Relazione: primo progetto intermedio

Programmazione II - Anno 2020/21

Interfaccia Post e Classe MyPost

Un oggetto Post, nel mio progetto, se associato ad un SocialNetwork rappresenta una transizione di stato. Ha però anche un'identità propria slegato da altre istanze di dati.

Esso è un insieme di Autore, Testo, Tempo di pubblicazione, Id e tipologia. Autore e testo sono implementati semplicemente con una stringa. Di seguito specifico invece le implementazioni degli altri tre dati:

- **timestamp**: memorizzato da un Calendar, rappresenta l'istante in cui il Post è stato creato;
- **id**: una stringa identificativa del Post, essa viene creata nel costruttore unendo le informazioni relative all'istanza: Tipo di post, Tempo di pubblicazione e autore (univoco poiché un autore non può creare più di un Post in un millesimo di secondo, la precisione con cui viene salvato il tempo di pubblicazione);
- **tipoDiPost**: una variabile intera che ho scelto di aggiungere al tipo di dato Post per facilitarne l'analisi da parte di un SocialNetwork. Prendendo un Post singolarmente questo dato non ha vincoli, invece non è necessariamente così se associato a social network che impostano regole di codifica del post.

Il tipo di dato astratto Post è immutabile, è fornito di tutti gli osservatori relativi ai campi prima elencati e di un metodo getTimestampInMillis che restituisce la data di pubblicazione misurata in millisecondi dal 01/01/1970 UTC/GMT.

È presente il metodo toString che crea una stringa rappresentante l'istanza e il metodo compareTo che crea un ordinamento totale in insiemi di Post.

Questo ordinamento si basa sul confronto tra le date di pubblicazione, in caso di parità si confronta il tipo di post e in caso di ulteriore parità si confronta l'autore lessicograficamente. Presi due post essi saranno uguali se e solo se sono uno la copia dell'altro, prendendo sempre in considerazione che è ragionevolmente impossibile creare due post nello stesso istante di tempo.

Interfaccia User e Classe MyUser

Per dare un'identità maggiore all'utente nel Social Network, per poter contenere dati importanti relativi allo stesso e per facilitare alcuni metodi del SocialNetwork ho scelto di creare il tipo di dato astratto User.

Esso è un dato mutabile che contiene un nome, una descrizione e una lista di post:

- **name**: è una stringa che definisce univocamente l'autore;

- **description**: una stringa che contiene informazioni aggiuntive relative all'autore che può essere modificata con l'apposito metodo modifyDes;
- **linkedPostList**: è una LinkedList che memorizza Post, in ordine di aggiunta, i cui campi Autore corrispondono al nome dell'autore.

La classe è fornita di osservatori adeguati per i dati sopra descritti, del metodo linkPost per aggiungere un Post alla lista e del metodo getLinkedPost che restituisce una lista navigabile ma non modificabile che corrisponde alla lista di post associata all'istanza di User.

La classe è fornita di metodo toString che restituisce una stringa, rappresentante l'istanza, e del metodo equals che restituisce TRUE, se il nome dello User passato come parametro è uguale al nome dello User con cui è stato chiamato il metodo, FALSE altrimenti.

Interfaccia SocialNetwork e Classe MySocialNetwork

Il tipo di dato astratto SocialNetwork è definito come un insieme di utenti e un insieme di post, rappresentato in memoria da diverse strutture dati principali:

- **socialPostList**: una LinkedList in cui sono memorizzati tutti i Post di tipologia 0 inseriti nel social network in ordine di aggiunta.
- **generalPostList**: una LinkedList in cui sono memorizzati tutti i Post con tipo diverso da 0 inseriti nel social in ordine di aggiunta.
- **Microblog**: una Map che associa stringhe a set di stringhe, implementata con una HashMap<String, Set<String>>. Questa Map memorizza quali sono gli utenti (identificati con stringhe univoche) seguiti da un utente.
- **userMap**: una Map che associa stringhe a User, implementata con una HashMap<String, User>. Questa Map collega stringhe con istanze di User il cui nome equivale alla stringa ad esso collegata.
- **maxInfluencers**: un influencer è definito come uno dei primi maxInfluencers (valore intero maggiore di 0) della lista di utenti ordinata in modo decrescente rispetto al numero di seguiti di ognuno. Il numero di seguiti è il numero di utenti che seguono un utente della rete sociale.

Regole di codifica del Post nel SocialNetwork

I post si dividono in 3 tipologie distinte dal valore intero contenuto nel campo tipoDiPost, esso può essere 0, 1 o 3 e ognuno descrive una modifica specifica, relativa anche allo stato attuale del SocialNetwork:

- 0: rappresenta un post sociale e la possibilità di ogni utente di esprimersi attraverso un testo di massimo 140 caratteri. È la tipologia più semplice di post e non comporta alcuna modifica delle strutture dati se non l'aggiunta di questo post alla lista di post sociali.

Affinché un post sociale sia accettato deve esistere nel Social Network un utente il cui nome corrisponde all'autore del post.

