2020년도 1학기

# 자바실무프로젝트 과목 – 10주차

## [교재 Day02-Class03 어드바이스 동작시점 ] p171~

# 스프링 실습 프로젝트 정리 실습

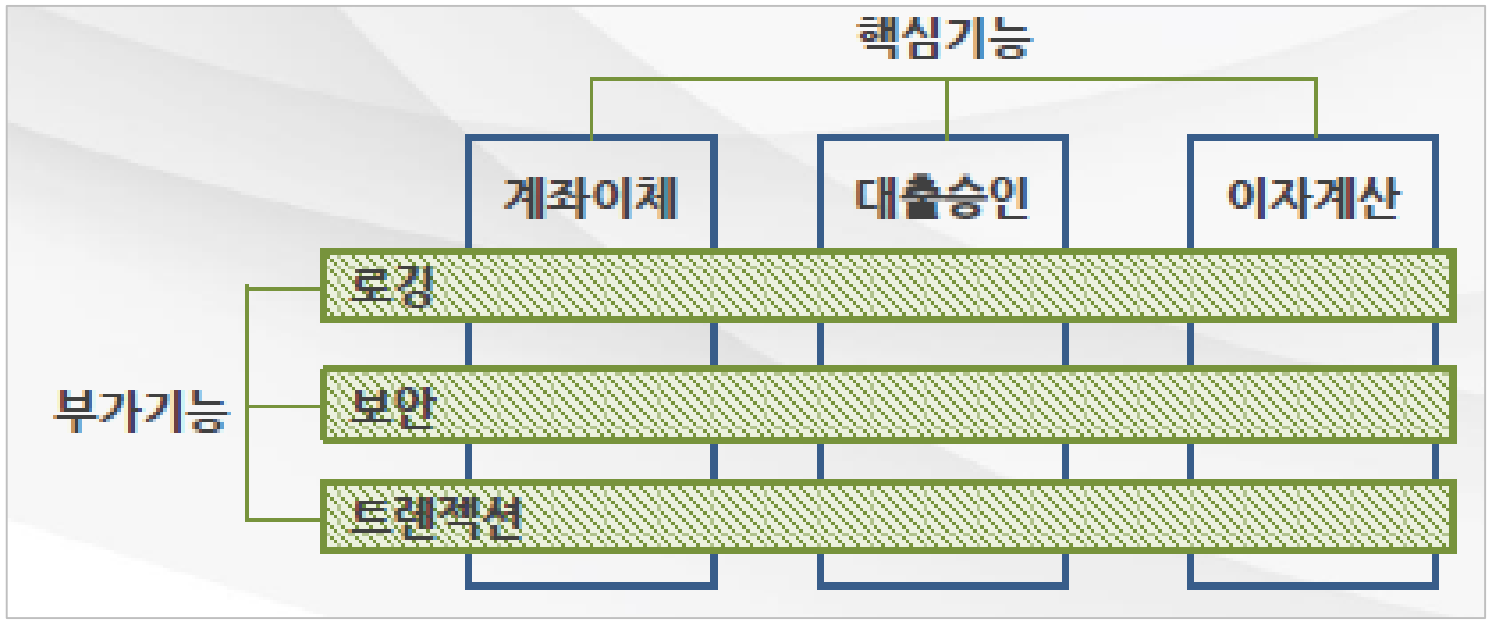
- 1주차 ~ 4주차 : //스프링을 위한 선수 지식 : Servlet 요점 정리
- SpringProject1: (5주차) //개발자에 의한 객체 결합 (교재 p39~48)
- SpringProject2: (6주차) //스프링컨테이너에 의한 객체 결합 (XML 방식) - IoC- (교재 p49~91)
- SpringProject3: (7주차) //스프링컨테이너에 의한 객체 결합 (어노테이션 방식) - IoC- (교재 p93~108)
- SpringProject4_BizStep1: (8주차) //비즈니스 컴포넌트 실습 1 (게시판) -Business Layer- (교재 p109~127)
- SpringProject4_BizStep2: (9주차) // 비즈니스 컴포넌트 실습 2 (User추가) -Business Layer- (교재 p129~139)
- SpringProject5_AOP_step1: (9주차) // 스프링 AOP 개념 파악 - AOP - (교재 p143~170)
- SpringProject5_AOP_step2: (10주차) // AOP 동작 시점 (5개) 실습- AOP - (교재 p171~182)
- SpringProject5_AOP_step3: (10주차) // AOP 동작 시점과 JointPoint/바인드변수 실습- AOP - (교재 p183~192)
- SpringProject5_AOP_step4: (10주차) // 어노테이션 방식의 AOP 실습- AOP - (교재 p193~192)

