1. AOP What?
  1)Application은 다양한 공통 기능을 필요로 한다.
  2)Logging과 같은 기본적인 기능에서부터 transaction이나 보안과 같은 기능에 이르기까지 application 전반에 걸쳐 적용되는 공통 기능이 존재한다.
  3)공통기능은 application의 핵심 business logic과는 구분되고 핵심기능을 도와 주는 부가적인 기능(logging, 보안 등)이다.
  4)핵심 business 기능과 구분하기 위해 공통 기능을 공통 관심 사항(Cross-cutting Concern)이라고 표현한다.
  5)핵심 logic(Biz logic을 포함하는 기능)을 핵심 관심 사항(Core Concern)이라고 표현한다.
    -핵심기능
      --계좌 이체, 대출 승인, 이자 계산
    -부가기능
      --Logging, 보안, transaction
    -그림 참조
    http://dev.anyframejava.org/docs/anyframe/plugin/foundation/4.6.0/reference/html/ch05.html

  6)객체지향의 기본 원칙을 적용하면서도 핵심기능에서 부가기능을 분리해서 모듈화하는 것은 매우 어렵다.
  7)Programming에서 공통적인 기능을 모든 module에 적용하기 위한 방법으로 상속이 있다.
  8)하지만 Java는 다중상속을 하지 않기 때문에 다양한 module에 상속 기법을 통한 공통 기능 부여에는 한계가 있다.
  9)AOP는 핵심기능과 공통 기능을 분리시켜놓고, 공통 기능을 필요로 하는 핵심 기능들에서 사용하는 방식이다.
  10)AOP(Aspect Oriented Programming)은 문제를 바라보는 관점(시점)을 기준으로 programming하는 기법이다.
  11)분리한 부가기능(공통 기능)을 Aspect라는 독특한 module 형태로 만들어서 설계하고 개발하는 방법이다.
  12)기본적인 개념은 공통 관심 사항을 구현한 code를 핵심 logic을 구현한 code 안에 삽입한다는 것이다.
  13)OOP를 적용하여도 핵심기능에서 부가기능을 쉽게 분리된 module로 작성하기 어려운 문제점을 AOP가 해결해 준다고 볼 수 있다.
  14)즉, AOP는 부가기능을 Aspect로 정의하여, 핵심기능에서 부가기능을 분리함으로써 핵심 기능을 설계하고 구현할 때 객체지향적인 가치를 지킬 수 있도록 도와주는 개념이다.
  15)AOP 기법에서는 핵심 logic을 구현한 code에서 공통 기능을 직접적으로 호출하지 않는다.
  16)핵심 logic을 구현한 code를 compile하거나, compile된 class를 loading하거나, loading한 class의 객체를 생성할 때 AOP가 적용되어 핵심 logic 구현 code안에 공통 기능이 삽입된다.
  17)AOP에서는 AOP library가 공통 기능을 알맞게 삽입해주기 때문에 개발자는 게시글 쓰기나 목록 읽기와 같은 핵심 logic을 구현할 때 transaction 적용이나 보안검사와 같은 공통 기능을 처리하기 위한 code를 핵심 logic code에 삽입할 필요가 없다.
  18)핵심 logic을 구현한 code에 공통 기능 관련 code가 포함되어 있지 않기 때문에 적용해야 할 공통 기능이 변경되더라도 핵심 logic을 구현한 code를 변경할 필요가 없다.
  19)단지, 공통 기능 code를 변경한 뒤 핵심 logic 구현 code에 적용만 하면 된다.
  20)AOP 개념을 적용하면 핵심기능 code 사이에 침투된 부가기능을 독립적인 Aspect로 구분해 낼 수 있다.
  21)구분된 부가기능 Aspect를 runtime 시에 필요한 위치에 동적으로 참여하게 할 수 있다.
  22)AspectJ Homepage : https://www.eclipse.org/aspectj/


2. AOP 용어
  1)Aspect : 여러 객체에 공통으로 적용되는 공통 관심 사항. 예)transaction이나 보안, logging 등...
    -Advice와 pointcut을 합친 것이다.
    -구현 하고자 하는 Cross-cutting Concern의 기능.
    -Application의 module화 하고자 하는 부분

  2)Target : 핵심 기능을 담고 있는 모듈로, Target은 부가기능을 부여할 대상이 된다.
    -Advice를 받는 객체.
    -Target은 우리가 작성한 class는 물론, 별도의 기능을 추가하고자 하는 third-party class가 될 수 있다.

  3)Advice : 언제 공통관심 기능을 핵심 logic에 적용할지를 정의한다.
    -즉, 부가기능을 정의한 code.

-Target에 제공할 부가기능을 담고 있는 module.
-PointCut에서 지정한 JoinPoint에서 실행(삽입)되어야할 code이다.
-Aspect의 실제 구현체
-예)'method를 호출하기 전'(언제)에 'transaction을 시작한다.'(공통기능) 기능을 적용한다는 것을 정의

4)JoinPoint : Advice를 적용해야 되는 지점
-Instance의 생성시점, method를 호출하는 시점, Exception이 발생하는 시점과 같이 application이 실행될 때 특정작업이 실행되는 시점을 의미한다.
-Aspect를 plugin 할 수 있는 application의 실행 지점
-즉, Target 객체가 구현한 interface의 모든 method는 JoinPoint가 된다.
-ex. field값 변경, method호출
-Spring에서는 method 호출만 해당

5)Pointcut : JoinPoint의 부분집합으로 실제로 Advice가 적용되는 JoinPoint 부분.
-Advice가 어떤 JoinPoint에 적용되어야 하는지 정의.
-명시적인 class의 이름, method의 이름이나 class나 method의 이름과 pattern이 일치하는 결합점을 지정 가능토록 해준다.
-Spring에서는 정규 표현식이나 AspectJ의 문법을 이용하여 정의한다.
-표현식은 execution으로 시작하고, method의 Signature를 비교하는 방법을 주로 이용

6)Weaving : Advice를 핵심 code에 적용하는 행위.
-공통 코드를 핵심 logic code에 삽입하는 것.
-AOP가 핵심기능(Target)의 code에 영향을 주지 않으면서 필요한 부가기능(Advice)을 추가할 수 있도록 해주는 핵심적인 처리과정
-Aspect를 Target 객체에 적용하여 새로운 proxy 객체를 생성하는 과정을 말한다.
-Aspect는 Target 객체의 지정된 JoinPoint에 엮인다.

