

# 4장 스프링 AOP 기능

강사 김영석

A top-down view of a wooden desk. On the desk, there is a silver laptop with a black keyboard, a pair of black-rimmed glasses, a white coffee cup with a yellow handle, and a small green succulent in a dark pot. The word "CONTENT" is written in large, white, sans-serif capital letters over the left side of the image.

# CONTENT

- 1 관점 지향 프로그래밍 등장
- 2 스프링에서 AOP 기능 사용하기
- 3 <AOP> 태그를 이용한 AOP 기능
- 4 애너테이션을 이용한 AOP 기능

# 1. 관점 지향 프로그래밍 등장

## ✓ 관점 지향 프로그래밍(AOP, Aspect-Oriented Programming)

### ■ 객체 지향 프로그래밍의 단점을 해소하기 위해서 등장했다.

- 모든 변수를 선언할 때 new 를 통해서 객체를 선언하고 생성했다.
- 객체를 재사용 한다는 측면에서는 효율적이었지만, 공통된 부가 기능에 대해서는 코드가 중복되고 반복된다는 단점이 있다.

```
class Test1 {  
    public void aaa() {  
        // 회원 여부 확인  
        // aaa 메서드 기능  
        // 작업 완료 확인  
    }  
}
```

```
class Test2 {  
    public void bbb() {  
        // 회원 여부 확인  
        // bbb 메서드 기능  
        // 작업 완료 확인  
    }  
}
```

```
class Test3 {  
    public void ccc() {  
        // 회원 여부 확인  
        // bbb 메서드 기능  
        // 작업 완료 확인  
    }  
}
```

- 각기 다른 클래스의 동일한 부가 기능이 들어가 있는 예 이다.

- 회원 여부 확인과 작업 완료 확인이 반복적으로 들어가 있는 것을 볼 수 있다. 중복된 코드는 소스코드가 많아지는 것은 물론 유지 보수를 할 때 한 가지 기능을 수정할 때 여러 군데를 다 같이 수정 해야 한다는 것이다.
- AOP는 이런 흩어진 공통 관심사를 모아서 모듈화 시키는 것이라 할 수 있다.
- 중복되는 코드를 흩어진 관심사라 하고 관심사 관점으로 생각하는 것이 바로 관점 지향 프로그래밍이다.



## 2. 스프링에서 AOP 기능 사용하기

✓ 여러 가지 AOP 관련 용어

용어	설명
aspect	구현하고자 하는 보조 기능을 의미한다.
advice	aspect의 실제 구현체(클래스)를 의미한다. 메서드 호출을 기준으로 여러 지점에서 실행 된다.
joinpoint	advice를 어디에 적용해야 하는 지점을 의미한다. 스프링은 method 결합점만 제공한다.
pointcut	advice가 구체적으로 실행될 대상을 지정한다. 패키지 이름/클래스 이름/메서드 이름을 정규식으로 지정하여 사용한다.
target	advice가 적용되는 클래스를 의미한다.
weaving	advice를 주기능에 적용하는 것을 의미한다.





## ✓ 스프링 API를 이용한 AOP 기능 구현 과정

- ① 타겟(target) 클래스를 지정한다.
- ② 어드바이스(Advice) 클래스를 지정한다.
- ③ 설정 파일에서 포인트컷(Pointcut)을 설정한다.
- ④ 설정 파일에서 어드바이스와 포인트컷을 결합하는 어드바이저를 설정한다.
- ⑤ 설정 파일에서 스프링의 ProxyFactoryBean 클래스를 이용해 타겟에 어드바이스를 설정한다.
- ⑥ `getBean()` 메서드로 빈 객체에 접근해 사용한다.