9주 과제

10주 과제

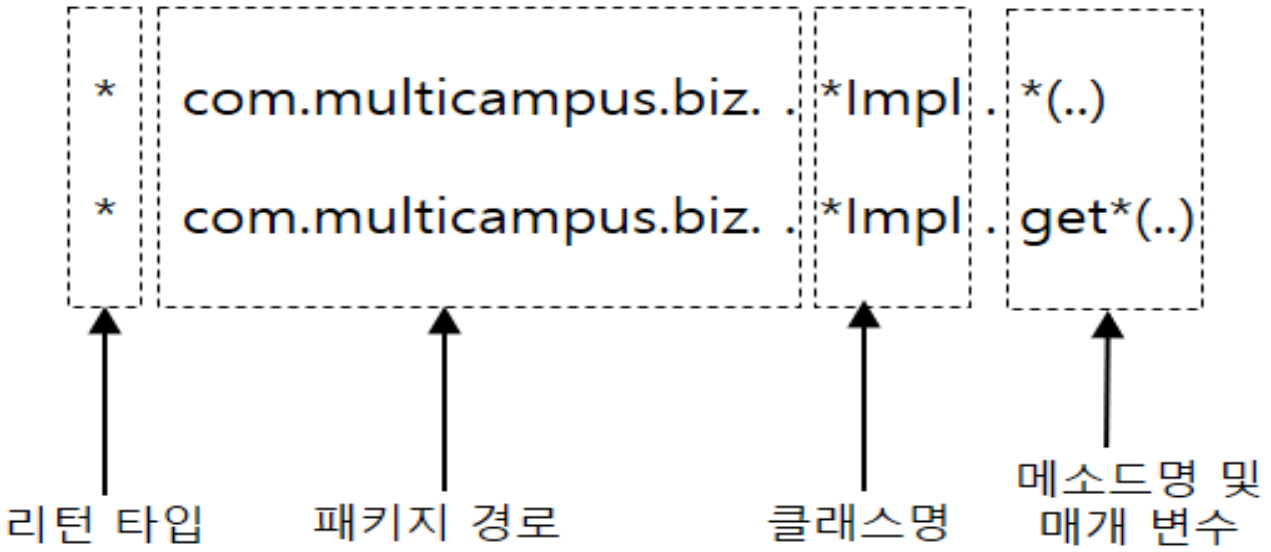
# AOP(Aspect Oriented Programming)의 개요

- 핵심기능과 부가기능
  - 업무(Biz) 로직을 포함하는 기능을 **핵심 기능(Core Concerns)**
  - 핵심기능을 도와주면서 공통적이고 반복적으로 처리하는 내용을 부가기능 **(Cross-cutting Concerns, 횡단관심사)** 이라고 부른다.
  - 프로그램 안에서 횡단 관심사에 해당하는 부분을 분리해서 한 곳으로 모으는 것을 횡단관심사의 분리라 하고, 이를 실현하는 방법을 관점지향 프로그래밍(AOP)이라 함



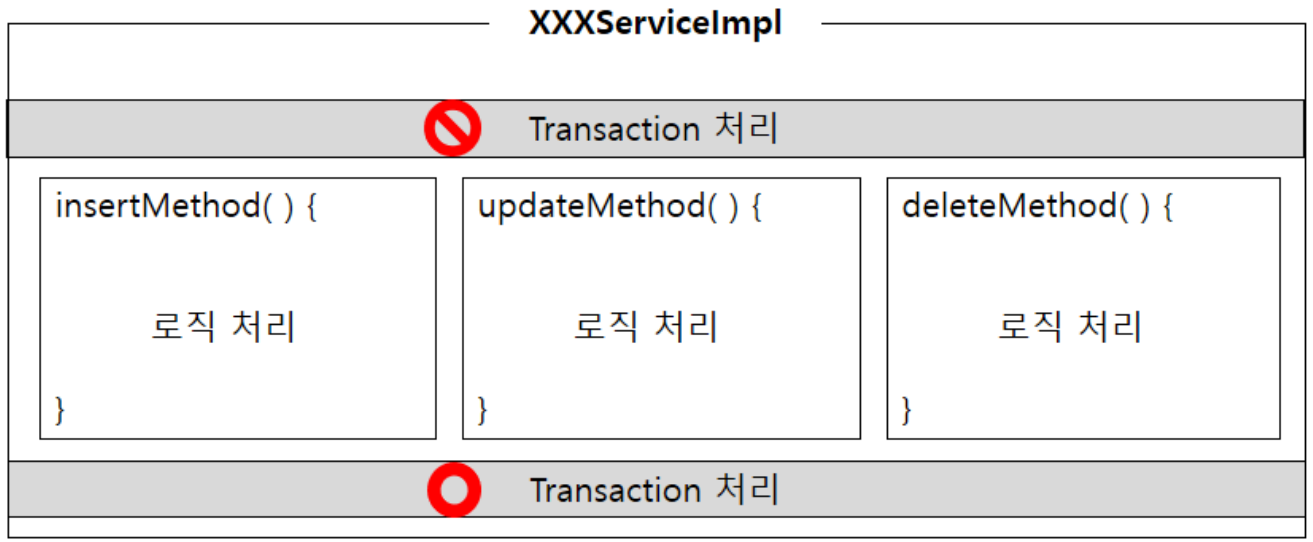
# AOP 용어

- **조인 포인트(Join Point)**
  - 클라이언트가 호출하는 모든 비즈니스 메소드
  - 포인트컷 대상 또는 포인트컷 후보
- **포인트 컷(Pointcut)**
  - 수많은 조인 포인트 중에서 실제로 어드바이스를 적용할 곳을 선별하기 위한 표현식을 말함. (필터링된 조인트포인트)
  - 스프링 AOP에서는 포인트 컷을 정의할 때 XML 기반 설정방식으로 빈 정의 파일을 만들거나, 어노테이션 기반 설정방식으로 소스코드에 주석 형태로 정의한다.
  - 포인트컷 표현식은 execution으로 시작하고, 메서드의 Signature를 비교하는 방법을 주로 이용한다.



