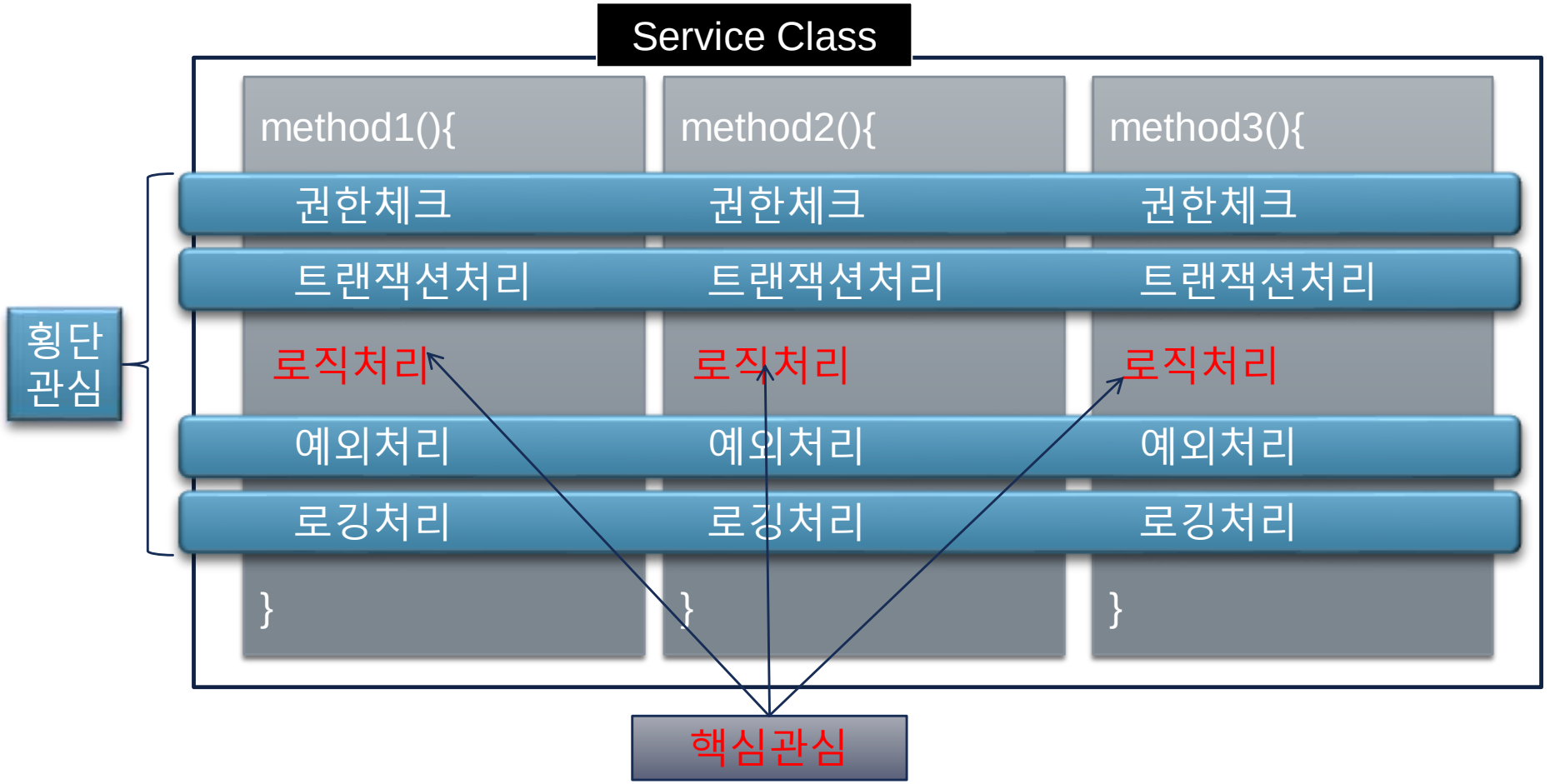
7.AOP

1. AOP 개요
2. AOP 용어
3. AOP 적용
4. 포언트 컷 표현식
5. 어드바이스 동작 시점
6. 어노테이션을 이용한 AOP 설정
7. 트랜잭션

1.AOP 개요

- Separation Of Concerns(관심분리)



