

5월 6일 시험 준비

MEMEBER DAO 만들면 @ 달아야 할 것

```
FeedDaoImpl.java
1 package com.kh.gw.feed.model.dao;
2
3 import java.util.List;
4
14
15 @Repository
16 public class FeedDaoImpl implements FeedDao{
17
18     @Override
19     public int insertFeed(SqlSessionTemplate sqlSession, FeedBoard f) {
20         return sqlSession.insert("Feed.insertFeed", f);
21     }
22
23     @Override
24     public int insertFeedAttachment(SqlSessionTemplate sqlSession, Attachment at) {
25         return sqlSession.insert("Feed.insertFeedAttachment", at);
26     }
27 }
```

DAO에서 AUTOWARIED 해야할 것 (DAO에는 없어서 Service 첨부)

```
FeedDaoImpl.java  FeedServiceImpl.java
1 package com.kh.gw.feed.model.service;
2
3 import java.util.List;
4
16
17 @Service
18 public class FeedServiceImpl implements FeedService{
19
20     @Autowired
21     private SqlSessionTemplate sqlSession;
22
23     @Autowired
24     private FeedDao fd;
25
26     @Override
27     public int insertFeed(FeedBoard f) {
28         return fd.insertFeed(sqlSession, f);
29     }
30 }
```

POM.XML DEPENDENCY에 작성하는 방법

<dependencies>와 <dependency>

기본적으로 생성된 pom.xml의 기술에서 하나 뿐인 매우 태그가 복잡한 부분이 있었다. <dependencies>라는 태그이다.

이것은 이미 언급한 바와 같이 의존 라이브러리 정보를 기술해 두기 위한 것이다. 하지만, 이 기술하는 태그가 많은 계층적으로 되어 있기 때문에, 상당히 이해하기 어려울지도 모른다. 이 태그는 기본적으로 다음과 같은 형태로 기술되어 있다.

```
<dependencies>
  <dependency>... 생략 ...</dependency>
  <dependency>... 생략 ...</dependency>
  ..... 중략 .....
</dependencies>
```

<dependencies>은 의존 라이브러리를 한곳에 모아 기술하기 위한 것이다. 각각의 의존 라이브러리 정보는 <dependency> 태그를 사용하여 작성한다. 이 <dependency> 태그를 필요한만큼 <dependencies> 태그 안에 기술한다.

<dependency> 태그

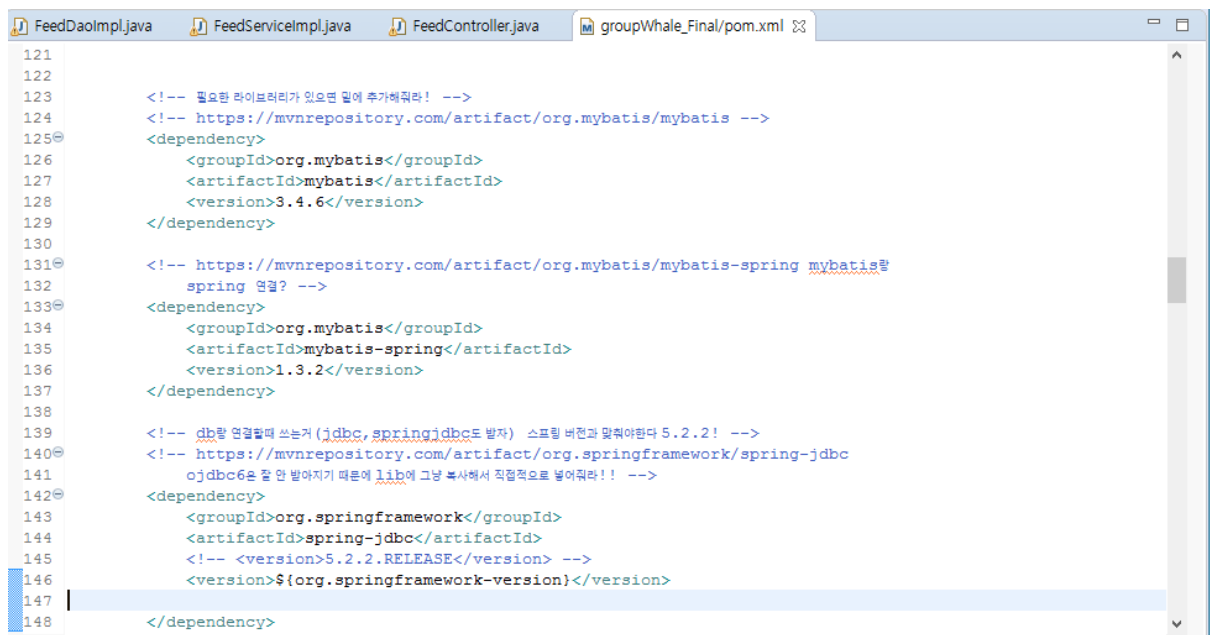
라이브러리 정보를 기술하는 <dependency> 태그는 태그 안에 몇 가지 정보를 기술하고 라이브러리 지정한다. 정리하면 다음과 같은 형태이다.

```
<dependency>
  <groupId>그룹 ID</groupId>
  <artifactId>아티팩트 ID</artifactId>
  <version>버전</version>
  <scope>범위</scope>
</dependency>
```

