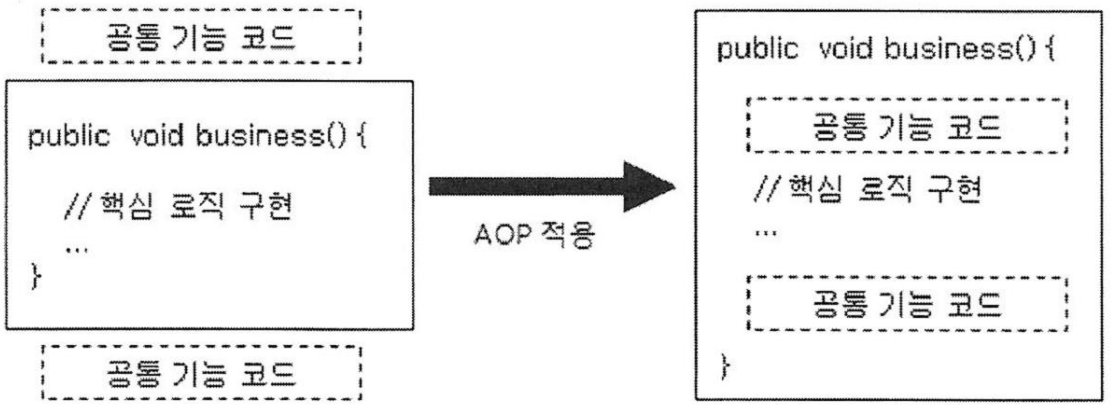
스프링 프레임워크
- AOP: Aspect Oriented Programming -

AOP 소개

❖ AOP

- ○ Aspect Oriented Programming
  - ▪ 문제를 바라보는 관점을 기준으로 프래밍하는 기법
    - 문제를 해결하기 위한 핵심 관심 사항, 전체에 적용되는 공통 관심 사항을 기준으로 프로그래밍
    - 공통 모듈을 여러 코드에 쉽게 적용 가능



- 공통 관심 사항을 구현한 코드를 핵심 로직을 구현한 코드 안에 삽입 (Weaving)

AOP 소개

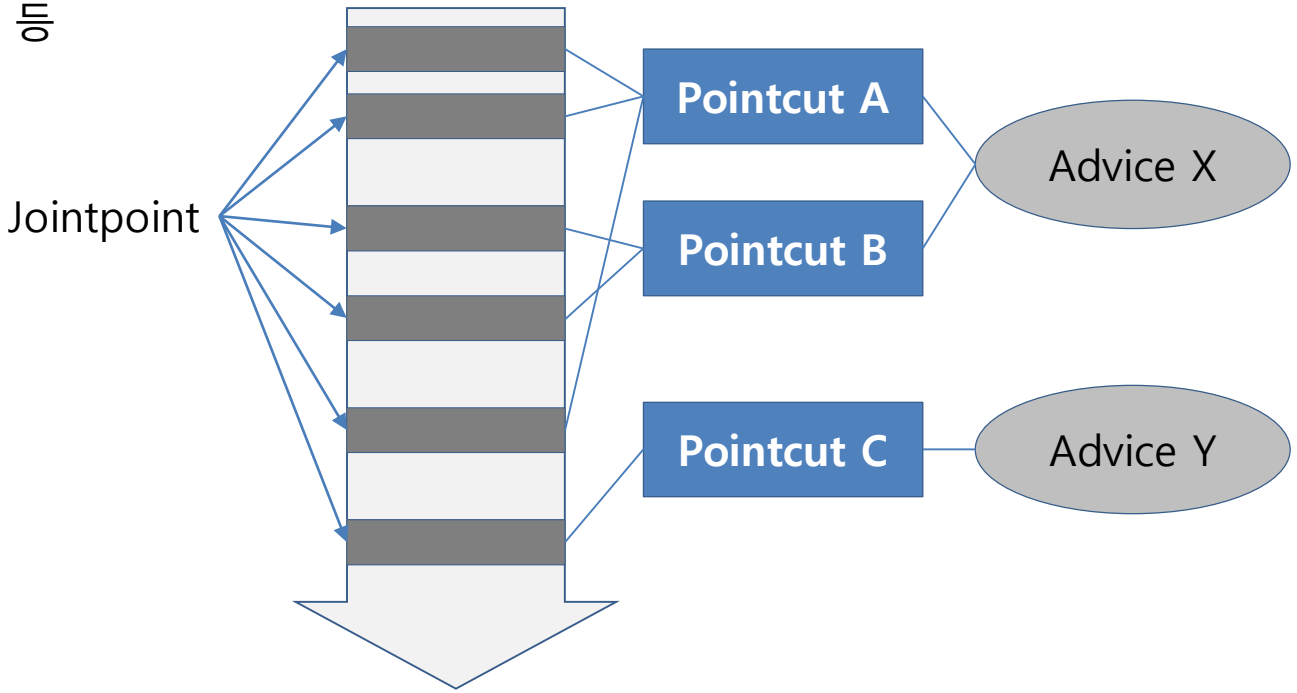
❖ 공통관심사(Common Concern)

```
public class ArticleService {
    public List<Article> getList(int page) {
        SqlSession session = SqlSessionProvider.getSqlSession();
        ...
        session.close();
        return list;
    }

    public List<Article> getListByUserId(HashMap map) {
        SqlSession session = SqlSessionProvider.getSqlSession();
        ...
        session.close();
        return list;
    }
    public Article getArticle(Article article) {
        SqlSession session = SqlSessionProvider.getSqlSession();
        ...
        session.close();
        return article;
    }
```

