1. AOP What?
1)어플리케이션은 다양한 공통 기능을 필요로 한다.
2)로깅과 같은 기본적인 기능에서부터 트랜잭션이나 보안과 같은 기능에 이르기까지 어플리케이션 전반에 걸쳐 적용되는 공통 기능이 존재한다.
3)공통기능은 어플리케이션의 핵심 비지니스 로직과는 구분되고 핵심기능을 도와 주는 부가적인 기능(로깅, 보안 등)이다.
4)핵심 비즈니스 기능과 구분하기 위해 공통 기능을 공통 관심 사항(Cross-cutting Concern)이라고 표현한다.
5)핵심 로직(Biz 로직을 포함하는 기능)을 핵심 관심 사항(Core Concern)이라고 표현한다.
-핵심기능
--계좌 이체, 대출 승인, 이자 계산
-부가기능
--로깅, 보안, 트렌젝션
-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)프로그래밍에서 공통적인 기능을 모든 모듈에 적용하기 위한 방법으로 상속이 있다.
8)하지만 자바는 다중상속을 하지 않기 때문에 다양한 모듈에 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 프로그래밍하는 기법이다.
11)분리한 부가기능(공통 기능)을 Aspect라는 독특한 모듈 형태로 만들어서 설계하고 개발하는 방법이다.
12)기본적인 개념은 공통 관심 사항을 구현한 코드를 핵심 로직을 구현한 코드 안에 삽입한다는 것이다.
13)OOP를 적용하여도 핵심기능에서 부가기능을 쉽게 분리된 모듈로 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 로직을 구현한 코드에서 공통 기능을 직접적으로 호출하지 않는다.
16)핵심 로직을 구현한 코드를 컴파일하거나, 컴파일된 클래스를 로딩하거나, 로딩한 클래스의 객체를 생성할 때 AOP가 적용되어 핵심 로직 구현 코드안에 공통 기능이 삽입된다.
17)AOP에서는 AOP 라이브러리가 공통 기능을 알맞게 삽입해주기 때문에 개발자는 게시글 쓰기나 목록 읽기와 같은 핵심 로직을 구현할 때 트랜잭션 적용이나 보안검사와 같은 공통 기능을 처리하기위한 코드를 핵심 로직 코드에 삽입할 필요가 없다.
18)핵심 로직을 구현한 코드에 공통 기능 관련 코드가 포함되어 있지 않기 때문에 적용해야 할 공통 기능이 변경되더라도 핵심 로직을 구현한 코드를 변경할 필요가 없다.
19)단지, 공통 기능 코드를 변경한 뒤 핵심 로직 구현 코드에 적용만 하면 된다.
20)AOP 개념을 적용하면 핵심기능 코드 사이에 침투된 부가기능을 독립적인 Aspect로 구분해 낼 수 있다.
21)구분된 부가기능 Aspect를 런타임 시에 필요한 위치에 동적으로 참여하게 할 수 있다.

2. AOP 용어
1)Aspect : 여러 객체에 공통으로 적용되는 공통 관심 사항. 예)트랜잭션이나 보안, 로깅 등...

2)Target : 핵심 기능을 담고 있는 모듈로, Target은 부가기능을 부여할 대상이 된다.

3)Advice : 언제 공통관심 기능을 핵심 로직에 적용할지를 정의한다.
-즉, 부가기능을 정의한 코드.
-Target에 제공할 부가기능을 담고 있는 모듈.
-예)'메소드를 호출하기 전'(언제)에 '트랜잭션을 시작한다.'(공통기능) 기능을 적용한다는 것을 정의

4)JoinPoint : Advice를 적용해야 되는 지점
-즉, Target 객체가 구현한 인터페이스의 모든 메소드는 JoinPoint가 된다.
-ex. 필드값 변경, 메소드호출
-Spring에서는 메소드호출만 해당

5)Pointcut : JoinPoint의 부분집합으로 실제로 Advice가 적용되는 JoinPoint 부분.
-Spring에서는 정규 표현식이나 AspectJ의 문법을 이용하여 정의한다.
-표현식은 execution으로 시작하고, 메소드의 Signature를 비교하는 방법을 주로 이용

6)Weaving : Advice를 핵심 코드에 적용하는 행위.
-공통 코드를 핵심 로직 코드에 삽입하는 것.
-AOP가 핵심기능(Target)의 코드에 영향을 주지 않으면서 필요한 부가기능(Advice)을 추가할 수 있도록 해주는 핵심적인 처리과정

7)즉, Aspect = Advice + PointCut 이다.

