Aspect-Oriented Programming

Overview

In this lab, you'll practise using simple aspect-oriented programming techniques in Spring Boot.

IntelliJ projects

Starter project: **`student\student-aop`**

Solution project: **`solutions\solution-aop`**

Roadmap

There are 5 exercises in this lab, of which the last exercise is “if time permits”. Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Adding AOP support
2. Defining some simple beans
3. Defining pointcuts
4. Defining an aspect class
5. (If Time Permits) Additional suggestions

Exercise 1: Adding AOP support

In the **`student-aop`** project, open the pom file and add the dependency for *Spring Boot Starter for AOP* (see the demo application for details, if you need a reminder).

Then locate the **`Application`** class and annotate it with **`@EnableAspectJAutoProxy`**. This enables Spring Boot to generate proxy object to achieve AOP proxy dynamic weaving.

Exercise 2: Defining some simple beans

Define an interface with some simple business methods (e.g. named **`doThis()`**, **`doThat()`** etc., plus some other methods with different names). Then define a component class that implements the methods in a simple way.

In the application class, add some code to get the bean and call its methods, to make sure it all works so far.

Exercise 3: Defining pointcuts

Define a new class to represent pointcuts. In this class, define two pointcuts:

- A pointcut that represents all methods in your package that start with "do".
- A pointcut that represents all methods in your package that don't start with "do".

Exercise 4: Defining an aspect class

Define a component class and annotate it with **`@Aspect`**. Inside the class, define the following advice methods:

- Define "before" advice that will apply before all the "do" methods.
- Define "before" advice that will apply before all the "non-do" methods.

Run the application, and ensure that Spring weaves-in your advice code.

Exercise 5 (If Time Permits): Additional suggestions

- In your advice methods, declare a **`JoinPoint`** parameter to access "context" information about the original target, the method name, etc.
- Refine your pointcut expressions so that they pertain only to a specific class and/or package, and verify that the advice code only executes when such methods are called.
- Experiment with pointcut expressions that relate to annotations (either on the target object or on the method return type).