제04장

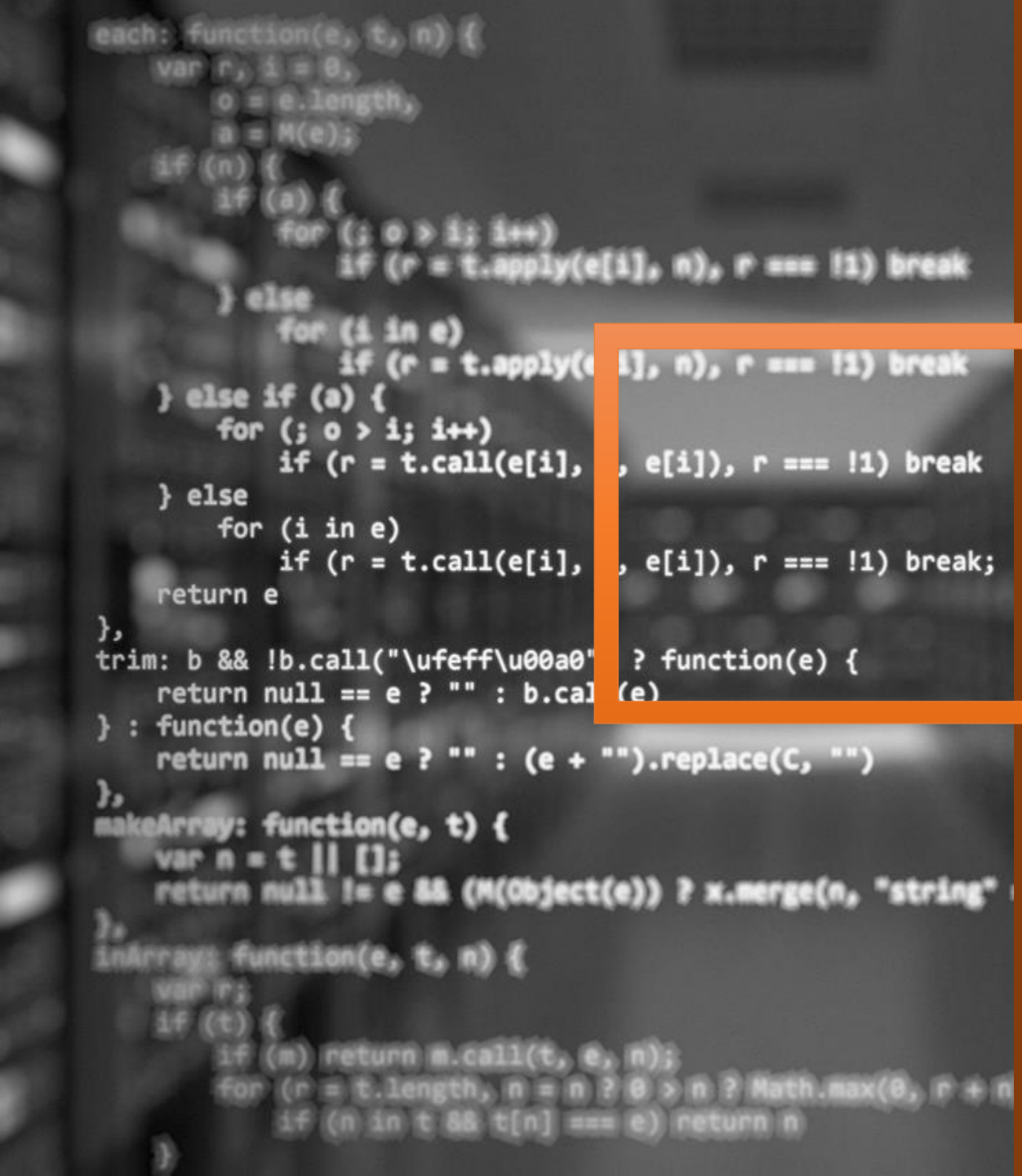
AOP

Spring

학습목표

1. AOP 개념에 대해서 알 수 있다.
2. AOP 설정에 대해서 알 수 있다.

목차



1. AOP 개념

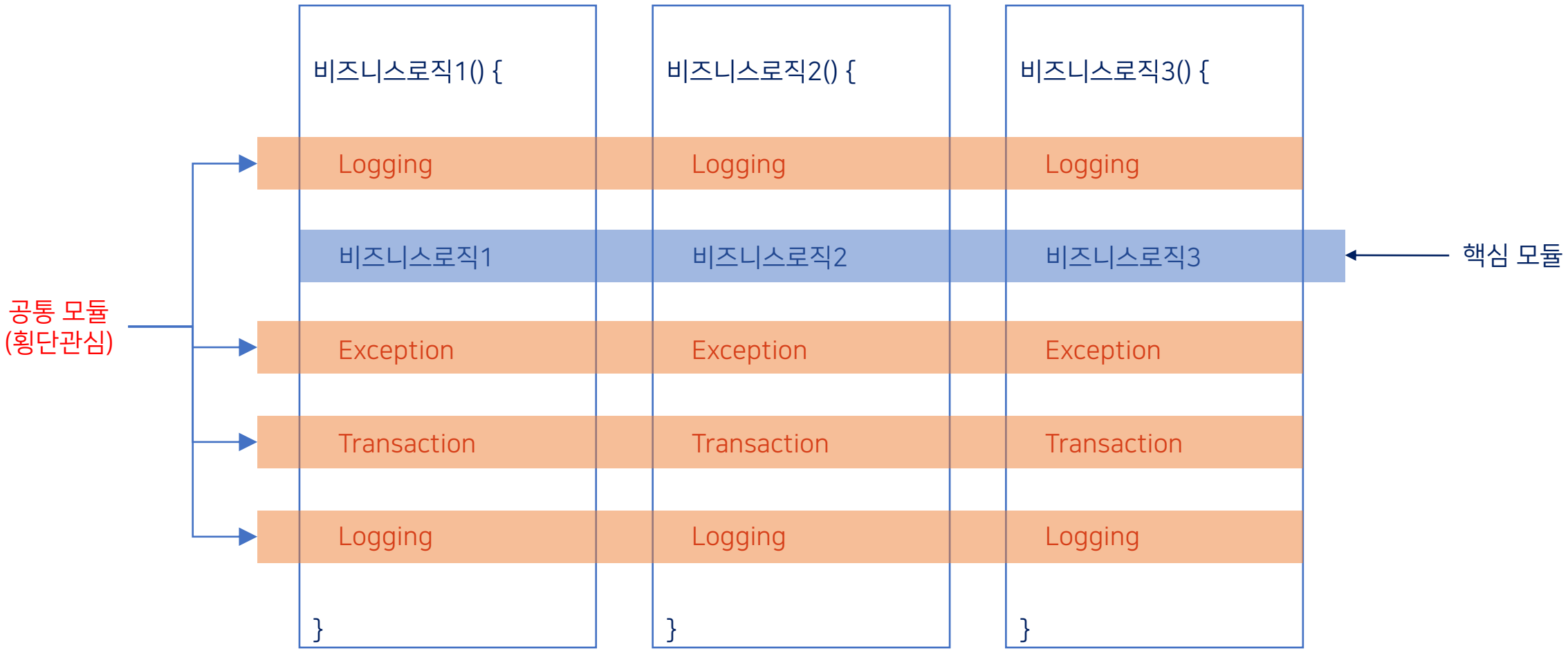
AOP

- Aspect Oriented Programming
  - 관점 지향 프로그래밍
  - 문제를 바라보는 관점을 기준으로 프로그래밍하는 패러다임
  - 각 문제를 해결하는 핵심로직과 모든 문제에 적용되는 공통로직을 기준으로 프로그래밍을 분리하는 것