8)Aspect는 AOP의 기본 모듈

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)컴파일 시 : AspectJ에서 사용하는 방식
2)클래스 로딩 시
3)런타임 시 : Proxy를 이용. 핵심 로직을 구현한 객체에 직접 접근하는 것이 아니라 중간에 Proxy를 생성하여 Proxy를 통해서 핵심 로직을 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 로직을 수행하고 난 후에 Target의 핵심 기능 로직을 호출한다.(전처리 Advice)
-또는 Target의 핵심기능 로직 메소드를 호출한 뒤에 부가기능(Advice)을 수행하는 경우도 있다.(후처리 Advice)

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

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

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

3)@Aspect 어노테이션 기반의 AOP 구현
-@Aspect 어노테이션을 이용해서 부가기능을 제공하는 Aspect 클래스를 작성
-이때, Aspect 클래스는 Advice를 구현하는 메소드와 Pointcut을 포함한다.
-XML 설정 파일에 <aop:aspectj-autoproxy />를 설정

4)@Aspect annotation
-Aspect 클래스를 선언할 때 @Aspect 어노테이션을 사용한다.
-AspectJ 5 버전에 새롭게 추가된 어노테이션이다.
-@Aspect 어노테이션을 이용할 경우 XML 설정 파일에 Advice와 Pointcut을 설정하는 것이 아니라 클래스 내부에 정의할 수 있다.
-<aop:aspectj-autoproxy> 태그를 설정파일에 추가하면 @Aspect 어노테이션이 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>
      -메소드 실행 전에 Advice실행
      -JoinPoint 앞에서 실행되는 Advice

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

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

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

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

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

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

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

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

    5)@AfterThrowing(pointcut="", throwing="")
      -Target 객체의 메소드가 예외가 발생하면 호출되는 Advice
      -발생된 예외를 참조할 때는 throwing 속성에 발생한 예외를 저장할 변수 이름을 지정해야 한다.
```

9. JoinPoint Interace

1)JoinPoint 는 Spring AOP 혹은 AspectJ에서 AOP가 적용되는 지점을 뜻한다.

2)해당 지점을 AspectJ에서 JointPoint라는 인터페이스로 나타낸다.

3)Methos

-getArgs() : 메소드 argument 반환
-getThis() : Proxy 객체를 반환
-getTarget() : 대상 객체를 반환
-getSignature() : Advice되는 메소드의 설명(description)을 반환
-toString() : Advice되는 메소드의 설명을 출력

4)모든 Advice는 org.aspectj.lang.JoinPoint 타입의 파라미터를 Advice 메소드에 첫 번째 매개변수로 선언 가능

5)Around Advice는 JoinPoint의 하위 클래스인 ProceedingJoinPoint 타입의 파라미터를 필수적으로 선언해야 함.

6)AspectJ Runtime API의 org.aspectj.lang의 JoinPoint interface 참조할 것

7)AspectJ Runtime API의 org.aspectj.lang의 ProceedingJoinPoint interface 참조할 것

10. AOP 설정

1)<aop:config> : AOP의 설정 정보임을 나타낸다.
2)<aop:aspect> : Aspect를 설정한다.
3)<aop:around pointcut="execution()"> : Around Advice와 Pointcut을 설정한다.
4)<aop:aspect> 태그의 ref속성은 Aspect로서 기능을 제공할 Bean을 설정할 때 사용함.
5)<aop:around> 태그의 pointcut 속성의 execution 지시자(designator)는 Advice를 적용할 패키지, 클래스, 메소드를 표현할 때 사용됨.
6)info.javaexpert.service 패키지 및 그 하위 패키지에 있는 모든 public 메소드를 Pointcut으로 설정하고 있다.
7)UserServiceImpl의 public 메소드가 호출될 때 PerformanceTraceAdvice Bean의 trace() 메소드가 호출되도록 설정하고 있다.

11. Lab : XML 스키마 기반의 AOP 구현

1)New > Spring Legacy Project > Simple Projects > Simple Spring Maven
Project Name : AopDemo

2)Create Package : src/main/java/info.javaexpert

3)info.javaexpert.Student.java

