Backend Seminar

AOP(Aspect Oriented Programming)

매시업 백엔드 9기

김다롬, 김승현

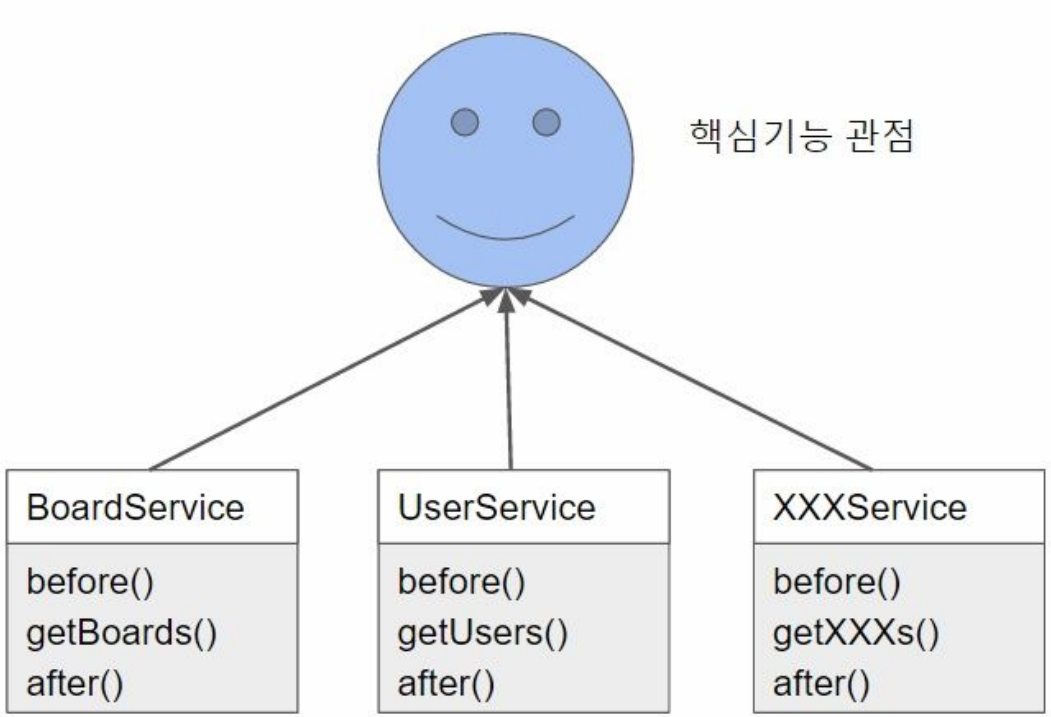
CONTENTS

1. AOP란?

1. AOP란?

```
1  class A {
2      method a() {
3          AAAA
4          ...
5          BBBB
6      }
7      method b() {
8          AAAA
9          …
10         BBB
11     }
12 }
13
14 class B {
15     method c() {
16         AAAA
17         …
18         BBBB
19     }
20 }
```