- AOP 도입 이유
  - 애플리케이션 개발에 필요한 다양한 공통 모듈(로그, 보안, 트랜잭션 처리 등)이 사용됨
  - 대부분의 비즈니스로직에서 필요하기 때문에 반복적으로 유사한 코드를 매번 작성해야 함
  - 공통 모듈을 비즈니스로직마다 작성해야 하는 문제를 개선하기 위해서 AOP가 도입

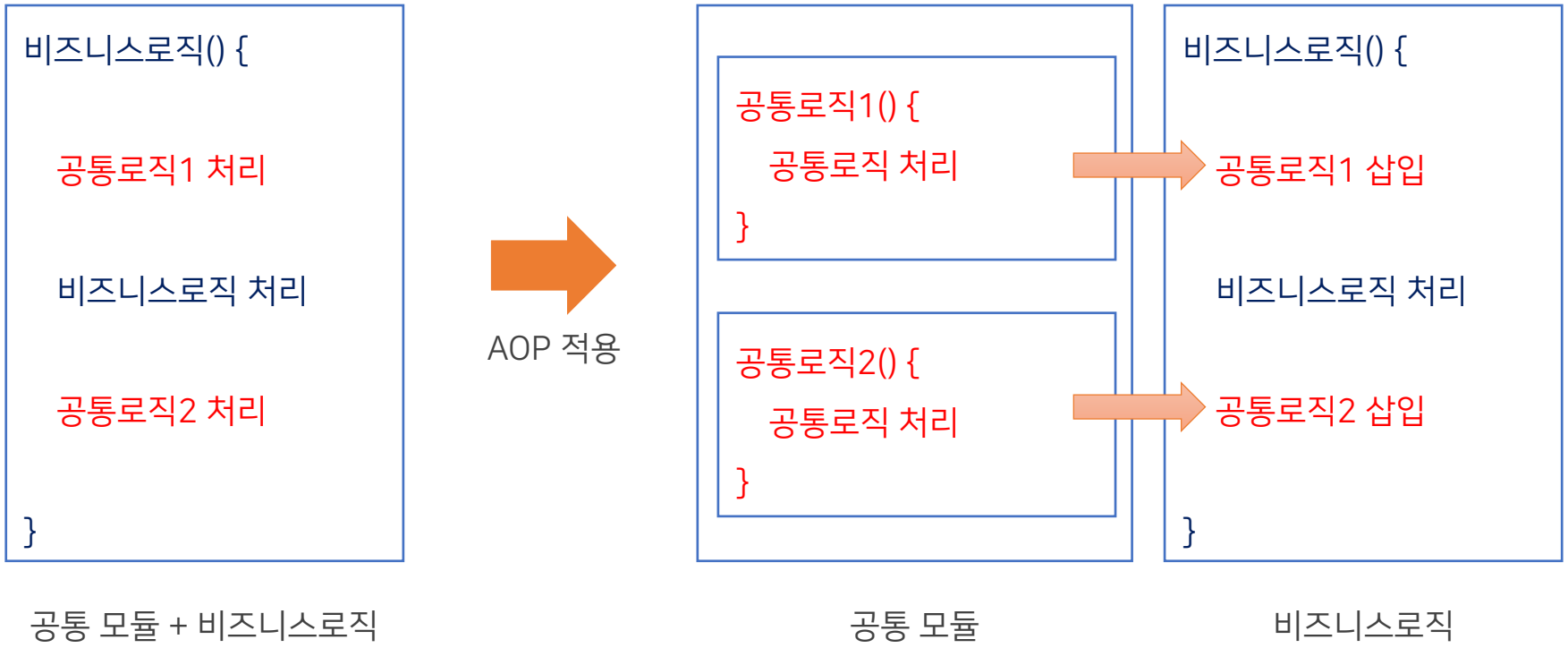
AOP

- AOP 개념


비즈니스로직1() {
비즈니스로직2() {
비즈니스로직3() {
Logging
Logging
Logging
비즈니스로직1
비즈니스로직2
비즈니스로직3
핵심 모듈
공통 모듈
(횡단관심)
Exception
Exception
Exception
Transaction
Transaction
Transaction
Logging
Logging
Logging
}
}
}