인터페이스	추상 메서드	설명
MethodBeforeAdvice	before(Method method, Object[] args, Object target) throws Throwable	해당 메서드가 실행되기 전 실행
<ul style="list-style-type: none"> <li>- Method method : 대상 객체에서 실행된 메서드를 나타내는 메서드 객체</li> <li>- Object[] args : 메서드 인자 목록</li> <li>- Object target : 대상 객체</li> </ul>		
AfterReturningAdvice	void afterReturning(Object returnValue, Method method, Object[] args, Object target) throws Throwable	해당 메서드가 실행된 후 실행
<ul style="list-style-type: none"> <li>- Object returnValue : 대상 객체의 메서드가 반환하는 값</li> <li>- Method method : 대상 객체에서 실행된 메서드를 나타내는 메서드 객체</li> <li>- Object[] args : 메서드 인자 목록</li> <li>- Object target : 대상 객체</li> </ul>		
ThrowsAdvice	void afterThrowing(Method method, Object[] args, Object target, Exception ex)	해당 메서드에서 예외 발생 시 실행
<ul style="list-style-type: none"> <li>- Method method : 대상 객체에서 실행된 메서드를 나타내는 메서드 객체</li> <li>- Object[] args : 메서드 인자 목록</li> <li>- Object target : 대상 객체</li> <li>- Exception ex : 발생한 예외 타입</li> </ul>		

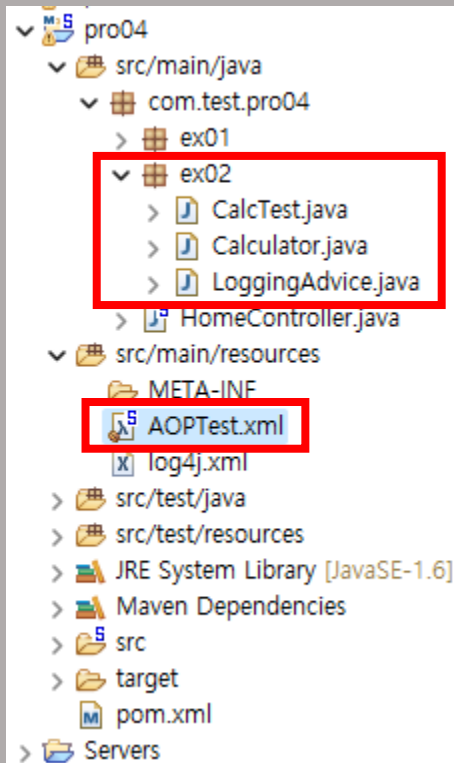


인터페이스	추상 메서드	설명
MethodInterceptor	Object invoke(MethodInvocation invocation) throws Throwable	해당 메서드의 실행 전/후와 예외 발생 시 실행
- MethodInvocation invocation : 대상 객체의 모든 정보를 담고 있는 객체 (호출된 메서드, 인자 등)		



## ✓ 스프링 API를 이용한 AOP 기능 실습


```
pro04/pom.xml
34      </dependency>
35
36      <!-- spring-aop -->
37      <dependency>
38          <groupId>org.springframework</groupId>
39          <artifactId>spring-aop</artifactId>
40          <version>${org.springframework-version}</version>
41      </dependency>
42      <dependency>
43          <groupId>cglib</groupId>
44          <artifactId>cglib</artifactId>
45          <version>2.2</version>
46          <type>jar</type>
47          <scope>compile</scope>
48      </dependency>
49
50      <dependency>
51          <groupId>org.aspectj</groupId>
52          <artifactId>aspectjweaver</artifactId>
53          <version>${org.aspectj-version}</version>
54      </dependency>
55
56      <!-- AspectJ -->
57      <dependency>
58          <groupId>org.aspectj</groupId>
59          <artifactId>aspectjrt</artifactId>
60          <version>${org.aspectj-version}</version>
61      </dependency>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">


    <bean id="calcTarget" class="com.test.pro04.ex02.Calculator"></bean>
    <bean id="logAdvice" class="com.test.pro04.ex02.LoggingAdvice"></bean>

    <bean id="proxyCal" class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="target" ref="calcTarget"></property>
        <property name="interceptorNames">
            <list>
                <value>logAdvice</value>
            </list>
        </property>
    </bean>
</beans>
```



```
package com.test.pro04.ex02;

public class Calculator {
    public void add(int x, int y) {
        int result = x + y;
        System.out.println("결과 : " + result);
    }
    public void subtract(int x, int y) {
        int result = x - y;
        System.out.println("결과 : " + result);
    }
    public void multiply(int x, int y) {
        int result = x * y;
        System.out.println("결과 : " + result);
    }
    public void divide(int x, int y) {
        int result = x / y;
        System.out.println("결과 : " + result);
    }
}
```



```
package com.test.pro04.ex02;


import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class LoggingAdvice implements MethodInterceptor{
    public Object invoke(MethodInvocation invocation) throws Throwable {
        System.out.println("[메서드 호출 전 : LogginAdvice]");
        System.out.println(invocation.getMethod() + "메서드 호출 전");

        Object object = invocation.proceed();

        System.out.println("[메서드 호출 후 : LogginAdvice]");
        System.out.println(invocation.getMethod() + "메서드 호출 후");
        return object;
    }
}
```






```
package com.test.pro04.ex02;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class CalcTest {
    public static void main(String[] args) {
        AbstractApplicationContext factory =
            new GenericXmlApplicationContext("AOPTest.xml");

        Calculator cal = (Calculator)factory.getBean("proxyCal");
        cal.add(100, 20);
        System.out.println();
        cal.subtract(100, 20);
        System.out.println();
        cal.multiply(100, 20);
        System.out.println();
        cal.divide(100, 20);
    }
}
```





```
[메서드 호출 전 : LogginAdvice]
public void com.test.pro04.ex02.Calculator.divide(int,int) 메서드 호출 전
결과 : 5
[메서드 호출 후 : LogginAdvice]
public void com.test.pro04.ex02.Calculator.divide(int,int) 메서드 호출 후
```