```
package info.javaexpert;

import java.util.ArrayList;

import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;

public class Student{
    private String name;
    private int age;
    private int grade;
    private int classNum;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
```

```
    }
    public int getGrade() {
        return grade;
    }
    public void setGrade(int grade) {
        this.grade = grade;
    }
    public int getClassNum() {
        return classNum;
    }
    public void setClassNum(int classNum) {
        this.classNum = classNum;
    }
    public void getStudentInfo(){
        System.out.println("Name : " + this.name);
        System.out.println("Age : " + this.age);
        System.out.println("Grade : " + this.grade);
        System.out.println("Class : " + this.classNum);
    }
}

4)info.javaexpert.Worker.java
  package info.javaexpert;

  public class Worker {
    private String name;
    private int age;
    private String job;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getJob() {
        return job;
    }
    public void setJob(String job) {
        this.job = job;
    }
    public void getWorkerInfo(){
        System.out.println("Name : " + this.name);
        System.out.println("Age : " + this.age);
        System.out.println("Job : " + this.job);
    }
  }

5)pom.xml : aop 코드 추가
  -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>

  -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>

      -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>

      -Maven Install

   6)info.javaexpert.LogAop.java
      package info.javaexpert;

      import org.aspectj.lang.ProceedingJoinPoint;

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

            //toShortString() : 메소드를 축약해서 표현한 문장을 구한다. 메소드의 이름만 구한다.
            //cf)toLongString() : 완전하게 표현된 문장. 리턴타입, 메소드이름, 파라미터 타입 모두
            //cf)getName() : 메소드의 이름을 구한다.
            System.out.println(signatureStr + " is start.");
            long start = System.currentTimeMillis();

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

   7)src/main/resources/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"
         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="info.javaexpert.LogAop" />

         <aop:config>
            <aop:aspect id="logger" ref="logAop">
               <aop:pointcut expression="within(info.javaexpert.*)*" id="publicMethod"/>
```

```
            <aop:around method="loggerAop" pointcut-ref="publicMethod"/>
        </aop:aspect>
    </aop:config>

    <!-- info.javaexpert아래 모든 클래스의 public Method를 호출할 때 LogAop의 loggerAop 메소드가 실행된다는 뜻 -->

    <bean id="student" class="info.javaexpert.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="info.javaexpert.Worker">
        <property name="name" value="설운도" />
        <property name="age" value="50" />
        <property name="job" value="개발자" />
    </bean>
</beans>
```

8)info.javaexpert.MainClass.java

```
package info.javaexpert;

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 : 14
Worker.getWorkerInfo() is start.
Name : 설운도
Age : 50
Job : 개발자
Worker.getWorkerInfo() is finished.
Worker.getWorkerInfo() 경과시간 : 7
```

10)Error 발생시 아래와 같이 조치한다.

-pom.xml 파일에서 아래를 수정한다.

```
    <dependencies>
        <!-- Spring and Transactions -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>4.3.9.RELEASE</version>     <--여기를 수정
        </dependency>
```

12. Lab

1)Advice 클래스 정보

-Class name : PerformanceTraceAdvice.java

-Class 기능 : Target 객체의 메소드 실행 시간을 계산해서 출력해 주는 부가기능 제공

-Advice 유형 : Around advice

--Target 객체의 메소드실행 전, 후의 시간을 측정하여 계산하면 Target 객체의 메소드 실행 시간을 알 수 있다.

-구현 메소드 이름 : trace(ProceedingJoinPoint joinPoint)

2)New > Spring Legacy Project > Simple Projects > Simple Spring Maven

Project Name : AopDemo1

3)Create Package : src/main/java/info.javaexpert

4)src/main/java/info.javaexpert.PerformanceTraceAdvice.java

```
package info.javaexpert;
import org.aspectj.lang.ProceedingJoinPoint;

public class PerformanceTraceAdvice {
   public Object trace(ProceedingJoinPoint joinPoint) throws Throwable {
      //타겟 메서드의 signature 정보
      String signatureString = joinPoint.getSignature().toShortString();
      System.out.println(signatureString + " 시작");
      //타겟의 메서드가 호출되기 전의 시간
      long start = System.currentTimeMillis();
      try {
         //타겟의 메서드 호출
         Object result = joinPoint.proceed();
         return result;
      } finally {
         //타겟의 메서드가 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 코드 추가

pom.xml : aop 코드 추가

-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>
```

-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>
```

