PROGRAMMAZIONE II - A.A. 2020-21
Primo Progetto Intermedio
Simone Passèra

Lo scopo del progetto è lo sviluppo di una componente software di supporto alla gestione e l'analisi di una rete sociale (Social Network) denominata MicroBlog. Una persona nella rete sociale è rappresentata e identificata in modo univoco dal nome. Il nome utente è compreso tra 5 e 15 caratteri, può contenere caratteri maiuscoli, minuscoli, numeri e sottolineato. La rete sociale consente agli utenti di inviare messaggi di testo di breve lunghezza, con un massimo di 140 caratteri, chiamati post. Un post è identificato in modo univoco da un numero naturale. Esiste una nozione di "follow", ciò significa che un utente può seguire un altro utente appartenente al social. Gli utenti della rete sociale non possono seguire se stessi ed ognuno può seguire un numero indefinito di utenti. Un utente può mettere un mi piace ad un post solo se segue il creatore del post e può menzionare un utente nei post, solo se segue l' utente menzionato. Intuitivamente un mi piace a livello grafico è rappresentato da un'icona cliccabile presente su ogni post. A livello di implementazione, quando un utente mette un mi piace ad un post, viene codificato all' interno della rete sociale con l'immissione di un post il cui campo di testo è formato esclusivamente da "#LIKE_id" dove id è l' identificatore univoco del post a cui è stato messo mi piace.
Per menzionare altri utenti appartenenti alla rete sociale, all'interno del post, è sufficiente scrivere il nome utente che si vuole menzionare preceduto dal simbolo (@) ad esempio @Nome_utente1 @Nome_utente2 …

Interfaccia Post

Tipo di dato astratto non modificabile.
Post rappresenta un generico messaggio utilizzato all'interno della rete sociale MicroBlog.
Un post è definito da un insieme di informazioni :

id ->	Identificatore univoco del post (numero naturale)
author ->	Utente univoco della rete sociale che ha scritto il post (il nome utente è compreso tra 5 e 15 caratteri, può contenere caratteri maiuscoli, minuscoli, numeri e sottolineato)
text ->	Testo (massimo 140 caratteri) del post
timestamp ->	Data e ora di invio del post

TE : <ID, AUTHOR, TEXT, TIMESTAMP>

Classe Message

Implementa l'interfaccia Post.

IR : id != null
author != null
text != null
timestamp != null

id >= 0
[L'identificatore del post è un numero naturale]

author.matches("[a-zA-Z_0-9]{5,15}") == true
[Il nome utente è compreso tra 5 e 15 caratteri, può contenere caratteri maiuscoli, minuscoli, numeri e sottolineato]

0 < text.length() <= 140
[Il testo del post non può essere vuoto ed ha un massimo di140 caratteri]

AF: AF(id) = identificatore univoco del post
AF(author) = utente univoco della rete sociale che ha scritto il post
AF(text) = testo del post
AF(timestamp) = data e ora di invio del post

Variabili d'istanza e costruttore

```java
static int id_generator = 0;

private final Integer id;
private final String author;
private final String text;
private final Timestamp timestamp;

// Crea un nuovo messaggio, con author, text e timestamp (parametri del costruttore)
public Message(String author, String text, Timestamp timestamp) throws NullPointerException, PostException {
    if(author == null || text == null || timestamp == null) throw new NullPointerException("One or more parameters are null");
    if(!author.matches("[a-zA-Z_0-9]{5,15}")) throw new PostException("Author " + author + " illegal format");
    if(text.isEmpty()) throw new PostException("The text of " + author + " is empty");
    if(text.length() > 140) throw new PostException("The text of " + author + " is too long (max 140 characters)");

    this.id = id_generator++;
    this.author = author;
    this.text = text;
    this.timestamp = (Timestamp) timestamp.clone();
}
/*
   @REQUIRES : author != null && text != null && timestamp != null
   @THROWS : NullPointerException, PostException
   @MODIFIES : id, author, text, timestamp
   @EFFECTS :  Initialize id, author, text, timestamp inferred from parameters
 */
```

Il costruttore per assegnare ad ogni post un identificatore univoco fa uso della variabile statica **id_genarator** inizializzata al valore zero, visibile da tutte le istanze della classe Message. Ad ogni invocazione del costruttore la variabile statica viene assegnata come id al post che si sta creando e subito dopo viene incrementata di uno. Questo fa si di rispettare il vincolo id >= 0 presente nell' invariante di rappresentazione.

