

## [문서 이용 전 요구사항: 실습을 위한 PC 필수 조건, 새로운 Windows 사용자 생성]

본 문서는 다음과 같은 사양이 구성된 PC로 모든 실습이 진행되었습니다.

- 메모리 크기: 8GB 사용됨

→ 실습 후반부에서 생성된 프로그램이 증가될수록 사용되는 메모리가 많아져 8GB 의 메모리도 부족하게 느껴집니다.

→ 프로그램 실행이나 개발 환경 실행이 느려짐

- 하드디스크 크기: 300GB 용량의 SSD 하드디스크가 사용됨

→ 개발 환경 구성과 프로그래밍 실습을 위해 20GB 이상의 빈 공간이 있으면 충분합니다.

- CPU: Intel(R) Core(TM) i5 CPU 760 @2.80GHz 프로세서가 사용됨

→ i5 이상의 CPU 면 충분함. 단 i3 CPU의 경우에도 실습은 무리없이 진행되지만, 처리속도가 다소 느립니다.

- 운영체제: **Windows 10 Pro 64비트**

→ 8GB 이상의 메모리를 사용하려면, 운영체제가 64비트가 되어야 합니다.

- 본 문서는 Windows 10 64 비트 운영체제를 사용하는 것을 기준으로 모든 실습을 진행합니다.

☞ Windows 운영체제 이 외의 애플 운영체제(iOS) PC나 리눅스 운영체제 PC를 사용하는 경우에는,

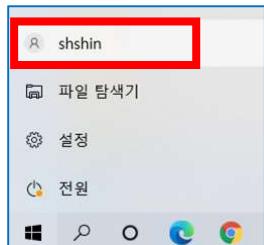
실습에서 차이난 부분은 스스로 웹 서칭 등을 통해 해결하시기 바랍니다.

특히, 오라클 데이터베이스 서버는 애플 PC에는 설치할 소프트웨어가 제공되지 않습니다.

- **Windows 운영체제 PC를 사용하여 개발환경 구성 시에 다음의 요구사항을 반드시 먼저 구현해야 합니다(중요!!!).**

→ 운영체제 사용자 확인: 시작(윈도우 로고) 클릭 → 사용자 모양 아이콘에 마우스 위치(클릭하지 말것)

→ 아래 그림처럼 현재 Windows 운영체제에 로그인 한 사용자이름이 표시됩니다.



→ 사용자이름이 그림처럼 **하나의 영어단어** 이어야만 합니다. 이 요구사항이 충족된 경우, 아래의 내용은 수행하지 않습니다.

사용자이름이 한글(예, 신상현)이거나, 2개 이상의 단어 (Shin SangHyun)인 경우, 실습 환경 구성 시에 설치 프로그램 실행이 않되는 등 문제가 발생됩니다. 특히 오라클 RDBMS를 설치 시에 문제가 많이 발생됩니다.

→ 따라서, Windows 운영체제의 사용자이름에 한글이 포함되거나 또는 2개 이상의 단어로 된 경우에는,

Windows 설정에서 새로운 사용자를 생성하여 관리자 권한을 부여한 후 사용해야 합니다.

**기존 사용자이름을 바꾸는 방법은 사용될 수 없습니다.!!!**

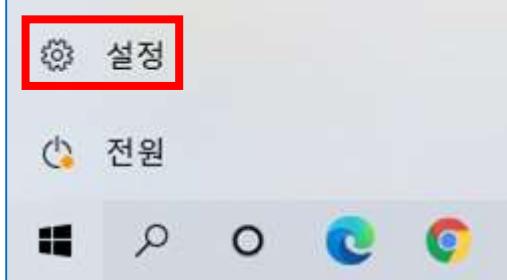
**반드시 하나의 영어 단어로 된 이름으로 새로운 사용자를 생성하고,**

**생성된 사용자로 로그인 한 후, 개발환경 구성부터 문서의 실습을 수행해야 합니다.**

→ 새로운 Windows 운영체제 사용자 생성(Windows 10 기준)

☞ Windows 10 업데이트 버전에 따라, 화면의 표시내용이 조금씩 다를 수 있지만, 클릭해야 하는 부분은 동일합니다.

