COMPSCI 335 Distributed Objects, Services and Programming
Week 3 Lab Work

Ian Warren

August 1, 2014

The aim of this week's lab work is for you to show that you can work with AOP *declaratively* and that you can solve problems involving more advanced AOP concepts and techniques.

Suggested tasks

Study the lecture materials on aspect-oriented programming and the associated source code. The Knights application demonstrates a number of common AOP situations using both the XML `aop` namespace and annotation-based approaches. Experiment with the supplied code to develop and test your understanding of the material.

This week's tasks primarily concern material presented in the second AOP lecture. They also require reading beyond lecture notes with respect to AOP details (for example, working with custom annotations in task 3) and concurrency control. For the former, you can use the Spring reference manual, in addition to other Spring resources that you find useful. You don't need any more knowledge regarding concurrency principles, other than knowing what a read-write lock is, and you'll find Java's `java.util.concurrent` package helpful.

You should complete the tasks using declarative techniques; unless indicated use either the XML `aop` namespace or annotations, whichever your prefer. Note how use of the declarative approaches simplify implementation by focusing on *what*'s required rather than *how* to set up and configure aspect code.

Task one: Develop a simple dynamic proxy interceptor

Define a couple of simple interfaces and a class that implements both of them. Working with Java's support for dynamic proxies, arrange for an object of your class to be proxied with some sort of useful interception logic. For example, this could be logic to log method invocations using using SLF4J's `Logger` interface, track the call's execution time, protect the object against concurrent access, or something else.

Note how Java's dynamic proxy support serves as the enabler for Spring's AOP approach using run-time proxies. Spring AOP extends the basic dynamic proxy with support for defining advice and pointcuts, along with the declarative techniques to express AOP requirements.

Task two: Declaratively specify AOP solutions

Modify your programmatic solutions to last week's authenticator and thread-safety problems (tasks 2 and 3 respectively from last week). Use the `aop` namespace to configure one of the applications, and the annotation-based approach for the other.

Task three: Develop a more efficient locking aspect

Use of exclusive locks is a simple solution to help ensure the integrity of shared objects in a multithreaded environment. However, being exclusive, such locks allow only a single thread to operate on a shared object at any one time. This can be inefficient, particularly if many of the threads do not change the state of a shared object. A Read-Write lock is one that allows many reader threads to concurrently operate on a shared object; for writer threads a Read-Write lock behaves as a conventional exclusive lock, allowing only a single writer thread to operate on a shared object.

Devise an AOP-based solution that allows an arbitrary class of object to be advised with logic that uses a Read-Write lock to protect the object from concurrent access. Any method that is prefixed with `get` should be be assumed not to change the state of the object; any method prefixed with `set` should be considered to change the object's state.

Class `com.aucklanduni.spring.labs.rwlock.ReadWriteLockMain` is supplied to help you test your solution. This program launches several reader and writer threads that operate on a couple of instances of `com.aucklanduni.spring.labs.rwlock.protect.ThreadMonitor`. The `ThreadMonitor` implementation tracks the number of concurrent threads that access it, and determines whether it has been protected correctly according to Read-Write lock semantics. A `ThreadMonitor` implementation will therefore serve as a target object, allowing you to test your AOP solution. Also supplied is a XML document that initialises a Spring container with the two `ThreadMonitor` beans.

Leveraging the JDK, interface `java.util.concurrent.locks.ReadWriteLock` represents a Read-Write lock, and `java.util.concurrent.locks.ReentrantReadWriteLock` is an implementation that will suffice for this task.

Task four: Utilise custom annotations to define pointcuts

Custom annotations offer a convenient means for defining pointcuts. You can define your own custom annotations and use them to represent pointcuts. Advice logic can then be applied to all methods or types with specific annotations.

For this task, you should further develop task 3's solution and introduce two custom annotations: `@Reader` and `@Writer`. These annotations should be used to decorate methods that can be concurrently executed by many reader threads or by a single write thread respectively. For example, `DefaultThreadMonitor`'s class definition would be amended as follows.

```
1  public class DefaultThreadMonitor implements ThreadMonitor {
2
3    @Writer
4    public void set() {
5      // Implementation omitted
6    }
7
8    @Reader
9    public int get() {
10     // Implementation omitted
11   }
12
13   @Reader
14   public void report() {
15     // Implementation omitted
16   }
17 }
```

The use of annotations in this case is an alternative to using `get` and `set` naming conventions as the basis for pointcut definition.

Defining pointcuts using custom annotations is treated in chapter 9 of the Spring manual. There are lots of Web-based references covering annotations, such as the JavaWorld article listed at the end of this document, that discuss custom annotation definition.

Task five: Develop a Modification tracker introduction

Consider *caching* as a separate concern. Caching involves storing a local copy of some resource so that it can be more quickly accessed. For example, rather than querying a database for some value every time it's needed, the result could be cached in an object – and the object held in memory. One of the problems introduced by caching, however, is that of consistency, where any changes to the state of a cached object should eventually be written back to the database.

For this task, you need to *introduce* functionality that can be used as part of a caching service. Specifically, it should be possible to take an arbitrary object and, at run-time, add the capability for the object's last modification time to be tracked. The idea is that the caching service would use this capability, during shutting down of the Spring container for example, to visit each object; if the object's been modified the caching service will write its current state back to the database.

Assume that the following interface needs to be introduced.

```
1  import org.joda.time.DateTime;
2
3  public interface ModificationTracker {
4    /**
5     * Called when an object has been updated. A target object's
6     * proxy should be responsible for making this call. A
7     * ModificationTracker implementation should store the current time
8     * as the latest modification time.
9     */
10   void objectUpdated();
11
12   /**
13    * Called to query the modification time held by the
14    * ModificationTracker implementation.
15    */
16   DateTime getLastModificationTime();
17 }
```

To determine when an object is being modified, use an appropriate pointcut – such as an expression that matches any method prefixed `set`, or a custom annotation to decorate any mutator method.

`org.joda.time.DateTime` (`http://www.joda.org/joda-time/`) is an alternative to the standard Java date and time classes.

Resources

- `http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/` This is the Spring reference manual and the definitive source of information for Spring. It can be supplemented with tutorials as needed.
- `http://docs.oracle.com/javase/7/docs/api/` The Javadoc documentation for Java. For this week's lab work, you will need to consult the documentation for `java.util.concurrent.`

`locks.ReadWriteLock` and `java.util.concurrent.locks.ReentrantReadWriteLock`.

- `http://www.javaworld.com/article/2074825/core-java/taming-tiger--part-3.html` JavaWorld is a useful source of information for Java development. This article includes how to define custom annotations.
- `http://mvnrepository.com/` The Maven repository website. In general, when looking for dependency information (for example `joda-time`), you can use this site to search for projects and obtain the required `<dependency>` tag content.
- `http://stackoverflow.com` As ever, this site is often useful when you have specific questions.

Supplementary tutorials can be found by using Web search engines.