AOP 소개

❖ AOP 용어

- Advice : 언제 공통 관심 기능을 핵심 로직에 적용할 지 정의
- Jointpoint : 모듈이 삽입되어 동작할수 있는 실행 가능한 특정 위치 (메소드호출, 리턴,필드 액세스,인트턴스 생성, 예외 처리)
- **Pointcut** : 어떤 클래스의 어느 조인트 포인트를 사용할 것인지 결정하는 선택 기능
- Weaving : Advice를 핵심 로직 코드에 적용하는 것(삽입)
- **Aspect**
  - 여러 객체에 공통으로 적용되는 공통 관심 사항
  - 트랜잭션, 보안 등



AOP 소개

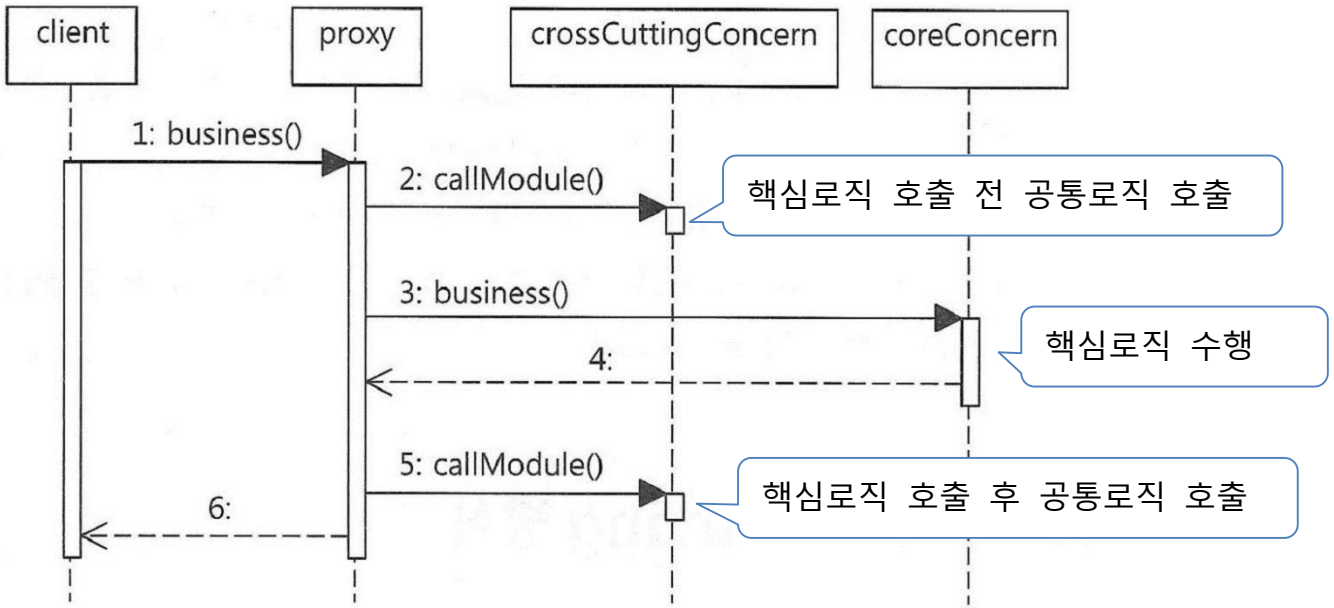
❖ 세 가지 Weaving 방식

- ○ Advice를 Weaving하는 방식
  - ▪ 컴파일 시에
    - - AspectJ에 사용하는 방식
    - - 핵심 로직을 구현한 자바 소스 코드를 컴파일 할 때에 알맞은 위치에 공통로직을 삽입
    - - AOP가 적용된 클래스 파일이 생성
  - ▪ 클래스 로딩 시에
    - - JVM이 클래스를 로딩할 때 클래스 정보를 변경할 수 있는 에이전트 제공
    - - 에이전트가 로딩한 클래스의 바이트에 공통로직을 삽입하여 새로운 바이트 코드 생성
  - ▪ 런타임 시에
    - - 소스 코드나 바이트 코드를 변경하지 않고 프록시를 이용하여 AOP를 적용
    - - 핵심 로직을 구현한 객체에 직접 접근하지 않고 프록시를 거쳐 접근

AOP 소개

❖ 세 가지 Weaving 방식

- ㅇ 프록시 기반의 AOP 적용 과정



❖ 스프링에서의 AOP

- 프록시 기반의 AOP 지원
  - 메서드 호출 Jointpoint 만 지원
  - 자바 기반
    - AspectJ는 자체 문법을 제공
    - 스프링은 자바 언어만 사용

- 스프링에서의 AOP 구현 방법
  - XML 스키마 기반의 POJO 클래스를 이용한 AOP 구현
  - AspectJ 5/6에서 정의한 @Aspect 어노테이션 기반의 AOP 구현
  - 스프링 API를 이용한 AOP 구현

❖ 스프링에서의 AOP

- ㅇ 메서드 실행 시간 측정
  - AOP를 사용하지 않는 경우

```
public void someMethod() {
    StopWatch stopWatch = new StopWatch();
    stopWatch.start();

    executeLogic();

    stopWatch.stop();
    long executionTime = stopWatch.getTotalTimeMillis();
}
```

- 실행 시간을 구해야 할 메서드가 많다면
- 조건에 따라서 실행 시간을 구해야 할 메서드를 선택해야 한다면

→ 요구가 변경될 때 마다 많은 코드를 변경해야 함

❖ AspectJ 라이브러리 추가

- mvnrepository.com에서 AspectJ Runtime, AspectJ Weaver 검색 (1.8.4 버전)
  - pom.xml에 추가

```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.8.4</version>
</dependency>

<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.4</version>
</dependency>
```

- LoggingAspect.java

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.aspectj.lang.ProceedingJoinPoint;
import org.springframework.util.StopWatch;

public class LoggingAspect {
    private Log log = LogFactory.getLog(getClass());

    public Object logging(ProceedingJoinPoint joinPoint)
                                            throws Throwable {
        log.info("기록 시작");
        StopWatch stopWatch = new StopWatch();
        try {
            stopWatch.start();
            Object retValue = joinPoint.proceed();
            return retValue;
        } catch (Throwable e) {
            throw e;
        }
```

- LoggingAspect.java