- 1: rappresenta la creazione di un utente. La nuova istanza di user creata avrà il nome uguale all'autore di questo post, la descrizione uguale al testo di questo post e la lista di post associati vuota. Se un utente con un nome uguale all'autore di questo post è stato già creato nel SocialNetwork, questo post viene riconosciuto come modifica alla descrizione dell'User.
- 3: rappresenta l'azione da parte dell'User, il cui nome è memorizzato nel campo autore di questo post, di seguire l'User il cui nome è memorizzato nel campo testo di questo post. Se questa azione è già stata memorizzata in precedenza nel SocialNetwork allora il post rappresenta l'azione contraria. Perché un post di questo tipo sia accettato devono esistere le istanze di User associate alle stringhe contenute in autore e in testo.

Descrizione metodi

La classe MySocialNetwork rappresenta non solo un tipo di dato astratto, ma anche una classe di supporto alla gestione di dati esterni, per questo presenta diversi metodi statici che lavorano su parametri passati in ingresso. Per rispettare il principio DRY alcuni metodi dinamici (esempio: getMentionedUser oppure containing) sfruttano quelli statici per manipolare i dati di istanza, altri, nonostante abbiano un corrispettivo metodo statico (esempio: writtenBy), sfruttano le strutture dati presenti nell'istanza per svolgere un lavoro più efficiente.

Descrivo di seguito i metodi derivati da scelte di implementazione che si discostano maggiormente dalle istruzioni del progetto:

- addPost: avendo interpretato SocialNetwork non solo come classe di analisi e gestione di dati ma anche come tipo di dato e avendo interpretato i Post come transizione di stato relativa ad un Social Network, ho implementato il metodo addPost che accetta un post in ingresso e fa avvenire i cambiamenti nello stato interno. Utilizza 3 diversi metodi privati a seconda del tipo di post in ingresso: addSocialPost, addNewUserPost e addFollowPost.
- addPostList: utilizza il metodo addPost per ogni post presente nella lista passata come parametro.
- containing, containingAll e containing(statico): l'ultimo è un metodo che restituisce i post, contenuti nella lista di post passata come parametro, che contengono almeno una parola della lista di parole passata come parametro. Il primo chiama il metodo statico containing con parametri la lista di parole passata in ingresso e la lista socialPostList. Il secondo chiama due volte il metodo statico containing, prima sulla lista generalPostList, poi sulla lista socialPostList e infine unisce i risultati. Ho fatto questo per dare al cliente una maggiore scelta delle aree di ricerca che sono o tutti i post o solo i post sociali.
- guessMicroblog e getMicroblog: guessMicroblog si sostituisce alla guessFollowers richiesta nelle istruzioni del progetto, il funzionamento del metodo è uguale, ma vista la variabile interna Microblog e la biunivocità tra liste di post e istanze di SocialNetwork, questa

funzione restituisce il possibile Microblog di una istanza di SocialNetwork a cui viene aggiunta la lista di post. getMicroblog è quindi il metodo associato dinamico, che restituisce un microblog navigabile e immutabile corrispondente a quello dell'istanza.

- Influencers: con questo nome sono implementati due metodi, uno statico e uno dinamico. Quello dinamico chiama quello statico utilizzando variabili di istanza, quello dinamico prende in ingresso un microblog e restituisce l'elenco degli influencers in ordine decrescente rispetto al numero di utenti che li seguono. Il metodo statico utilizza il metodo sort, proprio di Vector, per riordinare il vettore di stringhe rappresentanti istanze di User, utilizzando un comparatore privato nominato userFollowerComparator. Questo comparatore richiede nel costruttore una Map che associano String a Integer e ordina Stringhe rispetto al valore Integer a cui esse sono legate dalla Map. Quindi follower chiama sort fornendo un userFollowerComparator creato con una Map che collega nomi di User (stringhe) a numero di utenti che seguono questo User.

Cito anche il getMaxInfluencers e il setMaxInfluencers, che servono a modificare e osservare il parametro di istanza maxInfuencers, e il metodo writtenBy, che ha un'implementazione statica e una dinamica autonome per motivi di efficienza. I metodi containsUser, containsPost e AsegueB sono metodi che restituiscono un booleano e rispondono a interrogazioni dell'istanza del tipo:

- se nel SocialNetwork un'istanza User è presente,
- se un Post è presente,
- se un utente segue un altro utente (i cui nomi sono passati come parametri).

Interfaccia SocialNetworkFF e classe MySocialNetworkFF

Il tipo di dato SocialNetworkFF, dove FF sta per family friendly, è un'estensione gerarchica di SocialNetwork, che permette la segnalazione da parte di utenti di post ritenuti offensivi. La nuova interfaccia estende l'interfaccia precedente SocialNetwork e la classe implementa questa interfaccia ed estende la classe MySocialNetwork.

Le segnalazioni sono state codificate con il tipoDiPost 4, quindi abbiamo una precondizione di addPost, e conseguentemente anche di addPostList, più debole e che rispetta le regole di ereditarietà.

