Relazione progetto MicroBlog

Andrea Simone Costa

27 novembre 2020

Struttura del progetto

Le due cartelle principali del progetto sono:

- *MicroBlog* - contenente i file con il codice sorgente.
- *Tests* - contenente i test con i quali sincerarsi del corretto funzionamento dell'applicazione.

MicroBlog

In questa cartella troviamo quattro subdirectory e due file:

- *Exceptions* - contenente le eccezioni custom create per le possibili situazioni di errore relative alla logica intrinseca del blog.
- *Interfaces* - contenente le interfacce custom del progetto.
- *Implementations* - contenente le implementazioni delle suddette interfacce.
- *Utils* - contenente un paio di classi di generica utilità che sono state sfruttate all'interno del progetto.
- *SocialNetwork* - l'implementazione del social network aderente alle specifiche richieste.
- *SafeSocialNetwork* - l'estensione del social network che permette di segnalare contenuti offensivi.

Interfaces

Le tre interfacce create sono:

- *Post* - rappresentate il concetto di post del blog.
- *User* - rappresentate il concetto di utente del blog.
- *UserPostFactory* - rappresenta una semplice factory che permette di creare nuovi utenti e nuovi post.

Utils

Le due classi di generica utilità sono:

- *RandomIdGenerator* - una classe non istanziabile che provvede dei metodi statici per la generazione di id univoci.
- *StringMin1Max140* - una classe contenente una stringa di lunghezza compresa nell'intervallo $[1, 140]$.

Punti salienti del progetto

Post e User

L'idea alla base del progetto è quella di un database NoSQL dove sono facilmente definibili delle relazioni many-to-many tra le entità in gioco.
In particolare ogni `User` possiede dei riferimenti basati sugli **id** agli utenti seguiti, ai propri seguaci, ai post a cui ha lasciato like, ai post creati e ai post nei quali è stato menzionato (taggato). A sua volta ogni `Post` ha dei riferimenti all'autore, ai seguaci del post (ovvero chi vi ha messo like) e a chi è stato menzionato nel post stesso. Questo facilita notevolmente il recupero di una qualsiasi informazione legata ad una generica entità data.

È degno di nota che queste due entità fungono come banali contenitori di dati e non hanno cognizione alcuna della logica intrinseca al social network. Questo disaccoppiamento presenta almeno due vantaggi. Risulterà innanzitutto banale sia utilizzare diverse implementazioni di queste interfacce all'interno del medesimo social network, sia utilizzare le stesse implementazioni in social network aventi una logica interna differente da quella richiesta. Inoltre, le invarianti di queste due entità si semplificano di non poco.

UserPostFactory

Anziché legare il social network ad una implementazione specifica delle interfacce `Post` e `User`, dipendenza che avrebbe quasi completamente annullato i vantaggi precedentemente discussi, ho scelto di creare una semplice astrazione basata sul famoso design pattern delle factory in modo tale che l'operazione di istanziazione di nuovi utenti e nuovi post possa essere configurata alla creazione di una istanza del social, permettendo a diverse istanze di utilizzare diverse implementazioni delle interfacce in gioco.

Sistema di menzioni

Quando un utente crea un post ha la possibilità di menzionare (taggare) un qualsiasi utente del social network, anche se stesso in linea di massima. Questo sistema di tag ricalca quello di social network realmente esistenti e **non** ha nulla a che fare con il sistema di following, **non** andando quindi contro alle richieste presenti nelle specifiche. Ovvero, se un utente viene menzionato in un post **non** segue automaticamente né il post né l'autore.
La signature di una delle funzioni che permetteno di creare un nuovo post nel social network è:

```
public String createPost(String aN, StringMin1Max140 t, Set<String> mentions) {
    // ...
}
```

Le `mentions` scelte vengono semplicemente fornite come un `Set<String>`, parametro opzionale, che verrà memorizzato internamente nel post creato.

StringMin1Max140

Questa classe mantiene al livello del type system una informazione molto importante: la stringa in essa contenuta soddisfa la specifica del progetto per cui il testo di un post non deve essere maggiore di 140 caratteri. La classe non è direttamente istanziabile, espone invece un metodo statico che mappa una generica stringa `s` di tipo `String` in una stringa `s'` di tipo `StringMin1Max140` se e solo se la stringa `s` ha una lunghezza compresa nell'intervallo $[1, 140]$.
Ecco che questa specifica non fa direttamente parte di nessuna invariante di rappresentazione essendo direttamente scolpita nel type system.

Social Network

La classe `SocialNetwork` è l'unica a farsi carico di tutta la logica intrinseca del progetto. Per questo ha una invariante di rappresentazione piuttosto articolata la quale garantisce che le specifiche richieste siano soddisfatte. Internamente tiene traccia degli utenti iscritti al social e dei post presenti sulla rete.

Oltre ai metodi richiesti, questa classe espone:

- `createUser` - permette di creare un nuovo utente nel social network.
- `createPost` - permette di creare un nuovo post nel social network.
- `userLikeAPost` - permette di far lasciare un like ad un post da un utente, con conseguente follow dell'autore.
- `userBelongsToSocial` - permette di sapere se un determinato utente fa parte del social.
- `postBelongsToSocial` - permette di sapere se un determinato post fa parte del social.
- `getClonedPostById` - permette di ottenere una copia di uno specifico post.

Estensione per segnalare contenuti offensivi

Estendendo la classe `SocialNetwork` si può facilmente aggiungere un field privato che tenga traccia dei post segnalati, ad esempio un `Set<String>` contenente gli id dei post segnalati, ed un metodo pubblico che permetta di segnalare un post:

```
class SafeSocialNetwork extends SocialNetwork {

    Set<String> reportedPosts = new HashSet<String>();

    public reportPost(String postId) {
        // ...
    }

}
```

È possibile trovare una implementazione reale della classe `SafeSocialNetwork` nei file del progetto.

Compilazione

I file da compilare per utilizzare il social network sono quelli presenti nella cartella `Tests`. È consigliabile utilizzare almeno la versione 8 di Java, sebbene è doveroso tenere presente che sulla mia macchina è installata la versione 14.
Per compilare ed eseguire uno qualsiasi dei file di test è sufficiente rimanere nella **root directory** e utilizzare la seguente coppia di comandi:

```
javac Tests/NomeClasse_TEST.java
java Tests/NomeClasse_TEST
```

Note

Ove possibile è stato scelto un approccio funzionale nella stesura del codice, approccio basato sull'uso di metodi dichiarativi in unione con le lambda function al posto di costrutti iterativi, sull'uso dell'`Optional` al posto del `null` e sull'uso di tipi più restrittivi al posto delle eccezioni.