코드의 중복



1. AOP란?

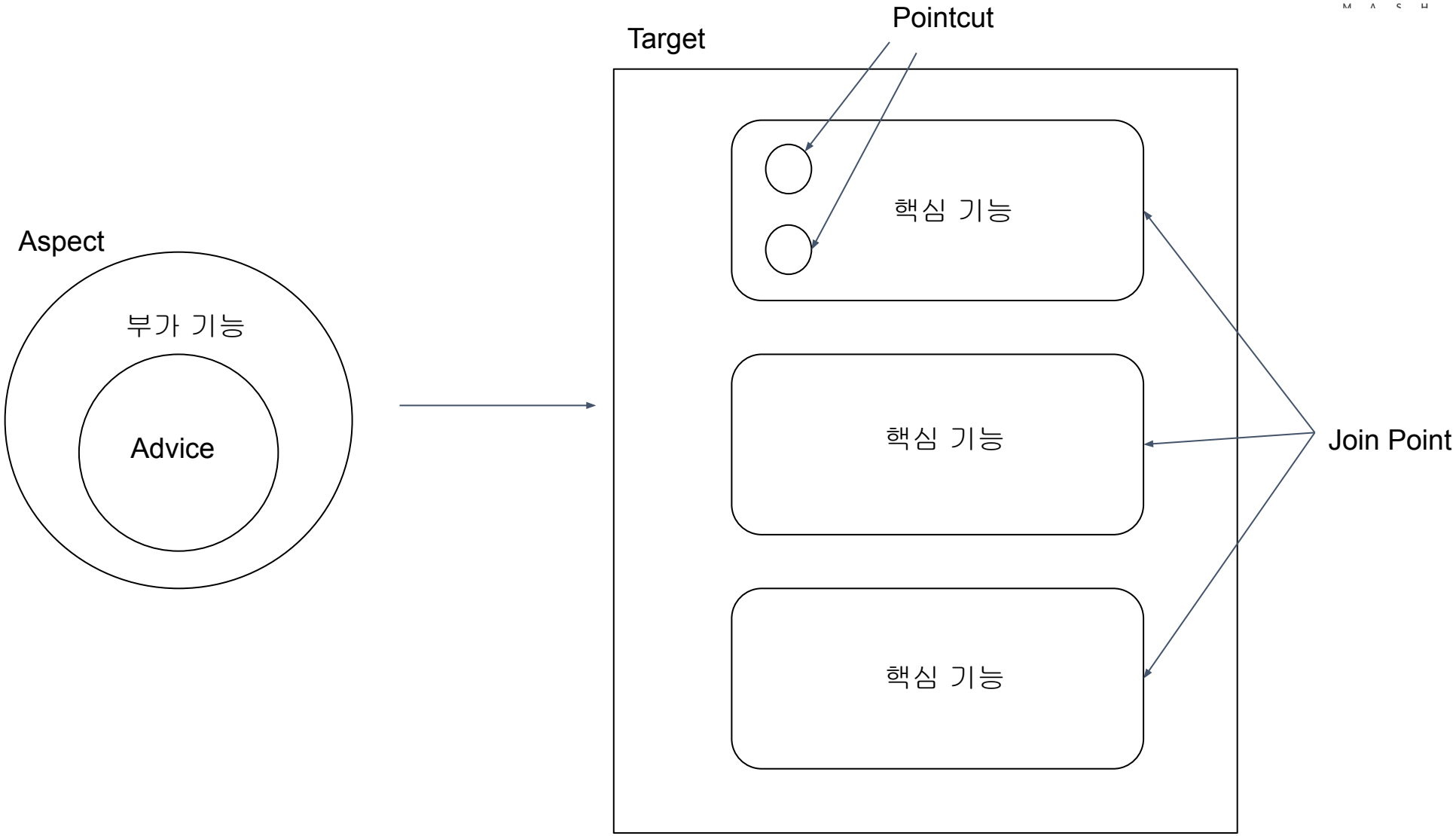
여러 오브젝트에 나타나는 공통적인 부가 기능을 모듈화 하여 재사용하는 기법

```java
@Aspect
public class Performance {

    @Around("execution(* com.blogcode.board.BoardService.getBoards(..))")
    public Object calculatePerformanceTime(ProceedingJoinPoint proceedingJoinPoint) {
        Object result = null;
        try {
            long start = System.currentTimeMillis();
            result = proceedingJoinPoint.proceed();
            long end = System.currentTimeMillis();

            System.out.println("수행 시간 : "+ (end - start));
        } catch (Throwable throwable) {
            System.out.println("exception! ");
        }
        return result;
    }
}
```