```
        finally {
           stopWatch.stop();
           log.info("기록 종료");
           log.info(joinPoint.getSignature().getName()
                 + "메서드 실행 시간 : "
                 + stopWatch.getTotalTimeMillis());
        }
    }
}
```

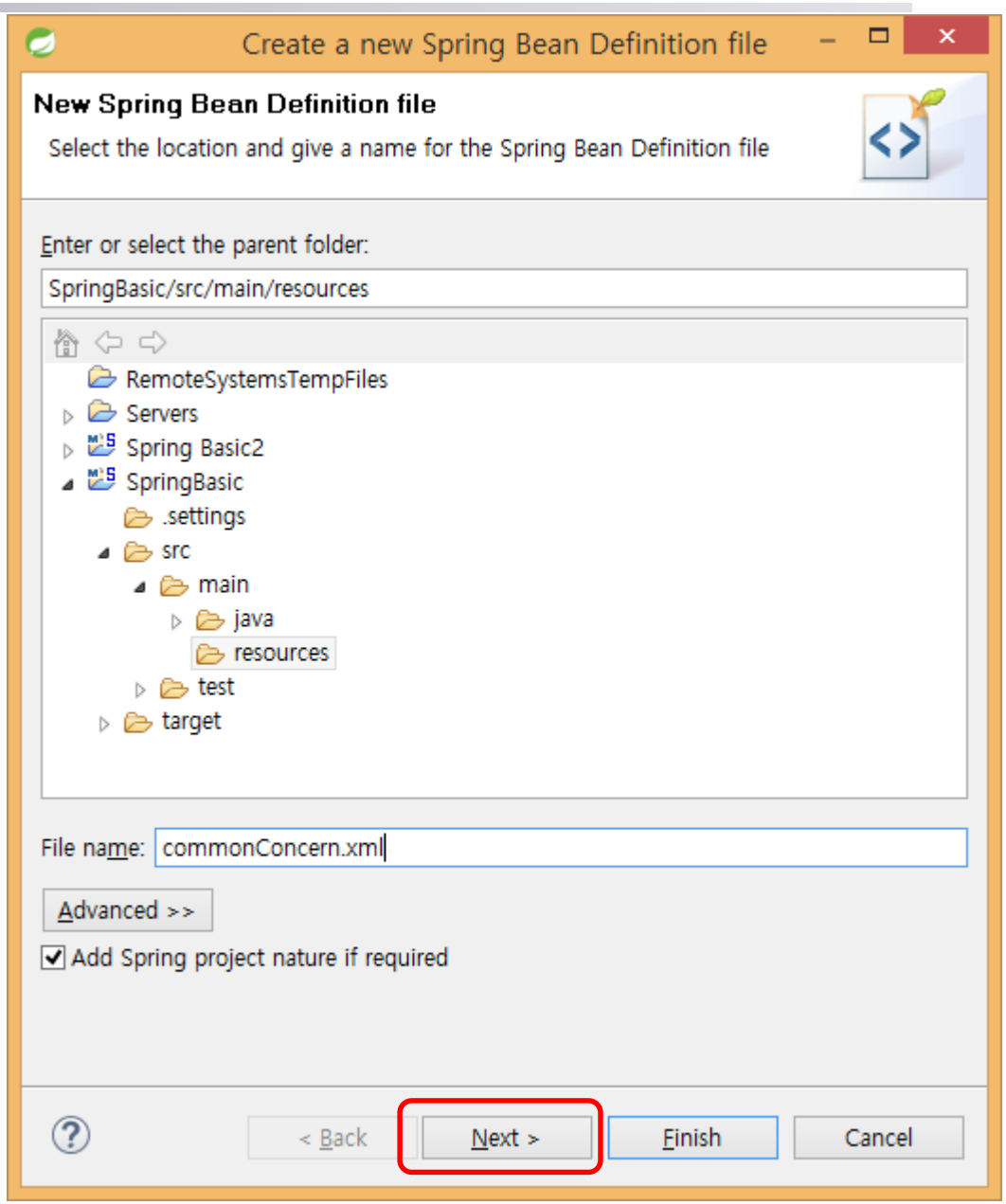
❖ 스프링에서의 AOP

- AOP를 이용한 메서드 실행 시간 측정
  - LoggingAspect 클래스는 핵심 기능을 구현한 클래스나 인터페이스에 전혀 의존하지 않음
