CMPE 275 Section 2
Lab 1 - Aspect Oriented Programming

Last updated: 10/01/2015

In this lab, you build a proof-of-concept secret storing and sharing service. One can create/read secretes, and share/unshare them with other users as well. The service is a proof of concept in that we do not intend to turn it into a fully featured system; instead, we leave most of the service implementation as either a dummy or a placeholder with an empty method, as we only want to concentrate on the access control and logging aspects with AOP. Please note this is an individual assignment.

```
package edu.sjsu.cmpe275.lab1;
import java.util.UUID;

public interface SecretService {
	/**
	* Store a secrete in the service. A new “secret” record is already created, identified by randomly
generated UUID, with the current user as the owner of the secret.
	* @param userId the ID of the current user
	* @param secret the secret to be stored. No duplication or null check is performed.
	* @return always return a new UUID for the given secret
	*/
	UUID storeSecret(String userId, Secret secret);

	/**
	* Read a secret by ID
	* @param userId the ID of the current user
	* @param secretId the ID of the secret being requested
	* @return the requested secret
	*/
	Secret readSecret(String userId, UUID secretId);

	/**
	* Share a secret with another user. The secret may not have been created by the current user.
	* @param userId the ID of the current user
	* @param secretId the ID of the secret being shared
	* @param targetUserId the ID of the user to share the secret with
	*/
	void shareSecret(String userId, UUID secretId, String targetUserId);

	/**
	* Unshare the current user's own secret with another user.
```

```
        * @param userId the ID of the current user
        * @param secretId the ID of the secret being unshared
        * @param targetUserId the ID of the user to unshare the secret with
        */
        void unshareSecret(String userId, UUID secretId, String targetUserId);
}
```

Suppose the existing implementation of the service forgot to enforce access control; i.e., the *shareSecret* method does not check whether the current user owns or is shared with the secret in the first place, and *readSecret* does not check that either. Your task is to use AOP to enforce the following access control, add logging, implement the *storeSecret* method and any other method that you deem necessary to satisfy the following requirements:

1. One can share the secrets he owns with anybody.
2. One can only read secrets that are shared with him, or his own secrets. In any other case, an *UnauthorizedException* is thrown.
3. If Alice shares her secret with Bob, Bob can further share ir with Carl.
4. One can only unshare his own secret. When unsharing a secret with someone that the secret is not shared by any means, the operation is silently ignored. When one attempts to unshare a secret he neither owns or is shared with, an *UnauthorizedException* is thrown. When attempts to unshare a secret he is shared with but does not own, the request is silently ignored.
5. Both sharing and unsharing of Alice's secret with Alice herself have no effect; i.e., Alice always has access to her own secret.
6. Log a message before each invocation of the *read*, *share*, and *unshare* methods and after the *store* method, with messages exactly as the following, one message per line, except you need to replace the user IDs with the right ones. All logs go to the console through *System.out.*
    a. Alice creates a secrete with ID x
    b. Alice reads the secret of ID x
    c. Alice shares the secret of ID x with Carl
    d. Alice unshares the secret of ID x with Carl

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 *readSecret* with *userId* as "Alice".

You have the option to use whatever storage mechanism you prefer, including maintaining all states in memory. For simplicity, you can also assume there service will be invoked linearly, and you do not need to worry about concurrency issues.

To help measure the correctness of your service, you need to create the following ten JUnit test cases.

A. testA: Bob cannot read Alice's secret, which has not been shared with Bob
B. testB: Alice shares a secret with Bob, and Bob can read it.

C. testC: Alice shares a secret with Bob, and Bob shares Alice's it with Carl, and Carl can read this secret.
D. testD: Alice shares her secret with Bob; Bob shares Carl's secret with Alice and encounters *UnauthorizedException*.
E. testE: Alice shares a secret with Bob, Bob shares it with Carl, Alice unshares it with Carl, and Carl cannot read this secret anymore.
F. testF: Alice shares a secret with Bob and Carl; Carl shares it with Bob, then Alice unshares it with Bob; Bob cannot read this secret anymore.
G. testG: Alice shares a secret with Bob; Bob shares it with Carl, and then unshares it with Carl. Carl can still read this secret.
H. testH: Alice shares a secret with Bob; Carl unshares it with Bob, and encounters *UnauthorizedException*.
I. testI: Alice shares a secret with Bob; Bob shares it with Carl; Alice unshares it with Bob; Bob shares it with Carl with again, and encounters *UnauthorizedException*.
J. testJ: Alice stores the same secrete object twice, and get two different UUIDs.

Please add proper Java Documents, and include a README.txt file with brief instructions on how to run your app.

Submission

Please submit through Canvas. Do not include jars or compiled class files. Please note this is a group assignment, even though you can be on your own group. The seven JUnit tests **must** be part of the submission, along with the a file OUTPUT.txt, which includes the **transcript** of the logging messages from running your JUnit tests from testA to testG. At the end of OUTPUT.txt, please also give the results of the each test, e.g.,
testA: pass
testB: fail
...

Due date

Pleaser refer to Canvas.

Grading:

This lab has a total point of 5, with 4 points for correctness, including using AOP to implement the access control control logic and logging. Code structure and documentation are worth 1 point. Note: you can choose to use either Spring AOP or Guice AOP.

Appendix

Related classes you might find helpful

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

```
import org.junit.Before;
import org.junit.Test;

public class SecretServiceTest {

    SecretService secretService;

    @Before
    public void setUp() throws Exception {
             //...
    }

    @Test(expected = UnauthorizedException.class)
    public void testA() {
              System.out.println("testA");
             UUID aliceSecret = secreteService.storeSecrete(“Alice”, new Secret());
              secretService.readSecret("Bob", aliceSecrte);
    }

    @Test
    public void testB(){
              System.out.println("testB");
             UUID aliceSecret = secreteService.storeSecrete(“Alice”, new Secret());
              secretService.shareSecret("Alice", aliceSecret, "Bob");
              secretService.readSecret("Bob",aliceSecret);
    }

    //...
}
```

======================================

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

public class Secret {

    public Secret() {
              // TODO Auto-generated constructor stub
    }

    // Add whatever you like
}
```

==

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

```
public class UnauthorizedException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public UnauthorizedException() {
            // TODO Auto-generated constructor stub
    }

    public UnauthorizedException(String arg0) {
            super(arg0);
            // TODO Auto-generated constructor stub
    }
}
```