Relazione sul Progetto "Microblog"

Luca De Paulis

29 novembre 2020

1 STRUTTURA GENERALE DEL PROGETTO

La specifica del progetto Microblog richiede di progettare dei tipi di dato astratti in grado di simulare un social network. In particolare, la specifica si divide in tre punti:

- definire un tipo di dato in grado di rappresentare un post;
- definire un tipo di dato per rappresentare il social network;
- estendere il social network per implementare un meccanismo di segnalazione dei post.

La descrizione dei dettagli implementativi dei tre punti sono rispettivamente nella sezione 3, nella sezione 4 e nella sezione 5 di questa relazione. Un'ultima sezione (la sezione 6) sarà dedicata a spiegare brevemente i tipi di eccezioni dichiarate.

Ogni interfaccia o classe dichiarata in questa implementazione è corredata da una *overview*, che spiega lo scopo del tipo di dato astratto, e da un *typical element*, che descrive un generico elemento tipico dell'interfaccia (o classe).

Inoltre ogni classe ha associato un invariante di rappresentazione (per distinguere gli elementi ben formati da quelli che non lo sono) e una funzione di astrazione (per ottenere il significato astratto di un elemento ben formato).

Infine, i vari metodi sono tutti dotati di un contratto d'uso, descritto attraverso commenti in stile `Javadocs` contenenti le clausole `requires`, `throws`, `modifies` e `effects`: il chiamante deve rispettare le precondizioni per esser certo che il metodo funzioni correttamente.

2 ESECUZIONE DEI TEST

Siccome il codice è stato scritto e testato usando solamente gli strumenti forniti dal JDK (in particolare `javac` e `java`), per eseguire i file di test è sufficiente:

- estrarre il contenuto del file compresso `Microblog`;
- posizionarsi nella cartella `src`;
- compilare i file `TestPost.java`, `TestSN.java` e `TestReport.java` tramite i comandi

```
$ javac microblog/tests/TestPost.java
$ javac microblog/tests/TestSN.java
$ javac microblog/tests/TestReport.java
```

- eseguire il test desiderato usando il comando `java` oppure `java -ea` per assicurarsi che gli invarianti di rappresentazione siano sempre verificati.

3 INTERFACCIA E IMPLEMENTAZIONE DEI POST

In questa implementazione del social network MicroBlog, un *post* rappresenta un qualsiasi tipo di interazione tra utenti. In particolare i post implementano il concetto di *post testuale*, di *like* e di *segnalazione*, da cui segue la necessità di una struttura gerarchica composta da diverse classi.

Interfaccia `Post`

L'interfaccia `Post` definisce le funzionalità di base di ogni tipo di post. Siccome il tipo di dato `Post` è immutabile ,come descritto nella clausola overview, l'interfaccia contiene solamente metodi osservatori, che verranno implementati in vari modi dalle classi `TextPost`, `LikePost` e `ReportPost`.

In particolare, oltre ai metodi richiesti dalla specifica, l'interfaccia `Post` definisce anche un metodo `getPostType()` che restituisce un oggetto di tipo `PostType`: questo tipo di dato è una semplice enumerazione che ci permette di distinguere i vari tipi di post.

Inoltre l'interfaccia definisce una costante intera `Post.MAX_POST_LENGTH`, che indica la lunghezza massima del contenuto testuale di un post e, come da specifica, è fissata a 140 caratteri. Questo ci permette di poter modificare la massima lunghezza ammessa per un post senza dover modificare il resto dell'implementazione.

Classe astratta `AbstractPost`

La classe astratta `AbstractPost` ha lo scopo di dare una implementazione generica dei metodi dell'interfaccia `Post` che sia valida per un qualsiasi post del social network. Infatti la maggior parte dei metodi richiesti da `Post` è realizzato allo stesso modo nei tre tipi di post, ma essi vengono costruiti a partire da dati diversi, quindi per favorire il riutilizzo del codice è stato necessario introdurre una *classe intermedia* che potesse dare un'implementazione di default per i vari metodi standard.

Classe `TextPost`

La classe `TextPost` si occupa di definire un post di natura testuale. Gli unici metodi da essa implementati sono i due costruttori, che si occupano di costruire un nuovo post testuale rispettivamente con un *timestamp* passato come parametro oppure fissato di default all'istante corrente (ottenuto usando il metodo `LocalDateTime.now()` definito dalla libreria standard).

I due costruttori si assicurano (in stile *defensive programming*) che la variabile di istanza `postType` sia sempre inizializzata a `PostType.TEXT` e che i parametri usati per creare il post rispettino alcune condizioni: in particolare l'autore del post, il timestamp e il contenuto non possono essere `null` e il contenuto del post deve avere lunghezza compresa tra 0 (escluso) e `Post.MAX_POST_LENGTH` (di default 140 caratteri).