7)즉, Aspect = Advice + PointCut 이다.

8)Aspect는 AOP의 기본 module

9)Aspect는 Singleton 형태의 객체로 존재한다.

10)Advisor = Advice + Pointcut
-Spring AOP에서만 사용되는 특별한 용어

11)그림참조
http://isstory83.tistory.com/90

3. 3 가지 Weaving 방식
1)Compile 시 : AspectJ에서 사용하는 방식
2)Class loading 시
3)Runtime 시 : Proxy를 이용. 핵심 logic을 구현한 객체에 직접 접근하는 것이 아니라 중간에 Proxy를 생성하여 Proxy를 통해서 핵심 logic을 구현한 객체에 접근
-Spring에서 AOP를 구현하는 방법 : Proxy를 이용한다.
-호출부(Client) --> Proxy(대행) --> Target(핵심기능)

4. Spring AOP의 특징
1)Spring은 Proxy 기반 AOP를 지원한다.
-Spring은 Target 객체에 대한 Proxy를 만들어 제공한다.
-Target을 감싸는 Proxy는 실행시간(Runtime)에 생성된다.
-Proxy는 Advice를 Target객체에 적용하면서 생성되는 객체이다.

2)Proxy가 호출을 intercept한다.
-Proxy는 Target 객체에 대한 호출을 가로챈 다음, Advice의 부가기능 logic을 수행하고 난 후에 Target의 핵심 기능 logic을 호출한다.(전처리 Advice)
-또는 Target의 핵심기능 logic method를 호출한 뒤에 부가기능(Advice)을 수행하는 경우도 있다.(후처리 Advice)

3)Spring AOP는 method JoinPoint만 지원한다.
-Spring은 동적 Proxy를 기반으로 AOP를 구현하기 때문에 method JoinPoint만 지원한다.
-즉, 핵심기능(Target)의 method가 호출되는 runtime 시점에만 부가기능(Advice)을 적용할 수 있다.
-반면에, AspectJ 같은 고급 AOP framework를 사용하면 객체의 생성, field값의 조회와 조작, static method 호출 및 초기화 등의 다양한 작업에 부가기능을 적용할 수 있다.

5. Spring에서 AOP 구현 방식
1)Spring API를 이용한 AOP 구현

2)XML schema 기반의 POJO class를 이용한 AOP구현 : Spring 2부터 사용
-부가기능을 제공하는 Advice class를 작성
-XML 설정 file에 <aop:config>를 이용해서 Aspect를 설정
-즉, Advice와 Pointcut을 설정

3)@Aspect annotation 기반의 AOP 구현
-@Aspect annotation을 이용해서 부가기능을 제공하는 Aspect class를 작성
-이때, Aspect class는 Advice를 구현하는 method와 Pointcut을 포함한다.
-XML 설정 file에 <aop:aspectj-autoproxy />를 설정

4)@Aspect annotation
-Aspect class를 선언할 때 @Aspect annotation을 사용한다.
-AspectJ 5 버전에 새롭게 추가된 annotaion이다.
-@Aspect annotaion을 이용할 경우 XML 설정 file에 Advice와 Pointcut을 설정하는 것이 아니라 class 내부에 정의할 수 있다.
-<aop:aspectj-autoproxy> tag를 설정file에 추가하면 @Aspect annotaion이 적용된 Bean을 Aspect로 사용 가능하다.

6. AspectJ와 Spring AOP library 설치
1)Runtime library 설치
-Maven Repository에서 'aspectj runtime'으로 검색
-aspectj runtime 1.8.10 버전을 pom.xml에 추가

```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.8.10</version>
</dependency>
```

2)AspectJ Weaver library 설치
-Maven Repository에서 'aspectj weaver'으로 검색
-aspectj weaver 1.8.10 버전을 pom.xml에 추가

```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.10</version>
```

```
        </dependency>

   3)Spring AOP library 설치
     -Maven Repository에서 'spring aop'으로 검색
     -aspectj weaver 4.3.9 버전을 pom.xml에 추가

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>4.3.9.RELEASE</version>
        </dependency>

   4)AspectJ Runtime API 문서
     -Google에서 'aspectj runtime aip doc'로 검색
     -https://eclipse.org/aspectj/doc/released/runtime-api/index.html


7. Advice의 종류
   1)<aop:before>
     -Method 실행 전에 Advice실행
     -JoinPoint 앞에서 실행되는 Advice

   2)<aop:after-returning>
     -정상적으로 method 실행 후에 Advice 실행
     -JoinPoint method 호출이 정상적으로 종료된 뒤에 실행되는 Advice

   3)<aop:after-throwing>
     -Method 실행 중 exception 발생시 Advice 실행
     -try-catch의 catch와 비슷

   4)<aop:after>
     -Method 실행 중 exception 이 발생하여도 Advice 실행
     -try-catch-finally에서 finally와 비슷

   5)<aop:around>
     -Target의 method 실행 전/후 및 exception 발생시 Advice 실행
     -JoinPoint 앞과 뒤에서 실행되는 Advice


8. Advice를 정의하는 annotation
   1)@Before("pointcut")
     -Target 객체의 method가 실행되기 전에 호출되는 Advice
     -JoinPoint를 통해 parameter 정보를 참조할 수 있다.

   2)@After("pointcut")
     -Target 객체의 method가 정상 종료됐을 때와 예외가 발생했을 때 모두 호출되는 Advice
     -Return값이나 예외를 직접 전달 받을 수는 없다.

   3)@Around("pointcut")
     -Target 객체의 method가 호출되는 전 과정을 모두 담을 수 있는 가장 강력한 기능을 가진 Advice

   4)@AfterReturning(pointcut="", returning="")
     -Target 객체의 method가 정상적으로 실행을 마친 후에 호출되는 Advice
     -Return값을 참조할 때는 returning 속성에 Return값을 저장할 변수 이름을 지정해야 한다.
```