그룹 ID, 아티팩트 ID 및 버전은 앞전에 프로젝트와 동일하다. 라이브러리들도 모든 Maven 프로그램으로 만드는 이상, 그룹 ID 및 아티팩트 ID가 설정되어 있다. 이렇게 지정함으로써 어떤 라이브러리를 사용하는지를 알 수 있다.

단, <scope>라는 것이 이해하기 어려울지도 모른다. 이것은 이 라이브러리가 이용되는 범위를 지정하는 것이다. 범위라고 하면 이해하기 어려울 수도 있지만, "어떤 때 사용하는지"를 나타내는 것이다. 이 응용 프로그램을 실행할 때 사용하는 경우는 특별히 지정하지 않아도 된다. 특별한 경우에만 준비하는 것이라고 생각해도 좋을 것이다.

- 예시

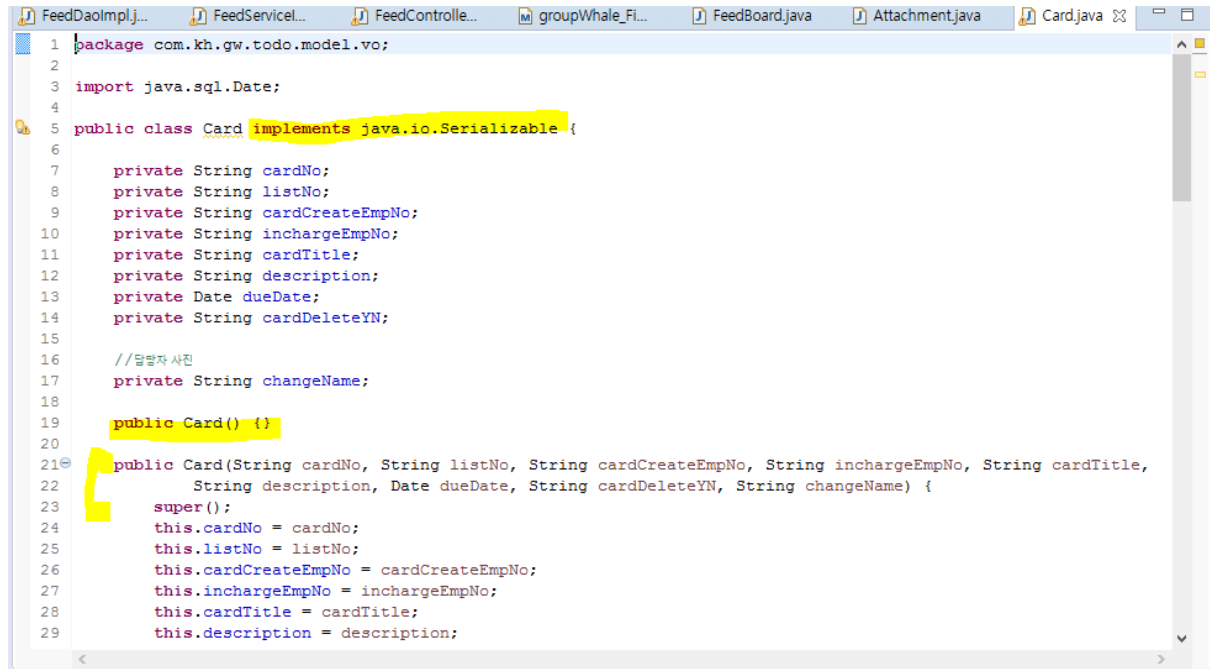


```
121
122
123      <!-- 필요한 라이브러리가 있으면 밑에 추가해줘라! -->
124      <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
125      <dependency>
126        <groupId>org.mybatis</groupId>
127        <artifactId>mybatis</artifactId>
128        <version>3.4.6</version>
129      </dependency>
130
131      <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring mybatis랑
132            spring 연결? -->
133      <dependency>
134        <groupId>org.mybatis</groupId>
135        <artifactId>mybatis-spring</artifactId>
136        <version>1.3.2</version>
137      </dependency>
138
139      <!-- db랑 연결할때 쓰는거 (jdbc, springjdbc도 발자) 스프링 버전과 맞춰야한다 5.2.2! -->
140      <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc
141            ojdbc6은 잘 안 받아지기 때문에 lib에 그냥 복사해서 직접적으로 넣어줘라!! -->
142      <dependency>
143        <groupId>org.springframework</groupId>
144        <artifactId>spring-jdbc</artifactId>
145        <!-- <version>5.2.2.RELEASE</version> -->
146        <version>${org.springframework-version}</version>
147      </dependency>
148
```