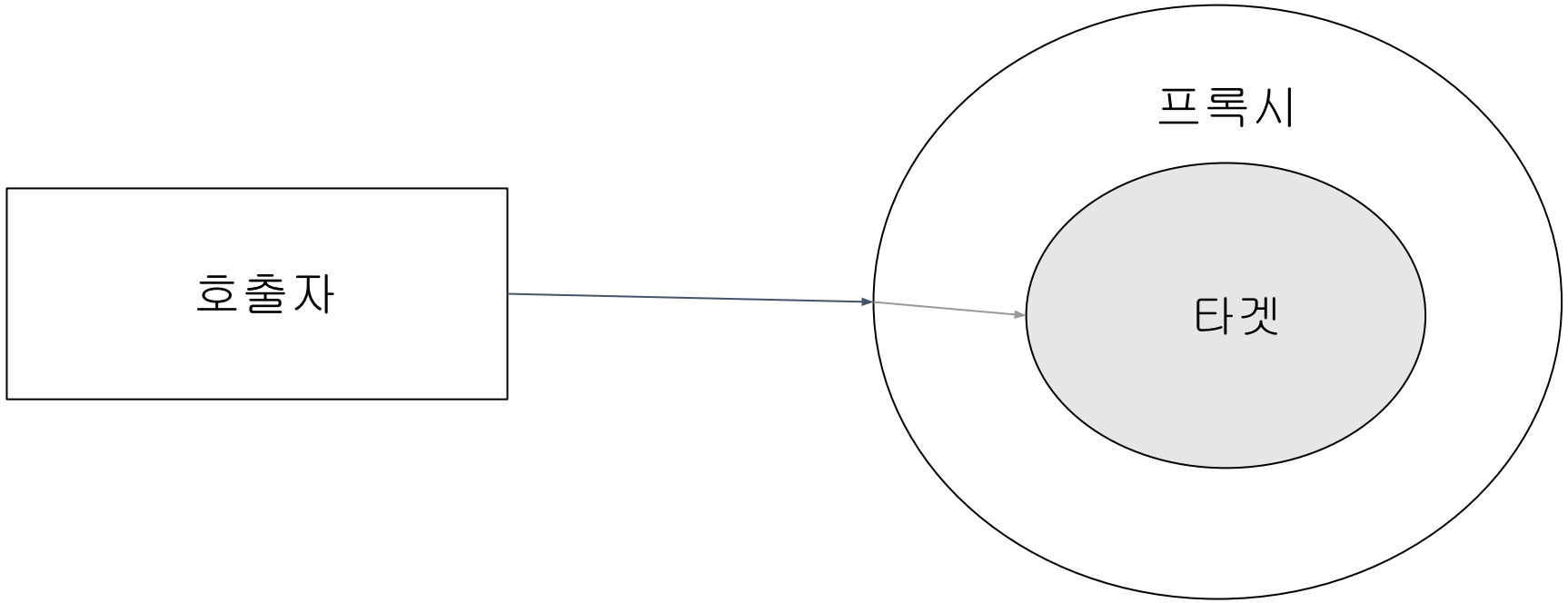
- 로그처리(성능)
- 보안처리(권한 확인)
- 트랜잭션처리

→ 애플리케이션 전체에 흩어진 공통 기능이 하나의 장소에서 관리, 다른 서비스 모듈은 본인의 목적에만 충실 가능

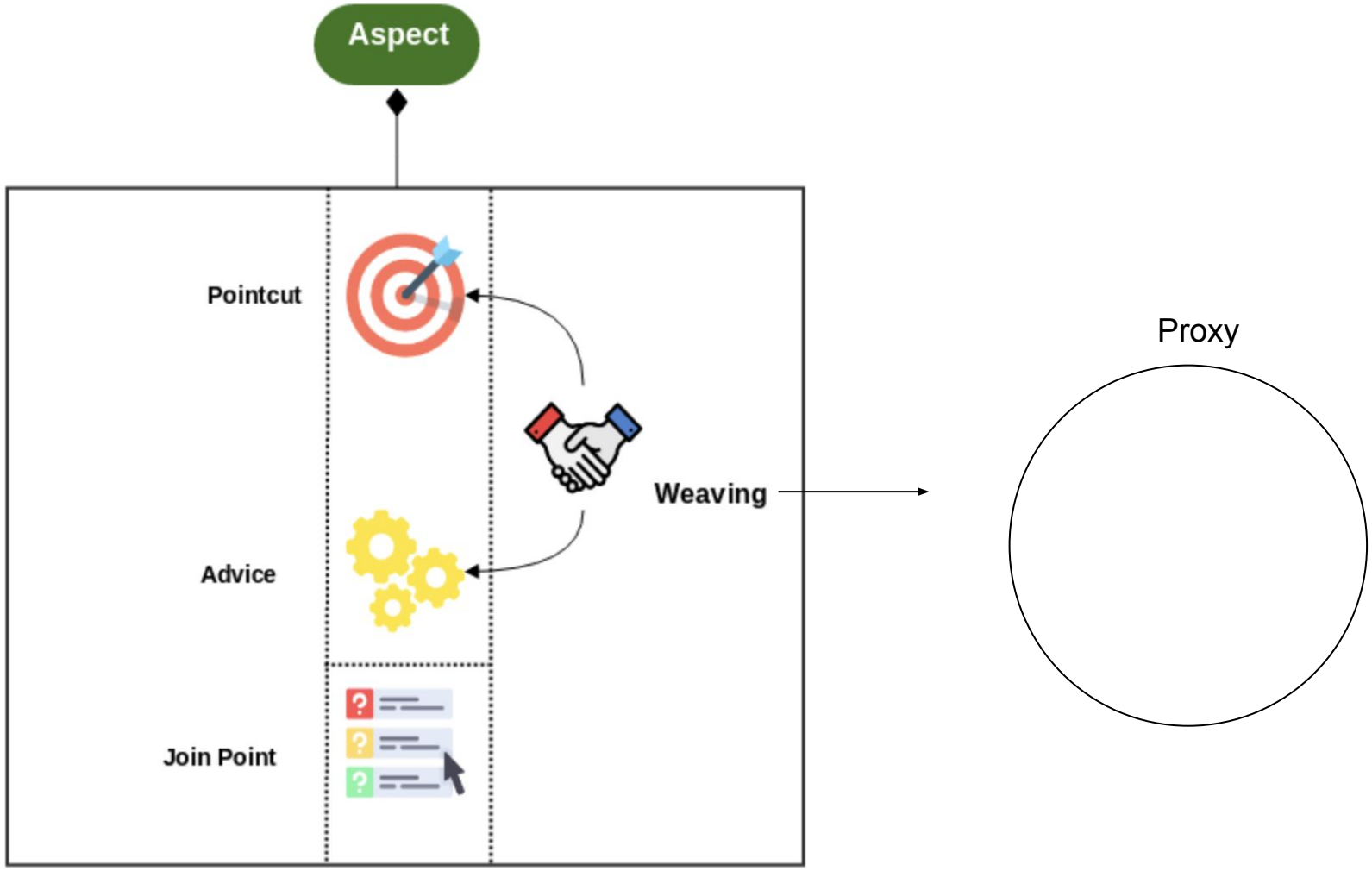
1. AOP란?


