RELAZIONE PRIMO PROGETTO – GIANLUCA PANZANI – CORSO A:

SUDDIVISIONE DEI FILE:

- `SocialNetwork.java` → interfaccia.
- `MicroBlog.java` → classe che implementa l'interfaccia SocialNetwork.
- MicroBlogSegnalation.java → sottoclasse di MicroBlog che la estende con l'operazione di segnalazione dei post.
- `PostInterface.java` → interfaccia.
- `Post.java` → classe che implementa l'interfaccia PostInterface.
- Main.java → file usato come batteria dei test del progetto.
- `Exception/BadLikeException.java` → classe usata per il sollevamento di eccezioni dovute al formato errato del nome utente.
- `Exception/OutOfRangeException.java` → classe usata per il sollevamento di eccezioni a causa di valori che non rispettano un range di valori ben definito.
- `Exception/EmptyPostException.java` → classe usata per il sollevamento di eccezioni a causa della creazione di post privi di testo.
- `Exception/PostNotFoundException.java` → classe usata per il sollevamento di eccezioni a causa della ricerca di post inesistenti nel Social Network.
- `Exception/UserNotFoundException.java` → classe usata per il sollevamento di eccezioni a causa della ricerca di utenti non presenti nel Social Network.
- `exec` → file eseguibile.

IMPLEMENTAZIONE DELLA CLASSE POST:

Un oggetto di tipo Post è formato dai seguenti campi privati:
`<id, counter, author, text, clock, mentiones>.`
Le proprietá dei campi sono elencate di seguito:

- id → è univoco e quindi un oggetto di tipo post si differenzia da un altro.
- counter → questo campo static rende possibile la differenziazione degli tra oggetti diversi in quanto viene incrementato ad ogni esecuzione del costruttore di Post e viene successivamente concatenato al nome dell'autore del post (formando cosí l'id). E' inoltre necessario aggiungere che si tratta di una variabile inesistente agli occhi dell'utente il quale per il principio di astrazione non è tenuto a sapere della sua esistenza e dei vari dettagli implementativi (anche le altre non sono visibili all'esterno ma l'utente puó interagire con esse attraverso i metodi, cosa che invece non accade con questa).
- author → stringa che indica l'autore del post.
- text → stringa che indica il testo del post.
- clock → corrisponde al timestamp indicato tra i campi privati della classe. Ci permette di memorizzare l'istante di creazione di un post e di conseguenza orario e data di creazione.
- mentiones → corrisponde al dato mentioned presente nei campi privati della classe. Si tratta dell'insieme contenente gli utenti (o autori) menzionati nel post.

Di seguito ho riportato le dichiarazioni dei metodi di questa classe con una breve descrizione di questi:

```
public Post(String author, String text, Set<String> mentioned);
```

Ha l'unico compito di inizializzare tutti i campi privati della classe e di creare un id univoco.

```
public void printInfo();
```

Stampa a video le informazioni relative al post (escluso i menzionati). Il compito di stampare questi ultimi è stato delegato ad un metodo della classe MicroBlog per il semplice fatto che non sarebbe corretto stampare utenti che non sono presenti sulla rete sociale. Cosa di cui un oggetto di tipo Post non puó essere a conoscenza.

```
public void printSignalatedInfo();
```

Stampa a video le informazioni di un post con il testo censurato. Questo metodo viene chiamato dal Social Network relativo alla classe MicroBlogSignalation se e solo se l'id del post è presente nella lista dei post segnalati. Puó essere chiamato anche dal singolo post ma non avrebbe senso in quanto questo non ha modo di capire se è stato segnalato o meno. Il compito di questo metodo è quello di fornire una funzionalitá ulteriore a chi la usa.

```
public String getDataAndHours();
```

Restituisce una copia delle informazioni relative al timestamp dell'oggetto, ovvero l'ora e la data di creazione del post (copy-out).

```
public String getId();
```

Ritorna una copia dell'id del post (copy-out).

```
public String getText();
```

Ritorna una copia del testo del post (copy-out).

```
public String getAuthor();
```

Ritorna una copia del nome utente associato al post, ovvero dell'autore del post (copy-out).

```
public HashSet<String> getMentioned();
```

Ritorna una copia dell'insieme degli utenti menzionati nel post (copy-out), se presenti.

IMPLEMENTAZIONE DELLA CLASSE MICROBLOG:

Un oggetto di tipo MicroBlog è formato dai tre campi privati che seguono:
`<postMap, socialMap, likeMap>`
Riporto anche una breve descrizione di questi:
- postMap → struttura dati che associa l'id di un post al post stesso.
- socialMap → struttura dati che associa ad un utente l'insieme degli utenti che segue, ovvero gli utenti che hanno caricato sul social network almeno un post e a cui l'utente chiave della map ha messo almeno un like.
- likeMap → struttura dati che associa all'id di un post l'insieme degli utenti che hanno messo like al post.

