Primo Progetto Intermedio

Leonardo Scoppitto

Novembre 2020

Contents

1 Compilazione ed esecuzione

Dopo aver scaricato l'archivio ed averlo estratto, ci si troverà davanti a tre cartelle: *Source*, *Eseguibili* e *Compilazione_Manuale*. Dentro la prima si torva tutto il codice sorgente con tutte le classi utilizzate per il progetto, in particolare dentro la cartella *Interfaces* si trovano le interfacce delle classi principali dove è scritta la specifica; l'invariante di rappresentazione e la funzione di astrazione di `SocialNetwork`, invece, sono nel file *SocialNetwork.java* Dentro la cartella *Eseguibili*, invece, si hanno due pacchetti:

- `TestSet.jar`: contiene una simulazione in cui 10 utenti interagiscono fra loro postando delle frasi celebri. Questa è la batteria di test richiesta.
- `MicroBlogCLI.jar`: contiene un'interfaccia testuale con cui poter provare manualmente tutto ciò che è stato implementato nel progetto. Per accedere al ripristino dei contenuti segnalati (implementazione della parte 3) è sufficiente eseguire il login come *admin* (password: *admin*).

I pacchetti si eseguono spostandosi da terminale all'interno della cartella *Eseguibili* e lanciandoli con il comando `java -jar NomePacchetto.jar`. Se si volesse compilare una delle due classi principali (`TestSet.java` o `MicroBlog.java`) è sufficiente spostarsi da terminale all'interno della cartella *Compilazione_Manuale* e compilare col comando `javac NomeClasse.java`, per poi eseguire il programma con `java NomeClasse.java`

2 Post

Nella classe `Post`, definita dall'interfaccia `PostInterface`, definisco la struttura che avranno i post all'interno di *MicroBlog* e i metodi per recuperare informazioni e visualizzarli. In particolare, rispetto alle informazioni richieste nella consegna del progetto, ho aggiunto:

1. `TreeSet<String> followers`: Questo è il set che salva gli utenti che seguono il post. Ho scelto un `TreeSet` per avere sempre gli utenti ordinati in ordine lessicografico una volta che andrò a stampare a schermo il post, rendendo quindi più fruibile quest'informazione (che alla fine è una di quelle che più interesserà all'utilizzatore di MicroBlog). Non ho scelto una lista così da non dover gestire i duplicati.
2. `boolean familyFriendly`: è un booleano che indica se il contenuto è adatto ad un pubblico sensibile (`true`) o no (`false`). Verrà utilizzato una volta implementata la sottoclasse `FamilyFriendlySocialNetwork` che fornisce le strutture e i metodi per segnalare e oscurare i post che possono risultare offensivi per qualcuno.

I metodi implementati nella classe `Post` sono quindi:

- `getAuthor`: restituisce la stringa contenente l'autore del post
- `getText`: restituisce il testo del post
- `getId`: restituisce l'ID del post
- `getTime`: restituisce il Timestamp della creazione del post
- `getFollowers`: restituisce il `Set` contenente i followers del post
- `addFollow`: riceve in input una stringa contenente il nome del follower e lo aggiunge al `Set Followers`
- `printPost`: stampa a schermo il post formattato
- `compareTo`: ridefinisce la funzione presente nell'interfaccia `comparable` per l'ordinamento secondo l'ID dei post
- `setFamiliFriendlyOn()`: modifica `this.familyFriendly` settandolo su `true`.
- `setFamilyFriendlyOff()`: modifica `this.familyFriendly` settandolo su `false`.