# AOP 용어

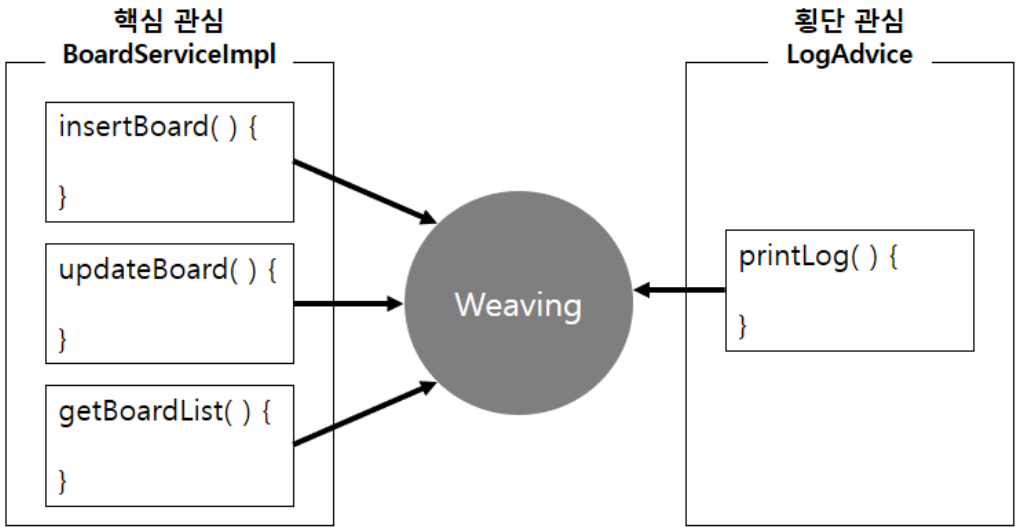
- **어드바이스(Advice)**
  - 특정 조인 포인트에서 실행되는 코드로 횡단 관심사를 실제로 구현해서 처리하는 부분
  - 어드바이스 동작 시점을 5가지로 지정 : before, after, after-returning, after-throwing, around




- **애스팩트(Aspect)** 또는 **어드바이저(Advisor)**
  - 애스팩트(어드바이저 ) = 어드바이스 + 포인트컷
  - 어떤 포인트컷 메소드에 대해서 어떤 어드바이스 메소드 실행할 지 결정
  - <aop:aspect> 또는 <aop:advisor> (트랜잭션 때 사용)

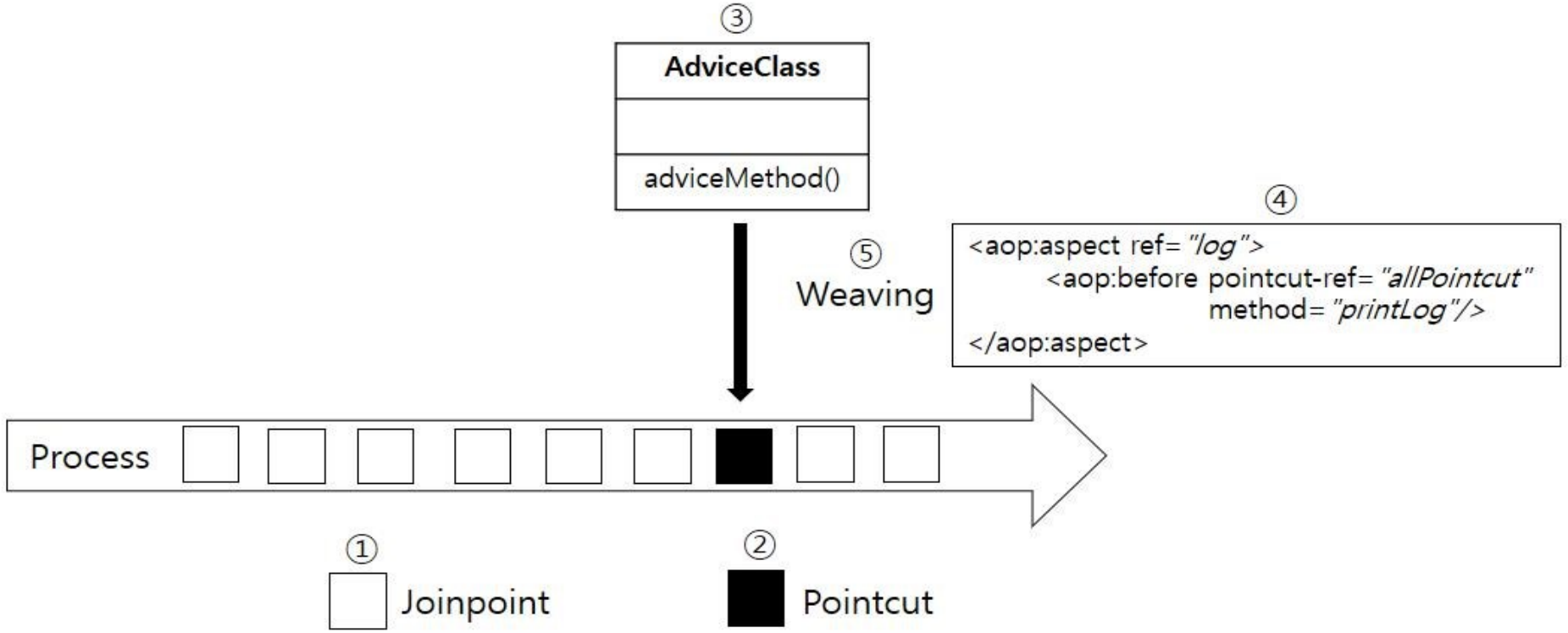
# AOP 용어

- **위빙(Weaving)**
  - 스프링 AOP는 기본적으로 실행 시점에 위빙한다.
  - 위빙은 포인트컷에 의해서 결정된 타겟의 조인 포인트에 부가기능(어드바이스)을 삽입하는 과정을 뜻한다.