# Quiz

- ✓ com.test.pro03.quiz 의 내용을 복사 해서 아래와 같이 구현해 보세요
  - boardAOP.xml 을 만들고 DI 기능과 AOP 둘다 설정해 보세요.
  - BoardAdvice 를 AOP 실행 클래스로 만든다.
  - BoardServiceImpl의 listBoard() 메서드 실행 전 “로그인 확인 메서드” 라고 출력하고 listBoard() 메서드 실행 후 “데이터 확인 메서드” 라고 출력하세요.

### 3. <AOP> 태그를 이용한 AOP 기능

#### ✓ execution 사용 방법

- execution은 Advice를 적용할 메서드를 명시할 때 사용한다.

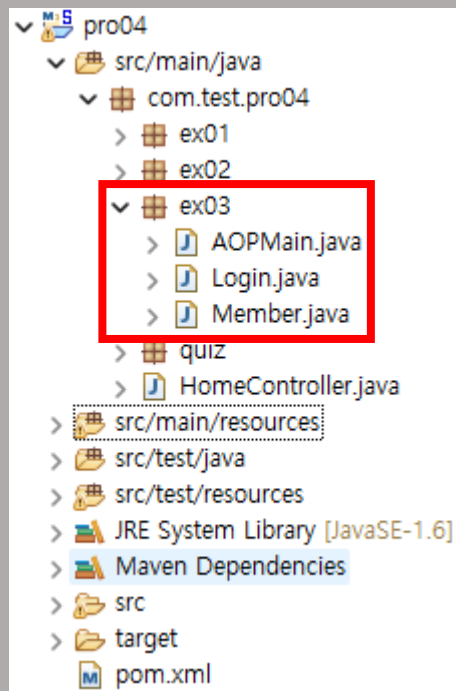
`execution([접근제한자] [리턴타입] [경로].[메서드](파라미터))`

- ① 접근제한자 : public, private, protected를 사용하며 생략 가능하다.
- ② 리턴타입 : 메서드의 리턴타입을 의미한다.
- ③ 경로 : 해당하는 모든 패키지의 이름을 지정한다.
- ④ 메서드 : 해당하는 메서드의 이름을 지정한다.
- ⑤ 매개변수 : 메서드의 매개변수의 개수를 표시 할 수 있다.