198
199 5)@AfterThrowing(pointcut="", throwing="")
200 -Target 객체의 method가 예외가 발생하면 호출되는 Advice
201 -발생된 예외를 참조할 때는 throwing 속성에 발생한 예외를 저장할 변수 이름을 지정해야 한다.
202
203
204 9. JoinPoint Interace
205 1)JoinPoint 는 Spring AOP 혹은 AspectJ에서 AOP가 적용되는 지점을 뜻한다.
206
207 2)해당 지점을 AspectJ에서 JointPoint라는 interface로 나타낸다.
208
209 3)Methos
210 -getArgs() : method argument 반환
211 -getThis() : Proxy 객체를 반환
212 -getTarget() : 대상 객체를 반환
213 -getSignature() : Advice되는 method의 설명(description)을 반환
214 -toString() : Advice되는 method의 설명을 출력
215
216 4)모든 Advice는 org.aspectj.lang.JoinPoint type의 parameter를 Advice method에 첫 번째 매개변수로 선언 가능
217
218 5)Around Advice는 JoinPoint의 하위 class인 ProceedingJoinPoint 타입의 parameter를 필수적으로 선언해야 함.
219
220 6)AspectJ Runtime API의 org.aspectj.lang의 JoinPoint interface 참조할 것
221
222 7)AspectJ Runtime API의 org.aspectj.lang의 ProceedingJoinPoint interface 참조할 것
223
224
225 10. AOP 설정
226 1)<aop:config> : AOP의 설정 정보임을 나타낸다.
227 2)<aop:aspect> : Aspect를 설정한다.
228 3)<aop:around pointcut="execution()"> : Around Advice와 Pointcut을 설정한다.
229 4)<aop:aspect> tag의 ref속성은 Aspect로서 기능을 제공할 bean을 설정할 때 사용함.
230 5)<aop:around> tag의 pointcut 속성의 execution 지시자(designator)는 Advice를 적용할 package, class, method를 표현할 때 사용됨.
231 6)com.example.service package 및 그 하위 package에 있는 모든 public method를 Pointcut으로 설정하고 있다.
232 7)UserServiceImpl의 public method가 호출될 때 PerformanceTraceAdvice Bean의 trace() method가 호출되도록 설정하고 있다.
233
234
235 11. Lab : XML schema 기반의 AOP 구현
236 1)In Java Perspective, New > Java Project >
237 Project Name : AopDemo
238 -JRE : Use default JRE (currently 'jdk 1.8.0_192')
239 -Maven Project Convert : project right-click > Configure > Convert to Maven Project
240 -Spring Project Convert : project right-click > Spring Tools > Add Spring Project Nature
241
242 2)Create Package : src/com.example
243
244 3)com.example.Student.java
245 package com.example;
246

```
247     public class Student{
248       private String name;
249       private int age;
250       private int grade;
251       private int classNum;
252       public String getName() {
253         return name;
254       }
255       public void setName(String name) {
256         this.name = name;
257       }
258       public int getAge() {
259         return age;
260       }
261       public void setAge(int age) {
262         this.age = age;
263       }
264       public int getGrade() {
265         return grade;
266       }
267       public void setGrade(int grade) {
268         this.grade = grade;
269       }
270       public int getClassNum() {
271         return classNum;
272       }
273       public void setClassNum(int classNum) {
274         this.classNum = classNum;
275       }
276       public void getStudentInfo(){
277         System.out.println("Name : " + this.name);
278         System.out.println("Age : " + this.age);
279         System.out.println("Grade : " + this.grade);
280         System.out.println("Class : " + this.classNum);
281       }
282     }
283
284   4)com.example.Worker.java
285     package com.example;
286
287     public class Worker {
288       private String name;
289       private int age;
290       private String job;
291       public String getName() {
292         return name;
293       }
294       public void setName(String name) {
295         this.name = name;
296       }
297       public int getAge() {
298         return age;
299       }
300       public void setAge(int age) {
```

```
301         this.age = age;
302       }
303       public String getJob() {
304         return job;
305       }
306       public void setJob(String job) {
307         this.job = job;
308       }
309       public void getWorkerInfo(){
310         System.out.println("Name : " + this.name);
311         System.out.println("Age : " + this.age);
312         System.out.println("Job : " + this.job);
313       }
314     }
315
316   5)Spring Context 설치
317     -Maven Repository 에서 'Spring Context'로 검색하여 dependency 추가하고 설치
318
319       <dependency>
320         <groupId>org.springframework</groupId>
321         <artifactId>spring-context</artifactId>
322         <version>4.3.20.RELEASE</version>
323       </dependency>
324
325     -'junit' 검색해서 추가
326
327       <dependency>
328          <groupId>junit</groupId>
329          <artifactId>junit</artifactId>
330          <version>4.12</version>
331          <scope>test</scope>
332       </dependency>
333
334     -pom.xml : aop code 추가
335     -Runtime library 설치
336       --Maven Repository에서 'aspectj runtime'으로 검색
337       --aspectj runtime 1.9.2 version을 pom.xml에 추가
338
339         <dependency>
340             <groupId>org.aspectj</groupId>
341             <artifactId>aspectjrt</artifactId>
342             <version>1.9.2</version>
343         </dependency>
344
345     -AspectJ Weaver library 설치
346       --Maven Repository에서 'aspectj weaver'으로 검색
347       --aspectj weaver 1.8.10 version을 pom.xml에 추가
348
349         <dependency>
350             <groupId>org.aspectj</groupId>
351             <artifactId>aspectjweaver</artifactId>
352             <version>1.9.2</version>
353         </dependency>
354
```

```
   -Spring AOP library 설치
     --Maven Repository에서 'spring aop'으로 검색
     --aspectj weaver 4.3.9 version을 pom.xml에 추가

       <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-aop</artifactId>
           <version>4.3.20.RELEASE</version>
       </dependency>

   -Maven Install

6)com.example.LogAop.java
   package com.example;

   import org.aspectj.lang.ProceedingJoinPoint;

   public class LogAop {
   //joinpoint 객체를 전달 받을 때에는 반드시 첫번째 parameter여야 한다.
     public Object loggerAop(ProceedingJoinPoint joinpoint) throws Throwable{
       String signatureStr = joinpoint.getSignature().toShortString();
       //Signature getSignature() : 호출되는 method에대한 정보를 구한다.
       //cf)Object getTarget() : 대상 객체를 구한다.
       //cf)Object [] getArgs() : parameter 목록을 구한다.

       //toShortString() : method를 축약해서 표현한 문장을 구한다. method의 이름만 구한다.
       //cf)toLongString() : 완전하게 표현된 문장. return type, method이름, parameter type 모두
       //cf)getName() : method의 이름을 구한다.
       System.out.println(signatureStr + " is start.");
       long start = System.currentTimeMillis();

       try{
         Object obj = joinpoint.proceed(); //대상객체의 실제 method 호출
         return obj;
       }finally{
         long end = System.currentTimeMillis();
         System.out.println(signatureStr + " is finished.");
         System.out.println(signatureStr + " 경과시간 : " + (end - start));
       }
     }
   }

7)beans.xml 설정
   -Project right-click > Build Path > Configure Build Path.. > Source tab
   -Add Folder > Select AopDemo > Click Create New Folder... >
   -Folder name : config > Finish > OK > Apply and Close

   -config right-click > New > Spring Bean Configuration File > File name : beans.xml
   -Namespace tab
     --aop Check

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

