CMPE 275 Section 1, Fall 2018

Lab 1 - Aspect Oriented Programming

Status: *published*

Last updated: 09/29/2018

In this lab, you implement the retry and stats concerns to a tweeting service through Aspect Oriented Programming (AOP). Please note this is an individual assignment.

The tweet service is defined as follows:

```
package edu.sjsu.cmpe275.lab1;

import java.io.IOException;

public interface TweetService {
  /**
   * @throws IllegalArgumentException if the message is more than 140 characters as measured by string length.
   * @throws IOException if there is a network failure
   */
  void tweet(String user, String message) throws IllegalArgumentException, IOException;

  /**
   * @throws IOException if there is a network failure
   */
  void follow(String follower, String followee) throws IOException;

  /**
    *
   * @throws IOException if there is a network failure
   */
  void block(String user, String followee) throws IOException;

}
```

Since network failure happens relatively frequently, you are asked to add the feature to automatically retry for up to three times for a network failure (indicated by an IOException). (Please note the three retries are in addition to the original failed invocation.) You are also asked to implement the following TweetStats service:

```
package edu.sjsu.cmpe275.lab1;

public interface TweetStats {

  /**
   * reset all the measurements and all the following/blocking relationship as well.
   */
  void resetStatsandSystem();

  /**
   * @return the length of longest message successfully sent or attempted since the beginning or last reset. Can be more than 140. If no messages succeeded, return 0.
   * Failed messages are counted for this as well.
   */
  int getLengthOfLongestTweet();
  /**
   * @return the user who has been followed by the biggest number of different users since the beginning or last reset. If there is a tie,
   * return the 1st of such users based on alphabetical order. If the follow action did not succeed, it does not count toward the stats. If no users were successfully followed, return null.
   */
  String getMostFollowedUser();

 /**
   * @return the message that has been shared with the biggest number of different followers when it is successfully tweaked. If the same message (based on string equality) has been tweeted more than once, it is considered as different message for this purpose.
   */
  String getMostPopularMessage();
```

```
  /**
   * The most productive user is determined by the total length of all the messages successfully tweeted since the beginning
   * or last reset. If there is a tie, return the 1st of such users based on alphabetical order. If no users successfully tweeted, return null.
   * @return the most productive user.
   */
  String getMostProductiveUser();

/**
    * @return the user who has been successfully blocked by the biggest number of
    *     different users since the beginning or last reset. If there is a
    *     tie, return the 1st of such users based on alphabetical order.
    *     If no follower has been successfully blocked by anyone, return null.
    */
   String getMostBlockedFollower();
}
```

Your implementation of the two concerns need to be done in the two files: *RetryAspect.java and StatsAspect.java*

You **do not** need to worry about multi-threading; i.e., you can assume invocations on the tweet service and stats service will come from only one thread.

W.r.t. *follow* and *block*, the two actions do not directly interfere with each other, i.e., Alice can block Bob, and after that Bob can still successfully follow Alice, as far the success of service invocations are considered. The end effect, however, is that when Alice sends a tweek, Bob cannot receive it, since he has been blocked. Both *follow* and *block* get cleared upon system reset.

For your testing purpose, you need to provide your own implementation of *TweetServiceImpl.java,* and simulate failures, but you do not need to submit this file, as the TA will use his own implementation(s) for grading purpose.

Project Setup

The setup of the project can be accessed HERE, including the build file with dependencies, application context, and Java files.

Example Stats

The following examples are assuming stats are reset() before running every single example. Additional test cases will be used for grading.

1. Tweet message as tweet(“foo”,”barbar”). Then getLengthOfLongestTweet() returns 6.
2. Alice follows Bob, Carl follows Bob (but fails to do so), and Bob follows Alice. getMostFollowedUser() returns “Alice”.
3. Successfully tweet a message ("Alice","[any message <= 140 chars]"), then getMostProductiveUser() returns “Alice”.

Submission

Please submit through Canvas, only the three java files, *RetryAspect.java, StatsAspect.java, and TweetStatsImpl.java.* The code you submit must compile with the given project setup. Your three java files CANNOT include any additional classes or packages, except those under java.util or those already provided in the given build dependencies. If your code does not compile with the TA’s code because of extra inclusion or dependency, you automatically lose most of your correctness points.

Due date

Pleaser refer to Canvas.

Grading:

This lab has total points of 6, all based on the correctness of the implementation.