RELAZIONE: PRIMO PROGETTO INTERMEDIO – PROGRAMMAZIONE II – A.A. 2020/21 – MARZEDDU SIMONE

Introduzione

Il primo progetto intermedio ha richiesto lo sviluppo di una componente software di supporto alla gestione e l'analisi di un Social Network, nella mia interpretazione, quest'ultimo si fonderà sulla pubblicazione di post da parte di utenti registrati nella rete e sulle interazioni tra questi.

Parte 1.1: TDA Post

La prima parte del progetto ha richiesto la realizzazione del TDA Post che nel caso della mia implementazione si articola attraverso la classe "MyPost" e l'interfaccia "Post" da essa implementata.

In particolare, le variabili di istanza che descrivono questo tipo di dato sono:

- Una stringa "**Author**": ovvero il nome dell'utente che ha pubblicato il post;
- Una stringa "**Text**": nei cui centoquaranta caratteri di lunghezza massima è rappresentato il testo del post;
- Una stringa "**ID**": ovvero un codice identificativo e univoco dell'istanza di "MyPost";
- Una stringa "**Timestamp**": ovvero una stringa capace di rappresentare data ed ora di pubblicazione del post, nello specifico l'ottenimento di questo valore dipende dall'uso dei metodi della API Calendar :
    - getInstance: restituisce un'istanza di Calendar;
    - getTime: restituisce data e ora nel momento della chiamata a partire dal 01/01/1970 UTC/GMT;
- Un insieme "**Mentions**": concretamente implementato come TreeSet di dati di tipo stringa, è una raccolta di possibili nomi di utenti che l'autore del post ha desiderato menzionare con la sua pubblicazione.

Parte 1.2: I Metodi Implementati

Dopo un'analisi delle specifiche e usi che avrebbero potenzialmente coinvolto questo tipo di dato, ho deciso di realizzare dieci metodi; tra questi:

- due costruttori (la cui scelta da parte del cliente deriva dall'uso esplicito o meno del campo "**Mentions**");
- cinque osservatori;
- un metodo "compareTo" implementato a partire dalla libreria Comparable;
- due metodi frutto di una rielaborazione (Override) dei metodi ereditati dalla superclasse "Object": "equals" e "toString".

Attraverso l'invocazione di questi metodi è assicurata l'immutabilità del TDA implementato.

Parte 1.3: Le Mie Scelte

La mia implementazione di questa specifica porzione di progetto non differisce particolarmente da quanto richiesto nelle istruzioni, se non per l'uso del campo "**Mentions**".

Nel dettaglio, la mia scelta è scaturita dal voler gestire in maniera semplice ed efficace la richiesta implementativa del metodo "getMentionedUser" analizzato più avanti, proprio in occasione di questa analisi discuterò di come questa variabile di istanza verrà sfruttata.

Ciò che è importante sottolineare in questa porzione di relazione è invece la possibilità, già evidenziata, di permettere all'utente la pubblicazione di un post sia con che senza il campo "**Mentions**" specificato, in questo ultimo caso si avrà un'inizializzazione ad "Insieme Vuoto" della variabile di istanza suddetta.

Una dovuta precisazione riguarda la produzione del codice identificativo univoco richiesto per ogni post, il così detto "**ID**" verrà di fatto generato dall'unione della stringa "**Author**" e dal valore ottenuto con la chiamata del metodo "getTimeInMillis" della API Calendar in una stringa assegnata dal costruttore al campo specifico ("**ID**").

Si noti dunque che la correttezza e l'univocità di questo codice è ottenuta considerando il fatto che un utente non potrà mai pubblicare due post in un istante così vicino da poter essere ugualmente approssimato dal metodo "getTimeInMillis".

Parte 2.0.0: il TDA User

Durante lo sviluppo del progetto ho ritenuto importante appoggiarmi ad un tipo di dato astratto non richiesto dalle istruzioni ricevute, ovvero quello rappresentato dalla classe "MyUser" e dalla relativa interfaccia "User".

I motivi dietro a questa idea sono molteplici:

- Donare alla figura dell'utente della rete sociale una più consistente identità;
- Poter sfruttare le variabili di istanza della classe "MyUser" per snellire le implementazioni dei metodi della classe "MySocialNetwork";
- Sfruttare le proprietà di specifiche strutture dati (come quelle offerte dalla API TreeSet).

Parte 2.0.1: La classe "MyUser"

La classe "MyUser" è rappresentativa di quelli che saranno gli utenti del Social Network, come anche per la classe "MyPost", le sue istanze verranno prima create al di fuori della vera e propria rete sociale e solo di seguito registrate/pubblicate nella stessa.

Ovviamente queste dinamiche di registrazione verranno approfondite nella specifica sezione della relazione.