```
409         xmlns:aop="http://www.springframework.org/schema/aop"
410         xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
411           http://www.springframework.org/schema/aop
              http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
412
413         <bean id="logAop" class="com.example.LogAop" />
414
415         <aop:config>
416           <aop:aspect id="logger" ref="logAop">
417             <aop:pointcut expression="within(com.example.*)*" id="publicMethod"/>
418             <aop:around method="loggerAop" pointcut-ref="publicMethod"/>
419           </aop:aspect>
420         </aop:config>
421
422         <!-- com.example아래 모든 class의 public Method를 호출할 때 LogAop의 loggerAop method가 실행
            된다는 뜻 -->
423
424         <bean id="student" class="com.example.Student">
425           <property name="name" value="한지민" />
426           <property name="age" value="15" />
427           <property name="grade" value="3" />
428           <property name="classNum" value="5" />
429         </bean>
430
431         <bean id="worker" class="com.example.Worker">
432           <property name="name" value="설운도" />
433           <property name="age" value="50" />
434           <property name="job" value="개발자" />
435         </bean>
436       </beans>
437
438     8)Junit Test Case 추가
439       -src > com.example.test package 추가
440       -com.example.test > right-click > New > Junit Test Case
441       -Select New JUnit 4 test
442       -Name : TestApp > Finish
443
444         package com.example.test;
445
446         import static org.junit.Assert.assertNotNull;
447         import static org.junit.Assert.fail;
448
449         import org.junit.Before;
450         import org.junit.Test;
451         import org.springframework.context.ApplicationContext;
452         import org.springframework.context.support.GenericXmlApplicationContext;
453
454         public class TestApp {
455           private ApplicationContext ctx;
456
457           @Before
458           public void init() {
459             this.ctx = new GenericXmlApplicationContext("classpath:beans.xml");
```

```
460         }
461         @Test
462         public void test() {
463           assertNotNull(this.ctx);
464         }
465       }
466
467     -right-click > Run as > JUnit test
468     -green bar
469
470   9)test() 수정
471     @Test
472     public void test() {
473       Student student = this.ctx.getBean("student", Student.class);
474       student.getStudentInfo();
475
476       Worker worker = this.ctx.getBean("worker", Worker.class);
477       worker.getWorkerInfo();
478     }
479
480   10)결과
481     -right-click > Run as > JUnit test
482     -green bar
483     Student.getStudentInfo() is start.
484     Name : 한지민
485     Age : 15
486     Grade : 3
487     Class : 5
488     Student.getStudentInfo() is finished.
489     Student.getStudentInfo() 경과시간 : 14
490     Worker.getWorkerInfo() is start.
491     Name : 설운도
492     Age : 50
493     Job : 개발자
494     Worker.getWorkerInfo() is finished.
495     Worker.getWorkerInfo() 경과시간 : 7
496
497
498 12. Lab : XML schema 기반의 AOP 구현
499   1)In Java Perspective, New > Java Project >
500     Project Name : AopDemo1
501     -JRE : Use default JRE (currently 'jdk 1.8.0_192')
502     -Maven Project Convert : project right-click > Configure > Convert to Maven Project
503     -Spring Project Convert : project right-click > Spring Tools > Add Spring Project Nature
504
505   2)Spring Context 설치
506     -Maven Repository 에서 'Spring Context'로 검색하여 dependency 추가하고 설치
507
508       <dependency>
509         <groupId>org.springframework</groupId>
510         <artifactId>spring-context</artifactId>
511         <version>4.3.20.RELEASE</version>
512       </dependency>
513
```

```
514     -'junit' 검색해서 추가
515
516       <dependency>
517         <groupId>junit</groupId>
518         <artifactId>junit</artifactId>
519         <version>4.12</version>
520         <scope>test</scope>
521       </dependency>
522
523     -Run as > Maven Install
524
525   3)Create Package : src/com.example
526   4)src/com.example.Animal interface
527
528     package com.example;
529
530     public interface Animal {
531       void walwal();
532     }
533
534   5)src/com.example.TomDog.java
535
536     package com.example;
537
538     public class TomDog implements Animal {
539
540         public void walwal() {
541           System.out.println("I'm Tomdog...");
542         }
543     }
544
545   6)src/com.example.AnimalAOP.java
546
547     package com.example;
548
549     public class AnimalAOP {
550         public void beforeWalwal() {
551             System.out.println("Hi~ Dog...");
552         }
553
554         public void afterWalwal() {
555             System.out.println("Good Bye Dog...");
556         }
557     }
558
559   7)src/com.example.TestClient.java
560
561     package com.example;
562
563     import org.springframework.beans.factory.BeanFactory;
564     import org.springframework.context.support.FileSystemXmlApplicationContext;
565
566     public class TestClient {
567
```

```
568         public static void main(String[] args) {
569           BeanFactory bean = new
              FileSystemXmlApplicationContext("classpath:applicationContext.xml");
570
571           Animal tomdog = (Animal)bean.getBean("tomdog");
572           tomdog.walwal();
573         }
574       }
575
576     8)-pom.xml : aop code 추가
577       -Runtime library 설치
578         --Maven Repository에서 'aspectj runtime'으로 검색
579         --aspectj runtime 1.9.2 version을 pom.xml에 추가
580
581           <dependency>
582               <groupId>org.aspectj</groupId>
583               <artifactId>aspectjrt</artifactId>
584               <version>1.9.2</version>
585           </dependency>
586
587       -AspectJ Weaver library 설치
588         --Maven Repository에서 'aspectj weaver'으로 검색
589         --aspectj weaver 1.8.10 version을 pom.xml에 추가
590
591           <dependency>
592               <groupId>org.aspectj</groupId>
593               <artifactId>aspectjweaver</artifactId>
594               <version>1.9.2</version>
595           </dependency>
596
597       -Spring AOP library 설치
598         --Maven Repository에서 'spring aop'으로 검색
599         --aspectj weaver 4.3.9 version을 pom.xml에 추가
600
601           <dependency>
602               <groupId>org.springframework</groupId>
603               <artifactId>spring-aop</artifactId>
604               <version>4.3.20.RELEASE</version>
605           </dependency>
606
607       -Maven Install
608
609     9)applicationContext.xml 설정
610       -Project right-click > Build Path > Configure Build Path.. > Source tab
611       -Add Folder > Select AopDemo > Click Create New Folder... >
612       -Folder name : config > Finish > OK > Apply and Close
613
614       -config right-click > New > Spring Bean Configuration File > File name : beans.xml
615       -Namespace tab
616         --aop Check
617
618         <bean id="tomdog" class="com.example.TomDog" />
619
620         <aop:config>
```