- **타겟(Target)**
  - 핵심기능을 담고 있는 모듈로, 타겟은 부가기능을 부여할 대상이 된다.

# AOP 용어 정리




① 사용자가 비즈니스 컴포넌트의 여러 조인포인트 호출
② 이때 특정 포인트컷으로 지정한 메소드가 호출되는 순간,
③ 어드바이스 객체의 어드바이스 메소드가 실행되는데..
④ 위 ③의 실행을 위한 설정 즉, 애스팩트에 의해서 실행이 된다.
⑤ 위의 ③④ 실행과정을 위빙이라한다.

# Spring AOP 설정

● AOP와 관련된 다양한 설정 : XML 방식과 어노테이션 방식

1) XML 기반의 POJO 클래스를 이용한 AOP 구현

- 부가기능을 제공하는 Advice 클래스를 작성한다.
- XML 설정 파일에 <aop:config>를 이용해서 애스펙트를 설정한다. (즉, 어드바이스와 포인트컷을 설정함)

2) @Aspect 어노테이션을 이용한 AOP 구현

- @Aspect 어노테이션을 이용해서 부가기능을 제공하는 Aspect 클래스를 작성한다. 이때 Aspect 클래스는 어드바이스를 구현하는 메서드와 포인트컷을 포함한다.
- XML 설정 파일에 <aop:aspectj-autoproxy />를 설정한다.

# Spring AOP 설정 by xml 방식

**LogAdvice.java**

```
public class LogAdvice {
        public void printLog() {
                System.out.println("[공통 로그] 비즈니스 로직 수행 전 동작");
        }
}
```

**applicationContext.xml**

```
<bean id="log" class="com.springbook.biz.common.LogAdvice"></bean>

<aop:config>
        <aop:pointcut id="allPointcut" expression="execution(* com.springbook.biz..*Impl.*(..))"/>

        <aop:pointcut id="getPointcut" expression="execution(* com.springbook.biz..*Impl.get*(..))"/>

        <aop:aspect ref="log">
                <aop:before pointcut-ref="getPointcut" method="printLog"/>

        </aop:aspect>
</aop:config>
```