## ✓ execution 의 예

- `execution(public Integer com.test.aop.*.*(*))`
  - `com.test.aop` 패키지에 속해 있고 매개변수가 1개인 모든 메서드
- `execution(* com.test..*.get*(..))`
  - `com.test` 패키지 및 하위 패키지에 속해 있고 이름이 `get`으로 시작하는 메서드로 매개변수가 0개 이상인 모든 메서드
- `execution(* com.test.aop.*Service.*(..))`
  - `com.test.aop` 패키지 및 하위 패키지에 속해 있고 이름이 `Service`로 끝나는 클래스의 매개변수가 0개 이상인 모든 메서드

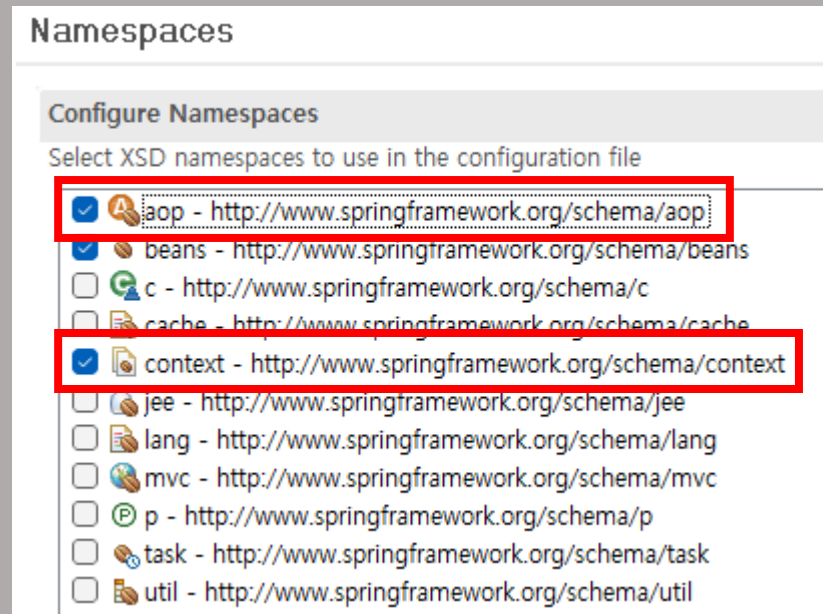
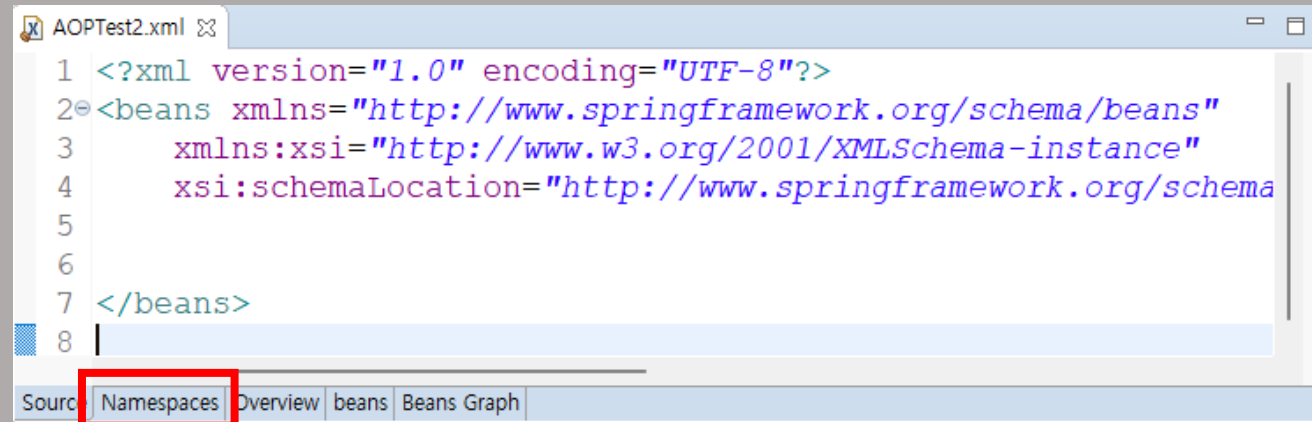
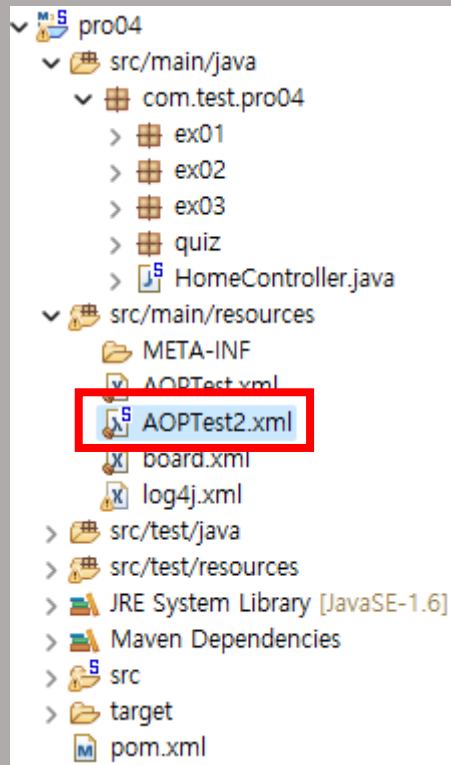


```
Login.java
1 package com.test.pro04.ex03;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class Login {
7     public void loginIn() {
8         System.out.println("로그인 중 입니다.");
9     }
10    public void logOut() {
11        System.out.println("로그 아웃 중입니다.");
12    }
13 }
```