Aspect
부가 기능
Advice
Target
Pointcut
핵심 기능
핵심 기능
핵심 기능
Join Point

1. AOP란?



1. AOP란?



2. Advice

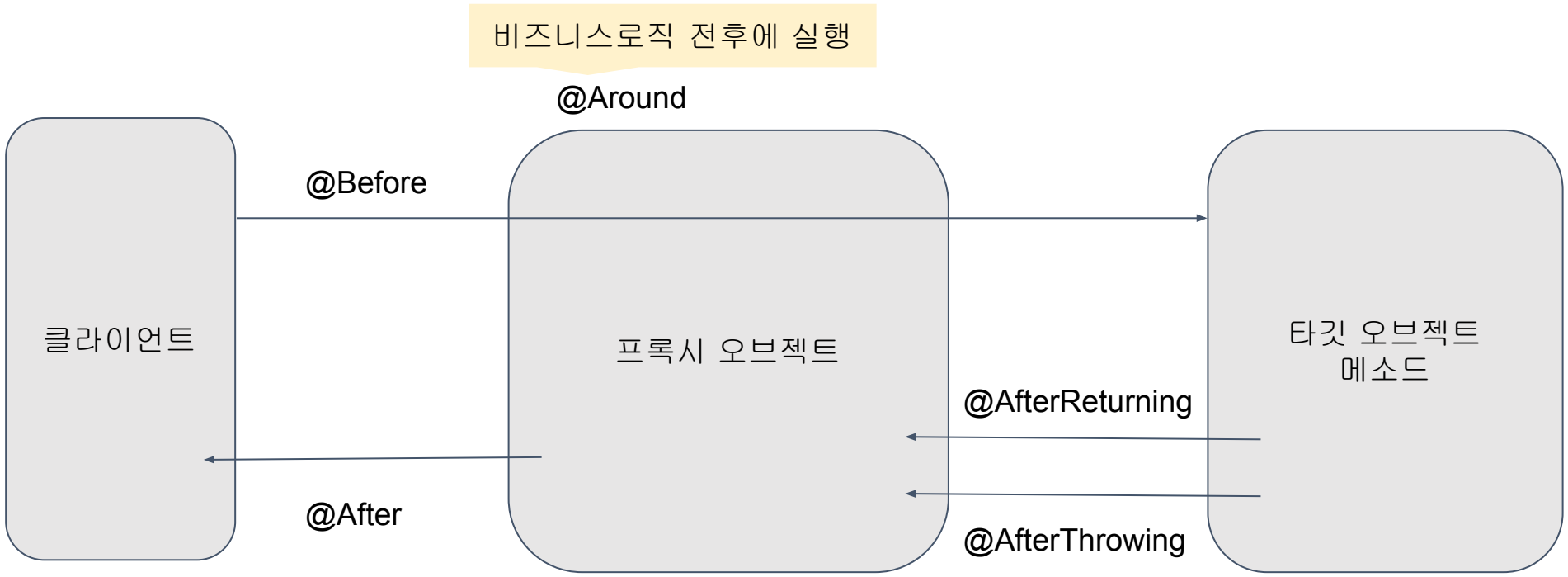
- Before
- After returning
- After throwing
- Around