② ④ ① ③

# Spring AOP 설정 by xml 방식

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans" …>

    <aop:config>

        <aop:pointcut …/>

        <aop:aspect …></aop:aspect>

    </aop:config>

</beans>
```

## 1) < aop:config > 엘리먼트

- AOP설정에서의 루트 엘리먼트 (여러 번 사용가능)
- 하위 엘리멘트로 <aop:pointcut>, <aop:aspect>

## 2) < aop:pointcut > 엘리먼트

- 포인트컷 지정
- <aop:config> 혹은 <aop:aspect> 의 자식 엘리먼트 사용 가능
  (단, <aop:aspect>의 자식으로 설정된 포인트컷은 해당 <aop:aspect>에서만 사용가능)

# Spring AOP 설정 by xml 방식

3) < aop:aspect > 엘리먼트

- 핵심관심에 해당하는 포인트컷 메소드와 횡단관심에 해당하는 어드바이스 메소드를 결합하기위해
- 애스팩트를 어떻게 설정하느냐에 따라 위빙 결과가 달라짐
- 어드바이스 객체의 아이디나 메소드 이름을 모르면 사용 불가능

4) < aop:advisor > 엘리먼트

- < aop:aspect > 엘리먼트와 같은 기능
- 트랜잭션 설정같은 몇몇 특수한 경우 < aop:aspect > 앨리먼트 대신에 사용

※ 포인트컷 표현식 (교재 p168~170)

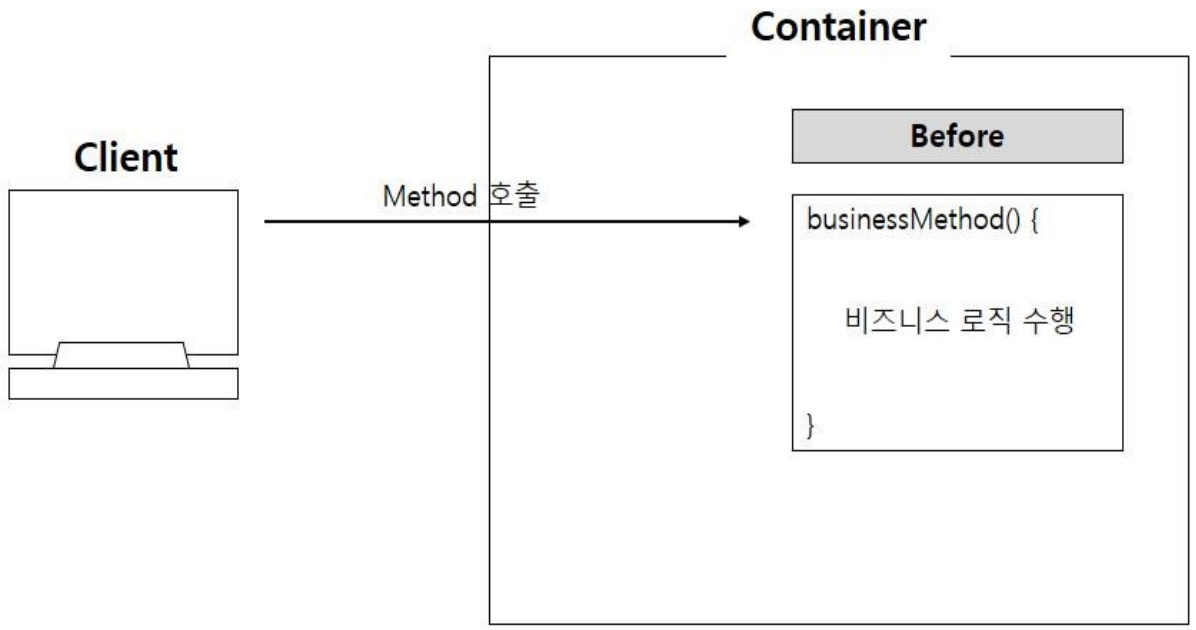
# 어드바이스 동작 시점

다음 5개의 동작 시점을 제공

| 동작 시점 | 설 명 |
| --- | --- |
| **Before** | 비즈니스 메소드 실행 전 동작 |
| **After** | - After Returning: 비즈니스 메소드가 성공적으로 반환되면 동작<br>- After Throwing: 비즈니스 메소드 실행 중 예외가 발생하면 동작<br>**(try~catch 블록에서 catch 블록에 해당)**<br>- After: 비즈니스 메소드가 실행된 후, 무조건 실행<br>**(try~catch~finally 블록에서 finally 블록에 해당)** |
| **Around** | Around는 메소드 호출 자체를 가로채 비즈니스 메소드 실행 전후에 처리할 로직을 삽입할 수 있음 |

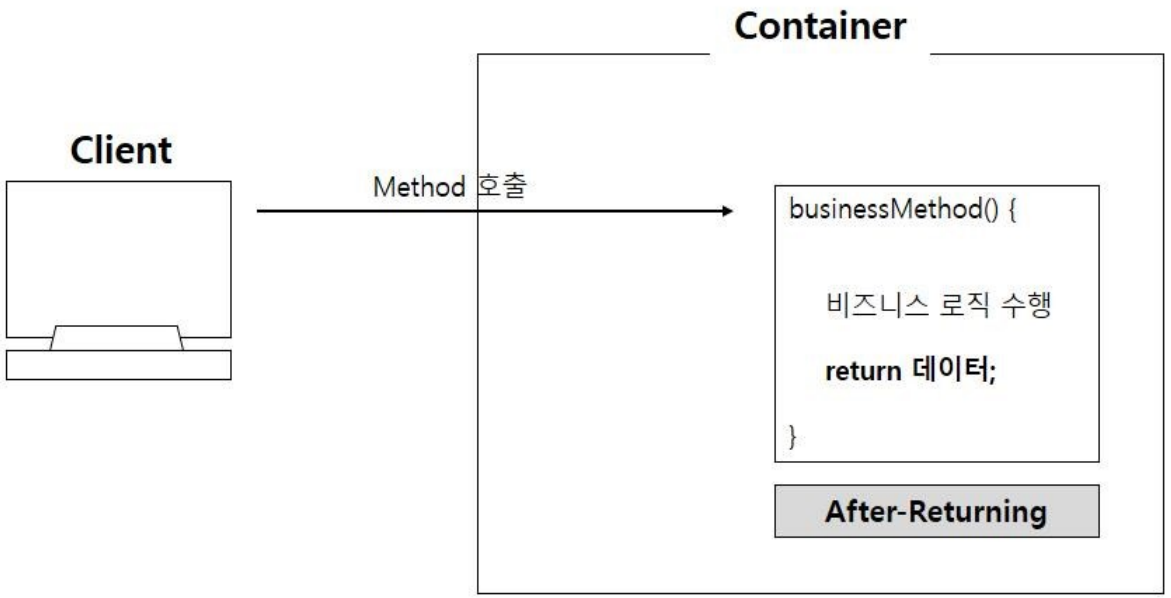
# 어드바이스 동작 시점

- **Before** 어드바이스 동작 시점




**실습예제)**
**SpringProject5_AOP_step2 :**
- **BeforeAdvice.class**
- BoardServiceClientBefore.class