Per quanto riguarda i metodi la scelta generale è stata di dichiarare quelli forniti dal testo del progetto nell'interfaccia SocialNetwork e di implementarli nella classe MicroBlog, nome del Social Network, per poi aggiungere anche i metodi che ho ritenuto necessari per la gestione di una rete sociale.
La dichiarazione completa dei metodi, con la relativa descrizione, è riportata qui sotto:

```
public MicroBlog()
```

Il costruttore ha l'unico compito di inizializzare i 3 campi privati della classe coma vuoti.

```
public void addPost(Post ps);
```

Permette di aggiungere un oggetto di tipo Post al Social Network. Un post puó essere di 2 tipi: un post con testo standard o un post il cui testo ha il formato di un like. In quest'ultimo caso, quello in cui il post è un "like" ad un altro post, avrá come formato del testo: "_like_idPost".

```
public boolean postMapContainsKey(String s);
```

Ritorna true se postMap contiene il post con id "s" passato come argomento, false altrimenti. Utile per permettere controlli di presenza di un post all'esterno della classe.

```
public Set<String> socialMapOf(String username);
```

Ritorna l'insieme di utenti seguiti dall'utente passato come parametro, se è presente nella rete sociale.

```
public void printPostInfo(Post ps);
```

Chiama il metodo `printInfo()` del post passato come parametro ed in piú stampa gli utenti menzionati.

Possibile domanda: perché non vengono stampati anche gli utenti menzionati nel post direttamente dal metodo `printInfo()`? Semplicemente perché gli utenti menzionati devono essere presenti nel social network e piú precisamente devono essere tra i seguiti dall'utente che li menziona. Quindi non é possibile tale verifica all'interno della classe Post in quanto non ha accesso alle strutture dati private della classe MicroBlog.

```
public void printAllPosts();
```

Stampa tutte le informazioni relative a tutti i post presenti nella rete sociale (tranne i post con formato "like"). Chiama il metodo `printPostInfo(Post ps)` per ogni post presente nella postMap.

```
public Map<String, Set<String>> guessFollowers(List<Post> psList);
```

Data una lista di post ritorna la rete sociale derivata dagli utenti, autori del post, ai quali é associato l'insieme di tutti gli altri utenti che lo seguono.

```
public List<String> influencers();
```

Ritorna la lista di stringhe formate dalla coppia (utente, #followers) per tutti quegli utenti il cui numero di followers é maggiore della media di followers tra tutti gli utenti nella rete sociale.

```
public Set<String> getMentionedUsers();
```

Ritorna l'insieme contenente tutti gli utenti menzionati nei post sulla rete sociale.

```
public Set<String> getMentionedUsers(List<Post> ps);
```

Ritorna l'insieme contenente tutti gli utenti menzionati nei post presenti nella lista passata come parametro del metodo.

```
public List<Post> writtenBy(String username);
```

Ritorna la lista di post presenti nella rete sociale, scritti dall'utente passato come parametro.

```
public List<Post> writtenBy(List<Post> ps, String username);
```

Ritorna la lista di post scritti dall'utente passato come secondo parametro e presenti nella lista passa come primo parametro.

```
public List<Post> containing(List<String> words);
```

Ritorna la lista di post contenenti almeno una delle parole presenti nella lista passata come parametro.

IMPLEMENTAZIONE DELLA CLASSE MICROBLOGSIGNALATION:

Questa classe estende la classe MicroBlog con un aggiunta di due campi privati. Quindi un oggetto di tipo MicroBlogSignalation é caratterizzato dai seguenti campi privati: `<postMap, socialMap, likeMap, wordsList, signaledList>`.

I campi aggiunti hanno le seguenti proprietá:

- wordsList → lista di tipo `final` le cui parole sono definite dal progettista del social network.
- signaledList → lista di post segnalati. Viene scansionata al momento della stampa per censurare i post contenuti in essa.

I metodi definiti all'interno di questa classe sono i seguenti:

```
public MicroBlogSignalation();
```

Costruttore della classe che inizializza tutti i campi privati (compresi quelli della superclasse chiamando il rispettivo costruttore con `super()`). In particolare aggiunge le 'bad words' alla lista `wordsList`.

```
public boolean postSignalation(Post ps);
```

Permette ad un utente anonimo di segnalare il post passato come parametro. Se il post contiene almeno una delle 'bad words' in `wordsList` allora aggiunge il post alla lista di post segnalati (che al momento di una stampa del post verranno censurati) e ritorna `true`, altrimenti ritorna `false`.

```
public void printPostInfo(Post ps);
```

Ridefinizione (override) del metodo definito in MicroBlog. La modifica che andava implementata era quella della censura dei post segnalati. Infatti é stato aggiunto un `if` per la decisione del tipo di stampa: stampa col metodo `printInfo()` o stampa col metodo `printSignalatedInfo()`.

Possiamo dimostrare con la Regola dei Metodi che vale il Principio di Sostituzione Liskov:

```
pre(super) ==> pre(sub) && [pre(super) && post(sub) ==> post(super)]
```

Allora:

```
1) pre(super) ==> pre(sub) == true
2) pre(super) && post(sub) ==> post(super) == true
```

Verifichiamole:

```
1)    [ps != null ==> ps != null] == true                    OK!
2)    [ps != null && print(ps) ==> print(ps)] == true        OK!
```

Il significato logico di print(ps) é definito nella sezione "specifica e notazione" (poco piú avanti nella relazione).