Le variabili di istanza che descrivono questo tipo di dato sono:

- Una Stringa "**Username**": ovvero il nome dell'utente (sarà univoco all'interno del Social Network);
- Un insieme di dati di tipo "MyPost" denominato "**UserPosts**": qui sono raccolti i post pubblicati dall'utente in questione (il campo "**Author**" di questi sarà equivalente al campo "**Username**");
- Un insieme di dati di tipo "MyPost" denominato "**LikedPosts**": qui sono raccolti i post verso i quali l'utente ha dimostrato apprezzamento (aggiunto un like/mi piace);
- Un insieme di stringhe denominato "**UserFollowed**": qui sono raccolti i nomi degli utenti seguiti dall'utente;
- Un insieme di stringhe denominato "**UserFollowers**": qui sono raccolti i nomi degli utenti che seguono l'utente;
- Un valore numerico "**FollowersCounter**": numero degli utenti che seguono l'utente.

NOTA: tutti gli insiemi cui si fa riferimento nella parte 2.0.1 sono di tipo TreeSet.

Parte 2.0.2: I Metodi Implementati

La classe "MyUser" conta diciassette metodi:

- Un costruttore richiedente come parametro una stringa (username dell'utente);
- Cinque metodi "default" (esclusivamente invocabili dalle classi del package "Social") utili in funzione dello sviluppo dei metodi utilizzati nella rete sociale e capaci di garantire la mutabilità del TDA "User" esclusivamente attraverso la classe "MySocialNetwork";

- Sei osservatori;
- Due metodi privati "addfollower" e "removefollower" ausiliari per lo sviluppo dei metodi "unfollow" e "follow";
- un metodo "compareTo" implementato a partire dalla libreria Comparable;
- due metodi frutto di una rielaborazione (Override) dei metodi ereditati dalla superclasse "Object": "equals" e "toString".

Parte 2.1: il TDA SocialNetwork

La seconda parte del progetto ha richiesto la realizzazione del TDA SocialNetwork che, nel caso della mia implementazione, si articola attraverso la classe "MySocialNetwork" e l'interfaccia "SocialNetwork" da essa implementata.

Come già esposto in precedenza questa classe conterrà per lo più la gestione dei tipi di dato già analizzati ("Post" e "User") e in particolare si occuperà di mantenere specifiche proprietà inviolate:

- Tutti gli utenti registrati nel Social Network devono disporre di un nome utente univoco, verrà impedita la registrazione di istanze di "MyUser" con equivalente campo "**Username**";
- La pubblicazione di un post nel Social Network è totalmente dipendente ed associata ad una specifica istanza di "MyUser" che deve essere già registrata nella rete, verrà impedita la pubblicazione di istanze di "MyPost" il cui campo "**Author**" differisca dal campo "**Username**" dell'utente associato;
- Qualsiasi interazione Utente->Utente ed Utente->Post dovrà avvenire esclusivamente tra utenti registrati nella rete sociale e post pubblicati nella stessa.

Le variabili di istanza che descrivono questo tipo di dato sono:

- Un insieme di istanze di "MyUser" denominato "**Users**": qui sono raccolti tutti gli utenti registrati nel Social Network secondo i criteri già analizzati;
- Una lista di istanze di "MyPost" denominata "**SocialPosts**": qui sono raccolti tutti i post pubblicati all'interno del Social Network secondo i criteri già analizzati, la struttura dati selezionata a questo scopo è la LinkedList;
- Una mappa denominata "**MicroBlog**": questa associa a stringhe ("**Username**" degli utenti registrati) insiemi di stringhe ("**Username**" degli utenti seguiti dai primi) e rappresenta il cuore pulsante della nostra rete sociale;
- Una mappa denominata "**LikesNumber**": questa associa ad istanze di "MyPost" (post pubblicati nel Social Network che hanno ricevuto almeno un apprezzamento (mi piace/ likes) in seguito alla propria pubblicazione) il valore numerico rappresentante il totale degli apprezzamenti ricevuti dal post.

Parte 2.2: I Metodi Implementati

Peculiarità distintiva della seconda parte del progetto è l'implementazione di sette metodi richiesti dalle stesse istruzioni, per questo affronterò in modo più dettagliato l'analisi di questi:

- guessFollowers: data in input una lista di post (istanze di "MyPost") il metodo deve restituire una rete sociale, ovvero una mappa, che similmente alla variabile di istanza "**MicroBlog**" associ a stringhe ("**Username**" degli utenti registrati) insiemi di stringhe ("**Username**" degli utenti seguiti dai primi).
  Per raggiungere questo obbiettivo ho deciso di utilizzare due iteratori:
  - Il primo iteratore è ottenuto attraverso il metodo "listIterator" facente parte dell'interfaccia List e come è intuibile ha lo scopo di scorrere tra i post della lista parametro;
  - Il secondo iteratore è invece ottenuto attraverso il metodo "descendingIterator" della classe TreeSet e ritrova la sua funzione nello scorrimento dell'intero insieme di utenti "**Users**" in cui, per ogni

iterazione del primo, si ricerca l'utente autore del post così da poter accedere al campo "**UserFollowed**" di questo ed aggiornare correttamente la mappa che sarà poi restituita.