3 SocialNetwork

La classe `SocialNetwork` è il fulcro del progetto e contiene tutto il necessario per creare e far funzionare una rete sociale bastata sulla pubblicazione di post. Nel corso della relazione si parlerà di `follower` e mai di `like`, questo per mantenere una coerenza di termini durante tutto lo sviluppo del progetto e della relazione, usando quindi la convenzione per cui *se una persona segue un post, segue l'utente*. Le strutture dati che ho scelto di usare per l'implementazione di `SocialNetwork` sono:

1. `HashMap<String, Set<String>> followers`: è una hashmap che ha come `Keys` gli utenti che hanno postato sul SocialNetwork, e come `Values` i followers di quell'utente.
2. `HashMap<String, Set<String>> followed`: è una hashmap che ha come `Keys` gli utenti che hanno avuto interazioni con gli altri utenti, anche se non hanno postato. Le `Values`, infatti sono `Set` di utenti che vengono seguiti dall'utente nella chiave, così da avere sia una mappa `utente -> persone che lo seguono` che una mappa `utente -> persone che sono seguite da utente`. Questo mi permette di implementare efficientemente il metodo `influencers`, come vedremo dopo, e di avere una panoramica dettagliata sulle interazioni che sono avvenute all'interno del `SocialNetwork` istanziato.
3. `HashSet<Post> postSet`: è un set contenente tutti i post pubblicati nel SocialNetwork. Ho scelto un `TreeSet` poiché in questo modo ho la garanzia che l'ordinamento sia mantenuto nel tempo, al prezzo di un inserimento in tempo logaritmico invece che costante (come negli `HashSet`).
4. `int idCounter`: è semplicemente un contatore inizializzato a 0 che viene incrementato ogniqualvolta viene inserito un post.

3.1 Metodi per l'interazione col SocialNetwork

3.1.1 `void addPost()`

Aggiunge un post all'interno di `postSet`, lanciando un'eccezione nel caso in cui il testo fosse più lungo di 140 caratteri (`IllegalLengthException`), nel caso il testo sia vuoto (`EmptyTextException`) o anche nel caso l'username inserito sia `null` o non contenga caratteri (`IllegalArgumentException`).

3.1.2 `void follow()`

Aggiunge l'autore del post che si vuole seguire (chiave) e il follower (valore) all'hashmap `followers`, lanciando l'eccezione `AutoFollowException` se l'utente prova a seguirsi da solo o `IllegalArgumentException` nel caso l'id o l'utente non rispettino i parametri della specifica. Questo metodo (pubblico), ne chiama uno privato (`void addFollowed()`), che popola la mappa `followed`, in modo che siano entrambe sempre pronte a essere utilizzate.

3.2 Metodi per la formattazione dell'output

Sono 3 metodi che stampano a schermo delle informazioni utili:

1. `printSocialNetwork()`: stampa la rete sociale contenuta in `this.followers`.
2. `printSocialNetworkbyFollowed()`: stampa la rete sociale contenuta in `this.followed`.
3. `printAllPosts()`: stampa tutti i post contenuti in `this.postSet`.

3.3 Metodi *getter*

I metodi *getter* sono 3:

1. `getPostSet()`: restituisce `this.postSet`.
2. `getFollowed()`: restuisce `this.followed`.
3. `guessFollowers()`: restutuisce `this.followers`. Questo metodo non si chiama `getFollowers()` poiché era richista l'implementazione di un metodo `Map<String, Set<String>> guessFollowers(List<Post> ps)` che ha la stessa funzione, ma opera su una lista di post data, mentre per come ho costruito io la classe `SocialNetwork`, ho già una mappa con le stesse proprietà.

3.4 Metodi di cui era richiesta l'implementazione

3.4.1 `Map<String, Set<String>> guessFollowers(List<Post> ps)`

Questo metodo restituisce una rete sociale creata a partire dalle interazioni fra gli utenti e i post, sempre secondo la logica *se una persona segue un post, segue l'utente*. Una volta chiamato il metodo, per prima cosa creo una mappa dove salvare l'output (`networkByFollowers`, sempre un'HashMap), dopodiché per ogni post all'interno di `ps` controllo se l'autore del post è già stato inserito all'interno di `networkByFollowers`: se è già stato inserito, aggiungo i follower del post al set associato all'autore; se non è stato inserito inizializzo un nuovo `HashSet` e inserisco la coppia chiave (autore del post) e valore (set appena creato contenente i followers) alla mappa. Di seguito il codice:

```
Map<String, Set<String>> networkByFollowers = new HashMap<>();
for (Post post : ps) {
    if (networkByFollowers.get(post.getAuthor()) == null)
        networkByFollowers.put(post.getAuthor(), post.getFollowers());
    else
        networkByFollowers.get(post.getAuthor()).addAll(post.getFollowers());
}
return networkByFollowers;
```