Classe `LikePost`

La classe `LikePost` si occupa di definire un post che rappresenti un *like*. Come la classe `TextPost` essa definisce solamente due costruttori, che si comportano analogamente ai costruttori della classe per i post testuali (dunque in particolare inizializzano il parametro `postType` a `PostType.LIKE` automaticamente).

La differenza principale è nei parametri: la classe `LikePost` prende come *contenuto del post* un intero che rappresenta l'identificatore del post a cui l'utente vuole mettere un like. Questo identificatore viene trasformato in una stringa, ma può essere trasformato nuovamente in un intero in totale sicurezza tramite il metodo della libreria standard `Integer.parseInt()`.

Classe `ReportPost`

La classe `ReportPost` si occupa di rappresentare una segnalazione. In questo caso i costruttori (che come nei casi precedenti inizializzano il parametro `postType` al valore corretto, cioè `PostType.REPORT`) prendono due parametri che rappresentano il contenuto del post: uno è l'identificatore del post da segnalare, l'altro è il contenuto effettivo della segnalazione.

Quest'ultimo è soggetto alle stesse limitazioni del contenuto dei post testuali: in particolare non può essere né vuoto né di lunghezza superiore ai 140 caratteri.

Osservazioni generali

In generale i vari tipi di dato che rappresentano i post non si occupano di verificare alcune condizioni, come ad esempio l'unicità dell'identificatore o l'esistenza del post a cui mettere un like: queste condizioni sono tutte controllate al livello del social network.

Inoltre la struttura ramificata delle varie classi ci permette di ampliare con pochissima fatica i tipi di post che si hanno a disposizione in quanto basta estendere l'enumerazione `PostType` e dichiarare una nuova classe che estende `AbstractPost`, reimplementandone alcuni metodi se necessario.

4 INTERFACCIA ED IMPLEMENTAZIONE DEL SOCIAL NETWORK

4.1 Interfaccia `SocialNetwork`

L'interfaccia `SocialNetwork` definisce i metodi per operare su un social network con la loro specifica: essi sono realizzati con il duplice scopo di permettere la *simulazione* e l'*analisi* dei contenuti di un social network.

I metodi `signUp`, `submitPost` e `addLike` servono a creare nuovi utenti e post nel social network. In particolare ci consentono di garantire alcune condizioni fondamentali:

- l'autore di un post deve essere sempre un utente registrato;
- non esistono due utenti con lo stesso *username* (e quindi ogni utente può essere identificato con il suo *username*);
- un utente non può mettere *like* ad un post che non è già contenuto nel social network, oppure ad un suo post, oppure ad un post a cui ha già messo *like*.

Per garantire queste ed altre condizioni (come l'unicità dell'identificatore dei post) l'interfaccia `SocialNetwork` obbliga il cliente a creare post attraverso il social network: questo consente di avere invarianti di rappresentazione molto più forti, che semplificano il lavoro di implementazione.

Gli altri metodi servono invece per analizzare l'evoluzione del social network (come ad esempio i metodi `getUserSet`, che restituisce l'insieme degli utenti iscritti al social network, oppure `influencers`, che restituisce gli utenti iscritti in ordine *decrescente* di followers) oppure per ricavare dati sui post pubblicati (come nel caso di `containing`).

4.2 Classe `SocialNetworkImpl`

La classe `SocialNetworkImpl` rappresenta l'implementazione dell'interfaccia `SocialNetwork`. Essa si basa su due strutture dati:

- una `Map<String, Set<String>>` chiamata `following` che, dato un utente u del social network (identificato attraverso il suo username), restituisce l'insieme degli utenti seguiti da u;

- una `Map<Post, Set<String>>` chiamata `postToLikes` che, dato un post p, restituisce l'insieme degli utenti che hanno messo like al post p.

L'insieme degli utenti e dei post è dato semplicemente dal dominio di queste funzioni, ovvero dal `keySet()` delle due mappe: in questo modo non è necessario duplicare l'informazione relativa agli utenti iscritti nel social network e ai post creati. Inoltre la variabile di istanza `idGenerator` ci permette di definire un identificatore unico per tutti i post creati all'interno del social network tramite il metodo `getAndIncrement()`.

Dettagli implementativi dei metodi

Il metodo `influencers` è degno di nota in quanto sfrutta una struttura dati di supporto (chiamata `StringIntegerPair`) per ordinare i vari utenti del social network per il numero di *followers*.