-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>

    -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 클래스를 Bean으로 등록
    -src/main/resources/beans.xml

        <!-- Advice 클래스를 Bean으로 등록 -->
        <bean id="performanceTraceAdvice" class="info.javaexpert.PerformanceTraceAdvice" />

7)beans.xml에 AOP 네임스페이스 추가
    -aop - http://www.springframework.org/schema/aop check

8)AOP 설정

        <aop:config>
            <aop:aspect id="traceAspect"   ref="performanceTraceAdvice">
                <aop:around  pointcut="execution(public * info.javaexpert.Hello.*(..))" method="trace"
                />
            </aop:aspect>
        </aop:config>

    -<aop:config> : AOP 설정 정보임을 나타낸다.
    -<aop:aspect> : Aspect를 설정한다.
    -<aop:around pointcut="execution()"> : Around Advice와 Pointcut을 설정한다.

9)Target Class 작성
    -/src/main/java/info.javaexpert.Hello.java

        package info.javaexpert;

        import org.springframework.beans.factory.annotation.Value;
        import org.springframework.stereotype.Component;

        @Component("hello")
        public class Hello {
            @Value("Spring")
            private String name;

            @Value("25")
            private int age;

            @Override
            public String toString() {
                return String.format("Hello [name=%s, age=%s]", name, age);
            }
        }

10)beans.xml 설정
    -Namespace Tab
    -Check context - http://www.springframework.org/schema/context

        <context:component-scan base-package="info.javaexpert" />

11)Around Advice와 AOP 설정 테스트
    -/src/main/java/info.javaexpert.MainClass.java
```

```
579
580         package info.javaexpert;
581
582         import org.springframework.context.ApplicationContext;
583         import org.springframework.context.support.GenericXmlApplicationContext;
584
585         public class MainClass {
586            public static void main(String[] args) {
587               ApplicationContext ctx = new GenericXmlApplicationContext("classpath:beans.xml");
588
589               Hello hello = ctx.getBean("hello", Hello.class);
590               System.out.println(hello);
591            }
592         }
593
594      12)결과
595
596         Hello.toString() 시작
597         Hello.toString() 종료
598         Hello.toString() 실행 시간 : 50 ms
599         Hello [name=Spring, age=25]
600
601
602   13. Lab : @Aspect 어노테이션 기반의 AOP 구현
603      1)New > Spring Legacy Project > Simple Projects > Simple Spring Maven
604         Project Name : AopDemo2
605
606      2)Create Package : src/main/java/info.javaexpert
607
608      3)info.javaexpert.Student.java
609         package info.javaexpert;
610
611         import java.util.ArrayList;
612
613         import org.springframework.beans.factory.DisposableBean;
614         import org.springframework.beans.factory.InitializingBean;
615
616         public class Student{
617            private String name;
618            private int age;
619            private int grade;
620            private int classNum;
621            public String getName() {
622               return name;
623            }
624            public void setName(String name) {
625               this.name = name;
626            }
627            public int getAge() {
628               return age;
629            }
630            public void setAge(int age) {
631               this.age = age;
632            }
633            public int getGrade() {
634               return grade;
635            }
636            public void setGrade(int grade) {
637               this.grade = grade;
638            }
639            public int getClassNum() {
640               return classNum;
641            }
642            public void setClassNum(int classNum) {
643               this.classNum = classNum;
644            }
645            public void getStudentInfo(){
```

```
            System.out.println("Name : " + this.name);
            System.out.println("Age : " + this.age);
            System.out.println("Grade : " + this.grade);
            System.out.println("Class : " + this.classNum);
        }
    }

4)info.javaexpert.Worker.java
    package info.javaexpert;

    public class Worker {
        private String name;
        private int age;
        private String job;
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public int getAge() {
            return age;
        }
        public void setAge(int age) {
            this.age = age;
        }
        public String getJob() {
            return job;
        }
        public void setJob(String job) {
            this.job = job;
        }
        public void getWorkerInfo(){
            System.out.println("Name : " + this.name);
            System.out.println("Age : " + this.age);
            System.out.println("Job : " + this.job);
        }
    }

5)pom.xml : 아래 코드 추가
    <!-- AOP -->
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.8.10</version>
    </dependency>

6)info.javaexpert.LogAop.java
    package info.javaexpert;

    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(info.javaexpert.*)*")
        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="info.javaexpert.LogAop" />

