```java
package com.myspring.pro27.common.log;

import java.util.Arrays;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAdvice {
    private static final Logger logger = LoggerFactory.getLogger

    // target 메서드의 파라미터등 정보를 출력합니다.
    /*@Before("execution(* com.myspring.pro27.member.service.*.*
            + "execution(* com.myspring.pro27.member.dao.*.*(..)
    @Before("execution(* com.myspring.pro27.*.service.*.*(..)) o
            + "execution(* com.myspring.pro27.*.dao.*.*(..))")
    public void startLog(JoinPoint jp) {

        logger.info("-------------------------------------");
        logger.info("-------------------------------------");

        // 전달되는 모든 파라미터들을 Object의 배열로 가져옵니다.
        logger.info("1:" + Arrays.toString(jp.getArgs()));

        // 해당 Advice의 타입을 알아냅니다.
        logger.info("2:" + jp.getKind());

        // 실행하는 대상 객체의 메소드에 대한 정보를 알아낼 때 사용합니다.
        logger.info("3:" + jp.getSignature().getName());

        // target 객체를 알아낼 때 사용합니다.
        logger.info("4:" + jp.getTarget().toString());

```

```
42          // Advice를 행하는 객체를 알아낼 때 사용합니다.
43          logger.info("5:" + jp.getThis().toString());
44
45      }
46
47      /*@After("execution(* com.myspring.pro27.member.service.*.*(
48              + "execution(* com.myspring.pro27.member.dao.*.*(..)
49      @After("execution(* com.myspring.pro27.*.service.*.*(..)) or
50              + "execution(* com.myspring.pro27.*.dao.*.*(..))")
51      public void after(JoinPoint jp) {
52          logger.info("---------------------------------------");
53          logger.info("---------------------------------------");
54
55          // 전달되는 모든 파라미터들을 Object의 배열로 가져옵니다.
56          logger.info("1:" + Arrays.toString(jp.getArgs()));
57
58          // 해당 Advice의 타입을 알아냅니다.
59          logger.info("2:" + jp.getKind());
60
61          // 실행하는 대상 객체의 메소드에 대한 정보를 알아낼 때 사용합니다.
62          logger.info("3:" + jp.getSignature().getName());
63
64          // target 객체를 알아낼 때 사용합니다.
65          logger.info("4:" + jp.getTarget().toString());
66
67          // Advice를 행하는 객체를 알아낼 때 사용합니다
68          logger.info("5:" + jp.getThis().toString());
69
70      }
71
72
73      // target 메소드의 동작 시간을 측정합니다.
74      /*@Around("execution(* com.myspring.pro27.member.service.*.*
75              + "execution(* com.myspring.pro27.member.dao.*.*(..)
76      @Around("execution(* com.myspring.pro27.*.service.*.*(..)) c
77              + "execution(* com.myspring.pro27.*.dao.*.*(..))")
78      public Object timeLog(ProceedingJoinPoint pjp) throws Throwa
79          long startTime = System.currentTimeMillis();
80          logger.info(Arrays.toString(pjp.getArgs()));
81
82          // 실제 타겟을 실행하는 부분이다. 이 부분이 없으면 advice가 적용된
```

```
83          Object result = pjp.proceed(); // proceed는 Exception 보
84
85          long endTime = System.currentTimeMillis();
86          // target 메소드의 동작 시간을 출력한다.
87          logger.info(pjp.getSignature().getName() + " : " + (endT
88          logger.info("================================");
89
90          // Around를 사용할 경우 반드시 Object를 리턴해야 합니다.
91          return result;
92      }
93
94  }
95
```