Metodi

```
// Restituisce l' identificatore univoco del post
public Integer getId() {
    return id;
}
// @RETURN : id

// Restituisce il nome utente dell' autore del post
public String getAuthor() {
    return author;
}
// @RETURN : author

// Restituisce il corpo del messaggio contenuto nel post
public String getText() {
    return text;
}
// @RETURN : text

// Restituisce data e ora di invio del post
public Timestamp getTimestamp() {
    return (Timestamp) timestamp.clone();
}
// @RETURN : timestamp
```

I metodi presenti sono di tipo getter, in quanto il tda Post non è modificabile.

Interfaccia SocialNetwork

Tipo di dato astratto modificabile.
SocialNetwork rappresenta la componente software di supporto alla gestione e l'analisi della rete sociale MicroBlog.

TE:

<USERS, POSTS>

USERS -> insieme di coppie <utente, insieme delle persone da lui seguite>
POSTS -> insieme di Post <id, utente, testo, data e ora di pubblicazione>

Classe MicroBlog

Implementa l'interfaccia SocialNetwork.

IR : users != null
followers != null
feed != null
mentioned != null

for all keys in users ==>
((key != null) && (key.matches("[a-zA-Z_0-9]{5,15}") == true))
[Il nome utente è compreso tra 5 e 15 caratteri, può contenere caratteri maiuscoli, minuscoli, numeri e sottolineato]

users.keySet().equals(followers.keySet())
[L'insieme dei nomi presenti in users e followers devono coincidere]

for all keys in users ==>
(users.get(key).contains(key) == false)
[Un utente non può seguire se stesso]

for all keys in followers ==> (followers.get(key) >= 0)
[Il numero dei followers appartenenti ad un utente deve essere >= 0]

for all keys in feed ==> (feed.get(key).getId().equals(key) == true)
[La mappa feed <id, post> contiene l'associazione tra id e il post che ha come id esattamente la chiave]

for all keys in feed ==>
(feed.get(key).getText.matches("#LIKE_[0-9]+") ==>
((feed.containsKey(Integer.parseInt("[0-9]+")) == true)
&&
(feed.get(Integer.parseInt("[0-9]+")).getText().matches("#LIKE_[0-9]+") == false)
&&
(users.get(feed.get(key).getAuthor()).contains(feed.get(Integer.parseInt("[0-9]+")).getAuthor()) == true)))
[Tutti i post che rappresentano un mi piace, devono aver messo mi piace ad un post che è presente nella rete sociale che non rappresenta un mi piace ma è semplicemente un post ed inoltre l'autore del mi piace deve seguire l'autore del post a cui a messo mi piace]

for all keys in feed ==>
(for all @user_mention in feed.get(key).getText() ==>
((users.contains(user_mention) == true)
&&
(users.get(feed.get(key).getAuthor()).contains(user_mention) == true)))
[In tutti i post che sono presenti degli utenti menzionati, questi utenti devono esistere all'interno del social network ed inoltre se l'autore di un post menziona un altro utente allora l'autore deve seguire l'utente menzionato]

users.keySet().containsAll(mentioned) == true
[L'insieme degli utenti menzionati presenti in mentioned sono un sottoinsieme di tutti gli utenti presenti nel social network]

AF: AF(users) = Insieme di coppie <utente, insieme delle persone da lui seguite>
AF(followers) = Insieme di coppie <utente, numero di followers>
AF(feed) = Insieme di coppie <id, post> dove post.getId() == id
AF(mentioned) = Sottoinsieme di utenti, che sono stati menzionati in tutti i post

Variabili d'istanza

```
protected Map<String, Set<String>> users;
private Map<String, Integer> followers;
protected Map<Integer, Post> feed;
protected Set<String> mentioned;
```

Users è la mappa richiesta dalla specifica.
La mappa **followers** serve ad ottimizzare il calcolo richiesto dal metodo influencers. **feed** contiene tutti i post del social network, ed è stata utilizza una mappa piuttosto che una lista per velocizzare la ricerca di un post tramite l'id. **mentioned** contiene gli utenti menzionati e l'utilizzo di essa serve per ottimizzare il calcolo di getMentionedUsers().