  - 오직 ProceedingJoinPoint 에만 의존
  - 코드 수정없이 LoggingAspect를 여러 클래스에 적용 가능

스프링과 AOP

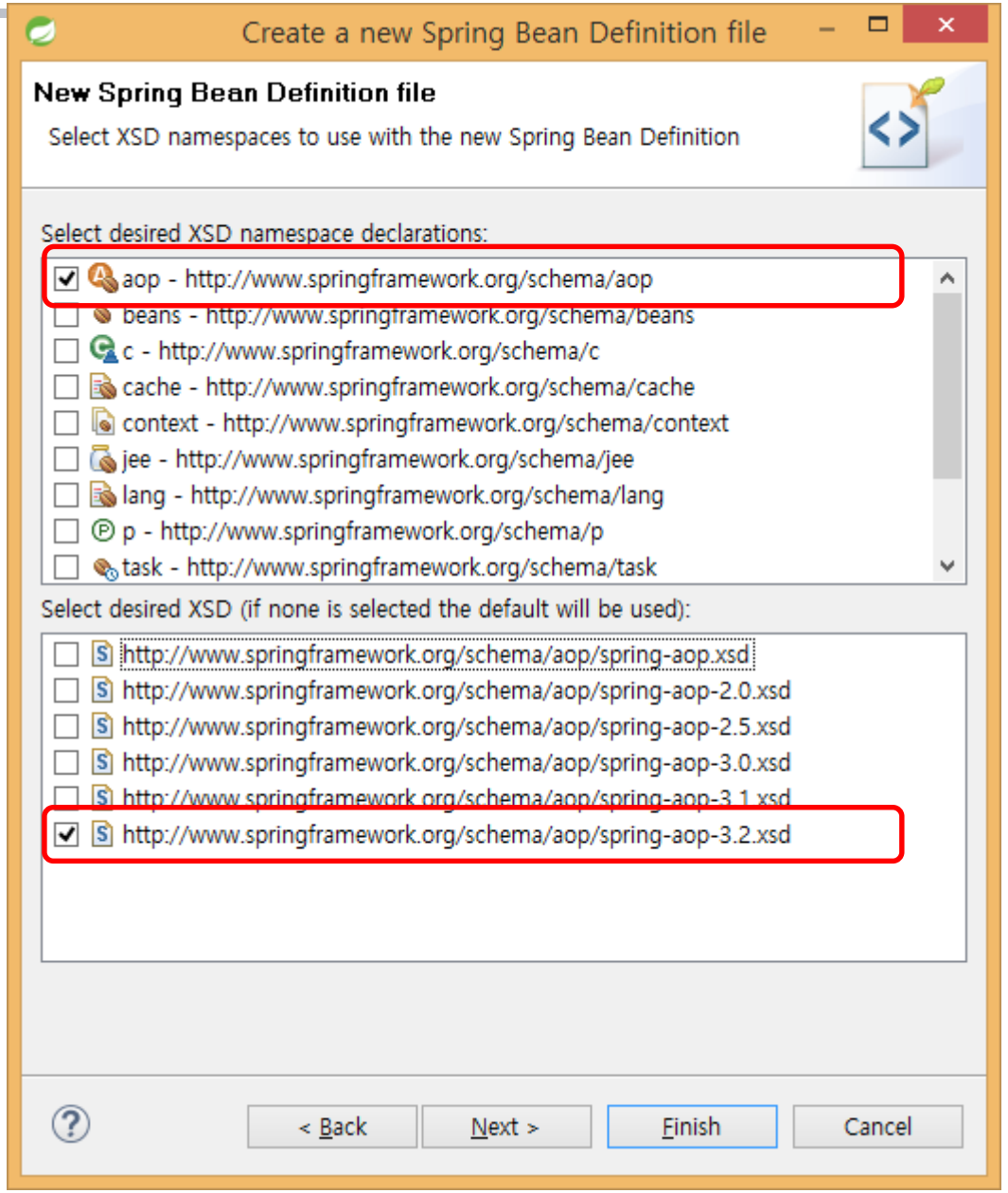
❖ 스프링에서의 AOP

- ○ Aop를 위한 스프링 설정 파일
  - ▪ 설정 : commonConcern.xml



❖ 스프링에서의 AOP

- ㅇ Aop를 위한 스프링 설정 파일
  - ▪ 설정 : commonConcern.xml



❖ 스프링에서의 AOP

- ○ Aop를 위한 스프링 설정 파일
  - ▪ 설정 : commonConcern.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">


</beans>
```