1.AOP 개요

- 관심분리
  - AOP는 애플리케이션에서의 관심사의 분리(기능의 분리) 즉, 핵심적인 기능에서 부가적인 기능을 분리한다. 분리한 부가기능을 Aspect라는 독특한 모듈형태로 만들어서 설계하고 개발하는 방법
  - OOP를 적용하여도 핵심기능에서 부가기능을 쉽게 분리된 모듈로 작성하기 어려운 문제점을 AOP가 해결

- 핵심기능과 부가기능
  - 업무 로직을 포함하는 기능을 **핵심 관심(Core Concerns)**이라 하고 핵심기능을 도와주는 부가적인 기능(로깅, 보안 등)을 횡단관심(Cross-cutting Conserns)이라고 부른다.

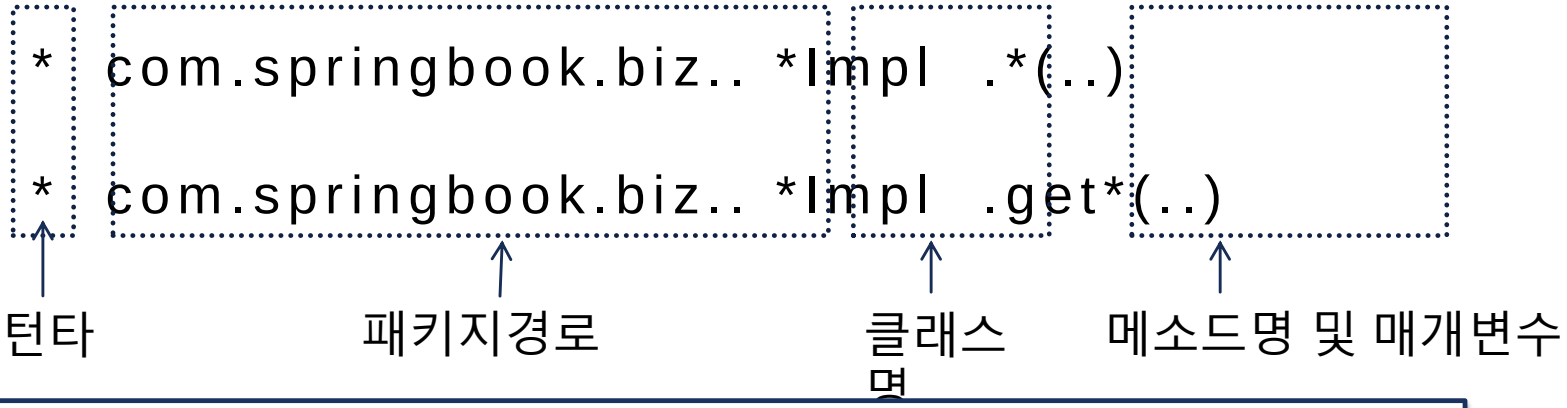
2.AOP 용어

- 어드바이스(Advice)
  - 횡단 관심에 해당하는 공통기능의 코드.
  - 독립된 클래스의 메소드로 작성

- Aspect (=Advisor)
  - Aspect는 포인트컷과 어드바이스의의 결합
  - 어떤 포인트컷 메소드에 대해서 어떤 어드바이스 메소드를 실행할 지 결정

AOP 용어

- 조인포인트(Joinpoint)
  - 클라이언트가 호출하는 모든 비즈니스 메소드
  - BoardServiceImpl, UserServiceImpl
- 포인트컷(Pointcut)
  - 필터링된 조인포인트
  - 특정 메서드에서만 공통기능을 수행하도록 메소드 필터링