서술형

VO 객체 만들고 해야하는 작업

- Serializable
- 생성자 2개
- getter, setter



```
1 package com.kh.gw.todo.model.vo;
2
3 import java.sql.Date;
4
5 public class Card implements java.io.Serializable {
6
7     private String cardNo;
8     private String listNo;
9     private String cardCreateEmpNo;
10    private String inchargeEmpNo;
11    private String cardTitle;
12    private String description;
13    private Date dueDate;
14    private String cardDeleteYN;
15
16    //담당자 사진
17    private String changeName;
18
19    public Card() {}
20
21    public Card(String cardNo, String listNo, String cardCreateEmpNo, String inchargeEmpNo, String cardTitle,
22                String description, Date dueDate, String cardDeleteYN, String changeName) {
23        super();
24        this.cardNo = cardNo;
25        this.listNo = listNo;
26        this.cardCreateEmpNo = cardCreateEmpNo;
27        this.inchargeEmpNo = inchargeEmpNo;
28        this.cardTitle = cardTitle;
29        this.description = description;
```

DATA SOURCE 만들때 URL, USERNAME, PASSWORD 작성하는 부분

```

<!-- DB에 연결할 설정에 대한 정보를 선언하는 부분 -->
<!-- default : 연결 설정을 여러 개 생성하여 아이디로 구분하고, 기본으로 연결할 설정 정보를 지정하는 속성 -->
<environments default="firstDev">
    <environment id="firstDev">
        <!-- 트랜잭션 매니저는 JDBC와 MANAGED 둘 중에 하나를 선택할 수 있음 -->
        <!--
            JDBC는 JDBC가 commit과 rollback의 기능을 직접 사용 가능하게 하는 옵션이다. (수동 commit)
            MANAGED는 트랜잭션에 대해 어떤 영향도 행사하지 않는다는 의미이다. (자동 commit)
        -->
        <transactionManager type="JDBC"/>

        <!-- DB접속에 관한 정보들을 넣는 태그이며, type 속성은 ConnectionPool을 사용할 것인지에 대한 여부임 -->
        <!-- POOLED와 UNPOOLED를 사용 가능함 -->
        <dataSource type="POOLED">
            <property name="driver" value="oracle.jdbc.driver.OracleDriver"/>
            <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
            <property name="username" value="mybatis"/>
            <property name="password" value="mybatis"/>
        </dataSource>
    </environment>
</environments>

```

POM.XML 태그 중에 라이브러리 구분하고 역할 작성하는 것 (예> MYBATIS, MYBATIS SPRING, SPRING JDBC 에 대해 설명)

- MyBais : SQL Mapper 라이브러리, jdbc만을 사용해서 작업할 때보다 코드를 상당히 많이 줄어들어서 개발 속도를 향상시킴
- MyBatis-Spring : 스프링과 MyBatis를 연결시킴
- Spring-jdbc : 자바와 DB의 연결시킴. DataSource에 필요한 클래스를 제공 (DataSource는 JDBC의 커넥션을 처리하는 기능을 가지고 있기 때문에 데이터베이스와 연동 작업에 반드시 필요)

MyBatis란?

