Relazione sul progetto intermedio in Java

Gianluca Morcaldi

November 29, 2020

1 Indicazioni al docente

- Il file *BatteriaTest.java* contiene una simulazione di utilizzo del social, nella quale si è cercato di coprire quanti più scenari di utilizzo possibili. Le eccezioni sono state catturate ed il loro esito stampato a schermo al fine di dimostrare l'effetto di comportamenti irregolari. Per eseguire la batteria è sufficiente compilare il file con *javac* ed eseguirlo, come un normale file *.java*.
- Nonostante una specifica quanto più formale possibile possa rendere minima la possibilità di ambiguità, si è deciso di utilizzare simboli matematici accompagnati da costrutti linguistici in lingua natuale, ad esempio usando l'espressione *in* al posto del simbolo $\in$ oppure *forall* al posto di $\forall$; questo per evitare eventuali problemi di compatibilità con editor di testo.
  Invece espressioni particolari come $oggetto_{pre}$ o $oggetto_{post}$ (usate rispettivamente per indicare lo stato precedente e successivo di un oggetto in seguito ad un cambiamento) sono state indicate utilizzando il carattere *underscore* (ad esempio $oggetto_{pre}$ è diventato *oggetto_pre*).
- In allegato a questa relazione ho aggiunto delle immagini che ho creato per dare una visualizzazione grafica dello "stato" del social durante alcuni scenari di utilizzo, al fine di facilitare la comprensione del suo funzionamento e della sua logica interna.

2 Logica interna al social

Il social network raccoglie al suo interno un insieme di utenti ed i post da loro pubblicati. Ogni utente presente all'interno del social ha la possibilità di creare dei post (di cui è ovviamente autore) e mettere like ai post altrui: se un utente *utenteX* ha messo like ad un qualsiasi post di un altro utente *utenteY*, allora *utenteX* segue *utenteY*.
Per poter associare utenti e post è stata usata la mappa **userPostMap**, la quale associa ad ogni utente (la chiave) il set di post da lui realizzati (il valore); analogamente è stata usata **userFollowsMap** per collegare ogni utente al set contenete gli utenti che esso segue e viceversa **userFollowersMap** svolge la

funzione inversa, creando una corrispondenza tra ogni utente e l'insieme di utenti che lo seguono.

3 Scelte di implementazione delle strutture dati

3.1 Classe Post

Il set degli utenti che hanno messo like al post è stato realizzato come HashSet in modo da garantire che non ci siano più like da parte dello stesso utente (proprietà del set) e per poter individuare immediatamente se un utente ha messo like ad al post (grazie ai tempi di lookup costanti).

3.2 Classe SocialNetwork

- **userFollowsMap** Questa mappa associa ad ogni utente il set di utenti che segue. La scelta di implementarla utilizzando una HashMap, che associa ad ogni stringa chiave un valore HashSet di stringhe, è dovuta principalmente al fatto che alla struttura sono richieste soprattutto operazioni che beneficiano in larga parte dei tempi di accesso costante di una Hashmap; allo stesso modo la decisione di implementare i valori associati alle chiavi come HashSet è dovuta al fatto che non le sono richieste operazioni di ordinamento o ricerca, mentre è necessario mantenere l'unicità dei valori salvati e permettere un accesso ad essi il più veloce possibile.

- **userPostMap** Per associare ogni utente all'insieme dei suoi post si è scelto di utilizzare una HashMap (per poter ottenere i post di un utente in tempo costante) e per l'insieme dei post si è usato un TreeSet, che ha la proprietà di mantenere l'insieme ordinato (in modo da poter eventualmente fornire i post dell'utente già ordinati, come funziona normalmente sui social).

- **userFollowersMap** La mappa associa ad ogni utente un HashSet di suoi followers, ed anche in questo caso si è usata una mappa hash per il lookup costante ed un HashSet per lookup costante e garanzia di unicità degli elementi.

4 Implementazione della segnalazione dei contenuti

La segnalazione di un contenuto potrebbe avvenire introducendo nella nuova sottoclasse del social un metodo per la segnalazione dei post e una struttura che, per ogni post segnalato, tiene traccia degli utenti che l'hanno effettuata.

```
private Map<Integer, Set<String>> reportedPostMap;
// f(idPost) -> {utenti che l'hanno segnalato}

public boolean reportPost(int idPost, String username){ ... }
```

Ogni volta che viene chiamato **reportPost**:

- se il post identificato da *idPost* non è già presente nella mappa, lo inserisce ed inizializza il set di stringhe, inserendo *username* dell'utente che ha segnalato il contenuto
- se il post identificato da *idPost* è già presente nella mappa, *username* viene inserito nel set associato al post (se non è già presente).

L'uso di una HashMap e di un HashSet non sono casuali: utilizzare HashMap come struttura mi permette di poter trovare il post con un tempo d'accesso costante, mentre usare un HashSet mi permette di tenere traccia degli utenti che hanno segnalato il post garantendo che uno stesso utente non possa segnalare il medesimo post più volte (senza contare che possiamo controllare se un utente ha già segnalato il post con un tempo pressochè costante).
Tenere salvati i post segnalati in una struttura, anzichè magari rimuoverli automaticamente, deriva dall'idea che la rimozione di un post debba comunque essere responsabilità di un moderatore umano.