```
        <aop:aspect ref="animalAOP">
          <aop:pointcut id="greeting"
            expression="execution(public * com.example.Animal.walwal(..))" />
          <aop:before pointcut-ref="greeting" method="beforeWalwal" />
          <aop:after-returning pointcut-ref="greeting"
            method="afterWalwal" />
        </aop:aspect>
      </aop:config>

      <bean id="animalAOP" class="com.example.AnimalAOP" />
```

10)TestClient 실행 결과

```
Hi~ Dog...
I'm Tomdog...
Good Bye Dog...
```

## 13. Lab

1)In Spring Perspective, New > Spring Legacy Project > Simple Spring Maven
- -Project name: AopDemo2
- -Finish

2)pom.xml의 Dependencies tab에서
- -spring-context, spring-test, junit을 제외하고 모두 제거
- -pom.xml source에서

```
<java.version>1.8</java.version> 수정
<!-- Spring -->
<spring-framework.version>4.3.20.RELEASE</spring-framework.version> 수정

<!-- Hibernate / JPA --> 제거
<hibernate.version>4.2.1.Final</hibernate.version> 제거
<!-- Logging --> 제거
<logback.version>1.0.13</logback.version> 제거
<slf4j.version>1.7.5</slf4j.version> 제거

<junit.version>4.12</junit.version> 수정
```

- -Maven Install
- -Meven > Update Project
- -Project right-click > Properties > Project facet > Java version을 최신 jdk로 변경

3)Create Package : src/main/java/com.example

4)src/main/java/com.example.TV.java

```
public interface TV {
  void powerOn();
  void powerOff();
  void soundUp();
  void soundDown();
}
```

5)src/main/java/com.example.SamsungTV.java

```
675
676     package com.javasoft;
677
678     public class SamsungTV implements TV{
679       private String name;
680       public SamsungTV(String name) {
681         this.name = name;
682         System.out.println(name + " : 방금 객체가 생성됐습니다.");
683       }
684
685       public void powerOn() {
686         System.out.println(name + " : 전원을 킨다.");
687       }
688       public void powerOff() {
689         System.out.println(name + " : 전원을 끈다.");
690       }
691       public void soundUp() {
692         System.out.println(name + " : 볼륨을 올린다.");
693       }
694       public void soundDown() {
695         System.out.println(name + " : 볼륨을 내린다.");
696       }
697     }
698
699   6)src/main/java/com.example.LgTV.java
700
701     package com.javasoft;
702
703     public class LgTV implements TV{
704       private String name;
705       public LgTV(String name) {
706         this.name = name;
707         System.out.println(name + " : 방금 객체가 생성됐습니다.");
708       }
709       public void powerOn() {
710         System.out.println(name + " : 전원을 킨다.");
711       }
712       public void powerOff() {
713         System.out.println(name + " : 전원을 끈다.");
714       }
715       public void soundUp() {
716         System.out.println(name + " : 볼륨을 올린다.");
717       }
718       public void soundDown() {
719         System.out.println(name + " : 볼륨을 내린다.");
720       }
721     }
722
723   7)src/main/java/com.example.LogAdvice.java
724
725     package com.javasoft;
726
727     import org.aspectj.lang.JoinPoint;
728
```

```
729     public class LogAdvice {
730       public void printLog(JoinPoint thisJoinPoint){
731         System.out.println("[Core Concern] 수행전 로그하기");
732       }
733     }
734
735   8)src/main/resources/beans.xml
736
737     <?xml version="1.0" encoding="UTF-8"?>
738     <beans xmlns="http://www.springframework.org/schema/beans"
739       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
740       xmlns:context="http://www.springframework.org/schema/context"
741       xmlns:aop="http://www.springframework.org/schema/aop"
742       xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
743         http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-3.2.xsd
744         http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
745
746
747       <aop:config proxy-target-class="true">
748         <aop:aspect ref="logAdvice" id="logger">
749           <aop:pointcut expression="within(com.javasoft.*)*" id="myPointcut" />
750           <aop:before method="printLog" pointcut-ref="myPointcut"/>
751         </aop:aspect>
752       </aop:config>
753
754       <bean id="samsungTV" class="com.javasoft.SamsungTV">
755         <constructor-arg value="samsungTV" />
756       </bean>
757       <bean id="lgTV" class="com.javasoft.LgTV">
758         <constructor-arg value="lgTV" />
759       </bean>
760     </beans>
761
762   9)src/main/java/test package 생성
763
764   10)src/main/java/test.Main.java 생성
765
766     package test;
767
768     import org.springframework.context.support.AbstractApplicationContext;
769     import org.springframework.context.support.GenericXmlApplicationContext;
770
771     import com.javasoft.LgTV;
772     import com.javasoft.SamsungTV;
773     import com.javasoft.TV;
774
775     public class Main {
776       public static void main(String[] args) {
777         AbstractApplicationContext ctx =
778             new GenericXmlApplicationContext("classpath:beans.xml");
779
```