→ 작업 표시줄 제일 왼쪽의 원도우 로고 클릭 → 톱니바퀴 모양의 아이콘(설정) 클릭



→ Windows 설정 창에서 [계정] 클릭 → 계정 창에서 왼쪽에 있는 가족 및 다른 사용자 클릭 → [이 PC에 다른 사용자 추가] 클릭

The left window shows the Windows Settings interface. The 'Account' section is highlighted with a red box. The right window shows the 'Family & other users' section with the 'Add another user for this PC' button highlighted with a red box.

→ [이 사람의 로그인 정보를 가지고 있지 않습니다] 클릭 → [Microsoft 계정 없이 사용자 추가] 클릭

→ 사용자 이름, 보안 암호 및 3개의 보안 질문을 모두 입력하여 설정 한 후, [다음] 클릭

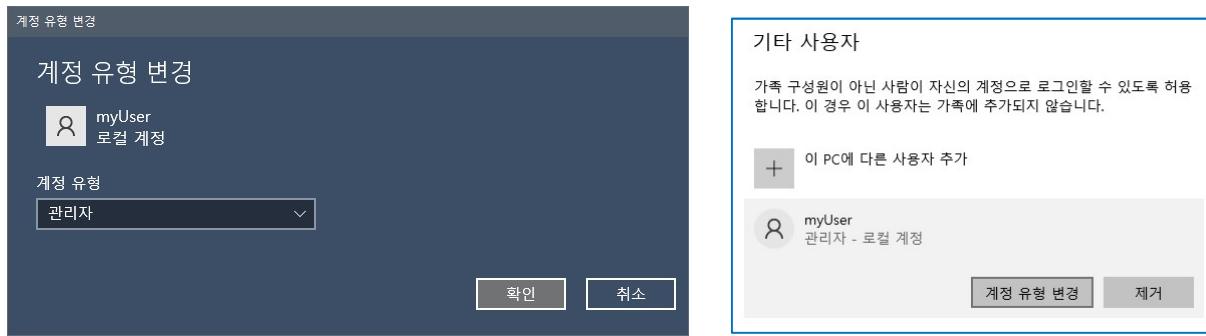
The three windows show the Microsoft Account setup process. The first window shows the 'Who's this person?' step with the 'This person doesn't have a Microsoft account' link highlighted with a red box. The second window shows the 'Create account' step with the 'Create account without Microsoft account' button highlighted with a red box. The third window shows the 'Create account for this PC' step with the 'Next' button highlighted with a red box.

→ 계정 창이 표시되면, 오른쪽의 기타 사용자에 표시된 생성된 사용자를 클릭 → 표시된 계정 유형 변경 버튼 클릭

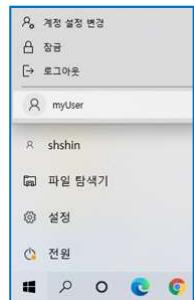
The window shows the 'Other users' section with a user named 'myUser' listed. The 'Change account type' button next to it is highlighted with a red box.

→ 계정 유형 변경 창에서 계정 유형을 표준 사용자에서 관리자로 변경 → 확인 클릭

→ 계정 창이 표시되고, 아래 그림처럼 관리자로 계정이 설정됩니다. → 설정 창을 닫습니다.



→ 작업표시줄 제일 왼쪽의 원도우 로고 클릭 → 사용자 모양 아이콘 클릭 → 생성된 사용자 클릭



→ 다시 로그인 하는 화면이 표시되면, 새로 생성한 하나의 영단어로 된 사용자로 로그인 합니다.

