CMPE 275 Section 1, Spring 2022

Lab 1 - Aspect Oriented Programming

Status: *published*

*Last updated: 03/11/2022*

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.aop.tweet;*

*import java.io.IOException;*

*public interface TweetService {*

*/\*\**

*\* A proper implementation of this method returns a unique key in UUID for the given*

*\* message. Every time the message is called, the returned key will be different even if*

*\* the message content is the same.*

***\* @throws*** *IllegalArgumentException if the message is more than 140 characters*

*\* as measured by string length, or any*

*\* parameter is null or empty.*

*\** ***@throws*** *IOException if there is a network failure.*

*\** ***@returns*** *a unique message ID*

*\*/*

***UUID*** *tweet(String user, String message)* ***throws*** *IllegalArgumentException, IOException;*

*/\*\**

*\* This method allows a user A to reply with message x to the original message y by another user B. The reply x is shared with B and all the active followers of the current user A. If A has blocked B or the original message y has not been shared with A, A cannot reply to y, hence an exception will be thrown. The replying message x will be treated as* ***a regular message for all practical purposes (e.g., metrics calculation) unless noted otherwise****.*

*\** ***@returns*** *a unique ID for the replying message, different every time this method is called.*

*\** ***@throws*** *IllegalArgumentException if any parameter is null or empty, the UUID is invalid, or*

*\* when a user attempts to directly reply to a message by themselves.*

*\** ***@throws*** *IOException if there is a network failure.*

*\** ***@throws*** *AccessControlException if the current user has not been shared with the original message or the current user has blocked the original sender.*

*\*/*

***UUID*** *reply(String user, UUID originalMessage, String message)* ***throws*** *IOException, IOException;*

*}*

*/\*\**

*\* If Alice follows Bob, and Bob has not blocked Alice before, any future*

*\* message or reply that Bob tweets after this are shared with Alice. If at*

*\* any point Bob blocks Alice, the sharing after blocking will be stopped.*

*\**

*\** ***@throws*** *IllegalArgumentException if either parameter is null or empty, or*

*\* when a user attempts to follow himself.*

*\** ***@throws*** *IOException if there is a network failure.*

*\*/*

***void*** *follow(String follower, String followee)* ***throws*** *IllegalArgumentException, IOException;*

*/\*\**

*\* This method allows a user to block a follower or a potential follower so that*

*\* subsequently tweets will not be shared with the latter. The*

*\* same block operation can be repeated.*

*\** ***@throws*** *IllegalArgumentException if either parameter is null or empty, or*

*\* when a user attempts to block himself.*

*\** ***@throws*** *IOException if there is a network failure.*

*\*/*

***void*** *block(String user, String follower)* ***throws*** *IOException, IOException;*

*/\*\**

*\* Show fondness of the message with the given message ID. The given user must*

*\* have been successfully shared with the given message in order*

*\* to like it. One can only like a message with a given message ID once.*

*\* As a special case, one is not allowed to like their own message. Another case of*

*\* interest to note is: suppose B replies to A’s message x with mess y, A can like*

*\* y even though A has not been following B - the key here is A has received y.*

*\**

*\**

*\** ***@throws*** *IllegalArgumentException if any parameter is null or empty*

*\** ***@throws*** *IOException if there is a network failure.*

*\** ***@throws*** *AccessControlException if the given user is not following the*

*\* sender of the given message, or the sender*

*\* has blocked the given user, the given*

*\* message does not exist, someone tries to like his own messages*

*\* or when the message with the given ID is already*

*\* successfully liked by the same user.*

*\*/*

***void*** *like(String user,* ***UUID*** *messageId)* ***throws*** *AccessControlException, IllegalArgumentException, IOException;*

Since network failures happen relatively frequently, you are asked to implement the crosscutting concern to automatically retry for up to *three* times for network failures (indicated by an IOException). (Please note the three retries are *in addition to* the original failed invocation, i.e., a total of four invocations should take place before an IOException is eventual thrown. )