2. Advice

타깃에게 제공할 부가기능을 담은 모듈, 어떤 부가기능을 언제 사용할지에 대한 정의

- **@Before** (이전)
  - 어드바이스 타겟 메소드가 호출되기 전에 어드바이스 기능을 수행
- **@After** (이후)
  - 타겟 메소드의 결과에 관계없이(즉 성공, 예외 관계없이) 타겟 메소드가 완료 되면 어드바이스 기능을 수행
- **@AfterReturning** (정상적 반환 이후)
  - 타겟 메소드가 성공적으로 결과값을 반환 후에 어드바이스 기능을 수행
- **@AfterThrowing** (예외 발생 이후)
  - 타겟 메소드가 수행 중 예외를 던지게 되면 어드바이스 기능을 수행
- **@Around** (메소드 실행 전후)
  - 어드바이스가 타겟 메소드를 감싸서 타겟 메소드 호출전과 후에 어드바이스 기능을 수행

2. Advice


비즈니스로직 전후에 실행
@Around
@Before
클라이언트
프록시 오브젝트
타깃 오브젝트
메소드
@AfterReturning
@After
@AfterThrowing

2. Advice : @Before

```java
package com.journaldev.spring.aspect;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class EmployeeAspect {

	@Before("execution(public String getName())")
	public void getNameAdvice(){
		System.out.println("Executing Advice on getName()");
	}

	@Before("execution(* com.journaldev.spring.service.*.get*())")
	public void getAllAdvice(){
		System.out.println("Service method getter called");
	}
}
```

- 구현할 클래스는 **@Aspect** 어노테이션을 붙여야 한다.
- 부수적인 업무에 해당하는 메소드에 **@Before** 어노테이션을 붙인다.
- getNameAdvice() 메소드는 `public string getName()` 이라는 시그니처를 가진 스프링 빈 메소드에 대해서 실행된다.
- PointCut 표현식에서 **와일드 카드**를 사용할 수 있다.
- getAllAdvice() 는 `com.journaldev.spring.service`에 속하면서 get으로 시작하는 모든 클래스에 적용된다.

2. Advice : @Around

```
@Aspect
public class EmployeeAroundAspect {

        @Around("execution(* com.journaldev.spring.model.Employee.getName())")
        public Object employeeAroundAdvice(ProceedingJoinPoint proceedingJoinPoint)
{
                System.out.println("Before invoking getName() method");
                Object value = null;
                try {
                        value = proceedingJoinPoint.proceed();
                } catch (Throwable e) {
                        e.printStackTrace();
                }
                System.out.println("After invoking getName() method. Return
value="+value);
                return value;
        }
}
```

- Around 어드바이스에서는 타겟 오브젝트에 어드바이스를 적용하기 위해서 항상 ProceedingJoinPoint의 proceed() 메소드를 사용해야 한다.
- 반환값이 있다면 caller 프로그램에게 반환해주고, 반환값이 없다면 null을 리턴해도 된다.