Metodi statici

```
public static Map<String, Set<String>> guessFollowers(List<Post> ps) throws NullPointerException, IllegalArgumentException
public static Set<String> getMentionedUsers(List<Post> ps) throws NullPointerException, IllegalArgumentException
public static List<Post> writtenBy(List<Post> ps, String username) throws NullPointerException, UsernameException, IllegalArgumentException
```

Questi metodi sono stati resi statici perché ognuno opera su una lista di post passata come parametro, quindi non hanno bisogno di essere invocate da specifiche istanze di MicroBlog.
guessFollowers per ottenere la rete sociale derivata dalla lista di post passata come parametro, utilizza come criterio di derivazione gli stessi vincoli dei post presenti all'interno di MicroBlog.

Interfaccia SocialNetworkReport

Una possibile soluzione per progettare una estensione gerarchica del tipo di dato SocialNetwork che permetta di introdurre un criterio per segnalare contenuti offensivi presenti nella rete sociale, sarebbe quella di permettere agli utenti di segnalare i post che ritengono offensivi, pensando che avvenga a livello intuitivo con un pulsante stile "mi piace" e ancora una volta codificare la segnalazione come un particolare post. In questo modo modificando opportunamente i metodi di SocialNetwork, sarebbe possibile implementare l'estensione e preservare il principio di sostituzione.
La soluzione che ho implementato invece non permette agli utenti di segnalare i post, ma il social network attraverso una lista di parole ritenute offensive già presenti di default, segnala in maniera autonoma i post che contengono almeno una delle parole offensive presenti nel campo di testo.

SocialNetworkReport è un tipo di dato astratto modificabile.

TE: <USERS, POSTS, REPORTS, WORDS>
REPORTS -> insieme dei post segnalati dal social network
WORDS -> lista di parole offensive

Classe MicroBlogReport

Implementa l'interfaccia SocialNetworkReport ed estende MicroBlog.

IR : reports != null
words != null
default_words != null

words.contains("") == false
[Lista di parole non può contenere stringhe vuote]

words.contains(null) == false
[Lista di parole non può contenere valori null]

for all i : 0 <= i < reports.size() ==>
((feed.containsKey(reports.get(i).getId()) == true)
&&
(EXISTS string in words | (reports.get(i).getText() contains string)))
[I post contenuti in reports sono un sottoinsieme di tutti i post presenti nella rete sociale ed ogni post in reports, all'interno del campo di testo c'è almeno una parola contenuta in words]

AF: AF(reports) = Insieme dei post segnalati dal social network
AF(words) = Lista di parole offensive

Variabili d'istanza

```
private Set<Post> reports;
private Set<String> words;
private String default_words = "idrogeno,interprete,canada,poligono,fantasma,freccette";
```

La variabile d'istanza **default_words** contiene la lista di parole offensive che viene assegnata a **words** al momento della creazione di MicroBlogReport.

Costruttori

```
// Crea un nuovo MicroBlogReport vuoto
public MicroBlogReport()
{
    super();

    reports = new HashSet<Post>();
    words = new HashSet<String>(Arrays.asList(default_words.split(",")));
}
/*
   @MODIFIES : users, followers, feed, mentioned, reports, words
   @EFFECTS : Initialize users, followers, feed, mentioned, reports, words
 */

// Crea un nuovo MicroBlogReport inizializzato con la lista di post ps (parametro del costruttore)
public MicroBlogReport(List<Post> ps) throws NullPointerException, IllegalArgumentException, MentionException, LikeException
{
    super(ps);

    reports = new HashSet<Post>();
    words = new HashSet<String>(Arrays.asList(default_words.split(",")));

    reports.addAll(containing(new ArrayList<String>(words)));
}
/*
   @REQUIRES : ps != null && ps.isEmpty() == false && ps.contains(null) == false
               && for all i : 0 <= i < ps.size() ==>
                           (((ps.get(i).getText().contains("@User") == true) ==> (ps.get(i).getAuthor().equals("User") == false))
                           && (ps.get(i).getTimestamp().before(current_time) == true)
                           && ((ps.get(i).getText().matches("#LIKE_[0-9]+") == true) ==>
                                       (EXISTS post in ps | post.getText().matches("#LIKE_[0-9]+") == false
                                                            && post.getAuthor().equals(ps.get(i).getAuthor()) == false
                                                            && post.getId().equals(Integer.parseInt("[0-9]+")) == true
                                                            && ps.get(i).getTimestamp().after(post.getTimestamp()) == true)))
   @THROWS : NullPointerException, IllegalArgumentException, MentionException, LikeException
   @MODIFIES : users, followers, feed, mentioned, reports, words
   @EFFECTS : Initialize users, followers, feed, mentioned inferred from ps list && Initialize reports, words
 */
```