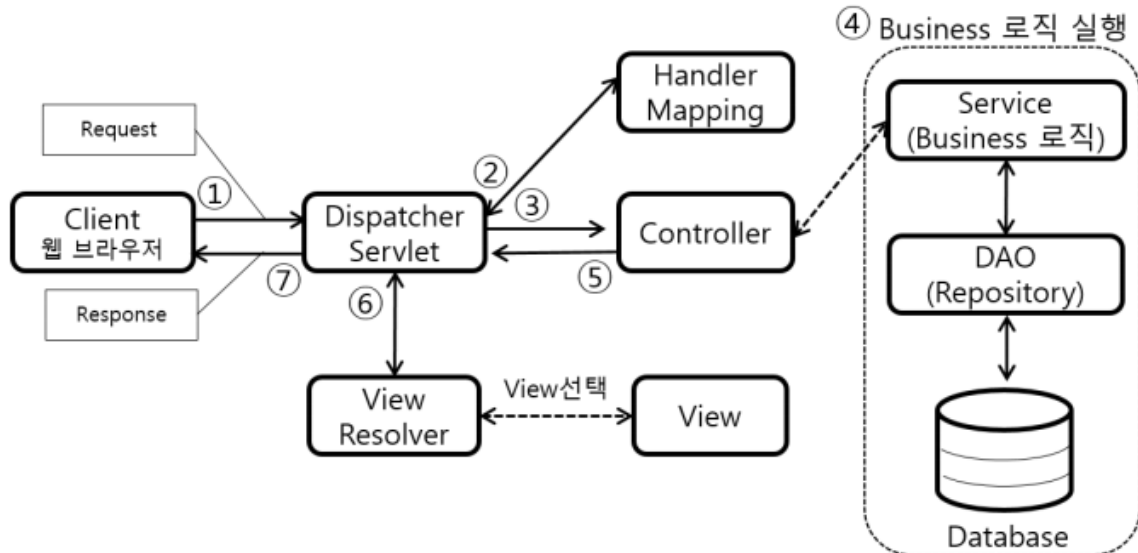
참고

1. 객체 지향 언어인 자바의 관계형 데이터베이스 프로그래밍을 보다 쉽게 도와주는 프레임워크
2. 자바에서는 관계형 데이터베이스 프로그래밍을 하기위해 JDBC를 제공
 - *JDBC(Java Data Connectivity): 자바 프로그래밍이 데이터베이스와 연결되어 데이터를 주고 받을 수 있게 해주는 프로그래밍 인터페이스이다(DriverClass, Connection, PerparedStatement, ResultSet etc..)
3. JDBC는 다양한 관계형 데이터베이스 프로그래밍을 위해 API 제공.

결론: MyBatis는 JDBC를 보다 편하게 사용하기 위해 개발되었다.

SPRING MVC 순서(그림 외우면 쓸수 있음)

Spring MVC 요청 처리 과정



Spring MVC 구성 요소

구성 요소	설명
Dispatcher Servlet	클라이언트의 요청을 전달받음 요청에 맞는 컨트롤러가 리턴한 결과값을 View에 전달하여 알맞은 응답을 생성
Handler Mapping	클라이언트의 요청 URL을 어떤 컨트롤러가 처리할지 결정
Controller	클라이언트의 요청을 처리한 뒤, 결과를 DispatcherServlet에게 리턴
Model AndView	컨트롤러가 처리한 결과 정보 및 뷰 선택에 필요한 정보를 담음
ViewResolver	컨트롤러의 처리 결과를 생성할 View를 결정
View	컨트롤러의 처리 결과 화면을 생성, JSP나 Velocity 템플릿 파일 등을 View로 사용

CONTROLLER에서 REQUEST PARAM 과 REQUEST BODY라는 어노테이션 에 대해 기술

@RequestParam

또다른 방법으로는 `@RequestParam` 어노테이션을 이용하면 간단하게 파라미터값을 가져올수 있다. 우선, 해당 어노테이션의 옵션값들에 대해 간략하게 확인하고 넘어가는데 좋을듯 싶다. [API문서 4.3.6 기준](#)

이름	타입	설명
name, value (Alias for name)	String	파라미터 이름
required	boolean	해당 파라미터가 반드시 필수인지 여부, 기본값은 true
defaultValue	String	해당 파라미터의 기본값

위 옵션값들을 조합하여 컨트롤러 메소드에 적용해보면 아래 소스와 같이 만들어지고, 이렇게 request에서 파라미터값을 가져올수 있다.

```
1 @RequestMapping("/")
2 public String home(@RequestParam(value="id", defaultValue="false") String id) {
3     return "home";
4 }
```

이 어노테이션을 이용하게되면 자칫 잘못하다간 에러를 만날수가 있는데 `required` 값을 true로 해놓고 (필수 파라미터 설정) 해당 파라미터를 사용하지 않고 요청을 보내게 되면 HTTP 400 에러를 받게 되니 각 옵션들을 정확히 확인하고 사용해야 할 것 같다.