AOP 처리 방식

- AOP 처리 방식
  - 비즈니스로직 작성 시 공통 모듈을 직접 호출하지 않음
  - 공통 모듈을 만들어 두고 AOP 기능을 적용하면 자동으로 비즈니스 로직에 공통로직이 삽입되는 방식으로 동작함

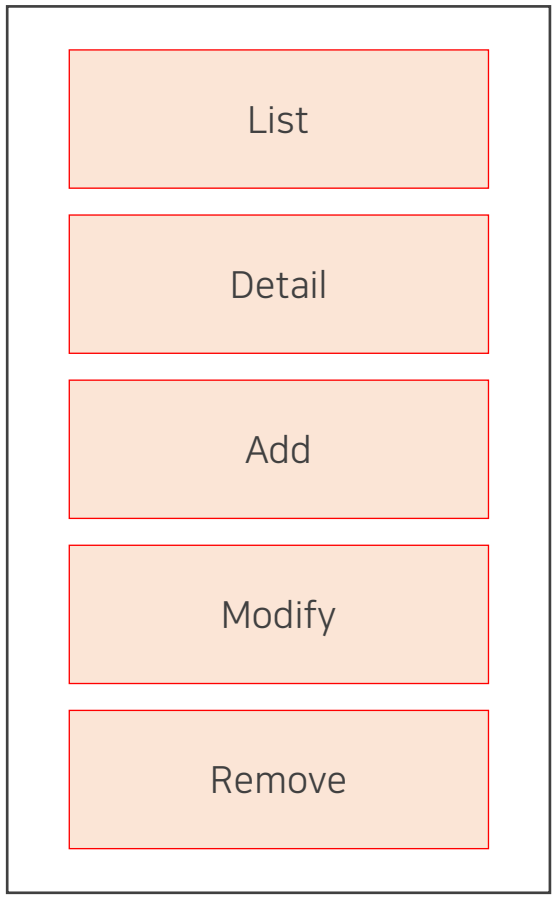


AOP 용어

- JoinPoint (조인포인트)
  - 클라이언트가 호출하는 모든 비즈니스 메소드를 의미함
  - JoinPoint 중 일부가 PointCut이 되기 때문에 PointCut 대상 또는 PointCut 후보라고도 함
- PointCut (포인트컷)
  - JoinPoint의 부분집합
  - 실제 Advice가 적용되는 JoinPoint를 의미함
  - 정규 표현식(Regular Expression) 또는 AspectJ 문법으로 PointCut을 정의함
- Advice (어드바이스)
  - 언제 공통 모듈을 비즈니스 메소드에 적용할 것인지 정의하는 것을 의미함
  - 비즈니스 메소드 실행 전, 후, 주변, 에러 발생시 넣을 수 있음

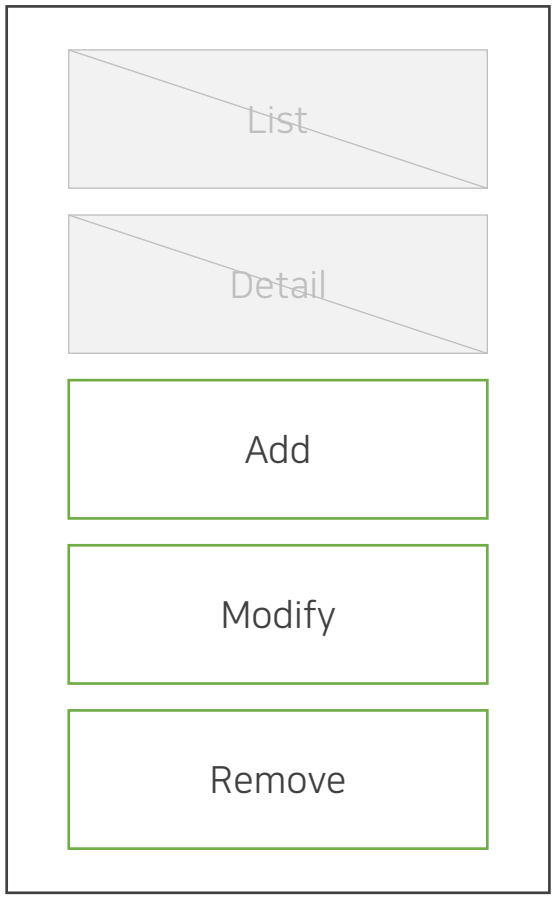
AOP 용어

전체 5개의 비즈니스 메소드가 있다.