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

         <bean id="student" class="info.javaexpert.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="info.javaexpert.Worker">
            <property name="name" value="설운도" />
            <property name="age" value="50" />
            <property name="job" value="개발자" />
         </bean>
      </beans>

   8)info.javaexpert.MainClass.java
      package info.javaexpert;

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

14. Lab

1)Aspect 클래스 정보

-클래스명 : LoggingAspect.java

-클래스 기능 : 이 Aspect 클래스는 4가지 유형의 Advice와 Pointcut을 설정하여 Target 객체의 파라미터와 리턴값, 예외 발생 시 예외 메시지를 출력하는 기능을 제공

-Advice 유형 : Before, AfterReturning, AfterThrowing, After

-구현 메소드명 : before(JoinPoint joinPoint), afterReturing(JoinPoint joinPoint, Object ret), afterThrowing(JoinPoint joinPoint, Throwable ex), afterFinally(JoinPoint joinPoint)

2)Aspect 클래스 선언 및 설정

-클래스 선언부에 @Aspect 어노테이션을 정의한다.

-이 클래스를 Aspect로 사용하려면 Bean으로 등록해야 하므로 @Component 어노테이션도 함께 정의한다.

```
package info.javaexpert;

import org.aspectj.lang.JoinPoint;

@Component
@Aspect
public class LoggingAspect {
...

<context:component-scan base-package="info.javaexpert" />
```

3)XML 설정파일에 <aop:aspectj-autoproxy /> 선언

-이 선언은 Bean으로 등록된 클래스 중에서 @Aspect가 선언된 클래스를 모두 Aspect로 자동 등록해주는 역할을 한다.

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

4)AopDemo4 Project 생성

-Spring Legacy Project > Simple Maven Project

5)info.javaexpert package 생성

-/src/main/java/info.javaexpert

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/info.javaexpert.LoggingAspect.java 생성

```
package info.javaexpert;

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;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;
@Component
@Aspect
public class LoggingAspect {
   @Before("execution(public * info.javaexpert..*(..))")
   public void before(JoinPoint joinPoint) {
      String signatureString = joinPoint.getSignature().getName();
      System.out.println("@Before [ " + signatureString + " ] 메서드 실행 전처리 수행");
      for (Object arg : joinPoint.getArgs()) {
         System.out.println("@Before [ " + signatureString + " ] 아규먼트 " + arg);
      }
   }
      @AfterReturning(pointcut="execution(public * info.javaexpert..*(..))", returning="ret")
   public void afterReturning(JoinPoint joinPoint, Object ret) {
      String signatureString = joinPoint.getSignature().getName();
      System.out.println("@AfterReturing [ " + signatureString + " ] 메서드 실행 후처리 수행");
      System.out.println("@AfterReturing [ " + signatureString + " ] 리턴값=" + ret);

   }

      @AfterThrowing(pointcut="execution(public * info.javaexpert..*(..))",
            throwing="ex")
   public void afterThrowing(JoinPoint joinPoint, Throwable ex) {
      String signatureString = joinPoint.getSignature().getName();
      System.out.println("@AfterThrowing [ " + signatureString + " ] 메서드 실행 중 예외 발생");
      System.out.println("@AfterThrowing [ " + signatureString + " ] 예외=" +
      ex.getMessage());
   }

      @After("execution(public * info.javaexpert..*(..))")
   public void afterFinally(JoinPoint joinPoint) {
      String signatureString = joinPoint.getSignature().getName();
      System.out.println("@After [ " + signatureString + " ] 메서드 실행 완료");
   }
}
```

8)beans.xml 파일 생성

-/src/main/resources/beans.xml

-Namespace Tab에서 'aop'와 'context' 체크할 것

```
<?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"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.2.xsd
      http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">

   <context:component-scan base-package="info.javaexpert" />
   <aop:aspectj-autoproxy />
</beans>
```

9)Target 객체 생성

-/src/main/java/info.javaexpert.Hello.java

```
package info.javaexpert;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("hello")
public class Hello {
   @Value("Spring")
   private String name;

   @Value("25")
   private int age;

   @Override
   public String toString() {
      return String.format("Hello [name=%s, age=%s]", name, age);
   }

   public void calculation(){
      System.out.println(5 / 0);
   }
}
```