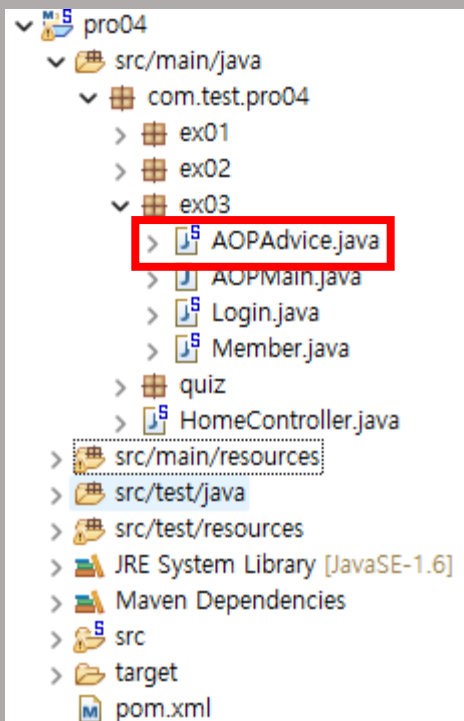
```
Member.java ✕
1 package com.test.pro04.ex03;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class Member {
7     public void memberJoin() {
8         System.out.println("회원 가입 중입니다.");
9     }
10    public void memberDel() {
11        System.out.println("회원 삭제 중입니다.");
12    }
13 }
```

```
AOPMain.java ✕
1 package com.test.pro04.ex03;
2
3 import org.springframework.context.support.AbstractApplicationContext;
4
5
6 public class AOPMain {
7     public static void main(String[] args) {
8         AbstractApplicationContext factory =
9             new GenericXmlApplicationContext("AOPTest2.xml");
10        Login lg = (Login) factory.getBean("login");
11        lg.loginIn();
12        lg.logOut();
13
14        Member mb = (Member) factory.getBean("member");
15        mb.memberJoin();
16        mb.memberDel();
17    }
18 }
```



## ✓ 메서드 실행 전 AOP 실행 하기

```
AOPTest2.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:aop="http://www.springframework.org/schema/aop"
5     xmlns:context="http://www.springframework.org/schema/context"
6     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
7     http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd
8     http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">
9
10     <context:component-scan base-package="com.test.pro04" />
11     <bean id="aopAdvice" class="com.test.pro04.ex03.AOPAdvice"></bean>
12     <aop:config>
13         <aop:aspect ref="aopAdvice">
14             <aop:pointcut expression="execution(public void com.test.pro04.ex03.*.*(..))" id="aopAdvicePointcut"/>
15             <aop:before method="aopMethodBefore" pointcut-ref="aopAdvicePointcut"/>
16         </aop:aspect>
17     </aop:config>
18 </beans>
```



```
AOPAdvice.java
1 package com.test.pro04.ex03;
2
3 public class AOPAdvice {
4     public void aopMethodBefore() {
5         System.out.println("실행 전 AOP 실행 입니다.");
6     }
7 }
```

실행 전 AOP 실행 입니다.  
로그인 중 입니다.  
실행 전 AOP 실행 입니다.  
로그 아웃 중입니다.  
실행 전 AOP 실행 입니다.  
회원 가입 중입니다.  
실행 전 AOP 실행 입니다.  
회원 삭제 중입니다.