→ 이 후 부터는 이 사용자로 Windows 에 로그인하여 문서의 실습을 수행합니다.

본 문서는, HTML5, JavaScript(jQuery), Java, Servlet-JSP, SQL, Spring Framework에 관한 기본 프로그래밍 지식이 있는 분들이 Eclipse 와 STS IDE를 이용하여 Spring-Framework 5 버전을 기반한 게시판 서비스를 구축하는 실습을 위한 내용들로 구성되어 있습니다. 따라서, 주제와 관련된 이론적인 설명은 포함되어 있지 않습니다.

## [목차]

### [개발 환경 구성]

#### [DE-01] [Java Development Toolkit 8 설치 및 운영체제 환경변수 설정](#)

1. Java Standard Edition Development Kit 8 다운로드 및 설치
2. Java Standard Edition Development Kit 8 운영체제 환경변수 구성

#### [DE-02] [이클립스 IDE\(Integrated Development Environment\) 설치](#)

#### [DE-03] [이클립스에 스프링 프레임워크 소프트웨어 설치](#)

#### [DE-04] [이클립스에 롬복\(Lombok\) 소프트웨어 설치](#)

#### [DE-05] [이클립스 워크스페이스 생성 및 UTF-8 인코딩 설정](#)

1. 이클립스 워크스페이스 폴더 생성
2. 워크스페이스를 JDK 8 사용 구성으로 변경(2021-03(4.19.0) 버전부터 필요)
3. 워크스페이스에 UTF-8 인코딩 적용

#### [DE-06] [톰캣 9 웹 애플리케이션 서버 설치](#)

#### [DE-07] [이클립스 워크스페이스에서 톰캣 사용 구성](#)

1. 이클립스 워크스페이스에서 톰캣 9 사용 구성: 이클립스에서 사용된 서버 자원 추가.
2. 톰캣 9를 서버 프로젝트로 등록: 프로젝트와 톰캣 9 서버 연동 준비.

#### [DE-08] [오라클 데이터베이스 11g R2 Express Edition 설치\(RDBMS 및 테스트 데이터베이스 생성 포함\)](#)

1. 오라클 데이터베이스 11gR2 Express Edition 다운로드 및 설치
2. 설치 확인 및 방화벽 예외 프로그램으로 등록
3. 오라클 XDB 서비스 포트 변경(8080 → 0) 및 서비스 사용 메모리 변경

#### [DE-09] [오라클 SQL\\*Developer 설치](#)

1. 오라클 SQL\*Developer 다운로드(JDK 포함되지 않은 버전 다운로드), 설치 및 실행
2. 오라클 데이터베이스 관리자 계정인 SYS 계정과 SYSTEM 계정의 XE 오라클 데이터베이스 서비스 접속 구성

### [프로젝트 준비]

#### [PP-01] [Spring Legacy Project 생성 및 기본구성](#)

1. Spring Legacy Project(Spring MVC Project Template 적용) 생성
2. 생성된 프로젝트 기본 구성 변경(Spring 5 기준)
3. 실습을 위해 필요한 추가 라이브러리 구성
4. 스프링 UTF-8 인코딩 필터(한글처리)

## [PP-02] 커넥션 풀, MyBatis 구성 및 오라클 데이터베이스와 접속 구성

1. 오라클 데이터베이스 접속 계정 생성
2. 커넥션 풀(Connection Pool)을 이용한 오라클 데이터베이스 서비스와의 접속 구성
  - JDBC 접속을 위한 정보가 저장된 properties 파일 생성
3. 스프링 빈 구성 설정파일 생성: mybatis-context.xml 파일
4. Mybatis, mybatis-spring 사용 구성
5. log4jdbc-log4j2 라이브러리를 통한 SQL 처리 로그 표시 구성
6. 오라클 데이터베이스 접속 및 SQL문 처리 관련 클래스들의 빈 설정 구성
7. 구성사항 테스트: 테스트 클래스 작성