SocialNetworkFF contiene al suo interno come variabili di istanza supplementari:

- **reportedPost**: una Map che associa stringhe a Integer, implementata con una HashMap<String, Set<String>>. Questa Map memorizza quali post sono stati segnalati e quante volte sono stati segnalati.
- **maxReport**: in SocialNetworkFF un post viene considerato rimosso se il numero di segnalazioni ricevute sono maggiori a maxReport (un numero intero maggiore di 0).

Descrizione metodi:

In questa classe sono presenti due metodi frutto di una rielaborazione (override) dei metodi forniti dalla super classe, questi sono addPost e addPostList. Mentre il secondo richiama il primo per ogni post nella lista ricevuta come parametro e non necessita descrizione, voglio approfondire il primo assieme agli altri metodi completamente inediti della classe:

- addPost: essendo un override di addPost della superclasse, accetta anche post di tipologia 4. Essi rappresentano il tipo di post segnalazione. Una segnalazione può essere fatta da qualsiasi utente nei confronti di un qualsiasi post sociale esistente in SocialNetworkFF. Questo metodo controlla la tipologia del post passato come parametro e se è uguale a 0, 1 o 3 allora chiama il metodo super.addPost, altrimenti utilizza un metodo specifico per un post di segnalazione addReportPost. Quest'ultimo aggiunge il post segnalato tra quelli segnalati, se non già presente, e aggiorna il numero di segnalazioni.
- getPostNotBanned: restituisce la lista di Post sociali presenti nell'istanza, rimuovendone tutti i post che hanno ricevuto un numero di segnalazioni superiore o uguale a maxReport. Restituisce quindi la lista di post sociali non rimossi.

  I restanti metodi non necessitano di una grande spiegazione: getMaxReport e getMaxReport servono a osservare e modificare il parametro maxReport; reportCount restituisce il numero di segnalazioni di un post sociale; getReportedPostList restituisce la lista degli id dei post segnalati almeno una volta.

Possibili estensioni di MySocialNetworkFF

L'estensione gerarchica del tipo di dato SocialNetwork, da me implementata, è semplice e non vieta l'abuso della stessa. Potenzialmente un utente può segnalare tante volte lo stesso post e rimuoverlo, inoltre una segnalazione, per quanto priva di senso, verrà comunque accettata dal SocialNetwork. Una possibile estensione di quest'ultima quindi deve andare a sopperire a queste debolezze. Alcune soluzioni sono:

- Porre un massimo di una segnalazione per post a utente.
- Differenziare tipi diversi di segnalazioni (linguaggio inappropriato, contenuto offensivo, incitamento all'odio eccetera) e testarne la veridicità basandosi sul numero di segnalazioni di un determinato post e dal tipo delle stesse.

Infine, parlando di possibili aggiunte alla classe, si potrebbe creare una "banca" di termini offensivi che si aggiorna e modifica a ogni segnalazione aggiunta nel SocialNetwork. Considerando offensive le parole presenti nel post segnalato e associando a queste un grado di offensività che aumenta, maggiore è la presenza di queste parole in post segnalati. Quando questo grado di offensività supera una determinata soglia, le parole in questione possono considerarsi proibite e quindi non accettate oppure censurate in nuovi post.

Eccezioni

Nel mio progetto ho creato varie eccezioni, tutte checked, relative a chiamate di metodi che non rispettano le precondizioni elencate nelle interfacce. La classe PostException è la classe padre di tutte le restanti eccezioni che riguardano i post che sono:

- PostAuthorNotFound: indica che l'autore del Post non è stato trovato all'interno della struttura dati relativa;
- PostNotFoundException: indica che il Post cercato non è presente all'interno della struttura dati relativa;
- PostNotAddedException: indica che il post non è stato aggiunto all'interno della struttura dati relativa;
- PostNotSupportedException: indica che il post non è supportato dalla struttura dati relativa (esempio: un utente non può seguire sé stesso);
- PostTargetNotFound: indica che il campo text non contiene nessuna stringa identificativa di altri tipi di dati astratti nella struttura dati relativa.

La classe UserNotFoundException estende Exception e indica che l'istanza di User relativa alla chiamata del metodo che lancia questa eccezione non è stata trovata.

Istruzioni per il codice:

nel mio progetto è presente una classe aggiuntiva chiamata TesterSupp in cui sono presenti tre metodi statici pubblici e uno statico privato. Questa classe è creata per fornire la batteria dei test associata al progetto e infatti i primi tre metodi sono:

- MyPostTest: che testa l'implementazione di MyPost creando diversi post e testando le eccezioni necessarie al corretto funzionamento.
- MyUserTest: che testa l'implementazione di MyUser anche qui creando varie istanze di User e testando tutte le eccezioni necessarie.
- MySocialNetworkTest: testa sia l'implementazione di MySocialNetwork che di MySocialNetworkFF. Lo fa utilizzando come supporto il metodo privato createColl che restituisce una lista di post che comprende Post di tipo 0,1 e 2, una lista di post corretta e la cui lunghezza cresce al crescere di n parametro in ingresso.

Questi metodi di test utilizzano tutti i metodi presenti nelle classi e ne testano il corretto funzionamento grazie ai comandi assert presenti. Quindi basterà chiamare questi metodi nel main e controllare che l'esecuzione del programma termini correttamente.