- **applicationContextBefore.xml**

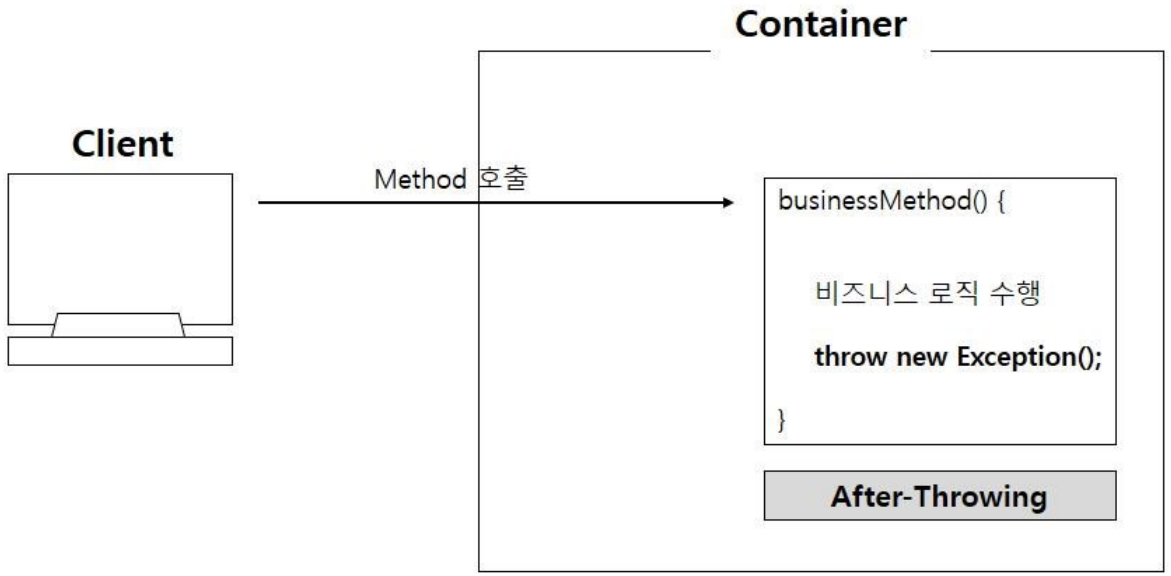
- **After Returning** 어드바이스 동작 시점




**실습예제)**
**SpringProject5_AOP_step2 :**
- AfterReturningAdvice.**class**
- BoardServiceClientAfterReturning.class
- **applicationContextAfterReturning.xml**

# 어드바이스 동작 시점

- **After Throwing** 어드바이스 동작 시점



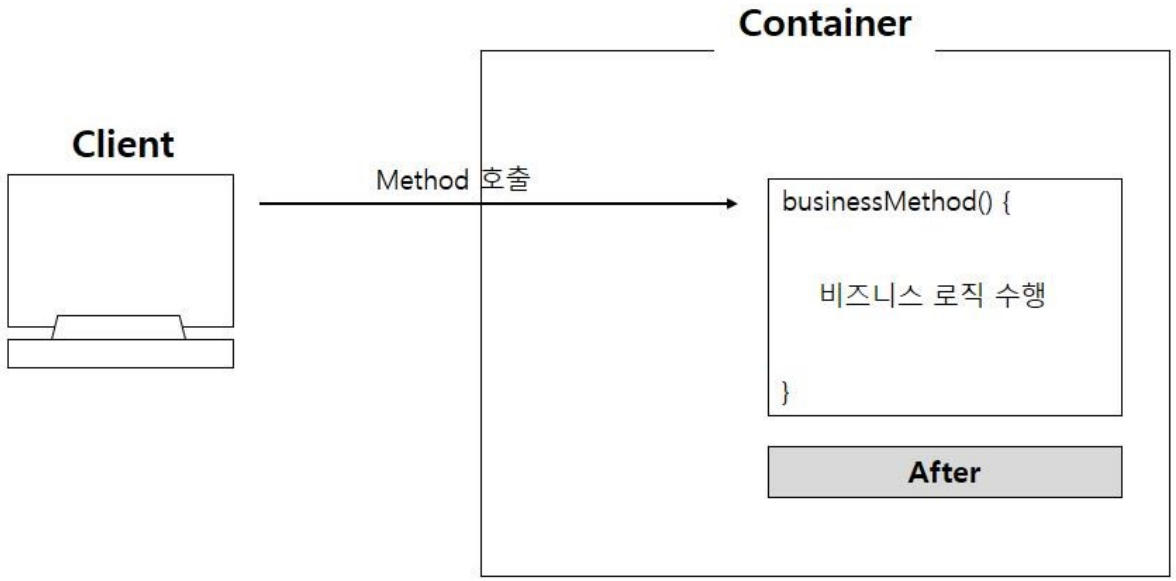

**실습예제)**

**SpringProject5_AOP_step2 :**

- AfterThrowingAdvice.**class**
- BoardServiceClientAfterThrowing.class
- applicationContextAfterThrowing.**xml**

(BoardServiceImpl.class 에러상황 추가!)

- **After** 어드바이스 동작 시점




**실습예제)**

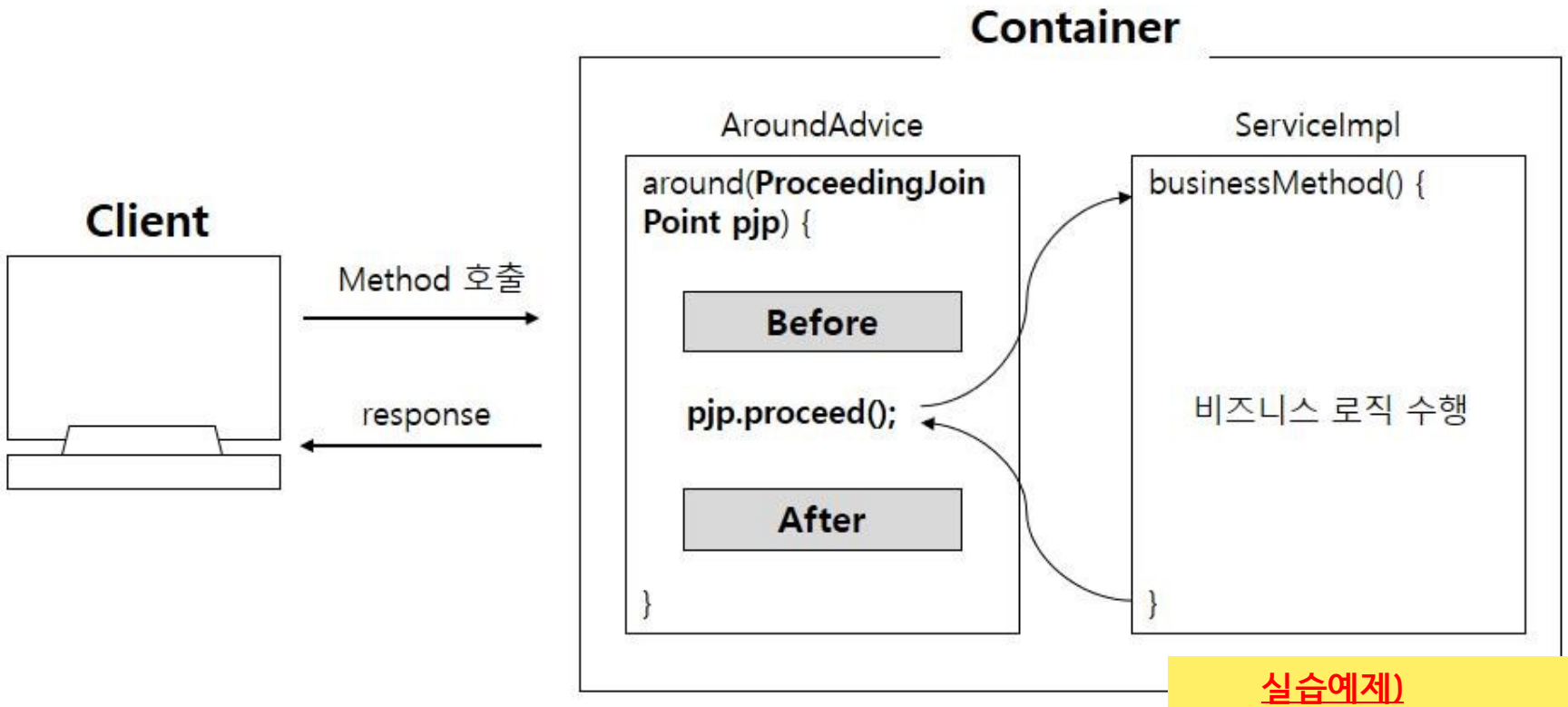
**SpringProject5_AOP_step2 :**

- AfterAdvice.**class**
- BoardServiceClientAfter.class
- applicationContextAfter.**xml**

(BoardServiceImpl.class 에러 有 / 無 에서 실행)

# 어드바이스 동작 시점

- **Around** 어드바이스 동작 시점




**실습예제)**

**SpringProject5_AOP_step2 :**

- AroundAdvice.**class**
- BoardServiceClientAround.class
- applicationContextAround.**xml**

# JoinPoint와 바인드 변수 p183

- **JointPoint 인터페이스 :** 횡단 관심에 해당하는 어드바이스 메소드를 의미있게 구현하기 위해 클라이언트가 호출한 비즈니스 메소드의 다양한 정보가 포함되어 있는 인터페이스
- Before / After Returning / After Throwing / After 어드바이스에서 사용
- JointPoint 메소드

| 메소드 | 설 명 |
|---|---|
| Signature getSignature( ) | - 클라이언트가 호출한 메소드의 시그니처(반환형, 이름, 매개변수) 정보가 저장된 Signature 객체 반환<br>- Signature객체의 메소드 :<br>● String getName( ) 클라이언트가 호출한 메소드 이름 리턴<br>● String toLongString( ) 클라이언트가 호출한 메소드의 리턴타입, 이름, 매개변수를 패키지경로까지 리턴<br>● String toShortString( ) 클라이언트가 호출한 메소드 시그너처를 축약한 문자열로 리턴 |