## [프로젝트 수행: 게시판 작성]

### [PD-03] 필요한 패키지 생성

1. 프로젝트 구성 클래스들이 저장될 패키지 생성

### [PD-04] 영속 계층(Persistence-Layer) 구현

1. 데이터베이스에 게시물 저장 구조(테이블, 인덱스, 시퀀스 등) 생성 및 테스트 데이터 입력
2. 도메인(VO) 클래스 생성
3. 매퍼-인터페이스 및 매퍼-XML 파일 생성 및 매퍼 사용 구성
  - 게시물-목록표시/게시물-등록/게시물-조회/게시물-수정/게시물-삭제 관련 구성
4. 테스트 수행

### [PD-05] 비즈니스 계층(Business-Layer) 구현: Service 인터페이스와 Service 구현-클래스

1. 서비스 인터페이스와 서비스 구현클래스 생성
  - 게시물-목록표시/게시물-등록/게시물-조회/게시물-수정/게시물-삭제 비즈니스 처리 메서드
2. 테스트 수행

### [PD-06] 제어 계층 구현: Controller 클래스

1. 컨트롤러 클래스 생성
  - 게시물 목록 표시/게시물 등록/게시물 조회/게시물 수정/게시물 삭제 요청 처리 메서드
2. 테스트 수행

### [PD-07] 프레젠테이션 계층 구현: JSP-뷰 페이지 생성(화면 구현, 디자인은 BootStrap 이용)

1. 공통 디자인 Bootstrap Template 구성
2. header/footer JSP 파일을 구성(각 JSP 파일에서 @include 지시자를 이용하여 설정)
3. JSP 페이지 생성: 게시물-목록표시/게시물-등록/게시물-세부정보 표시/게시물-수정 및 삭제
4. 페이지 이동 구성: 게시물 목록 페이지 → 특정 게시물 세부정보 페이지 → 게시물-수정-삭제 페이지

### [PD-08] 게시물 목록 페이지: 페이징 구현

1. 페이징 데이터 준비(SQL\*Developer에서 작업)
2. 페이징 데이터 저장 객체(DTO) 생성(MyBoardPagingDTO 클래스)
3. MyBoardMapper.xml 파일과 MyBoardMapper 인터페이스 수정
4. MyBoardService와 MyBoardController 수정
5. 게시물 목록화면 페이징 처리 구현

## 6. 화면 이동간 페이징 정보 유지 구현

### [PD-09] [게시물 목록 페이지: 게시물 검색 구현](#)

1. 대표적인 게시물 검색 기준 확인 및 검색 기능 구현 시 고려사항
2. 매퍼 XML 파일에 검색을 위한 SQL 작성: MyBatis의 동적 태그 이용
3. 검색기능 구현
4. 화면 이동간 검색-페이징 유지 구현

### [PD-10] [게시물 댓글처리 - REST-API와 jQuery의 Ajax 이용](#)

1. REST-API 개요 및 Spring의 REST-API 간단 소개
2. 댓글처리를 위한 데이터베이스 구성
3. 댓글처리를 위한 도메인 클래스 생성
4. 댓글처리를 위한 매퍼 XML 파일 및 매퍼 인터페이스 생성 및 테스트
5. 비즈니스 계층(서비스)과 제어계층(컨트롤러) 구현 밀 테스트
6. 화면 구성
  - JavaScript 클로저(Closure) 개요
  - JavaScript ajax를 이용한 데이터 이동 구현
  - 댓글 화면 구성
  - 테스트

### [PD-11] [파일 업로드 처리](#)

1. 톰캣 내장 업로드 라이브러리를 이용한 업로드 구성
2. form을 이용한 업로드와 Ajax를 이용한 업로드 개요 실습
3. ajax 업로드를 기반한 프로젝트 업로드 구성