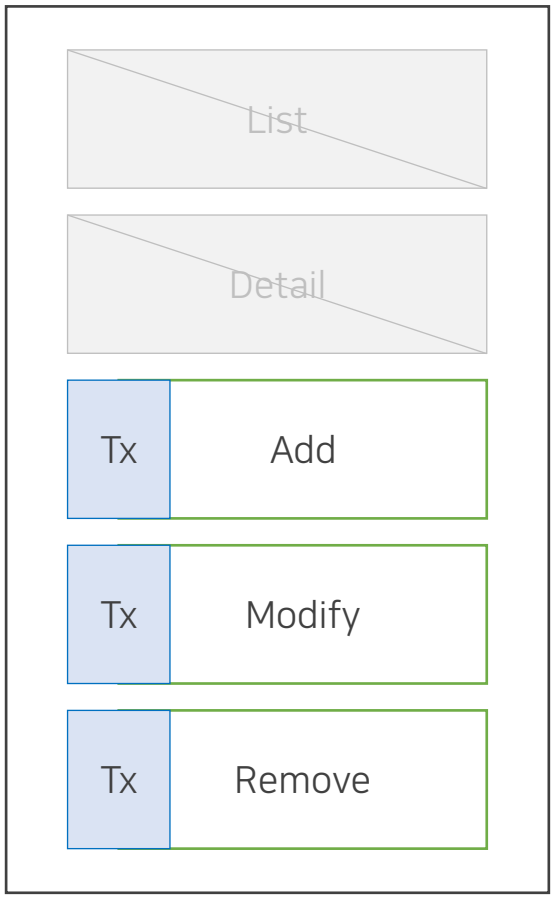
JoinPoint

이 중 3개의 비즈니스 메소드에
공통 모듈을 적용하고자 한다.

