Aspect-Oriented Programming

1. Getting started with AOP
2. Simple AOP example
3. More AOP syntax

Demo project: **`demo-aop`**

1. Getting Started with AOP

- What is AOP?
- How to utilize AOP
- AOP technologies
- Essential AOP concepts

What is AOP?

- AOP allows you to modularize cross-cutting concerns, such as:
  - Logging and tracing
  - Transaction management
  - Security
  - Error handling

- AOP allows you to define cross-cutting logic in one place...
  - Avoids code tangling
    - E.g. mixing security checks in with your real application logic
  - Avoids code scattering
    - E.g. security checks in every method

How to Utilize AOP

1. Implement your mainstream application logic
   - Focus on business rules, algorithms, etc.
2. Write aspects to implement cross-cutting concerns
   - Spring provides many predefined aspects
3. Weave the aspects into your application
   - Add the cross-cutting behaviours to the appropriate places

AOP Technologies

- AspectJ
  - The original AOP technology (first appeared in 1995)
  - Full-blown AOP language
  - Uses Java byte-code modification for aspect weaving

- Spring AOP (aka @AspectJ)
  - Annotation-based AOP framework for Spring
  - This is what you'll probably use when you use AOP in Spring apps
  - Uses subset of AspectJ expression syntax
  - Can use dynamic proxies for aspect weaving

Essential AOP Concepts

- Join point
  - A point in the execution of an application, e.g. a method call
- Pointcut
  - An (AspectJ) expression that selects one or more join points
- Advice
  - Code to execute at a join point that's selected by a pointcut
- Aspect
  - A class that encapsulates pointcuts and advice

2. Simple AOP Example

- Configuring Maven dependencies
- Defining a simple aspect
- Using named pointcuts
- Enabling @AspectJ support
- Accessing context information

Configuring Maven Dependencies

- Add the "Spring Boot Starter for AOP" dependency to your POM file:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

pom.xml

Defining a Simple Aspect (1 of 2)

- Problem statement:
  - Log a message every time a property is about to change
  - Here's a sample class with some properties that might change...

```
public class MySampleClass implements MySampleInterface {
    public void setPropertyA(String value) {…}
    public void setPropertyB(int value)    {…}
    …
}
```

MySampleClass.java

- How can we define an aspect that runs before these methods are executed?
  - See following slide…

Defining a Simple Aspect (2 of 2)

- Here's an aspect that runs before the "setter" methods in `MySampleClass` are executed

```
@Aspect
@Component
public class PropertySetTracker {

    @Before("execution(void MySampleClass.set*(*))")
    public void logPropertyChange() {
        System.out.println("A property is about to be set…");
    }
}
```

PropertySetTracker.java

- This is *before* advice
  - You can also define *after* advice and *around* advice – see later

Using Named Pointcuts

- It's generally good practice to define named pointcuts, and apply them by name in join point(s)...

```
@Aspect
@Component
public class PropertySetTrackerV2 {

    @Pointcut("execution(void MySampleClass.set*(*))")
    public void setterMethodExecuted() {}

    @Before("setterMethodExecuted()")
    public void logPropertyChange() {
        System.out.println("A property is about to be set…");
    }
}
```

PropertySetTrackerV2.java

Enabling @AspectJ Support

- You must enable @AspectJ support in your application

```
@SpringBootApplication
@EnableAspectJAutoProxy
public class Application {
    …
}
```

Application.java

Accessing Context Info (1 of 2)

- Advice methods can access context info
  - Declare `JoinPoint` param in advice method, and call getters...

JoinPoint getter	Description
`getThis()`	The currently executing object that has been intercepted (i.e. Spring dynamic proxy)
`getTarget()`	The target of the execution (typically your object)
`getSignature()`	The signature of the join point
`getArgs()`	The method arguments passed to the join point

Accessing Context Info (2 of 2)

- Example:

```
@Aspect
@Component
public class AnotherPropertySetTracker {

    @Pointcut("execution(void MySampleClass.set*(*))")
    public void setterMethodExecuted() {}

    @Before("setterMethodExecuted()")
    public void logPropertyChange(JoinPoint jp) {

      System.out.println("Target: "    + jp.getTarget() +
                         " Method: "    + jp.getSignature().getName() +
                         " value: "     + jp.getArgs()[0]);
    }
}
```

AnotherPropertySetTracker.java

3. More AOP Syntax

- Separating pointcuts from aspects
- Formal syntax for execution pointcuts
- Combining pointcuts

Separating Pointcuts from Aspects

```
public final class MyPointcuts {

    @Pointcut("execution(* MyBean.*(*))")
    public void callToMyBean() {}

    @Pointcut("within(MyBean)")
    public void withinMyBean() {}
}
```

This class just defines pointcuts

MyPointcuts.java

```
@Aspect @Component
public class MyAspect {

    @Before("MyPointcuts.callToMyBean()")
    public void logCallToBean() {
        System.out.println("***Call to bean");
    }

    @Before("MyPointcuts.withinMyBean()")
    public void logWithinBean() {
        System.out.println("***Within bean");
    }
}
```

This aspect class uses the pointcuts

MyAspect.java

Formal Syntax for Execution Pointcuts (1 of 2)

- Formal syntax:

  [Modifiers] ReturnType [ClassType] MethodName ([Args]) [throws ExcType]

- Wildcards:

  `*` Any return type, method name fragment, class name fragment, param

  `..` Zero or more sub-packages, zero or more additional parameters

Formal Syntax for Execution Pointcuts (2 of 2)

- Discuss 😀

```
1) execution(void com.osl.MyBean.* (int))

2) execution(* send*(int))

3) execution(void send(*))

4) execution(void send(Date, ..))

5) execution(* com.osl..*.*(..))

6) execution(public * *(..))

7) execution(@org.spingframework.transaction.annotation.Transactional void *(..))

8) execution((@myannotationspackage.MyAnnotation *) *(..))
```

Combining Pointcut Expressions

- You can combine pointcut expressions using these operators:

&&	And
||	Or
!	Not

- Example:

```
public final class MyPointcuts {

    @Pointcut(" execution(* com.osl..*.*(..))     &&
               !execution(* com.osl..*.set*(..)) &&
               !execution(* com.osl..*.get*(..))" )
    public void callToBizMethod() {}
    …
}
```

Summary

- Getting started with AOP
- Simple AOP example
- More AOP syntax