3.4.2 `Set<String> getMentionedUser(List<Post> ps)`

Questo metodo restituisce la lista degli utenti che hanno contribuito al SocialNetwork scrivendo almeno un post. Il metodo quindi prende una lista di post e la scorre aggiungendo l'autore ad un `TreeSet` di stringhe, così da averle ordinate lessicograficamente, che verrà restituito al termine dell'operazione. Anche in questo caso ho definito un altro metodo `Set<String> getMentionedUser()` che non prende in input nessun parametro, ma esegue la stessa operazione su `this.postSet`, restituendo la lista di tutti gli utenti che hanno postato sul SocialNetwork. Di seguito il codice:

```
Set<String> mentionedUsers = new TreeSet<>();
for (Post post : ps)
    mentionedUsers.add(post.getAuthor());
return mentionedUsers;
```

3.4.3 `List<String> influencers(Map<String, Set<String>> followers)`

Questo metodo restituisce una lista di utenti che hanno più *followers* che *followed* a partire da una mappa `followers` fornita come parametro. Per verificare quali utenti soddisfino questa condizione, costruisco una `followedMap` con la stessa proprietà di `this.followed` e poi confronto per ogni utente (`Key`) la grandezza dei `Set` (`Value`) contenente i *seguiti* e i *seguaci*. Chi soddisfa la condizione viene inserito nella lista `influecers`, che verrà poi ordinata e restituita. Di seguito il codice dell'implementazione:

```
List<String> influencers = new ArrayList<>();
for (Map.Entry<String, Set<String>> entry : followers.entrySet()) {
  for (String people : entry.getValue()) {
      if (followedMap.get(people) == null) {
          toadd = new HashSet<>();
          toadd.add(entry.getKey());
          followedMap.put(people, toadd);
      } else
          followedMap.get(people).add(entry.getKey());
  }
}
// Abbrevio followedMap con fMap altrimenti non entra nei margini
for (Map.Entry<String, Set<String>> entry : followers.entrySet()) {
  if ((fMap.get(entry.getKey()) == null && !entry.getValue().isEmpty())
  || entry.getValue().size() > fMap.get(entry.getKey()).size())
    influencers.add(entry.getKey());
}
Collections.sort(influencers);
return influencers;
```

Ho poi definito un metodo `List<String> influencers()` che opera solo con `this` e non riceve parametri in input, così da poter calcolare i followers dell'istanza di `SocialNetwork` con cui stiamo lavorando. Il codice è molto simile, infatti manca solo la parte della costruzione della mappa `followedMap` di supporto, in quanto in questo caso abbiamo già tutto quello che ci serve per il calcolo:

```
List<String> influencers = new ArrayList<>();
for (Map.Entry<String, Set<String>> entry : followers.entrySet()) {
  if ((followed.get(entry.getKey()) == null && !entry.getValue().isEmpty())
          || entry.getValue().size() > followed.get(entry.getKey()).size())
      influencers.add(entry.getKey());
}
Collections.sort(influencers);
return influencers;
```

Come possiamo vedere, se abbiamo già le mappe a disposizione, l'operazione viene svolta molto più velocemente ed efficientemente.