```
780         TV tv = ctx.getBean("lgTV", LgTV.class);
781         tv.powerOn();
782         tv.powerOff();
783         tv.soundUp();
784         tv.soundDown();
785         ctx.close();
786       }
787     }
788
789   11)결과
790
791     samsungTV : 방금 객체가 생성됐습니다.
792     lgTV : 방금 객체가 생성됐습니다.
793     [Core Concern] 수행전 로그하기
794     lgTV : 전원을 킨다.
795     [Core Concern] 수행전 로그하기
796     lgTV : 전원을 끈다.
797     [Core Concern] 수행전 로그하기
798     lgTV : 볼륨을 올린다.
799     [Core Concern] 수행전 로그하기
800     lgTV : 볼륨을 내린다.
801
802
803 14. Lab
804   1)Advice class 정보
805     -Class name : PerformanceTraceAdvice.java
806     -Class 기능 : Target 객체의 method 실행 시간을 계산해서 출력해 주는 부가기능 제공
807     -Advice 유형 : Around advice
808       --Target 객체의 method실행 전, 후의 시간을 측정하여 계산하면 Target 객체의 method 실행 시간을 알 수 있
          다.
809     -구현 method 이름 : trace(ProceedingJoinPoint joinPoint)
810
811   2)New > Spring Legacy Project > Simple Projects > Simple Spring Maven
812     Project Name : AopDemo2
813
814   3)Create Package : src/main/java/com.example
815
816   4)src/main/java/com.example.PerformanceTraceAdvice.java
817
818       package com.example;
819       import org.aspectj.lang.ProceedingJoinPoint;
820
821       public class PerformanceTraceAdvice {
822         public Object trace(ProceedingJoinPoint joinPoint) throws Throwable {
823           //타겟 method의 signature 정보
824           String signatureString = joinPoint.getSignature().toShortString();
825           System.out.println(signatureString + " 시작");
826           //타겟의 method가 호출되기 전의 시간
827           long start = System.currentTimeMillis();
828           try {
829             //타겟의 method 호출
830             Object result = joinPoint.proceed();
831             return result;
832           } finally {
```

```
            //타겟의 method가 호출된 후의 시간
            long finish = System.currentTimeMillis();
            System.out.println(signatureString + " 종료");
            System.out.println(signatureString + " 실행 시간 : " +
               (finish - start) + " ms");
          }
        }
      }

5)pom.xml : aop code 추가
  pom.xml : aop code 추가
  -Runtime library 설치
    --Maven Repository에서 'aspectj runtime'으로 검색
    --aspectj runtime 1.8.10 version을 pom.xml에 추가

      <dependency>
          <groupId>org.aspectj</groupId>
          <artifactId>aspectjrt</artifactId>
          <version>1.8.10</version>
      </dependency>

  -AspectJ Weaver library 설치
    --Maven Repository에서 'aspectj weaver'으로 검색
    --aspectj weaver 1.8.10 version을 pom.xml에 추가

      <dependency>
          <groupId>org.aspectj</groupId>
          <artifactId>aspectjweaver</artifactId>
          <version>1.8.10</version>
      </dependency>

  -Spring AOP library 설치
    --Maven Repository에서 'spring aop'으로 검색
    --aspectj weaver 4.3.9 version을 pom.xml에 추가

      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-aop</artifactId>
          <version>4.3.9.RELEASE</version>
      </dependency>

  -Maven Install

      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-context</artifactId>
          <version>4.3.9.RELEASE</version>     <--여기를 수정
      </dependency>

  -Maven Clean -> Maven Install

6)Advice class를 Bean으로 등록
  -src/main/resources/beans.xml
```

```
887         <!-- Advice class를 Bean으로 등록 -->
888         <bean id="performanceTraceAdvice" class="com.example.PerformanceTraceAdvice" />
889
890     7)beans.xml에 AOP namespace 추가
891       -aop - http://www.springframework.org/schema/aop check
892
893     8)AOP 설정
894
895         <aop:config>
896           <aop:aspect id="traceAspect"   ref="performanceTraceAdvice">
897             <aop:around  pointcut="execution(public * com.example.Hello.*(..))" method="trace"
                />
898           </aop:aspect>
899         </aop:config>
900
901       -<aop:config> : AOP 설정 정보임을 나타낸다.
902       -<aop:aspect> : Aspect를 설정한다.
903       -<aop:around pointcut="execution()"> : Around Advice와 Pointcut을 설정한다.
904
905     9)Target Class 작성
906       -/src/main/java/com.example.Hello.java
907
908         package com.example;
909
910         import org.springframework.beans.factory.annotation.Value;
911         import org.springframework.stereotype.Component;
912
913         @Component("hello")
914         public class Hello {
915           @Value("Spring")
916           private String name;
917
918           @Value("25")
919           private int age;
920
921           @Override
922           public String toString() {
923             return String.format("Hello [name=%s, age=%s]", name, age);
924           }
925         }
926
927     10)beans.xml 설정
928       -Namespace Tab
929       -Check context - http://www.springframework.org/schema/context
930
931         <context:component-scan base-package="com.example" />
932
933     11)Around Advice와 AOP 설정 test
934       -/src/main/java/com.example.MainClass.java
935
936         package com.example;
937
938         import org.springframework.context.ApplicationContext;
939         import org.springframework.context.support.GenericXmlApplicationContext;
```

```
940
941         public class MainClass {
942           public static void main(String[] args) {
943             ApplicationContext ctx = new GenericXmlApplicationContext("classpath:beans.xml");
944
945             Hello hello = ctx.getBean("hello", Hello.class);
946             System.out.println(hello);
947           }
948         }
949
950     12)결과
951
952         Hello.toString() 시작
953         Hello.toString() 종료
954         Hello.toString() 실행 시간 : 50 ms
955         Hello [name=Spring, age=25]
956
957
958   15. Lab : @Aspect annotation 기반의 AOP 구현
959     1)New > Spring Legacy Project > Simple Projects > Simple Spring Maven
960       Project Name : AopDemo3
961
962     2)Create Package : src/main/java/com.example
963
964     3)com.example.Student.java
965       package com.example;
966
967       import java.util.ArrayList;
968
969       import org.springframework.beans.factory.DisposableBean;
970       import org.springframework.beans.factory.InitializingBean;
971
972       public class Student{
973         private String name;
974         private int age;
975         private int grade;
976         private int classNum;
977         public String getName() {
978           return name;
979         }
980         public void setName(String name) {
981           this.name = name;
982         }
983         public int getAge() {
984           return age;
985         }
986         public void setAge(int age) {
987           this.age = age;
988         }
989         public int getGrade() {
990           return grade;
991         }
992         public void setGrade(int grade) {
993           this.grade = grade;
```