FILE MAIN (BATTERIA DEI TEST):

Il file `Main.java` contiene il main appunto, il quale contiene il codice corrispondente alla batteria dei test. Il metodo migliore per comprendere facilmente l'esecuzione é quello di seguire contemporaneamente il file `Main.java` e il terminale con le varie stampe dovute all'esecuzione della batteria dei test.

I commenti nel file indicano cosa stiamo testando e in che modo, ai quali corrispondono delle stampe del formato "***TEST DEL METODO xxxxx (tipo di test sul metodo)***".

Gli output corrispondenti ad ogni test saranno di 3 tipi:

- Ok! → se il test non doveva stampare nulla e cosí é stato.
- stampe riferite al metodo → che possono essere solo stampe corrette o stampe corrette miste a messaggi dovuti a sollevamenti di eccezioni (i quali sono indicati con un commento nel file, tranne per i sollevamenti di eccezioni impliciti e intuibili dall'header del test).
- sollevamenti di eccezioni → in questi casi verrá stampato solamente il messaggio associato all'eccezione sollevata (e gestita).

SPECIFICA E NOTAZIONE:

La specifica é stata definita appena sopra l'implementazione dei vari metodi ed é suddivisa in due parti: una parte informale e una parte formale. La prima é una spiegazione scritta in linguaggio comune, la seconda invece é una formula logica.

La notazione usata nelle formule logiche si basa su:

- Dot Notation → usata ad esempio per accedere al testo associato al post ps con la seguente segnatura `ps.text`.
- Sintassi per le Map → xxxxxMap.key = chiave generica
  xxxxxMap(key) = oggetto associato alla chiave key

- Funzioni usate:
    * print(x) → indica la scrittura a video delle informazioni relative all'oggetto x.
    * return(x) → indica che il ritorno del metodo é l'oggetto x.
    * init(x) → inizializzazione dell'oggetto x a "vuoto".
    * quelle presenti anche nel linguaggio java, seppur trattate in modo astratto.
- forall x . … → indica "per ogni x t.c. …", quindi possiamo dire che si tratta di una versione astratta di un ciclo for (talvolta anche di un for dentro un altro for).

Parlando di Invariante di Rappresentazione, analizzerei i seguenti Metodi Mutators:
1) per MicroBlog:
→ addPost: modifica i 3 campi privati (anche se non sempre tutti). Prendiamo in esame le modifiche sulle diverse strutture dati separatamente. La struttura dati postMap non contiene key di post diversi che mappano sugli stessi oggetti in quanto univoche e vengono aggiornati per ogni post che supera le condizioni imposte; la struttura dati likeMap aggiunge sempre l'autore del like e non contiene duplicati perché usa sempre come keys gli id dei post; la struttura dati socialMap aggiunge sempre gli utenti che non avevano mai postato sulla rete e viene aggiornata ogni volta che un utente mette like per tenere traccia dei follows.
Inoltre, siccome ogni volta che si aggiunge un post alla rete questo sará diverso dai precedenti, il numero di post aumenta di 1 ad ogni esecuzione di addPost (priva di sollevamento di eccezione). Il numero di keys nella socialMap corrisponde al numero di utenti e quindi possono esserci al piú N utenti se ci sono N post. Il numero di post nella likeMap deve essere necessariamente inferiore al numero di post esistenti nella postMap in quanto non é possibile mettere like ad un post se la rete sociale é vuota (oltre al fatto che gli id dei post con formato di like non finiscono nella likeMap. Piú precisamente il numero di post nella likeMap sará al piú N/2 visto che, affinché un post risulti nella likeMap, deve ricevere almeno 1 like. Quindi se mettiamo 1 like per ogni post aggiunto massimizzeremo il numero di post nella likeMap e avremo, su un totale di N post, N/2 post di like (che non saranno nella likeMap in quanto semplici post) e N/2 post nella likeMap per il like ricevuto. Inoltre il like Questo dimostra che il RepInv non viene violato.
2) per MicroBlogSignalation:
→ addPost: essendo uguale a quello di MicroBlog non aggiungo altro.
→ postSignalation: modifica solo la struttura dati che tiene traccia dei post segnalati. Ha l'unico compito di aggiungere il post nel caso in cui sia stato giustamente segnalato. Questo non viola il RepInv in quanto questo impone solo il fatto che non debba essere null. Ma visto che viene istanziato nel costruttore e visto che non sono state implementate rimozioni, non va contro il RI. Inoltre il numero di post segnalabili é al piú N, cioé pari al numero di post sulla rete sociale.

ESECUZIONE DEL CODICE:

L'esecuzione puó essere fatta con l'istruzione ./`exec`, cioé eseguendo uno script bash sul quale é stato applicato il comando `chmod 755`.
Questo compilerá ed eseguirá tutti i file java all'interno della cartella. Inoltre rimuoverá i file class creati dalla compilazione del codice.

FINE