- influencers: data una rete sociale come parametro del metodo, questo ha lo scopo di restituire una lista di stringhe equivalenti ai nomi degli utenti ("**Username**") disposti in ordine discendente rispetto al numero di utenti che seguono i primi.
  Nella mia implementazione ho voluto far sì che la lista restituita sia costituita da non più di dieci elementi, questa particolarità è stata ottenuta con la chiamata del metodo "subList" dell'interfaccia List.
  Nella classe "MySocialNetwork" è presente anche una versione alternativa di questo metodo, questa non richiede parametri specifici e si limita ad invocare il metodo influencers con parametro uguale al "**MicroBlog**" di questa istanza.
  Implementazione:
  - Viene inizializzata e popolata, con chiavi uguali agli "**Username**" degli utenti del social, una mappa che associa ad ognuno di questi il numero di utenti che seguono il primo;
  - Viene inizializzata e popolata una mappa definibile come inversa della prima la quale associa ad ogni diverso "**FollowersCounter**" la lista di "**Username**" di utenti aventi campo equivalente ad esso;
  - La seconda mappa viene visitata in ordine discendente rispetto alle chiavi e le stringhe corrispondenti vengono inserite in modo appropriato all'interno della lista che sarà poi restituita.

- getMentionedUser: data una lista di post come parametro del metodo, restituisce l'insieme degli "**Username**" di utenti menzionati nella prima.
  Nella mia implementazione gli utenti menzionati nei post sono, come già preannunciato, raccolti nel campo "**Mentions**" in ognuno di questi, pertanto il metodo si occupa di raccogliere tutte le possibili menzioni in un unico insieme e di privare questo di tutte quelle che sono le menzioni non rilevanti, ovvero, tutte le stringhe che non identificano "**Username**" di utenti registrati in "**Users**".
  Un'altra delle implementazioni richieste prevede un metodo getMentionedUser senza parametro specifico, questo è stato ottenuto rispettando il principio "DRY" semplicemente invocando il metodo suddetto utilizzando come lista di post "**SocialPosts**";

- writtenBy: questo metodo viene richiesto in due specifiche versioni, la prima restituisce la lista di post appartenenti ad una lista parametro il cui autore ("**Author**") equivale alla stringa anch'essa parametro del metodo.
  L'implementazione consiste in una semplice visita della lista parametro della quale soltanto i post che soddisfano il requisito saranno inseriti nella lista di output.
  La seconda versione del metodo risponde al principio "DRY" e consiste nella chiamata della prima fornendo come lista parametro la lista "**SocialPosts**" e come stringa parametro la stringa che è unico parametro di questa seconda versione;

- containing: questo metodo restituisce la lista dei post della rete sociale contenenti almeno una delle parole raccolte nella lista di stringhe parametro.
  L'implementazione è basata su una visita di "**SocialPosts**" dove per ogni campo "**Text**" delle istanze di "MyPost" viene verificata la presenza di almeno una delle stringhe ricevute in ingresso sfruttando il metodo "contains" della classe String.

Oltre ai metodi sopra analizzati, la classe "MySocialNetwork", conta ulteriori dodici metodi, tra questi:

- un costruttore che non richiede parametri specifici;
- sei metodi che permettono la gestione e la registrazione/pubblicazione di utenti, post e interazioni tra questi;
- cinque osservatori.

Parte 2.3: Le Mie Scelte

La principale personalizzazione della classe "MySocialNetwork" da me implementata riguarda la meccanica degli apprezzamenti, infatti, ogni utente registrato nel Social Network ha la possibilità di dimostrare apprezzamento verso uno specifico post pubblicato nella rete o di rimuovere questo qualora fosse sua volontà.

In particolare, per quanto ogni utente sia libero di applicare un apprezzamento ad un post più volte senza ricevere degli errori, solo un apprezzamento per singola coppia utente/post verrà effettivamente conteggiato; questa meccanica è integrata nel fatto che i post "piaciuti" ad un utente siano raccolti in un insieme ("**LikedPosts**").

Parte 3.1: Il TDA ROLSocialNetwork

La terza parte di questo progetto riguarda l'implementazione di un'estensione gerarchica della classe "MySocialNetwork" nella quale sia possibile segnalare eventuali contenuti offensivi.

La progettazione da me realizzata si concretizza nella classe "ROLMySocialNetwork" e nella rispettiva interfaccia implementata "ROLSocialNetwork" (Report Offensive Language Social Network).