2. Advice : @After

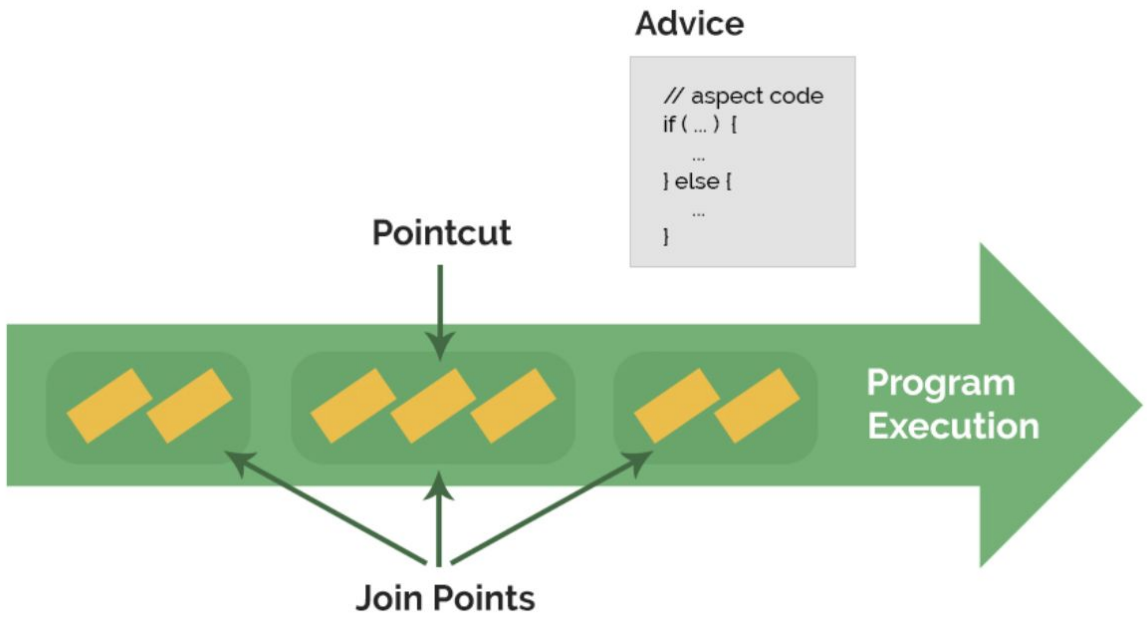
```
@Aspect
public class EmployeeAfterAspect {

	@After("args(name)")
	public void logStringArguments(String name){
		System.out.println("Running After Advice. String argument
passed="+name);
	}

	@AfterThrowing("within(com.journaldev.spring.model.Employee)")
	public void logExceptions(JoinPoint joinPoint){
		System.out.println("Exception thrown in Employee
Method="+joinPoint.toString());
	}

	@AfterReturning(pointcut="execution(* getName())",
returning="returnString")
	public void getNameReturningAdvice(String returnString){
		System.out.println("getNameReturningAdvice executed. Returned
String="+returnString);
	}

}
```

3. 정리

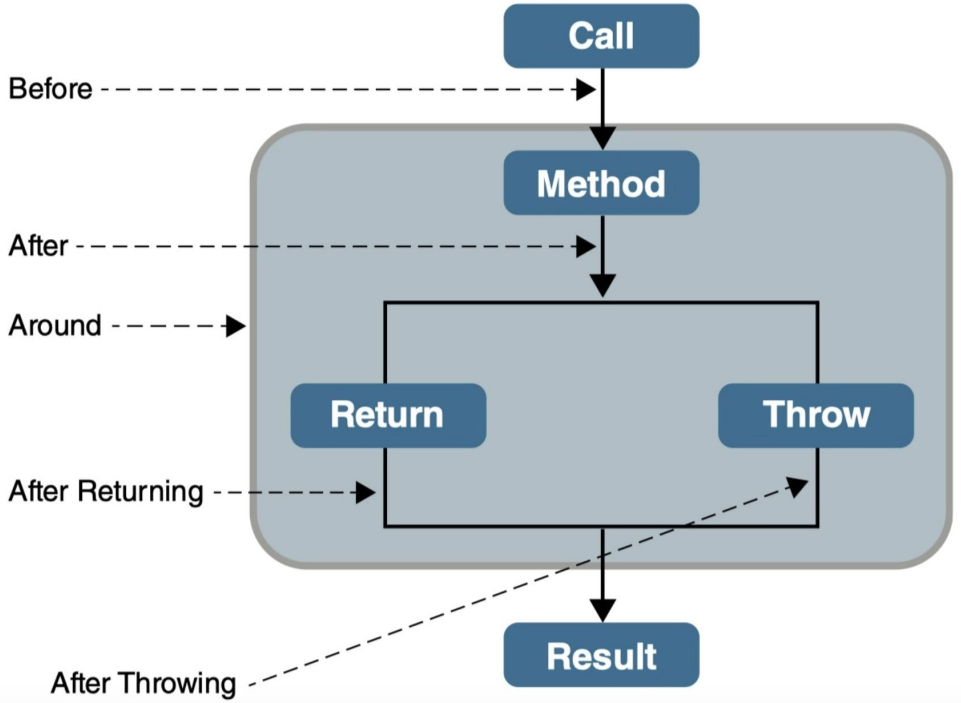
3. 정리

AOP(Aspect Oriented Programming)

= 여러 오브젝트에 나타나는 공통적인 부가 기능을 모듈화 하여 재사용하자!



어떤 부가 기능을 언제 사용할까?



참고문서

유튜브

https://www.youtube.com/watch?v=y2JkXjOocZ4

https://www.youtube.com/watch?v=WQR_VQnz7Yg

블로그

https://jojoldu.tistory.com/71

https://www.journaldev.com/2583/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations

THANKS FOR WATCHING

감사합니다