Altri metodi interessanti sono i metodi `writtenBy` e `getMentionedUsers`: entrambi hanno due varianti che consentono di operare rispettivamente sui post del social network e su un insieme di post forniti dall'esterno. Tuttavia, in entrambi i casi questi metodi richiedono che i dati siano relativi all'istanza di `SocialNetworkImpl` utilizzata:

- nel caso di `writtenBy` si richiede in ogni caso che l'utente passato come parametro sia un utente del social network;
- nel caso di `getMentionedUsers` i *tag* restituiti sono della forma `@<user>`, dove `<user>` è un utente del social network considerato.

L'unico metodo che non sfrutta i dati dell'istanza corrente è `guessFollowers`: esso prende una lista di post (contenente quindi post di tipo testuale e likes) come parametro e ne deduce la rete sociale formata dai vari utenti.

Per ottenere l'insieme degli utenti e l'insieme dei post pubblicati la classe mette a disposizione quattro metodi, due di essi restituiscono un `Set`, gli altri due una `List`: questi metodi effettuano un *copy-out* dei dati dell'oggetto per non esporre la rappresentazione interna e quindi per impedire la modifica indesiderata dello stato.

Infine la classe `SocialNetworkImpl` contiene alcuni metodi protetti per astrarre alcune funzionalità usate frequentemente, come il controllo dell'esistenza di un utente oppure la ricerca di un post tra quelli pubblicati dato il suo identificatore.

5 IMPLEMENTAZIONE DEL SISTEMA DI SEGNALAZIONE

Per implementare il sistema di segnalazione dei post richiesto dal terzo punto della specifica è stata definita una classe `SocialNetworkReport` che estende la classe `SocialNetworkImpl`.

Il meccanismo di segnalazione è il seguente: un utente registrato nel social network può segnalare un post pubblicato nel social network attraverso il metodo `reportPost`, specificando l'identificatore del post da segnalare e dando una breve spiegazione per la sua decisione.

Questa implementazione è più flessibile delle segnalazioni automatiche: non è necessario decidere a priori una lista di parole da censurare, ma si permette agli utenti di segnalare ciò che ritengono opportuno. Inoltre, siccome nessun post viene eliminato automaticamente, questa implementazione rispetta il Principio di Sostituzione.

Dettagli implementativi

La classe `SocialNetworkReport` usa una struttura dati aggiuntiva rispetto alla sua superclasse: una `Map<Post, Set<ReportPost>>` chiamata `reports` che associa ad un post pubblicato p tutte le segnalazioni relative a p.

Per fare in modo che ogni post pubblicato sia segnalabile, i metodi `submitPost` e `addLike` vengono modificati in modo tale che il post creato venga incluso nell'insieme delle chiavi di `reports`.

Al contrario, le segnalazioni create con `reportPost` non vengono aggiunte all'insieme dei post pubblicati: questa decisione deriva dal fatto che una segnalazione non può essere considerata come un evento pubblico al pari della creazione di un post o dell'aggiunta di un like ad un post esistente, ma deve essere visibile solo ai *moderatori* del social network (o, in questo caso, a chi lo simula).

Da ciò segue che una segnalazione non può ricevere like da altri utenti e non viene considerata quando si chiamano i metodi della superclasse `SocialNetworkImpl`. Per ottenere le segnalazioni vengono quindi definiti i metodi `getReportList` e `reportedPost`: il primo restituisce una lista di tutte le segnalazioni, il secondo restituisce la mappa `reports` con dominio limitato ai post che hanno ricevuto almeno una segnalazione.

6 ECCEZIONI USATE

Per imporre il rispetto dei contratti d'uso (in stile *defensive programming*) sono state dichiarate diverse eccezioni, ognuna con un significato ed uno scopo diverso.

- Una `EmptyTextException` è sollevata quando il testo di un post è vuoto.
- Una `TextTooLongException` è sollevata quando il testo di un post supera il massimo consentito.
- Una `UserAlreadyExistsException` è sollevata quando si cerca di registrare un utente che è già registrato nel social network.
- Una `IllegalUsernameException` è sollevata quando si cerca di registrare un utente con un nome contenente caratteri diversi da numeri, lettere o *underscores*.
- Una `UserNotFoundException` è sollevata quando si cerca di compiere un'operazione con un utente non registrato.
- Una `IllegalLikeException` è sollevata quando un utente cerca di mettere like ad un suo post, oppure cerca di mettere like ad un post a cui ha già messo like.
- Una `IdNotFoundException` è sollevata quando si cerca un post tramite un identificatore `id`, ma nessun post nel social network ha come identificatore `id`.

Tutte le eccezioni dichiarate sono *checked* e incluse nella firma dei metodi, per permettere al cliente di catturarle e gestirle nel modo corretto.