### [PD-12] [스프링 시큐리티](#)

1. 영속계층 구현: 인증 및 승인 시에 필요한 데이터 유지 및 데이터베이스 액세스 구성
  - 데이터베이스 저장구조 구성
  - VO 클래스 생성 및 Mybatis 매퍼 구성
2. 스프링 시큐리티 기본구성:
  - 스프링 시큐리티 라이브러리 설치
  - 승인/인증을 위해 필요한 데이터 사용 구성: 데이터 사용을 위해 필요한 스프링 시큐리티 빈 클래스 생성
  - 스프링 시큐리티 컨텍스트 설정파일 생성 및 설정 구성
  - 사용자 암호에 대한 스프링 시큐리티 암호화 구성
  - 스프링 시큐리티 어노테이션 사용 활성화 설정.
3. 스프링 시큐리티 활용
  - 로그인/로그아웃 처리
    - 커스텀 로그인/로그아웃 JSP 페이지 사용 구성
    - 로그인/로그아웃 후, 특정 URL 페이지로 이동 구성
  - 권한 부족으로 인한 접근 금지 오류 시의 처리 로직 구현
    - AccessDeniedHandler 구현 클래스 이용
  - 로그인 성공 후, 수행될 내용 구성: 관련 빈 클래스 생성 및 설정
  - 스프링 시큐리티 어노테이션과 스프링 시큐리티 표현식을 이용한 액세스 제어 구현

# [개발 환경 구성]

- 본 파트에서는 자신의 PC에 "Spring Framework 5 기반의 자바 웹 애플리케이션 개발"을 위한 개발 환경을 구성하는 방법을 설명합니다.
  - 본 문서의 실습은 Windows 10 Pro 64 비트 운영체제의 PC에서 모든 과정이 진행됩니다.
  - 실습 시에 반드시 Windows 10 운영체제에 하나의 영어 단어로 사용자이름으로 로그인하여 실습해야만 합니다.  
Windows 사용자 이름에 한글이 포함되거나 2개 이상의 단어로 된 경우, 문서 제일 앞에 새로운 Windows 사용자 생성 단원을 참고하여 계정을 생성한 후, 생성된 계정으로 Windows 운영체제에 로그인하여 실습하십시오.
  - 실습에 사용되는 PC에는, 최소 8 GB 이상의 메모리(16GB 이상 권장)와 20 GB 이상의 하드디스크 빈 공간이 필요합니다  
프로세서(CPU)는 인텔 i5 이상이면, 실습이 답답하지 않게 진행될 수 있습니다.
  - 본 파트에서는, 개발 환경을 구성하는 각 과정에서의 주의사항들을 다음의 순서에 따라 각 단원별 순서에 맞추어 설명합니다.
    1. Java Standard Edition Development Kit 8 설치 및 운영체제 환경변수 설정
    2. 이클립스 IDE(Integrated Development Environment) 설치
    3. 이클립스에 스프링 프레임워크 소프트웨어 설치
    4. 이클립스에 룸복(Lombok) 소프트웨어 설치
    5. 이클립스 워크스페이스 생성 및 UTF-8 인코딩 설정
    6. 톰캣 웹 애플리케이션 서버 설치
    7. 이클립스 프로젝트에서의 톰캣 사용 구성
    8. 오라클 데이터베이스 11g R2 Express Edition 설치(RDBMS 및 테스트 데이터베이스 생성 포함)
    9. 오라클 SQL\*Developer 설치 및 접속 구성
- ☞ 위의 단계들 중, 1 단계는 다른 단계보다 먼저 구성되어야 하며, 1 번 단계 구성 이 후에 2 번부터 7 번 단계까지는 위의 순서대로 진행하는 것을 적극 권장합니다.
- ☞ 개발 환경에서 오라클 RDBMS 를 사용하는 경우에는, 위의 모든 단계들이 구성되기 전에, 오라클 RDBMS 를 설치하는 것을 적극 권장합니다(불필요한 오류가 발생되는 횟수가 적어집니다). 문서에서는 제일 마지막 단계에서 오라클 RDBMS 를 설치합니다.
- ☞ 문서에서는 Java Standard Edition Development Kit 8 버전을 기준으로 모든 실습을 진행합니다.  
오라클 다운로드 사이트에서는 Java SE 16, Java SE 11 버전도 사용할 수 있지만, 국내 웹 애플리케이션 개발에서는 Java SE 8 이 대부분 사용되므로, 문서에서는 Java SE 8 을 사용하겠습니다.
- ☞ 오라클 RDBMS 는 개인용 테스트 버전으로 배포되는 오라클 데이터베이스 Express Edition 을 사용합니다.  
Express Edition 의 경우 18c 버전이 최신 버전이지만, 실습 데이터 구성이 제외되어 있고, PC 상태에 따라 설치 상에 오류가 많으므로, 개인적으로는 11gR2 버전을 사용하는 것을 추천합니다.  
문서에서는 오라클 데이터베이스 11g R2 Express Edition (64 비트)를 설치하여 사용합니다.