물론 컨트롤러의 메소드에서 해당 어노테이션을 사용하지 않고도 아래 코드처럼 바로 받을수 있다. 이렇게 바로 받을 경우는 필수 파라미터값이 false로 설정이 되고 변수명과 동일한 파라미터만 받을수 있게 되며 기본값 설정을 할수는 없다. 방법의 차이라서 상황에 따라 맞춰 사용하면 될듯 하다.

```
1 @RequestMapping("/")
2 public String home(String id) {
3     return "home";
4 }
```

@RequestBody

`@RequestBody` 어노테이션을 사용할 경우 반드시 POST형식으로 응답을 받는 구조여야만 한다. 이를테면 JSON 이나 XML같은 데이터를 적절한 `messageConverter`로 읽을때 사용하거나, POJO형태의 데이터 전체로 받을경우에 사용된다. 단, 이 어노테이션을 사용하여 파라미터를 받을 경우 별도의 추가 설정(POJO 의 get/set 이나 json/xml 등의 `messageConverter` 등)을 해줘야 적절하게 데이터를 받을수가 있다.

```
1 @PostMapping("/")
2 public String home(@RequestBody Student student) {
3     return "home";
4 }
```

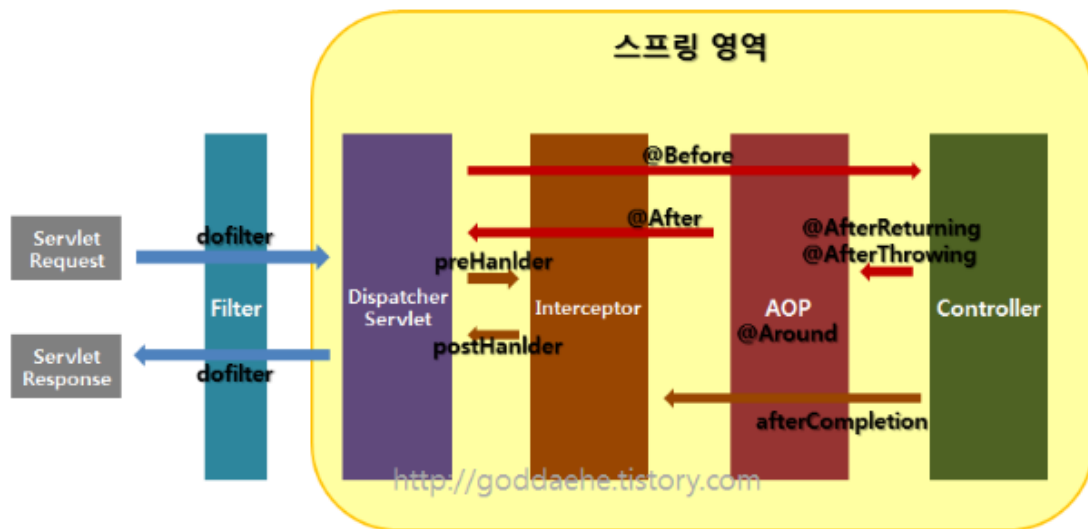
FILTER와 INTERCEPTER의 차이점

[정리]

둘다 컨트롤러가 호출되기 전에 호출되어 실행된다. 하지만 Filter는 Dispatcher가 호출되기전에, Interceptor는 Dispatcher가 호출되고 난뒤 호출된다.

필터는 일반적으로 서블릿 컨테이너에서 제어하기 때문에 web.xml에서 정의하여야 하며, 스프링 애플리케이션에서 관리하는 빈을 DI로 받을 수 없다.

인터셉터는 dispatcher-servlet.xml 에서 컨트롤러에 대해 인터셉터를 정의하며, 스프링 애플리케이션에서 관리하는 빈을 DI로 받아서 사용한다.



1. Filter(필터)

말그대로 요청과 응답을 거른뒤 정제하는 역할을 한다.

서블릿 필터는 DispatcherServlet 이전에 실행이 되는데 필터가 동작하도록 지정된 자원의 앞단에서 요청내용을 변경하거나, 여러가지 체크를 수행할 수 있다.

또한 자원의 처리가 끝난 후 응답내용에 대해서도 변경하는 처리를 할 수가 있다.

보통 web.xml에 등록하고, 일반적으로 인코딩 변환 처리, XSS방어 등의 요청에 대한 처리로 사용된다.