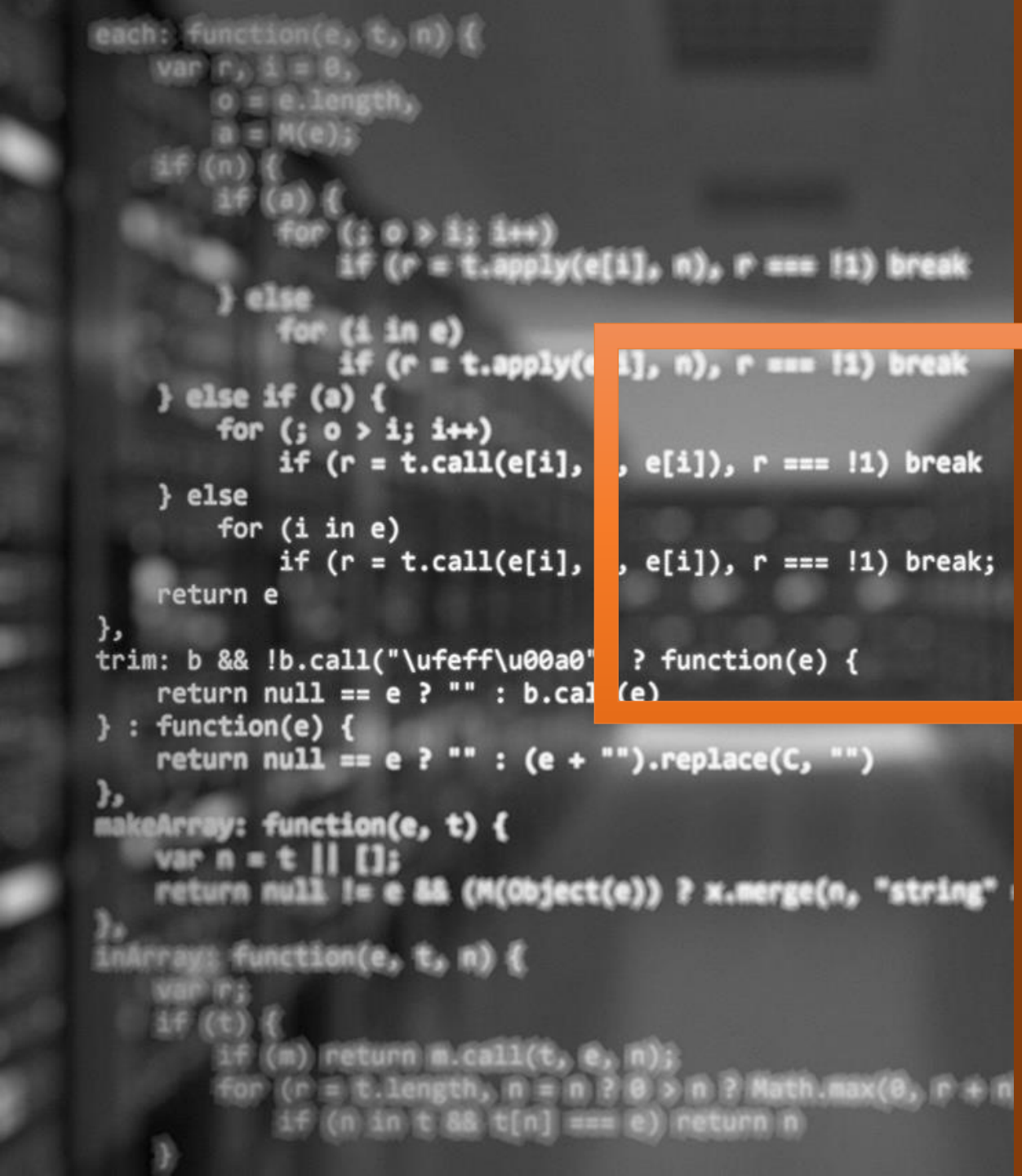


PointCut

적용할 공통 모듈은
트랜잭션처리이다.



Advice



2. AOP 설정

디펜던시

- AspectJ Runtime Library

```
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjrt</artifactId>
  <version>${org.aspectj-version}</version>
</dependency>
```

- AsectJ Weaving

```
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>${org.aspectj-version}</version>
  <scope>runtime</scope>
</dependency>
```

Weaving

- Weaving
  - 위빙
  - Java 클래스를 로딩하는동안 Advice를 불러올 수 있도록 하는 것
  - Advice를 비즈니스 메소드에 적용하는 것을 의미함

- Weaving 방식
  - 컴파일 타임에 Weaving 하기
    : 컴파일할 때 공통 코드가 삽입되고 AOP가 적용된 클래스 파일(.class)을 생성함
  - 클래스 로딩타임에 Weaving 하기
    : 원본 클래스를 수정하지 않고, 클래스를 메모리에 로딩할 때 JVM이 AOP를 적용한 바이트 코드를 사용함
  - 런타임에 Weaving 하기
    : 스프링이 지원하는 방식으로 Proxy를 사용함

JoinPoint

- JoinPoint 인터페이스
  - JoinPoint는 Spring AOP 또는 AspectJ에서 AOP가 적용되는 지점을 의미함
  - AspectJ에서는 이 지점을 JoinPoint라는 인터페이스로 나타낼 수 있음
  - 모든 어드바이스(Advice)는 JoinPoint 인터페이스 타입의 파라미터를 첫 번째 매개변수로 선언하고 사용할 수 있음 (org.aspectj.lang.JoinPoint)
  - Around 어드바이스는 반드시 ProceedingJoinPoint 타입의 파라미터를 필수로 선언해야 함

```
@Around("execution(* com.spring.app.controller.*.*(..))")
public Object logging(ProceedingJoinPoint joinPoint) throws Throwable {

    return null;
}
```