10)테스트 클래스 작성

-src/main/java/info.javaexpert.MainClass.java

```
package info.javaexpert;

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

public class MainClass {
   public static void main(String[] args) {
      ApplicationContext ctx = new GenericXmlApplicationContext("classpath:beans.xml");

      Hello hello = ctx.getBean("hello", Hello.class);
      System.out.println(hello);
      hello.calculation();
   }
}
```

11)결과

```
@Before [ toString ] 메서드 실행 전처리 수행
@After [ toString ] 메서드 실행 완료
@AfterReturing [ toString ] 메서드 실행 후처리 수행
@AfterReturing [ toString ] 리턴값=Hello [name=Spring, age=25]
Hello [name=Spring, age=25]
@Before [ calculation ] 메서드 실행 전처리 수행
@After [ calculation ] 메서드 실행 완료
@AfterThrowing [ calculation ] 메서드 실행 중 예외 발생
@AfterThrowing [ calculation ] 예외=/ by zero
```

15. AspectJ Pointcut 표현식

1)표현식은 Pointcut 지시자를 이용하여 작성

2)가장 대표적인 지시자는 execution()이다.

3)Pointcut 을 지정할 때 사용하는 표현식으로 AspectJ 문법을 사용한다.

-* : 모든

-. : 현재

-.. : 0개 이상

4)execution

-Usage

execution([접근제한자 패턴] 리턴타입패턴 [타입패턴.] 이름패턴(파라미터타입패턴 | "..", ...) [throws 예외패턴])
--접근제한자 패턴 : public, private과 같은 접근 제한자, 생략가능
--리턴타입패턴 : 리턴값의 타입 패턴
--타입패턴 : 패키지와 클래스 이름에 대한 패턴, 생략가능. 사용할 때 "."를 사용해 연결함.
--이름패턴 : 메소드 이름 타입 패턴
--파라미터 타입패턴 : 파라미터의 타입 패턴을 순서대로 넣을 수 있다. 와일드카드를 이용해서 파라미터 갯수에 상관없는 패턴을 만들 수 있다.
--예외패턴 : 예외 이름 패턴

-예
"execution(* aspects.trace.demo.*.*(..))"
-* : Any return type
-aspects.trace.demo : package
-* : class
-* : method
-(..) : Any type and number of arguments

-execution(* hello(..))
--hello라는 이름을 가진 메소드를 선정
--파라미터는 모든 종류를 다 허용

-execution(* hello())
--hello 메소드 중에서 파라미터가 없는 것만 선택함.

-execution(* info.javaexpert.service.UserServiceImpl.*(..))
--info.javaexpert.service.UserServiceImpl 클래스를 직접 지정
--이 클래스가 가진 모든 메소드를 선택

-execution(* info.javaexpert.user.service.*.*(..))
--info.javaexpert.user.service 패키지의 모든 클래스에 적용
--하지만 서브패키지의 클래스는 포함하지 않는다.

-execution(* info.javaexpert.user.service..*.*(..))
--info.javaexpert.user.service 패키지의 모든 클래스에 적용
--그리고 '..'를 사용해서 서브패키지의 모든 클래스까지 포함

-execution(* *.. Target.*(..))
--패키지에 상관없이 Target이라는 이름의 모든 클래스에 적용
--다른 패키지의 같은 이름의 클래스가 있어도 적용이 된다는 점에 유의해야 함.

@Pointcut("executeion(public void get*(..))") : public void인 모든 get메소드
@Pointcut("executeion(* info.javaexpert.*.*())") : info.javaexpert 패키지에 파라미터가 없는 모든 메소드
@Pointcut("executeion(* info.javaexpert..*.*())") : info.javaexpert 패키지 & kr.co.javaexpert 하위 패키지에 파라미터가 없는 모든 메소드
@Pointcut("executeion(* info.javaexpert.Worker.*())") : info.javaexpert.Worker 안의 모든 메소드

2)within
@Pointcut("within(info.javaexpert.*)") : info.javaexpert 패키지 안에 있는 모든 메소드
@Pointcut("within(info.javaexpert..*)") : info.javaexpert 패키지 및 하위 패키지 안에 있는 모든 메소드
@Pointcut("within(info.javaexpert.Worker)") : info.javaexpert.Worker 모든 메소드

3)bean
@Pointcut("bean(student)") : student 빈에만 적용
@Pointcut("bean(*ker)") : ~ker로 끝나는 빈에만 적용