realtà, non sono offensivi (di fatto abusando della possibilità di segnalare i Post), si assegna meno peso alla segnalazione di quell'iscritto.

In ogni caso, entrambe le migliorie sono state considerate al di fuori dello scopo del progetto.

3.3.2: Reporting

Reporting rappresenta una segnalazione di un post. Un iscritto può inviare una segnalazione tramite il metodo pubblico report(String author, Post post), mentre le segnalazioni automatiche vengono gestite tramite un metodo privato (automaticReport(Post p)).

3.3.3: Scelte implementative in ModeratedSocialNetwork e ModeratedSocialNetworkImpl

ModeratedSocialNetwork rappresenta l'estensione di un SocialNetwork in cui viene gestito il problema della segnalazione dei contenuti offensivi.
Sono stati aggiunti ad essa dei metodi volti all'invio di segnalazioni e all'analisi di esse, con la possibilità di ottenere tutti i post del ModeratedSocialNetwork in ordine di sommatoria dei pesi delle loro segnalazioni (getControversialPosts()).

La creazione di una segnalazione avviene tramite il ModeratedSocialNetwork: nella sua implementazione la gravità della segnalazione è determinata automaticamente a seconda che si tratti di una segnalazione inviata autonomamente dal sistema o di una inviata da un iscritto. Così come se non si è iscritti non è possibile mettere like o pubblicare post, allo stesso modo in ModeratedSocialNetwork, se non si è iscritti non è possibile inviare segnalazioni.

ModeratedSocialNetworkImpl implementa ModeratedSocialNetwork e aggiunge due attributi: reportings e forbiddenWords.
Reportings è una Map<Post, Set<Reporting>> che collega ogni post all'insieme di segnalazioni che lo riguardano, mentre forbiddenWords è la lista delle parole considerate offensive o inappropriate all'interno del MicroBlog e viene passata come parametro al costruttore. È comunque possibile modificare questa lista aggiungendo o rimuovendo delle parole tramite gli appositi metodi addForbiddenWord e removeForbiddenWord.

Il cambiamento delle parole proibite non è retroattivo in nessuno dei due casi.

È stato inoltre effettuato l'override dei metodi:

- publishPost, per poter aggiungere a reportings un nuovo TreeSet con il post pubblicato come chiave a (e poter quindi verificare se un post ha ricevuto segnalazioni o se un post è già stato segnalato da un certo utente).
- deletePost, per poter rimuovere le segnalazioni ricevute da un post cancellato.
- removeUser, per poter rimuovere tutte le segnalazioni inviate da un utente rimosso.

4: Eccezioni

Nel progetto sono state definiti nuovi tipi di eccezione, in modo da poter gestire errori tipici del MicroBlog. Tutte le eccezioni aggiunte sono checked, in modo tale da rendere gli utenti consapevoli del fatto che potrebbero essere lanciate, e in modo da permettere loro di gestirle come meglio credono.

- AutoLikeException: lanciata quando si tenta di mettere like a un post creato da sé stessi.
- LikeNotFoundException: lanciata quando si tenta di rimuovere un like che non è mai stato registrato.
- EmptyContentException: lanciata quando si tenta di creare un Post con contenuto vuoto.
- PostAlreadyPublishedException: lanciata quando si tenta di pubblicare nel MicroBlog un post che è già stato pubblicato.
- PostLengthExceededException: lanciata quando si crea un post con un testo più lungo di 140 caratteri.
- PostNotFoundException: lanciata quando un post non viene trovato all'interno del MicroBlog.
- AutoReportException: lanciata quando un utente cerca di segnalare un suo stesso post.
- ReportingAlreadySentException: lanciata quando un utente tenta di segnalare più volte lo stesso post.
- InvalidUsernameException: lanciata se il nome dell'utente non è valido (perché troppo lungo o perché contiene caratteri non consentiti).
- UserAlreadyRegisteredException: lanciata se si sta cercando di registrare un utente il cui nome è già presente all'interno del MicroBlog.
- UserNotFoundException: lanciata quando un utente non viene trovato all'interno del MicroBlog.