UNIVERSITÀ DI PISA

Dipartimento di Informatica

Corso di Laurea in Informatica

Relazione del progetto midterm in Java per il corso di Programmazione 2

A cura di Alessandro Cheli

Insegnanti:

Prof. Gianluigi Ferrari

Prof.ssa Francesca Levi

Studente:

Alessandro Cheli

583350 Corso A

Sessione autunnale

Anno Accademico 20190/2020

Capitolo 1

Relazione del Progetto

1.1 Istruzioni

Il progetto è stato realizzato utilizzando **Apache Maven** come strumento di build e gestione delle dipendenze. Alcune clausole sono specificate con la sintassi **Javadoc** (i parametri sono documentati con la clausola *@params* invece che con la clausola *@requires*). Dopo aver installato JDK di versione uguale o superiore alla 8 e la dipendenza Apache Maven si può compilare ed eseguire il progetto eseguendo.

```
mvn compile
mvn package
java -jar target/application-1.0-SNAPSHOT.jar
```

In alternativa si può compilare il progetto utilizzando javac ma non verranno spiegati nella relazione i dettagli. Apache Maven scaricherà e compilerà automaticamente le dipendenze necessarie (per generare Javadoc e creare un archivio `.jar` corretto). Non vengono utilizzate librerie esterne a `java.util`.

Per visualizzare la documentazione **Javadoc** si può eseguire, dopo la compilazione

```
# entro nella directory dove risiedono i file compilati
cd application/target
# creo una cartella dove estrarre i file html
mkdir javadoc
```

```
# estraggo il contenuto dell'archivio jar dove
# sono contenuti i Javadoc
unzip application-1.0-SNAPSHOT-javadoc.jar -d javadoc/
# eseguire un server http all'interno della
# directory contenente i file html
# ad esempio darkhttpd
darkhttpd javadoc/
darkhttpd/1.12, copyright (c) 2003-2016 Emil Mikulic.
listening on: http://0.0.0.0:8080/
# aprire un browser all'indirizzo localhost:8080
```

1.2 Dettagli di Implementazione

La classe `Board<E extends DataElement>` implementa l'interfaccia `DataBoard<E extends DataElement>` mantenendo i contenuti (post) all'interno di una tabella hash `private HashMap<String, TreeSet<E>> contents;`. Dove la chiave è la categoria e il contenuto è un insieme di post ordinato per numero di like e lessicograficamente. (Si veda `DataElement.compareTo` per l'ordinamento dei post). Gli amici vengono contenuti in una tabella hash `private HashMap<String, HashSet<String>> friends;` Dove la chiave è la categoria e i valori sono gli insiemi di amici che possono visualizzare tale categoria.

La seconda implementazione `Board2<E extends DataElement>` utilizza una classe `Category<E extends DataElement>` che contiene, oltre al nome della categoria, un `TreeSet<E>` per mantenere i post ordinati e un `HashSet<String>` per mantenere i nomi degli amici che possono visualizzare la categoria. Nella classe `Board2` si usa una `ArrayList<Category>` per mantenere la lista delle categorie e si effettua il controllo dell'unicità a mano sul campo `c.category` $\forall$ `Category c` tale che `c` è contenunta all'interno della board.

Entrambe le implementazioni effettuano rigorosi controlli sull'unicità, clonazione e correttezza dei valori dove tali attributi sono richiesti. In un'applicazione reale si utilizzerebbe, oltre che ad un database *SQL* per la permanenza dei dati serializzati, anche dei

metodi di crittografia come *BCRYPT+Salt* per la sicurezza della gestione delle password e dei contenuti, ed un server che espone endpoint di una *API HTTP*. Per semplicità, il progetto non richiede nessuna di tali funzionalità e l'implementazione e la batteria di test non gestiscono crittografia o permanenza dei dati.

1.3 Dettagli sulla batteria di test

La batteria di test è realizzata per semplicità **a mano** nel metodo main e non con **Junit**. Viene controllata la correttezza delle eccezioni lanciate all'interno di *worst case* appositi dove si intende ottenere tali eccezioni, e vengono eseguite operazioni comuni su tutti i metodi di entrambe le classi `Board` e `Board2`. Nel caso le operazioni di *normal usage* lancino un eccezione non aspettata viene terminata l'esecuzione del programma di test con un messaggio di *FATAL ERROR*. Se un eccezione lanciata in un *worst case* non rispetta il tipo di eccezione che si aspettava viene riportato un messaggio *FAIL*, altrimenti un messaggio *PASS*. Per controllare che l'esecuzione del main di test non contenga messaggi *FAIL* o *FATAL* si può eseguire

```
java -jar application/target/application-1.0-SNAPSHOT.jar 2>&1 \
    | grep -F -e FAIL -e FATAL
```