## ✓ 메서드 실행 후 AOP 실행 하기

```
AOPTest2.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:aop="http://www.springframework.org/schema/aop"
5       xmlns:context="http://www.springframework.org/schema/context"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/sp
7                           http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-
8                           http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">
9
10    <context:component-scan base-package="com.test.pro04" />
11    <bean id="aopAdvice" class="com.test.pro04.ex03.AOPAdvice"></bean>
12    <aop:config>
13      <aop:aspect ref="aopAdvice">
14        <aop:pointcut expression="execution(public void com.test.pro04.ex03.*.*(..))" id="aopAdvicePointcut"/>
15        <aop:before method="aopMethodBefore" pointcut-ref="aopAdvicePointcut"/>
16        <aop:after method="aopMethodAfter" pointcut-ref="aopAdvicePointcut"/>
17      </aop:aspect>
18    </aop:config>
19 </beans>
```



```
AOPAdvice.java
1 package com.test.pro04.ex03;
2
3 public class AOPAdvice {
4     public void aopMethodBefore() {
5         System.out.println("실행 전 AOP 실행 입니다.");
6     }
7     public void aopMethodAfter() {
8         System.out.println("실행 후 AOP 실행 입니다.");
9     }
10 }
```

실행 전 AOP 실행 입니다.  
로그인 중 입니다.  
실행 후 AOP 실행 입니다.  
실행 전 AOP 실행 입니다.  
로그 아웃 중입니다.  
실행 후 AOP 실행 입니다.  
실행 전 AOP 실행 입니다.  
회원 가입 중입니다.  
실행 후 AOP 실행 입니다.  
실행 전 AOP 실행 입니다.  
회원 삭제 중입니다.  
실행 후 AOP 실행 입니다.

## ✓ 메서드 실행 전후로 AOP 기능 실행하기

```
AOPTest2.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:aop="http://www.springframework.org/schema/aop"
5     xmlns:context="http://www.springframework.org/schema/context"
6     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spr
7         http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-
8         http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">
9
10    <context:component-scan base-package="com.test.pro04" />
11    <bean id="aopAdvice" class="com.test.pro04.ex03.AOPAdvice"></bean>
12    <aop:config>
13        <aop:aspect ref="aopAdvice">
14            <aop:pointcut expression="execution(public void com.test.pro04.ex03.*.*(..))" id="aopAdvicePointcut"/>
15            <aop:before method="aopMethodBefore" pointcut-ref="aopAdvicePointcut"/>
16            <aop:after method="aopMethodAfter" pointcut-ref="aopAdvicePointcut"/>
17            <aop:around method="aopMethodAround" pointcut-ref="aopAdvicePointcut"/>
18        </aop:aspect>
19    </aop:config>
20 </beans>
```

```

1 package com.test.pro04.ex03;
2
3 import org.aspectj.lang.ProceedingJoinPoint;
4
5 public class AOPAdvice {
6     public void aopMethodBefore() {
7         System.out.println("실행 전 AOP 실행 입니다.");
8     }
9     public void aopMethodAfter() {
10        System.out.println("실행 후 AOP 실행 입니다.");
11    }
12    public Object aopMethodAround(ProceedingJoinPoint joinPoint) throws Throwable {
13        Object result = null;
14
15        try {
16            System.out.println("Around 실행 전 AOP 실행 입니다.");
17            result = joinPoint.proceed();
18            System.out.println("Around 실행 후 AOP 실행 입니다.");
19        } catch (Exception e) {
20            // TODO: handle exception
21            e.printStackTrace();
22        }
23        return result;
24    }
25 }