Le nuove variabili di istanza che descrivono questo tipo di dato sono:

- Una mappa denominata "**PostReports**": questa associa a dati di tipo post "MyPost" un insieme di stringhe "**Username**" degli utenti segnalatori del post;
- Una mappa denominata "**UserReports**": questa associa a stringhe "**Username**" di dati di tipo "MyUser" un insieme di stringhe "**Username**" degli utenti segnalatori di post dell'utente.

Parte 3.2: I Metodi Implementati

La classe "ROLMySocialNetwork" conta sei metodi, tra questi:

- un costruttore che non richiede parametri specifici;
- un metodo report: che dati in ingresso un utente (istanza di "MyUser") ed un post (istanza di "MyPost") procede a registrare la segnalazione secondo criteri e principi stabiliti;
- Due metodi devoti all'individuazione del fatto che un utente (istanza di "MyUser") od un post (istanza di "MyPost") abbiano o meno ricevuto un numero di segnalazioni critico, questi metodi prendono rispettivamente i nomi: "checkUserBan" e "checkPostBan";
- Due metodi osservatori.

Parte 3.3: Le Mie Scelte

Come intuibile la mia implementazione della terza parte del progetto consiste nella possibilità da parte degli utenti di segnalare specifici post, ogni segnalazione compiuta con successo verso un determinato post causa una segnalazione anche dell'utente autore del post.

Come evincibile dai metodi "checkUserBan" e "checkPostBan" ho introdotto due soglie critiche che qualora sorpassate in termini di numero di segnalazioni ricevute porterebbero ad una eventuale (non implementata perché non richiesta) rimozione di post segnalati e sospensione di utenti autori di questi.

Parte 3.4: Altre Possibilità

Durante la fase di progettazione di questa estensione gerarchica ho pensato anche ad una seconda soluzione che mi piacerebbe riassumere in poche righe qui di seguito.

Sarebbe possibile implementare un sistema di segnalazione che oltre alle meccaniche già realizzate consenta all'utente segnalatore di specificare le parole od i passaggi del post segnalato risultati per lui offensivi.

In questo modo si potrebbe implementare una raccolta di frammenti di linguaggio offensivo utili per eseguire un controllo alla creazione di ogni nuovo post che consenta di notificare all'utente autore il fatto che il post in pubblicazione possa risultare o meno irrispettoso verso altri utenti.

Ovviamente sono conscio del fatto che la gestione di queste dinamiche richieda una particolare attenzione verso casi di "falsa positività", a questo proposito penso che una parziale soluzione sarebbe la presa in considerazione di frammenti di linguaggio offensivo segnalati un numero di volte non irrisorio, così che l'algoritmo abbia una maggiore possibilità di identificare come oltraggiosi contenuti che lo siano da un punto di vista oggettivo.

Un idea forse troppo fantasiosa, ma che ha stuzzicato il mio interesse, riguarda la possibilità da parte degli utenti della rete sociale di iscriversi ad un eventuale programma di moderazione del linguaggio offensivo, così che la sensibilità di una molteplicità di utenti appartenenti a diverse culture e con diverse prospettive possa essere determinante nella tutela di un clima positivo all'interno del Social Network.

Istruzioni all'esecuzione del codice

Il materiale consegnato comprende una classe "Test" nella quale sono stati raccolti diversi casi di esecuzione utili ad esplorare ed evidenziare tutti i possibili comportamenti derivati dall'esecuzione del codice.

In particolare, le varie proprietà da preservare vengono studiate con delle specifiche istruzioni "assert", pertanto il mancato lancio di eccezioni "AssertionError" implica la correttezza di queste porzioni di codice.

Inoltre, tutte le eccezioni potenzialmente lanciabili vengono gestite da opportune istruzioni "try-catch" che si occupano di notificare il corretto lancio di queste; ulteriori istruzioni "assert" verificano che tutte le eccezioni progettate siano state considerate e che dunque tutti i test siano stati eseguiti.

I Test di ogni singolo metodo sono stati isolati e resi indipendenti attraverso la strutturazione a blocchi, il contenuto di ogni blocco è stato raccolto in un commento, per la sua esecuzione è necessaria una rimozione dei "/* e */" che lo identificano.

NOTA: Tutte le classi implementate tranne la classe "Test" sono parte del package "Social" questa caratteristica è essenziale per garantire che i metodi "default" della classe "MyUser" non siano invocabili dall'esterno.

NOTA: Tutte le classi implementate all'interno del package "Social" dispongono di un metodo privato "checkRep", questo ha ricoperto un ruolo importante durante la fase di debug ed è stato conservato (per quanto inutilizzabile) nella versione finale del progetto esclusivamente a scopo illustrativo.