3.4.4 `List<Post> writtenBy(String username)`

Questo metodo prende come parametro un nome utente e restituisce la lista dei post pubblicati da quell'utente, opera come `getMentionedUser()`, ma salva l'intero post e non solo il nome utente. Il metodo `List<Post> writtenBy(List<Post> ps, String username)` funziona nello stesso modo, semplicemente esegue la ricerca su `ps` e non su `this.postSet`. Di seguito il codice di:

```
if (username.isBlank())
            throw new IllegalArgumentException("Username non valido");
  List<Post> wroteBy = new ArrayList<>();
  for (Post post : this.postSet)
      if (post.getAuthor().equals(username))
          wroteBy.add(post);

  if (wroteBy.isEmpty()) {
      System.out.println("Nessun post trovato :(");
      return wroteBy;
  }
  Collections.sort(wroteBy);
  return wroteBy;
```

3.4.5 `List<Post> containing(List<String> words)`

Questo metodo ricerca una lista di parole all'interno dei post pubblicati all'interno del SocialNetwork. Il suo funzionamento consiste nel dividere il testo di ogni post e confrontare ogni parola con quelle prestenti in `words`, restituendo una lista di post tali che contengano almeno una parola definita dall'espressione regolare `word.toLowerCase()[a-z]*` dove `word` appartiene a `words`. Inoltre, uso un flag `boolean found` per indicare quando una parola è stata trovata, così da interrompere la ricerca e passare al post successivo. Di seguito il codice:

```
List<Post> contains = new LinkedList<>();
String[] parsedText;
boolean found;
for (Post toScan : this.postSet) {
    found = false;
    for (String word : words) {
        if (word.isBlank())
            throw new IllegalArgumentException("Stringa non valida");
        parsedText = toScan.getText().split("[^a-zA-Z]+");
        for (String parsedWord: parsedText) {
            if (parsedWord.toLowerCase().matches(word.toLowerCase() + "[a-z]*")) {
                contains.add(toScan);
                found = true;
                break;
            }
        }
        if (found)
            break;
    }
}
if (contains.isEmpty()) {
    System.out.println("Non ho trovato risultati :(");
    return contains;
}
return contains;
```

4 FamilyFriendlySocialNetwork (Parte 3)

Ho pensato di estendere la classe `SocialNetwork` con la sottoclasse `FamilyFriendlySocialNetwork` introducendo dei metodi che permettessero di segnalare i post sia direttamente tramite il loro ID, che tramite una lista di *badWords*, così da popolare un dizionario, inizialmente vuoto, di parole non adatte ad un pubblico sensibile. Ho scelto di implementare questo metodo di segnalazioni progressive in modo che sia l'utenza stessa a decidere cosa è ammesso e cosa no all'interno della loro rete sociale.

4.1 Funzionamento

Ogni volta che viene aggiunta una parola al dizionario, la lista dei post ammessi viene aggiornata e verrà impedita la visualizzazione e la creazione di post non conformi alle nuove regole. Rimane comunque possibile ripristinare un post segnalato per errore e visualizzare l'id dei post che sono stati rimossi, come anche rimuovere una parola dal dizionario e ripristinare di conseguenza tutti i post che contenevano quella parola. Inoltre, ho effettuato un override dei metodi di `SocialNetwork` che richiedono di interagire con i post, così da bloccare i contenuti offensivi ogniqualvolta vengono chiamati tali metodi. In particolare non sarà possibile seguire un post segnalato (`follow()`) e i metodi `writtenBy()` e `containing()` restituiranno solo i post in cui il flag `familyFriendly` è `true`. Per vedere l'implementazione dell'override si può consultare il file *FamilyFriendlySocialNetwork.java*, per brevità qui riporterò le funzioni più interessanti, ovvero la segnalazione mediante una stringa contente le parole da bannare e la rimozione di una stringa di parole da quelle bannate con conseguente ripristino dei post. I due metodi sono sostanzialmente una rivisitazione di `containig()` con la differenza che si ha una condizione più stringente sul controllo delle stringhe (`equals` invece che `matches`). Per quanto riguarda gli altri metodi di segnalazione/ripristino tramite il solo ID (`void reportPost(int id)` e `void removeFlag(int id)`), semplicemente viene cercato il post incriminato `super.postSet` e successivamente oscurato/ripristinato. Di seguito il codice dei metodi principali.