```

실행 전 AOP 실행 입니다.  
 Around 실행 전 AOP 실행 입니다.  
 로그인 중 입니다.  
 실행 후 AOP 실행 입니다.  
 Around 실행 후 AOP 실행 입니다.  
 실행 전 AOP 실행 입니다.  
 Around 실행 전 AOP 실행 입니다.  
 로그 아웃 중입니다.  
 실행 후 AOP 실행 입니다.  
 Around 실행 후 AOP 실행 입니다.  
 실행 전 AOP 실행 입니다.  
 Around 실행 전 AOP 실행 입니다.  
 회원 가입 중입니다.  
 실행 후 AOP 실행 입니다.  
 Around 실행 후 AOP 실행 입니다.  
 실행 전 AOP 실행 입니다.  
 Around 실행 전 AOP 실행 입니다.  
 회원 삭제 중입니다.  
 실행 후 AOP 실행 입니다.  
 Around 실행 후 AOP 실행 입니다.

## ✓ 메서드 정상 종료 후 AOP 실행 하기

```
*AOPTest2.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:aop="http://www.springframework.org/schema/aop"
5     xmlns:context="http://www.springframework.org/schema/context"
6     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
7     http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd
8     http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">
9
10    <context:component-scan base-package="com.test.pro04" />
11    <bean id="aopAdvice" class="com.test.pro04.ex03.AOPAdvice"></bean>
12    <aop:config>
13        <aop:aspect ref="aopAdvice">
14            <aop:pointcut expression="execution(public void com.test.pro04.ex03.*.*(..))" id="aopAdvicePointcut"/>
15            <aop:before method="aopMethodBefore" pointcut-ref="aopAdvicePointcut"/>
16            <aop:after method="aopMethodAfter" pointcut-ref="aopAdvicePointcut"/>
17            <aop:around method="aopMethodAround" pointcut-ref="aopAdvicePointcut"/>
18            <aop:pointcut expression="execution(public void com.test.pro04.ex03.*.*n*(..))" id="aopAdvicePointcut2"/>
19            <aop:around method="aopMethodReturning" pointcut-ref="aopAdvicePointcut2"/>
20        </aop:aspect>
21    </aop:config>
22 </beans>
```



```

AOPAdvice.java
1 package com.test.pro04.ex03;
2
3 import org.aspectj.lang.ProceedingJoinPoint;
4
5 public class AOPAdvice {
6     public void aopMethodBefore() {
7         System.out.println("실행 전 AOP 실행 입니다.");
8     }
9     public void aopMethodAfter() {
10        System.out.println("실행 후 AOP 실행 입니다.");
11    }
12    public Object aopMethodAround(ProceedingJoinPoint joinPoint) throws Throwable {
13        Object result = null;
14
15        try {
16            System.out.println("Around 실행 전 AOP 실행 입니다.");
17            result = joinPoint.proceed();
18            System.out.println("Around 실행 후 AOP 실행 입니다.");
19        } catch (Exception e) {
20            // TODO: handle exception
21            e.printStackTrace();
22        }
23        return result;
24    }
25    public void aopMethodReturning() {
26        System.out.println("JoinPoint 메서드가 정상 실행 한 후");
27    }
28 }

```

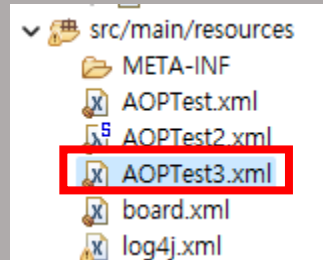
실행 전 AOP 실행 입니다.  
 Around 실행 전 AOP 실행 입니다.  
 JoinPoint 메서드가 정상 실행 한 후  
 Around 실행 후 AOP 실행 입니다.  
 실행 전 AOP 실행 입니다.  
 Around 실행 전 AOP 실행 입니다.  
 로그 아웃 중입니다.  
 실행 후 AOP 실행 입니다.  
 Around 실행 후 AOP 실행 입니다.  
 실행 전 AOP 실행 입니다.  
 Around 실행 전 AOP 실행 입니다.  
 JoinPoint 메서드가 정상 실행 한 후  
 Around 실행 후 AOP 실행 입니다.  
 실행 전 AOP 실행 입니다.  
 Around 실행 전 AOP 실행 입니다.  
 회원 삭제 중입니다.  
 실행 후 AOP 실행 입니다.  
 Around 실행 후 AOP 실행 입니다.