| Object getTarget( ) | 클라이언트가 호출한 비즈니스 메소드를 포함하는 비즈니스 객체 반환 |
| Object[] getArgs( ) | 클라이언트가 메소드를 호출할 때 넘겨준 인자 목록을 Object 배열로 반환 |

- **ProceedingJointPoint 인터페이스 :**
  JointPoint를 상속받은 인터페이스이며, JointPoint메소드+proceed( )메소드 추가
- Around 어드바이스에서 사용

# JoinPoint와 바인드 변수 - 실습

- **Before** 어드바이스

실습예제)
SpringProject5_AOP_step3 :
- **BeforeAdvice.class**
- BoardServiceClientBefore.class
-**applicationContextBefore.xml**

- **After Returning** 어드바이스

실습예제)
SpringProject5_AOP_step3 :
- AfterReturningAdvice.**class**
- BoardServiceClientAfterReturning.class
- UserServiceClientAfterReturning.class
-**applicationContextAfterReturning.xml**

Board와 User 두개 다 테스트

- **After Throwing** 어드바이스

실습예제)
SpringProject5_AOP_step3 :
- AfterThrowingAdvice.**class**
- BoardServiceClientAfterThrowing.class
-applicationContextAfterThrowing.**xml**
(BoardServiceImpl.class 에러상황 추가!)

- **After** 어드바이스

실습예제)
SpringProject5_AOP_step3 :
- AfterAdvice.**class**
- BoardServiceClientAfter.class
-applicationContextAfter.**xml**
(BoardServiceImpl.class 에러 有 / 無 에서 실행)

- **Around** 어드바이스

실습예제)
SpringProject5_AOP_step3 :
- AroundAdvice.**class**
- BoardServiceClientAround.class
-applicationContextAround.**xml**

# Spring AOP 설정 by 어노테이션 방식 p193

- 먼저 스프링 설정파일에 <aop:aspect-autoproxy> 엘리먼트 선언