4.1.1 `void reportPost(int id)`

```
if (id <= 0 || id > super.postSet.size())
    throw new IllegalArgumentException("Input errato");
for (Post post: super.postSet){
    if (post.getId() == id) {
        post.setFamilyFriendlyOff();
        this.reportedId.add(post.getId());
        return;
    }
}
```

4.1.2 `void reportPost(int id, String badWords)`

```
if (id <= 0 || id > super.postSet.size() || badWords.isBlank())
    throw new IllegalArgumentException("Input errato");
reportPostsByWord(badWords);
String[] toFilter = badWords.split("[^a-zA-Z]+");
for (Post post: super.postSet){
    if (post.getId() == id && post.getFlag()) {
        post.setFamilyFriendlyOff();
        this.reportedId.add(post.getId());
        return;
    }
    else if (post.getId() == id && !post.getFlag())
        return;
}
```

4.1.3 void reportPostsByWord(String badWords)

```
if (badWords.isBlank())
    throw new IllegalArgumentException("Stringa non valida");
// Divido le parole contenute nell'input
String[] toFilter = badWords.split("[^a-zA-Z]+");
boolean found;
// Aggiungo le parole segnalate al set this.badWords
this.badWords.addAll(Arrays.asList(toFilter));
String[] parsedText;
for (Post post: super.postSet){
    found = false;
    parsedText = post.getText().split("[^a-zA-Z]+");
    for (String word: toFilter){
        for (String textWord: parsedText){
            if (word.toLowerCase().equals(textWord.toLowerCase())){
              // Se trovo almeno una parola segnalata, esco dal loop
                found = true;
                post.setFamilyFriendlyOff();
                this.reportedId.add(post.getId());
                break;
            }
        }
        if (found)
            break;
    }
}
```

4.1.4 void restoreWords(String goodWords)

```
if (goodWords.isBlank())
    throw new IllegalArgumentException("Stringa non valida");
String[] toRestore = goodWords.split("[^a-zA-Z]+");
this.badWords.removeAll(Arrays.asList(toRestore));
boolean clean; // Se true, il post non contiene più badWords
boolean found;
String[] parsedText;
for (Post post: super.postSet){
    found = false;
    clean = true;
    parsedText = post.getText().split("[^a-zA-Z]+");
    for (String word: this.badWords){
        for (String textWord: parsedText){
            if (textWord.toLowerCase().equals(word.toLowerCase())) {
                clean = false; // Il post contiente ancora parole proibite.
                found = true;
                break;
            }
        }
        if (outOfLoop)
            break;
    }
    if (clean){
        post.setFamilyFriendlyOn();
        this.reportedId.remove(post.getId());
    }
}
```

4.2 Note conclusive

Le funzioni di ripristino, in un'ipotetica implementazione del social network, dovrebbero essere ad uso esclusivo di un moderatore/admin (in un primo momento scelto dagli utenti del social network) che controlla i post che sono stati segnalati e modera i contenuti presenti nella rete, così da non creare confusione fra gli utenti stessi. Si trova un'implementazione di questa soluzione nel pacchetto `MicroBlogCLI`. Questo sistema è molto rudimentale ed è adatto ad una piccola rete sociale in cui gli utenti che la popolano non sono molti e hanno interessi comuni, in cui un sistema di moderazione dei contenuti è superfluo per la maggior parte delle situazioni. Un'evoluzione naturale di questo approccio potrebbe essere il controllo di quante persone segnalano un determinato post o dei termini specifici, così da attivare un campanello d'allarme quando il contatore di tali segnalazioni supera una certa soglia, permettendo ai moderatori/admin della rete di controllare quel contenuto per poi eventualmente oscurarlo o, al contrario, resettare le segnalazioni. Infine, se, ad esempio, il contatore delle segnalazioni supera la metà del numero degli utenti iscritti o un'altra soglia significativa, si potrebbe oscurare quel contenuto preventivamente per poi far validare questa azione automatica ai moderatori, che possono o mantenere l'oscuramento o ripristinare il post.