Around Advice 예시

JoinPoint 인터페이스 주요 메소드

- JoinPoint 인터페이스 주요 메소드

반환타입	이름	의미
Object[]	getArgs()	메소드로 전달되는 argument 반환
Signature	getSignature()	어드바이스 되는 메소드의 시그니처 반환
Object	getTarget()	대상 객체 반환
String	toString()	어드바이스 되는 메소드의 문자열 설명 반환

PointCut 표현식

PointCut 표현식

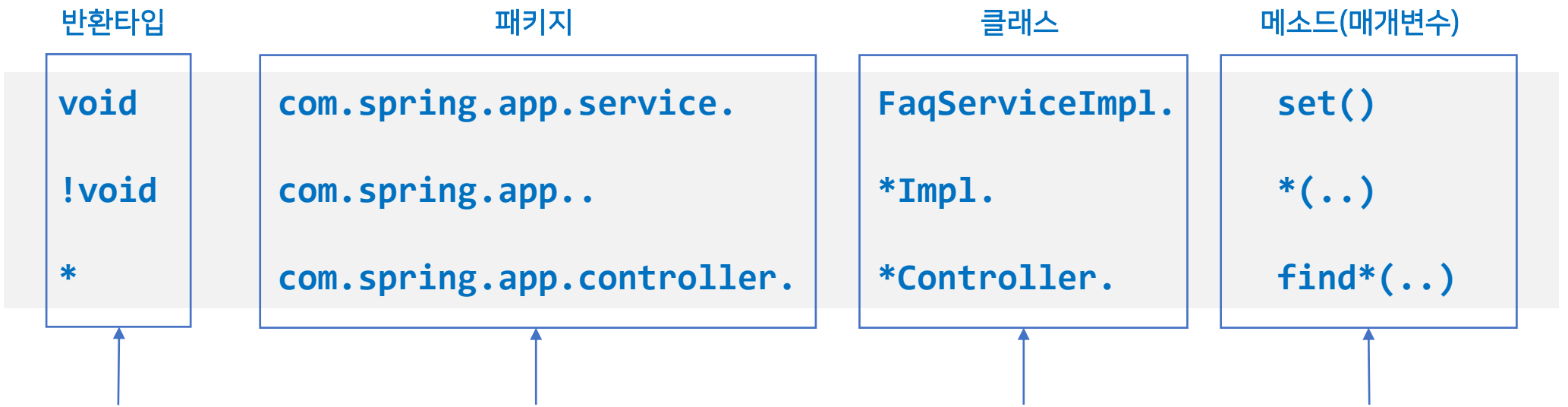
- AspectJ 포인트컷 표현식은 execution() 포인트컷 지시자를 이용하여 작성
- 문법 구조
  - execution([접근제한자] [반환타입] 패키지.클래스.메소드(매개변수))

execution() 지시자

구분	종류	의미
접근제한자	private, public	메소드의 접근제한자를 의미함. 생략가능
반환타입	*, void, !void	메소드의 반환타입을 의미함
메소드	메소드명, *	PointCut되는 메소드를 의미함
매개변수	*, ..	..는 개수 상관없이 모든 매개변수, *는 1개의 모든 매개변수

PointCut 표현식

- PointCut 표현식 작성 예시



Advice

Advice

- 어드바이스
- 언제 공통 관심 기능(공통 모듈)을 핵심 로직(비즈니스 로직)에 적용할 것인지를 정의함
- 언제 적용할 것인지에 따라 5가지 종류로 나눌 수 있음

Advice 종류

종류	동작시점	반환타입	매개변수타입
@Before	메소드 실행 이전	void	JointPoint
@After	메소드 실행 이후	void	JointPoint
@Around	메소드 실행 이전/이후	Object	ProceedingJoinPoint
@AfterReturning	예외 없는 메소드 실행 이후	void	JointPoint
@AfterThrowing	예외 발생한 메소드 실행 이후	void	JointPoint