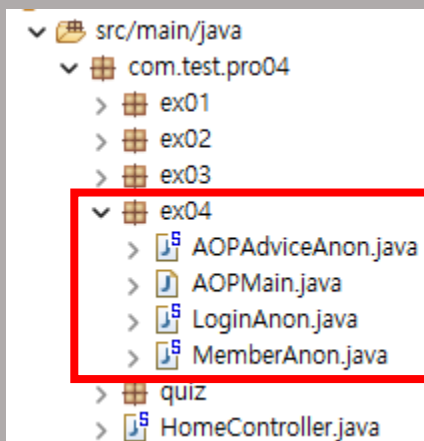
# Quiz

- ✓ quiz01 패키지의 AOP를 xml 로 아래와 같이 변환 해 보세요.
  - Pointcut 과 Advice 를 <aop> 태그를 이용해서 변환 하세요.
  - <aop:before> 로 BoardServiceImpl 의 모든 메서드를 처리 해 보세요.
  - <aop:after> 로 BoardDAOImpl 의 모든 메서드를 처리 해 보세요.
  - 내용은 ‘AOP 실행’ 으로 해 보세요.

# 4. 애너테이션을 이용한 AOP 기능



```
AOPTest3.xml 83
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:aop="http://www.springframework.org/schema/aop"
5       xmlns:context="http://www.springframework.org/schema/context"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.s
7                           http://www.springframework.org/schema/context http://www.springframework
8                           http://www.springframework.org/schema/aop http://www.springframework.org
9
10      <context:component-scan base-package="com.test.pro04.ex04" />
11      <bean id="aopAdviceAnon" class="com.test.pro04.ex04.AOPAdviceAnon"></bean>
12      <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
13 </beans>
```



```
AOPMain.java
1 package com.test.pro04.ex04;
2
3 import org.springframework.context.support.AbstractApplicationContext;
4
5
6 public class AOPMain {
7     public static void main(String[] args) {
8         AbstractApplicationContext factory =
9             new GenericXmlApplicationContext("AOPTest3.xml");
10        LoginAnon lg = (LoginAnon)factory.getBean("loginAnon");
11        lg.loginIn();
12        lg.logOut();
13
14        MemberAnon mb = (MemberAnon)factory.getBean("memberAnon");
15        mb.memberJoin();
16        mb.memberDel();
17    }
18 }
```

```
*AOPAdviceAnon.java
1 package com.test.pro04.ex04;
2
3 import org.aspectj.lang.ProceedingJoinPoint;
10
11 @Aspect
12 @Component
13 public class AOPAdviceAnon {
14     @Before("execution(public void com.test.pro04.ex04.*(..))")
15     public void aopMethodBefore() {
16         System.out.println("실행 전 Anon AOP 실행 입니다.");
17     }
18     @After("execution(public void com.test.pro04.ex04.*(..))")
19     public void aopMethodAfter() {
20         System.out.println("실행 후 Anon AOP 실행 입니다.");
21     }
22     @Around("execution(public void com.test.pro04.ex04.*(..))")
23     public Object aopMethodAround(ProceedingJoinPoint joinPoint) throws Throwable {
24         Object result = null;
25
26         try {
27             System.out.println("Around Anon 실행 전 AOP 실행 입니다.");
28             result = joinPoint.proceed();
29             System.out.println("Around Anon 실행 후 AOP 실행 입니다.");
30         } catch (Exception e) {
31             // TODO: handle exception
32             e.printStackTrace();
33         }
34         return result;
35     }
36     @AfterReturning("execution(public void com.test.pro04.ex04.*.*n*(..))")
37     public void aopMethodReturning() {
38         System.out.println("JoinPoint 메서드가 Anon 정상 실행 한 후");
39     }
40 }
```

# Quiz

✓ quiz01 패키지의 AOP를 애터네이션으로 변환 해 보세요.



A photograph of a server room with rows of server racks on both sides of a central aisle. The racks have glass doors and are filled with server units, some of which have glowing blue indicator lights. The ceiling has several long, rectangular light fixtures. The overall atmosphere is dimly lit, with the primary light sources being the server lights and the ceiling fixtures.

수고하셨습니다.