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

2.AOP 용어

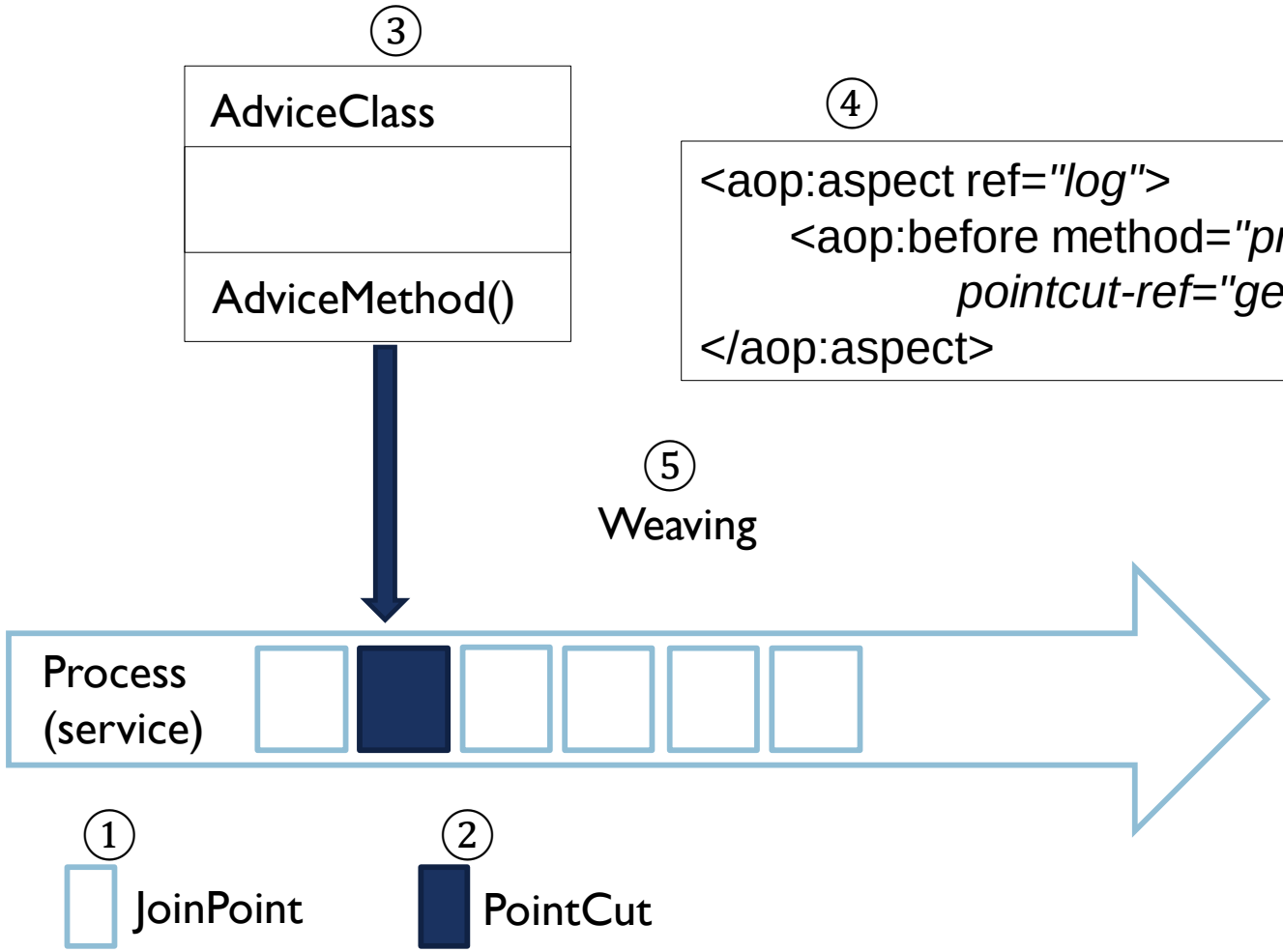
- 위빙(Weaving)
  - 포인트컷으로 지정한 핵심관심 메소드가 호출될 때, 어드바이스에 해당하는 횡단 관심 메소드가 삽입되는 과정을 의미
  - 위빙을 통해서 비즈니스 메소드를 수정하지 않고도 횡단관심에 해당하는 기능을 추가하거나 변경가능 함
  - 동작시점을 지정
    - before : 포인트컷 메소드 실행되기 전에
    - after : 포인트컷 메소드 실행 후 예외발생여부 관계없이 무조건 실행
    - after-returning : 포인트컷 메소드가 정상적으로 실행 후 리턴 시점
    - after-throwing : 포인트컷 메소드가 실행되다가 예외 발생 시점
    - around : 메소드 실행 전과 실행 후에 모두 동작