Il costruttori rispettano il principio di sostituzione in quanto chiamano i costruttori padre e inizializzano le variabili d'istanza, quindi la post condizione dei metodi è rafforzata e questo rispettano la regola dei metodi.

Metodo addPost

```
@Override
// Aggiunge un post creato da username, e se il testo del post contiene
// almeno una parola offensiva, il post viene segnalato dal social network
public Integer addPost(String username, String text) throws NullPointerException, UsernameException, LikeException, MentionException, PostException {
    if(username == null || text == null) throw new NullPointerException();
    if(!users.containsKey(username)) throw new UsernameException("Username " + username + " not exists");
    if(text.isEmpty()) throw new PostException("The text of " + username + " is empty");
    if(text.length() > 140) throw new PostException("The text of " + username + " is too long (max 140 characters)");

    if(text.matches("#LIKE_[0-9]+"))
    {
        int id = Integer.parseInt(text.substring(6));

        if(!feed.containsKey(id)) throw new LikeException("Post " + id + " not exists");
        if(feed.get(id).getText().matches("#LIKE_[0-9]+")) throw new LikeException("Post " + id + " is a like");
        if(!users.get(username).contains(feed.get(id).getAuthor())) throw new LikeException("You can't like " + id);
    }
    else
    {
        Pattern mention = Pattern.compile("@([a-zA-Z_0-9]{5,15})");
        Matcher m = mention.matcher(text);
        String u;
        Set<String> mentioned = new HashSet<String>();
```

```
        while(m.find())
        {
            u = m.group(1);
            mentioned.add(u);

            if(!users.containsKey(u)) throw new MentionException("Username mentioned " + u + " not exists");
            if(!users.get(username).contains(u)) throw new MentionException("You can't mention " + u);
        }

        this.mentioned.addAll(mentioned);
    }

    Post post = new Message(username, text, new Timestamp(System.currentTimeMillis()));
    feed.put(post.getId(), post);

    StringBuilder regex = new StringBuilder();

    for(String w : words)
    {
        regex.append(w).append("|");
    }

    Matcher words_list = Pattern.compile(regex.deleteCharAt(regex.length()-1).toString()).matcher(text);

    if(words_list.find()) reports.add(post);

    return post.getId();
}
/*
   @REQUIRES : username != null && text != null && users.containsKey(username) == true
   @THROWS : NullPointerException, UsernameException, LikeException, MentionException, PostException
   @MODIFIES : feed, mentioned, reports
   @EFFECTS : feed.put(new_post.getId(), new_post) && (reports.add(post) if EXISTS string in words | (post.getText() contains string))
   @RETURN : new_post.getId()
 */
```

Il metodo **addPost** viene sovrascritto per controllare se il post inserito deve essere segnalato. Anche in questo caso la modifica che viene fatta rafforza la post condizione, e dunque il principio di sostituzione è preservato.

Eccezioni personalizzate

PostException ->	Segnala errori presenti nei post relativi al testo oppure all'autore.
FollowerException ->	Segnala errori relativi ai vincoli sulla nozione di "follow".
LikeException ->	Segnala errori relativi ai vincoli sui "mi piace"
MentionException ->	Segnala errori presenti nei testi dei post relativi agli utenti menzionati.
UsernameException ->	Segnala errori relativi all'esistenza degli utenti oppure al formato degli utenti.

Classe Test

Nella classe Test è presente una batteria di test, che va a testare le precondizioni dei metodi e i valori restituiti.
L'esecuzione della classe Test mostra nel terminale l'esito di ogni test con il seguente formato:

<codice eseguito> **->** <messaggio di errore contenuto nelle eccezioni>
oppure
<codice eseguito> **->** <risultato del metodo testato>

Per eseguire la classe Test è sufficiente compilare i file sorgenti nella cartella src ed eseguire ila classe Test :

```
cd src
javac *
java Test
```