# CMPE 275 Section 1, Spring 2020
# Lab 1 - Aspect Oriented Programming
## Status: *published*

*Last updated: 03/5/2020*

In this lab, you implement the retry, validation, 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 parameter is null, empty, more than 140
characters as measured by string length, or the user parameter is null or empty.
   * @throws IOException if there is a network failure
   */
  void tweet(String user, String message) throws IllegalArgumentException, IOException;

  /**
   * @throws IllegalArgumentException if the follower or followee is null or empty.
   * @throws  UnsupportedOperationException if follower and followee are two non-empty
strings that equal to each other.
   * @throws IOException if there is a network failure
   */
  void follow(String follower, String followee) throws  IllegalArgumentException,
UnsupportedOperationException, IOException;

  /**
```

```
    * This method allows a user to block a follower or a potential follower so that subsequently
tweeted messages will not be shared with the latter, unless unblocked. The same block
operation can be repeated.
     * @throws IllegalArgumentException if the user or follower is null or empty.
       @throws UnsupportedOperationException if attempts to block himself.
    * @throws IOException if there is a network failure
    */
   void block(String user, String follower) throws IllegalArgumentException,
UnsupportedOperationException, IOException;

/**
     *  This method undoes previous blocks.
    * @throws IOException if there is a network failure
    * @throws IllegalArgumentException if the follower or followee is null or empty.
    * @throws UnsupportedOperationException  if follower is not currently blocked by user, or
user attempts to unblock himself.
    */
   void unblock(String user, String follower) throws IllegalArgumentException
, UnsupportedOperationException, 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 relationships as well.
    */
   void resetStatsandSystem();
   /**
    * @return the length of the longest message successfully sent since the beginning or last
reset. Cannot be more than 140. If no messages succeeded, return 0.
    * Failed messages are not counted for this.
    */
   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 the alphabetical order. If the follow action did not
succeed (e.g., due to network errors), it does not count toward the stats. If no users were
successfully followed, return null.  Blocking or not does not affect this metric.
    */
   String getMostFollowedUser();

  /**
    * @return the message that has been shared with the biggest number of different followers
when it is successfully tweeted. If the same message (based on string equality, case sensitive)
has been tweeted more than once, it is considered as the same message for this purpose.
Return based on dictionary order if there is a tie.
    */
   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 for the biggest number tweets
     *  since the beginning or last reset. If there is a
     *  tie, return the 1st of such users based on alphabetical order.
     *  If the same message is tweeted twice, it's considered as two tweets.
     *     If no follower has been successfully blocked by anyone, return null.
     */
    String getMostBlockedFollowerByNumberOfMissedTweets();

/**
     * @return the user who is currently 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. Please
     *     also note that if Alice is not currently following Bob, Bob is not counted toward
     *     this metric for Alice even if Bob is currently blocking Alice, as Alice is
     *     technically not a follower for Bob. If no follower has been successfully
     *     blocked by anyone, return null.
     */
    String getMostBlockedFollowerByNumberOfFollowees();
}
```

Your implementation of the three concerns need to be done in the three files: *RetryAspect.java, ValidationAspect.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 the starter 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 four java files, *RetryAspect.java, ValidationAspect.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

Please refer to Canvas.

## Grading:

This lab has total points of 8, all based on the correctness of the implementation.