3. 1 라이브러리 설치

```
<properties>
   <org.aspectj-version>1.8.3</org.aspectj-version>
</properties>


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

3. 2 AOP 적용과정


③
AdviceClass
AdviceMethod()
④
<aop:aspect ref="log">
<aop:before method="printLogging"
pointcut-ref="getPointcut"/>
</aop:aspect>
⑤
Weaving
Process
(service)
①
JoinPoint
②
PointCut

3.3 XML 기반 AOP 설정

```
<aop:config  proxy-target-class="true">>
  <aop:pointcut  expression="execution(* com.yedam.app..*Impl.*(..))"  id="allpointcut"/>
  <aop:aspect ref="log4j">
      <aop:before method="printLogging" pointcut-ref="getPointcut"/>
  </aop:aspect>
</aop:config>
```

4. 포인트컷 표현식

- 리턴타입 지정

표현식	설명
*	모든 리턴타입 허용
void	리턴타입이 void인 메서드 선택
!void	리턴타입이 void가 아닌 메서드 선택

- 패키지 지정

표현식	설명
com.springbook.biz	정확하게 일치하는 패키지만 선택
com.springbook.biz..	com.springbook.biz 패키지로 시작하는 모든 패키지 선택
com.springbook..impl	com.springbook 패키지로 시작하면서 마지막 패키지 이름이 impl로 끝나는 패키지 선택

4. 포인트컷 표현식

- 클래스 지정

표현식	설명
BoardServiceImpl	정확하게 BoardServiceImpl 클래스만 선택
*Impl	클래스 이름이 Impl로 끝나는 클래스만 선택
BoardService+	클래스 이름 뒤에 '+'가 붙으면 해당 클래스로부터 파생된 모든 자식 클래스 선택. 인터페이스 뒤에 '+'가 붙으면 해당 인터페이스를 구현한 모든 클래스 선택

- 메소드 지정

표현식	설명
*()	가장 기본 설정으로 모든 메서드 선택
get*(..)	메소드 이름이 get으로 시작되는 모든 메소드 선택

4. 포인트컷 표현식

- 클래스 지정

표현식	설명
(..)	가장 기본 설정으로서 매개변수의 개수와 타입에 제약이 없음을 의미
(*)	반드시 1개의 매개변수를 가지는 메서드만 선택
(com.springbook.userUerVO)	매개변수는 UserVO를 가지는 메서드만 선택, 이때 클래스의 패키지 경로가 반드시 포함되어야 함.
(!com.springbook.user.UserVO)	매개변수는 UserVO를 가지지 않는 메서드만 선택
(Integer, ..)	한 개 이상의 매개변수를 가지되, 첫 번째 매개변수의 타입이 Integer 인 메서드만 선택
(Integer, *)	반드시 두 개의 매개변수를 가지되, 첫 번째 매개변수의 타입인 Integer인 메소드만 선택

5. JoinPoint

- JoinPoint 메서드

메소드	설명
Signature GetSignature()	클라이언트가 호출한 메소드의 시그니처(리턴타입, 이름, 매개변수) 정보가 저장된 Signature 객체 리턴
Object getTarget()	클라이언트가 호출한 비즈니스 메소드를 포함하는 비즈니스 객체 리턴
Objet[] getArgs()	클라이언트가 메소르를 호출할 때 넘겨준 인자 목록을 Object 배열로 리턴

- Signature 메서드

메소드명	설명
String getName()	클라이언트가 호출한 메소드 이름 리턴
String toLongString()	클라이언트가 호출한 메소드의 리턴타입, 이름 매개변수 패키지 경로까지 포함하여 리턴
String toShortString()	클라이언트가 호출한 메소드 시크니처를 축약한 문자열로 리턴

6. 어노테이션 기반 AOP

- 어노테이션 사용을 위한 스프링 설정
  - <aop:aspectj-autoproxy>

- 애스펙트 설정
  - @Aspect

- 포인트 컷 설정