2. Interceptor(인터셉터)

요청에 대한 작업 전/후로 가로챈다고 보면 된다.

필터는 스프링 컨텍스트 외부에 존재하여 스프링과 무관한 자원에 대해 동작한다.

하지만 인터셉터는 스프링의 DispatcherServlet이 컨트롤러를 호출하기 전, 후로 끼어들기 때문에 스프링 컨텍스트(Context, 영역) 내부에서 Controller(Handler)에 관한 요청과 응답에 대해 처리한다.

스프링의 모든 빈 객체에 접근할 수 있다.

인터셉터는 여러 개를 사용할 수 있고 로그인 체크, 권한체크, 프로그램 실행시간 계산작업 로그확인 등의 업무처리

INTERCEPTER에 3가지 시점에 대해 기술

- 1.요청하자마자 동작 (서블릿으로 도착하고 매핑된 메소드로 가기 직전에) - preHandle
- 2.응답되고 동작 (View화면 보이기전) - PostHandle
- 3.응답이 View화면이 보이고 난 이후 동작 - afterCompletion

LOG4J 안에 들어가는 태그들 설명

중요 컴포넌트 설명

1. Logger : 로그의 주체 (로그 파일을 작성하는 클래스) - Log4j의 심장부에 위치하며, 개발자가 로그출력 여부를 런타임에 조정되도록 해준다.

로거는 로그레벨을 가지고 있으며, 로그의 출력여부는 로그문의 레벨과 로거의 레벨을 가지고 결정된다.

- 어플리케이션을 작성하기전 어떤 로거를 사용해야 할지 정해야 한다. ex) static
Logger logger = Logger.getLogger(SimpleLog.class);

[참고] Commons-Logging 는 레퍼클래스도 존재함

2. Appender : 로그를 출력하는 위치

- 로그를 출력하는 위치를 의미하며,

Log4J API문서의 XXXAppender로 끝나는 클래스들의 이름을 보면, 출력위치를 어느정도 짐작할 수 있다.

3. Layout : Appender의 출력포맷 - 일자, 시간, 클래스명등 여러가지 정보를 선택하여 로그정보내용으로 지정할 수 있다.

패턴 레이아웃 설명

ex) "[%d{yyyy-MM-dd HH:mm:ss}] %-5p [%l] - %m%n

C : 클래스명을 출력한다. {1}과 같은 설정을 추가하여 클래스 이름 또는 특정 패키지 이상만 출력하도록 설정할 수 있다.

d : 로그 시간을 출력한다. java.text.SimpleDateFormat에서 적절한 출력 포맷을 지정할 수 있다.

F : 파일 이름을 출력한다. 로그시 수행한 메소드, 라인번호가 함께 출력된다.

L : 라인 번호만 출력한다.

m : 로그로 전달된 메시지를 출력한다.

M : 로그를 수행한 메소드명을 출력한다.

n : 줄 바꿈

p : 로그 이벤트명 (DEBUG등)

r : 로그 처리시간 (milliseconds)

AOP ADVICE 시점 4가지 이상 쓰기

- @Before (이전) : 어드바이스 타겟 메소드가 호출되기 전에 어드바이스 기능을 수행
- @After (이후) : 타겟 메소드의 결과에 관계없이(즉 성공, 예외 관계없이) 타겟 메소드가 완료 되면 어드바이스 기능을 수행
- @AfterReturning (정상적 반환 이후)타겟 메소드가 성공적으로 결과값을 반환 후에 어드바이스 기능을 수행
- @AfterThrowing (예외 발생 이후) : 타겟 메소드가 수행 중 예외를 던지게 되면 어드바이스 기능을 수행
- @Around (메소드 실행 전후) : 어드바이스가 타겟 메소드를 감싸서 타겟 메소드 호출전과 후에 어드바이스 기능을 수행

용 어	설 명	
Aspect	여러 객체에 공통으로 적용되는 기능을 분리하여 작성한 클래스	
Joinpoint	클래스의 객체(인스턴스) 생성 지점, 메소드 호출 시점, 예외 발생 시점 등 특정 작업이 시작되는 시점	
Advice	Joinpoint에 삽입되어 동작될 코드, 메소드	
	Before advice	Joinpoint 앞에서 실행
	Around Advice	Joinpoint 앞과 뒤에서 실행
	After Advice	Joinpoint 호출이 리턴되기 직전에 실행
	After Returning Advice	Joinpoint 메소드 호출이 정상적으로 종료된 후에 실행
	After Throwing Advice	예외가 발생했을 때 실행