# [DE-01] Java Development Toolkit 8 설치 및 운영체제 환경변수 설정

- 다음의 작업들이 진행됩니다.
  - Java Standard Edition Development Kit 8 다운로드
  - Java Standard Edition Development Kit 8 설치
  - Java Standard Edition Development Kit 8 운영체제 환경변수 구성

## 1. Java Standard Edition Development Kit 8 다운로드

→ 웹 브라우저에서 <https://www.oracle.com/kr/java/technologies/javase/javase-jdk8-downloads.html> 사이트에 접속합니다.

→ 자신의 PC 운영체제에 해당하는 Java SE Development Kit 8 의 설치파일을 다운로드 합니다

문서에서는 Windows X64 운영체제에 사용되는 Java SE Development Kit 8u281 버전(2021-04)을 다운로드 했습니다.

☞ 다운로드 사이트에서 제공하는 가장 최근 버전의 업데이트 버전을 다운로드 받습니다.

→ 다운로드 시에, 로그인을 요청합니다. → 미리 생성해 둔 계정으로 로그인 합니다 → 다운로드가 진행됩니다.

☞ 회원가입을 하지 않은 경우, 표시된 로그인 페이지에서 "계정 만들기"를 클릭하여 회원가입을 수행한 후,  
다시 다운로드를 수행합니다.

## 2. Java Standard Edition Development Kit 8 설치

- 다운로드 된 jdk-8u281-windows-x64.exe 파일을 실행하여 자신의 PC에 설치합니다.
- 설치 시에, C:\myJavaDev\jdk1.8.0\_281 폴더를 생성한 후, 설치합니다.

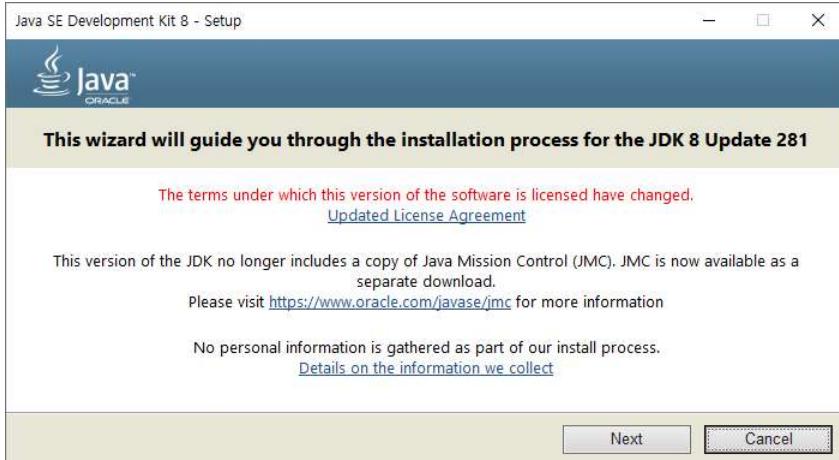
→ 파일탐색기를 실행하여 C:\ 디렉토리에 myJavaDev 폴더를 생성합니다.