```
@Pointcut("execution(* com..* *(..))")
public void allpointcut() { }
```

- 어드바이스 설정
  - @Before(“allpointcut”)

6. 어노테이션 기반 AOP

- 어노테이션 사용을 위한 스프링 설정

1. xml 기반 설정 : Servlet-context.xml 에 설정

```
<aop:aspectj-autoproxy proxy-target-class="true“/>
<beans:bean id="logAdvice" class="com.dbal.app.common.aop.LogAdvice"></beans:bean>
```

2. 자바 기반 설정

```
@Configuration
@EnableAspectJAutoProxy(proxyTargetClass = true)
public class AopConfig {
	@Bean
	public BeforeAdvice boforeAdvice() {
		return new BeforeAdvice();
	}
}
```

6. 어노테이션 기반 AOP

- Aspect 클래스 = advice + pointcut

```
@Aspect
@Component
public class BeforeAdvice{

  @Pointcut("execution(* com.yedam..*Impl.*(..))")
  public void allpointcut(){}

  @Before("allpointcut()")
  public void beforeLog(JoinPoint jp){
    String methodName = jp.getSignature().getName();
    System.out.println("[사전처리] beforeLog" + methodName);
  }
}
```

Aspect 설정

컨테이너에 빈 등록

pointcut 설정

weaving 설정

7. 1 트랜잭션 JDBC

```
try{
	conn = db.connect();
	conn.setAutoCommit(false);

	String sql1 = "UPDATE table1 SET data1 = ? WHERE data1 = 100";
	String sql2 = "UPDATE table2 SET data1 = ? WHERE data1 = 200";

	pstmt1 = conn.prepareStatement(sql1);
	pstmt1.setInt(1,a);
	pstmt1.executeUpdate();

	pstmt2 = conn.prepareStatement(sql2);
	pstmt2.setInt(1,a);
	pstmt2.executeUpdate();
	conn.commit();

} catch (Exception se) {
	conn.rollback();
	se.printStackTrace();
} finally {
	db.close(conn);
}
```

7.2 트랜잭션 AOP 설정

- 트랜잭션 메니저 등록

```
<!-- Transaction Manager 설정 -->
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<property name="dataSource" ref="dataSource" />
</bean>
```

- @Transactional 어노테이션 지정

```
@Service
public class SampleTxServiceImpl implements SampleTxService {

 @Autowired private Sample1Mapper mapper1;
 @Autowired private Sample2Mapper mapper2;

 @Transactional
 public void addData(String value) {
  mapper1.insertCol1(value);
  mapper2.insertCol2(value);
 }
```

@Transactional 적용순서
1. 메서드의 설정
2. 클래스의 설정
3. 인터페이스의 설정

7.2 트랜잭션 AOP 설정

- 테스트 코드

```
public class SamepleTxServiceTests {

 @Autowired private SamepleTxService service;

 @Test
 public void testLong(){
   String str="안녕하세요!!!";
   service.addData(str);
 }
```

7.2 트랜잭션 AOP 설정

- 어노테이션 없이 포인트컷으로 트랜잭션 적용

```
<!-- Transaction Advice 설정 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
 <tx:attributes>
  <tx:method name="get*" read-only="true" />
  <tx:method name="*" />
 </tx:attributes>
</tx:advice>

<!-- Transaction AOP 설정 -->
<aop:config>
 <aop:pointcut expression="execution(public * com..*Impl.*(..))" id="txPointCut" />
 <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointCut" />
</aop:config>
```

@Transactional 속성

- 전파(Propagation) 속성
- 격리(Isolation) 레벨
- Read-only 속성
- rollback-for-예외

참고사이트

- https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#aop