```
994         }
995         public int getClassNum() {
996           return classNum;
997         }
998         public void setClassNum(int classNum) {
999           this.classNum = classNum;
1000        }
1001        public void getStudentInfo(){
1002          System.out.println("Name : " + this.name);
1003          System.out.println("Age : " + this.age);
1004          System.out.println("Grade : " + this.grade);
1005          System.out.println("Class : " + this.classNum);
1006        }
1007      }
1008
1009    4)com.example.Worker.java
1010      package com.example;
1011
1012      public class Worker {
1013        private String name;
1014        private int age;
1015        private String job;
1016        public String getName() {
1017          return name;
1018        }
1019        public void setName(String name) {
1020          this.name = name;
1021        }
1022        public int getAge() {
1023          return age;
1024        }
1025        public void setAge(int age) {
1026          this.age = age;
1027        }
1028        public String getJob() {
1029          return job;
1030        }
1031        public void setJob(String job) {
1032          this.job = job;
1033        }
1034        public void getWorkerInfo(){
1035          System.out.println("Name : " + this.name);
1036          System.out.println("Age : " + this.age);
1037          System.out.println("Job : " + this.job);
1038        }
1039      }
1040
1041    5)pom.xml : 아래 code 추가
1042      <!-- AOP -->
1043      <dependency>
1044        <groupId>org.aspectj</groupId>
1045        <artifactId>aspectjweaver</artifactId>
1046        <version>1.8.10</version>
1047      </dependency>
```

6)com.example.LogAop.java

```
package com.example;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class LogAop {

  @Pointcut("within(com.example.*)*")
  private void pointcutMethod(){}

  @Around("pointcutMethod()")
  public Object loggerAop(ProceedingJoinPoint joinpoint) throws Throwable{
    String signatureStr = joinpoint.getSignature().toShortString();
    System.out.println(signatureStr + " is start.");
    long start = System.currentTimeMillis();

    try{
      Object obj = joinpoint.proceed();
      return obj;
    }finally{
      long end = System.currentTimeMillis();
      System.out.println(signatureStr + " is finished.");
      System.out.println(signatureStr + " 경과시간 : " + (end - start));
    }
  }

  @Before("within(kr.co.javaexpert.*)*")
  public void beforeAdvice(){
    System.out.println("Called beforeAdvice()");
  }
}
```

7)src/main/resources/beans.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">

  <bean id="logAop" class="com.example.LogAop" />

  <aop:config>
    <aop:aspect id="logger" ref="logAop">
      <aop:pointcut expression="within(com.example.*)*" id="publicM"/>
      <aop:around method="loggerAop" pointcut-ref="publicM"/>
```

```
        </aop:aspect>
      </aop:config>

      <bean id="student" class="com.example.Student">
        <property name="name" value="한지민" />
        <property name="age" value="15" />
        <property name="grade" value="3" />
        <property name="classNum" value="5" />
      </bean>

      <bean id="worker" class="com.example.Worker">
        <property name="name" value="설운도" />
        <property name="age" value="50" />
        <property name="job" value="개발자" />
      </bean>
    </beans>
```

8)com.example.MainClass.java

```
package com.example;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class MainClass {
  public static void main(String[] args) {
    AbstractApplicationContext context = new
    GenericXmlApplicationContext("classpath:beans.xml");
    Student student = context.getBean("student", Student.class);
    student.getStudentInfo();

    Worker worker = context.getBean("worker", Worker.class);
    worker.getWorkerInfo();

    context.close();
  }
}
```

9)결과

```
Student.getStudentInfo() is start.
Name : 한지민
Age : 15
Grade : 3
Class : 5
Student.getStudentInfo() is finished.
Student.getStudentInfo() 경과시간 : 12 ms
Worker.getWorkerInfo() is start.
Name : 설운도
Age : 50
Job : 개발자
Worker.getWorkerInfo() is finished.
Worker.getWorkerInfo() 경과시간 : 7 ms
```

16. Lab

1)Aspect class 정보
-Class명 : LoggingAspect.java
-Class 기능 : 이 Aspect class는 4가지 유형의 Advice와 Pointcut을 설정하여 Target 객체의 parameter와 return값, 예외 발생 시 예외 message를 출력하는 기능을 제공
-Advice 유형 : Before, AfterReturning, AfterThrowing, After
-구현 method명 : before(JoinPoint joinPoint), afterReturing(JoinPoint joinPoint, Object ret), afterThrowing(JoinPoint joinPoint, Throwable ex), afterFinally(JoinPoint joinPoint)

2)Aspect class 선언 및 설정
-Class 선언부에 @Aspect annotation을 정의한다.
-이 class를 Aspect로 사용하려면 Bean으로 등록해야 하므로 @Component annotation도 함께 정의한다.

```
package com.example;

import org.aspectj.lang.JoinPoint;

@Component
@Aspect
public class LoggingAspect {
...

<context:component-scan base-package="com.example" />
```

3)XML 설정파일에 <aop:aspectj-autoproxy /> 선언
-이 선언은 bean으로 등록된 class 중에서 @Aspect가 선언된 class를 모두 Aspect로 자동 등록해주는 역할을 한다.

```
<aop:aspectj-autoproxy />
```

4)AopDemo4 Project 생성
-Spring Legacy Project > Simple Maven Project

5)com.example package 생성
-/src/main/java/com.example

6)pom.xml에 Aspectj 종속성 추가 및 설치

```
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.8.10</version>
</dependency>
```