❖ 스프링에서의 AOP

- ㅇ Aop를 위한 스프링 설정 파일
  - 설정 : commonConcern.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

   <bean id="logging" class="com.lecture.spring.basic.LoggingAspect" />

    <aop:config>
        <aop:pointcut id="servicePointcut"
            expression="execution(* *..*Service.*(..))" />

        <aop:aspect id="loggingAspect" ref="logging">
            <aop:around pointcut-ref="servicePointcut" method="logging" />
        </aop:aspect>
    </aop:config>
</beans>
```

❖ 스프링에서의 AOP

- AOP 설정
  - AOP XML 스키마 지정

```
<beans ...
       http://www.springframework.org/schema/aop
          :
       http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
```

❖ 스프링에서의 AOP

- AOP 설정
  - AOP 적용 대상 설정
    - 이름이 Service로 끝나는 인터페이스를 구현한 모든 클래스(*** *..*Service**)의 모든 메서드(**.*(..)**)에 LoggingAspect가 적용됨

```
<bean id="logging" class="com.lecture.spring.basic.LoggingAspect" />

 <aop:config>
     <aop:pointcut id="servicePointcut"
         expression="execution(* *..*Service.*(..))" />

     <aop:aspect id="loggingAspect" ref="logging">
         <aop:around pointcut-ref="servicePointcut" method="logging" />
     </aop:aspect>
 </aop:config>
```

❖ 스프링에서의 AOP

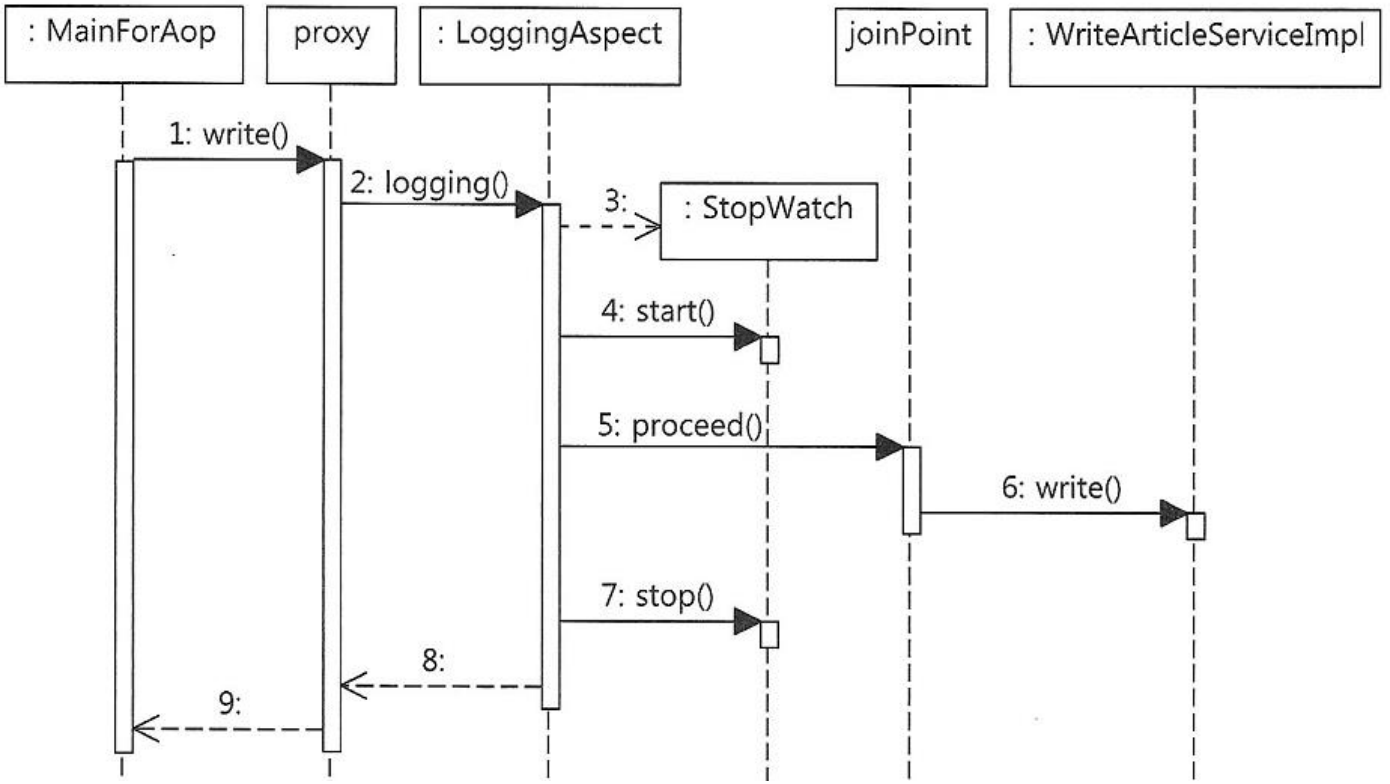
- AOP를 이용한 메서드 실행 시간 측정
  - AopTest.java
    - WriteArticleServiceImpl의 write 메서드 실행 시간 측정

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class AopTest {

   public static void main(String[] args) {
      String[] configLocations = new String[] { "applicationContext.xml",
            "commonConcern.xml" };

      ApplicationContext context = new ClassPathXmlApplicationContext(
                                                   configLocations);

      WriteArticleService articleService = (WriteArticleService)
                                 context.getBean("writeArticleService");
      articleService.write(new Article());
   }
}
```

복수개의 설정파일로 빈을 준비

❖ 스프링에서의 AOP

- LoggingAspect의 실행 순서



- 스프링 AOP
  - 핵심 로직 코드의 수정 없이 웹 어플리케이션의 보안, 로깅, 트랜잭션과 같은 공통 관심 사항을 AOP로 간단히 적용 가능