You are also asked to implement the following TweetStatsService:

*package edu.sjsu.cmpe275.aop.tweet;*

*public interface TweetStatsService {*

*/\*\**

*\* Reset all the measurements. For the purposes of this lab, it also resets the*

*\* following and blocking records as if the system is starting fresh for any*

*\* purpose related to the metrics below.*

*\*/*

***void*** *resetStatsAndSystem();*

*/\*\**

*\** ***@returns*** *the length of longest message a user successfully sent since the*

*\* beginning or last reset. Replied messages count as well, but each replying message is an independent message on its own.*

*If no messages were successfully tweeted,*

*\* return 0.*

*\*/*

***int*** *getLengthOfLongestTweet();*

*/\*\**

*\** ***@returns*** *the user who is being 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 any follower*

*\* has been blocked by the followee, this follower Still count; i.e.,*

*\* Blocking or not does not affect this metric. If someone follows*

*\* him/herself, it does not count. If no users are followed by*

*\* anybody, return null.*

*\*/*

*String getMostFollowedUser();*

*/\*\**

*\** ***@returns*** *the message that has been shared with the biggest number of*

*\* unique recipients when it is successfully tweeted. If two messages have the same string content but different UUIDs, they are considered different for the purpose here.*

*\* If there is a tie, return the message whose UUID is smaller. If no shared messages, return null.*

*\* The very original sender of a message will NOT be counted*

*\* toward the number of shared users for this purpose, unless somebody else*

*\* has successfully shared the same message (based on string equality) with him.*

*\*/*

*UUID getMostPopularMessage();*

*/\*\**

*\** ***@returns*** *the ID of the message that has been successfully liked by the biggest*

*\* number of unique recipients when it is successfully tweeted. If two messages*

*\* are equal based on string equality but have different message IDs,*

*\* they are considered as different message for this*

*\* purpose.*

*\* If there is a tie in the number of different recipients, return the smallest message*

*\* ID. If no shared messages, return null.*

*\*/*

*UUID getMostLikedMessage();*

*/\*\**

*\* The most productive replier is determined by the total length measured in character count of all the*

*\* messages successfully tweeted as a reply to another message 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.  
 \**

*\** ***@returns*** *the most productive user.*

*\*/*

*String getMostProductiveReplier();*

*/\*\**

*\** ***@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. If no*

*\* follower has been successfully blocked by anyone, return null.*

*\*/*

*String getMostUnpopularFollower();*

*/\*\**

*\** ***@return*** *find the longest message thread by the number of messages in the message/reply path, and return the message ID of the last reply or message in the path. When there is a tie, return the smallest UUID.*

*\*/*

*UUID getLongestMessageThread();*

*}*

You are expected to implement the above mentioned concerns / features in: *AccesControlAspect.java, RetryAspect.java,ValidationAspect.jva, StatsAspect.java, and TweetStatsServiceImpl.java.*

For example, the permission check for *like* needs to be done through *AccessControlAspect.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 follow Alice. The end effect, however, is that when Alice sends a tweet, Bob cannot receive it, since he has been blocked. Both *follow* and *block* get cleared upon system reset.

For your testing purposes, you need to provide your own implementation of *TweetServiceImpl.java,* and simulate failures, but you ***must not*** submit this file, as the TA will use his own implementation(s) for grading purposes. For the version of *TweetServiceImpl.java* to be used for grading, you can assume the *tweet* and *reply* methods will correctly return unique UUIDs if not running into validation, access, or IO errors*.*

## Project Setup

The setup of the project can be accessed [*here*](https://drive.google.com/file/d/1YuJ14IM-TayB8mG0aI-1c6_x9UxQjUlH/view?usp=sharing), 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 ***five*** java files, *ValidationAspect.java, RetryAspect.java, AccessControl.java, StatsAspect.java, and TweetStatsServiceImpl.java.* The code you submit must compile with the given project setup. Your four 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 10, all based on the correctness of the implementation.