# CMPE 275 Section 2, Fall 2017

# Lab 1 - Aspect Oriented Programming

*Last updated: 10/04/2017*

In this lab, you implement the retry and authorization concerns for a blog service through Aspect Oriented Programming (AOP). The blog service allows access to and sharing of blogs. One can share his blog with others, and can share with others the blogs he is shared with too.

The blog service is defined as follows:

```
package edu.sjsu.cmpe275.aop;

public interface BlogService {
        /**
        * Read the blog of another user or oneself.
        * @param userId the ID of the current user
        * @param blogUserId the ID of user, whose blog is being requested
        * @return the blog for blogUserId
        */
        Blog readBlog(String userId, String blogUserId)  throws AccessDeniedExeption,
NetworkException, IllegalArgumentException;

        /**
        * Comment on another user's blog with a message.
        * @param userId the ID of the current user
        * @param blogUserId the ID of user, whose blog is being commented
        */
        void commentOnBlog(String userId, String blogUserId, String message)  throws
AccessDeniedExeption, NetworkException, IllegalArgumentException;

        /**
        * Share a blog with another user. The blog may or may not belong to the current user.
        * @param userId the ID of the current user
        * @param blogUserId the ID of the user, whose blog is being shared
        * @param targetUserId the ID of the user to share the blog with
        */
        void shareBlog(String userId, String blogUserId, String targetUserId)  throws
AccessDeniedExeption, NetworkException, IllegalArgumentException;

        /**
        * Unshare the current user's own blog with another user.
        * @param userId
```

```
     * @param targetUserId
     */
     void unshareBlog(String userId, String targetUserId)  throws AccessDeniedExeption,
NetworkException,IllegalArgumentException;
}
```

The actual implementation of the service, the "official" version of *BlogServiceImpl.java*, however, does not provide enforce access control; i.e., the *shareBlog* method does not check whether the current user is shared with the blog in the first place, and *readBlog* does not to check that either. Part of your task is to use AOP to enforce the following authorization policies.

1. Once can share his blog with anybody.
2. One can only read blogs that are shared with him, or his own blog. In any other case, an *AccessDeniedExeption* is thrown.
3. If Alice shares her blog with Bob, Bob can further share Alice's blog with Carl. If Alice attempts to share Bob's blog with Carl while Bob's blog is not shared with Alice in the first place, Alice gets an *AccessDeniedExeption.*
4. One can only unshare his own blog. When unsharing a blog with Bob that the blog is not shared by any means with Bob in the first place, the operation throws an *AccessDeniedExeption*.
5. Both sharing and unsharing of Alice's blog with Alice have no effect; i.e. Alice always has access to her own blog, and can share and unshare with herself without encountering any exception, even these operations do not take any effect.

BlogService has the following validation rules, which the "official" version of *BlogServiceImpl.java* fails to implement, and you are expected to enforce it with an aspect.

6. For all the methods, every parameters for user ID must of a string of at least 3 and maximum 16 unicode characters, or an *IllegalArgumentException* is thrown.
7. For any blog that is shared with Alice, she can comment on it with a message that is up to 100 unicode characters. If the message is longer than 100, or null, or empty, an *IllegalArgumentException* is thrown.

Please note that our access control here assumes that authentication is already taken care of elsewhere, i.e., it's outside the scope of the project to make sure only Alice can call *readBlog* with *userId* as "Alice".

All the methods in BlogService can also run into network failures, in which case, an *NetworkException* will be thrown, and the method takes no effect at all. Actually, since network failure happens relatively frequently, you are asked to add the feature to automatically retry for up to two times for a network failure (indicated by an NetworkException). Please note the two retries are in addition to the original failed invocation; if you still encounter NetworkException on your second (i.e., final) retry, you should *throw out* this NetworkException so that the caller knows its occurrence.

Your implementation of the above concerns need to be done in the three files: *AuthorizationAspect.java, RetryAspect.java and. ValidationAspect.java.* For example, *RetryAspect.java should look like the following:*

```
package edu.sjsu.cmpe275.aop.aspect;
import org.aspectj.lang.annotation.Aspect;  // if needed
import org.aspectj.lang.annotation.Before;  // if needed

@Aspect
public class RetryAspect {
    ...
}
```

You do not need to worry about multi-threading; i.e., you can assume invocations on the sharing service will be linear and never occur concurrently.

For your testing purpose, you need to provide your own implementation of *BlogServiceImpl.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

A sample project with build file with dependencies, application context, and Java files is now available for download here.

## Submission

Please submit through Canvas, only the **two** java files, *RetryAspect.java, ValidationAspect.java and AuthorizationAspect.java.* The code you submit must compile with the given sample project setup (to be provided). Your two 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 is worth 6 points and will be graded based on the correctness if your implementation.