-Maven Install

7)/src/main/java/com.example.LoggingAspect.java 생성

```
package com.example;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
```

```
1204        import org.aspectj.lang.annotation.Before;
1205        import org.springframework.stereotype.Component;
1206        @Component
1207        @Aspect
1208        public class LoggingAspect {
1209          @Before("execution(public * com.example..*(..))")
1210          public void before(JoinPoint joinPoint) {
1211            String signatureString = joinPoint.getSignature().getName();
1212            System.out.println("@Before [ " + signatureString + " ] 메서드 실행 전처리 수행");
1213            for (Object arg : joinPoint.getArgs()) {
1214              System.out.println("@Before [ " + signatureString + " ] 아규먼트 " + arg);
1215            }
1216          }
1217            @AfterReturning(pointcut="execution(public * com.example..*(..))", returning="ret")
1218          public void afterReturning(JoinPoint joinPoint, Object ret) {
1219            String signatureString = joinPoint.getSignature().getName();
1220            System.out.println("@AfterReturing [ " + signatureString + " ] method 실행 후처리 수
              행");
1221            System.out.println("@AfterReturing [ " + signatureString + " ] 리턴값=" + ret);
1222
1223          }
1224
1225            @AfterThrowing(pointcut="execution(public * com.example..*(..))",
1226                throwing="ex")
1227          public void afterThrowing(JoinPoint joinPoint, Throwable ex) {
1228            String signatureString = joinPoint.getSignature().getName();
1229            System.out.println("@AfterThrowing [ " + signatureString + " ] method 실행 중 예외 발
              생");
1230            System.out.println("@AfterThrowing [ " + signatureString + " ] 예외=" +
              ex.getMessage());
1231          }
1232
1233            @After("execution(public * com.example..*(..))")
1234          public void afterFinally(JoinPoint joinPoint) {
1235            String signatureString = joinPoint.getSignature().getName();
1236            System.out.println("@After [ " + signatureString + " ] method 실행 완료");
1237          }
1238        }
1239
1240    8)beans.xml 파일 생성
1241      -/src/main/resources/beans.xml
1242      -Namespace Tab에서 'aop'와 'context' 체크할 것
1243
1244        <?xml version="1.0" encoding="UTF-8"?>
1245        <beans xmlns="http://www.springframework.org/schema/beans"
1246          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1247          xmlns:aop="http://www.springframework.org/schema/aop"
1248          xmlns:context="http://www.springframework.org/schema/context"
1249          xsi:schemaLocation="http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans.xsd
1250            http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context-3.2.xsd
1251            http://www.springframework.org/schema/aop
                http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
```

```
1252
1253          <context:component-scan base-package="com.example" />
1254          <aop:aspectj-autoproxy />
1255        </beans>
1256
1257    9)Target 객체 생성
1258      -/src/main/java/com.example.Hello.java
1259
1260        package com.example;
1261
1262        import org.springframework.beans.factory.annotation.Value;
1263        import org.springframework.stereotype.Component;
1264
1265        @Component("hello")
1266        public class Hello {
1267          @Value("Spring")
1268          private String name;
1269
1270          @Value("25")
1271          private int age;
1272
1273          @Override
1274          public String toString() {
1275            return String.format("Hello [name=%s, age=%s]", name, age);
1276          }
1277
1278          public void calculation(){
1279            System.out.println(5 / 0);
1280          }
1281        }
1282
1283    10)테스트 class 작성
1284      -src/main/java/com.example.MainClass.java
1285
1286        package com.example;
1287
1288        import org.springframework.context.ApplicationContext;
1289        import org.springframework.context.support.GenericXmlApplicationContext;
1290
1291        public class MainClass {
1292          public static void main(String[] args) {
1293            ApplicationContext ctx = new GenericXmlApplicationContext("classpath:beans.xml");
1294
1295            Hello hello = ctx.getBean("hello", Hello.class);
1296            System.out.println(hello);
1297            hello.calculation();
1298          }
1299        }
1300
1301    11)결과
1302        @Before [ toString ] method 실행 전처리 수행
1303        @After [ toString ] method 실행 완료
1304        @AfterReturing [ toString ] method 실행 후처리 수행
1305        @AfterReturing [ toString ] 리턴값=Hello [name=Spring, age=25]
```

1306 Hello [name=Spring, age=25]
1307 @Before [ calculation ] method 실행 전처리 수행
1308 @After [ calculation ] method 실행 완료
1309 @AfterThrowing [ calculation ] method 실행 중 예외 발생
1310 @AfterThrowing [ calculation ] 예외=/ by zero
1311
1312
1313 17. AspectJ Pointcut 표현식
1314 1)표현식은 Pointcut 지시자를 이용하여 작성
1315
1316 2)가장 대표적인 지시자는 execution()이다.
1317
1318 3)Pointcut 을 지정할 때 사용하는 표현식으로 AspectJ 문법을 사용한다.
1319 -* : 모든
1320 -. : 현재
1321 -.. : 0개 이상
1322
1323 4)execution
1324 -Usage
1325 execution([접근제한자 pattern] return type pattern [type pattern.] 이름 pattern(parameter type pattern | "..", ...) [throws 예외pattern])
1326 --접근제한자 pattern : public, private과 같은 접근 제한자, 생략가능
1327 --Return type pattern : return값의 type pattern
1328 --Type pattern : 패키지와 class 이름에 대한 pattern, 생략가능. 사용할 때 "."를 사용해 연결함.
1329 --이름 pattern : method 이름 type pattern
1330 --Parameter 타입pattern : parameter의 type pattern을 순서대로 넣을 수 있다. wildcard를 이용해서 parameter 갯수에 상관없는 pattern을 만들 수 있다.
1331 --예외 pattern : 예외 이름 pattern
1332
1333 -예
1334 "execution(* aspects.trace.demo.*.*(..))"
1335 -* : Any return type
1336 -aspects.trace.demo : package
1337 -* : class
1338 -* : method
1339 -(..) : Any type and number of arguments
1340
1341 -execution(* hello(..))
1342 --hello라는 이름을 가진 method를 선정
1343 --Parameter는 모든 종류를 다 허용
1344
1345 -execution(* hello())
1346 --hello method 중에서 parameter가 없는 것만 선택함.
1347
1348 -execution(* com.example.service.UserServiceImpl.*(..))
1349 --com.example.service.UserServiceImpl class를 직접 지정
1350 --이 class가 가진 모든 method를 선택
1351
1352 -execution(* com.example.user.service.*.*(..))
1353 --com.example.user.service package의 모든 class에 적용
1354 --하지만 sub-package의 class는 포함하지 않는다.
1355
1356 -execution(* com.example.user.service..*.*(..))
1357 --com.example.user.service package의 모든 class에 적용

1358 --그리고 '..'를 사용해서 sub-package의 모든 class까지 포함
1359
1360 -execution(* *.. Target.*(..))
1361 --Package에 상관없이 Target이라는 이름의 모든 class에 적용
1362 --다른 package의 같은 이름의 class가 있어도 적용이 된다는 점에 유의해야 함.
1363
1364 @Pointcut("executeion(public void get*(..))") : public void인 모든 get method
1365 @Pointcut("executeion(* com.example.*.*())") : com.example package에 parameter가 없는 모든 method
1366 @Pointcut("executeion(* com.example..*.*())") : com.example package & kr.co.javaexpert 하위 package에 parameter가 없는 모든 method
1367 @Pointcut("executeion(* com.example.Worker.*())") : com.example.Worker 안의 모든 method
1368
1369 2)within
1370 @Pointcut("within(com.example.*)") : com.example package 안에 있는 모든 method
1371 @Pointcut("within(com.example..*)") : com.example package 및 하위 package 안에 있는 모든 method
1372 @Pointcut("within(com.example.Worker)") : com.example.Worker 모든 method
1373
1374 3)bean
1375 @Pointcut("bean(student)") : student bean에만 적용
1376 @Pointcut("bean(*ker)") : ~ker로 끝나는 bean에만 적용