```
13    <context:component-scan base-package="com.springbook.biz">
14    </context:component-scan>
15
16    <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
17
18 </beans>
```

- AOP 관련 어노테이션들은 어드바이스 클래스에 설정

**< 어노테이션>**

```
 6 import org.springframework.stereotype.Service;
 7
 8 @Service
 9 @Aspect
10 public class LogAdvice {
11     @Pointcut("execution(* com.springbook.biz..*Impl.
12     public void allPointcut(){}
13
14     @Pointcut("execution(* com.springbook.biz..*Impl.get*(..))")
15     public void getPointcut(){}
16
17     @Before("allPointcut()")
18     public void printLog() {
19         System.out.println("[공통 로그] 비즈니스 로직 수행 전 동작");
20     }
```

**< XML >**

```
<bean id="log" class="com.springbook.biz.common.Befo
<aop:config>
    <aop:pointcut id="allPointcut" expression="execu
    <aop:pointcut id="getPointcut" expression="execu
    <aop:aspect ref="log">
        <aop:before pointcut-ref="getPointcut" metho
    </aop:aspect>
</aop:config>
```

# Spring AOP 설정 by 어노테이션 방식

- 어드바이스 동작시점과 관련된 어노테이션

| 어노테이션 | 설 명 |
|---|---|
| **@Before** | 비즈니스 메소드 실행 전에 동작 |
| **@AfterReturning** | 비즈니스 메소드가 성공적으로 반환되면 동작 |
| **@AfterThrowing** | 비즈니스 메소드 실행 중 예외가 발생하면 동작<br>(마치 try~catch 블록에서 catch 블록에 해당). |
| **@After** | 비즈니스 메소드가 실행된 후, 무조건 실행<br>(try~catch~finally 블록에서 finally 블록에 해당) |
| **@Around** | 호출 자체를 가로채 비즈니스 메소드 실행 전후에 처리할 로직을 삽입할 수 있음 |

# Spring AOP 설정 by 어노테이션 방식

실습예제) SpringProject5_AOP_step4 :

- **Before** 어드바이스

```
@Service
@Aspect
public class BeforeAdvice {
	@Pointcut("execution(* com.springbook.biz..*Impl.*(..))")
	public void allPointcut() {}

	@Before("allPointcut()")
	public void beforeLog(JoinPoint jp) {
		String method = jp.getSignature().getName();
		Object[] args = jp.getArgs();

		System.out.println("[사전 처리] " + method +
			"() 메소드 ARGS 정보 : " + args[0].toString());
	}
}
```

- **After Returning** 어드바이스

```
@Service
@Aspect
public class AfterReturningAdvice {
	@Pointcut("execution(* com.springbook.biz..*Impl.get*(..))")
	public void getPointcut() {}

	@AfterReturning(pointcut="getPointcut()", returning="returnObj")
	public void afterLog(JoinPoint jp, Object returnObj) {
		String method = jp.getSignature().getName();

		System.out.println("[사후 처리] " + method +
			"() 메소드 리턴값 : " + returnObj.toString());
	}
}
```

# Spring AOP 설정 by 어노테이션 방식

실습예제) SpringProject5_AOP_step4 :

- **After Throwing** 어드바이스

```
@Service
@Aspect
public class AfterThrowingAdvice {
	@Pointcut("execution(* com.springbook.biz..*Impl.*(..))")
	public void allPointcut() {}

	@AfterThrowing(pointcut="allPointcut()", throwing="exceptObj")
	public void exceptionLog(JoinPoint jp, Exception exceptObj) {
		String method = jp.getSignature().getName();
		System.out.println(method + "() 메소드 수행 중 예외 발생!");

		if(exceptObj instanceof IllegalArgumentException) {
			System.out.println("부적합한 값이 입력되었습니다.");
		}
	}
}
```

- **After** 어드바이스

```
@Service
@Aspect
public class AfterAdvice {
	@Pointcut("execution(* com.springbook.biz..*Impl.*(..))")
	public void allPointcut() {}

	@After("allPointcut()")
	public void finallyLog() {
		System.out.println("[사후 처리] 비즈니스 로직 수행 후 무조건 동작");
	}
}
```

# Spring AOP 설정 by 어노테이션 방식

실습예제) SpringProject5_AOP_step4 :

- **Around** 어드바이스

```
@Service
@Aspect
public class AroundAdvice {
	@Pointcut("execution(* com.springbook.biz..*Impl.*(..))")
	public void allPointcut(){}

	@Around("allPointcut()")
	public Object aroundLog(ProceedingJoinPoint pjp) throws Throwable {
		String method = pjp.getSignature().getName();
			System.out.println(method + "() 메소드 수행에 걸린 시간 : "
			+ stopWatch.getTotalTimeMillis() + "(ms)초");
		return obj;
	}
}
```

# Spring AOP 설정 by 어노테이션 방식

실습예제) SpringProject5_AOP_step4 :

- 외부 Pointcut 참조

```
@Aspect
public class PointcutCommon {
        @Pointcut("execution(*
com.springbook.biz..*Impl.*(..))")
        public void allPointcut() {}
        @Pointcut("execution(*
com.springbook.biz..*Impl.get*(..))")
        public void getPointcut() {}
}
```

```
@Service
@Aspect
public class BeforeAdvice {
        @Before("PointcutCommon.allPointcut()")
        public void beforeLog(JoinPoint jp) {

        }
}
```