→ 생성된 myJavaDev 폴더로 이동하여 jdk1.8.0\_281 폴더와 jre1.8.0\_281 폴더를 생성합니다.

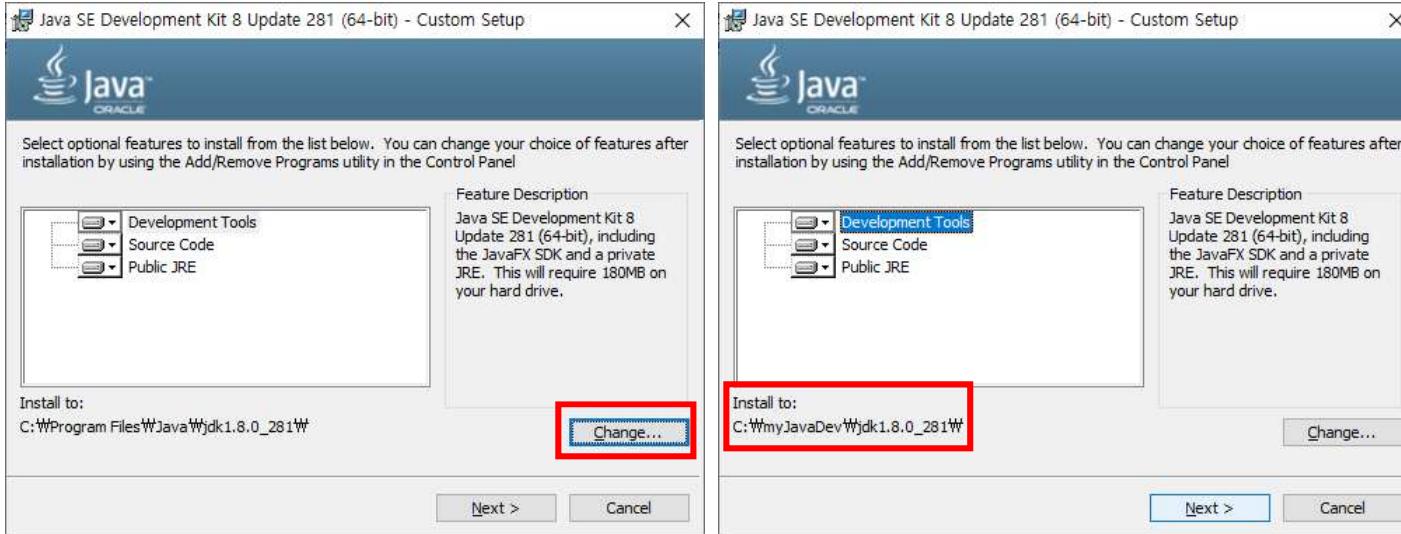
☞ 폴더 이름 뒤에 있는 문자열(1.8.0\_281)은 설치되는 Java SE Development Kit의 버전입니다.

→ 다운로드 된 jdk-8u281-windows-x64.exe 설치파일을 실행합니다.

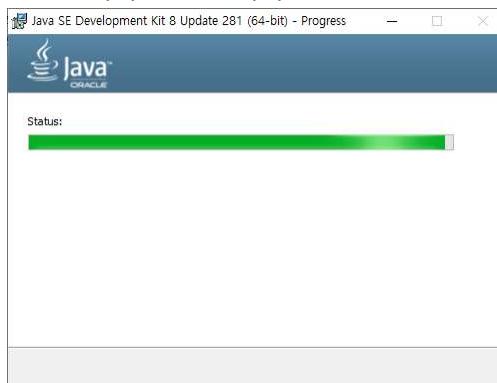
→ 라이선스 동의 공지창에서 Next 클릭



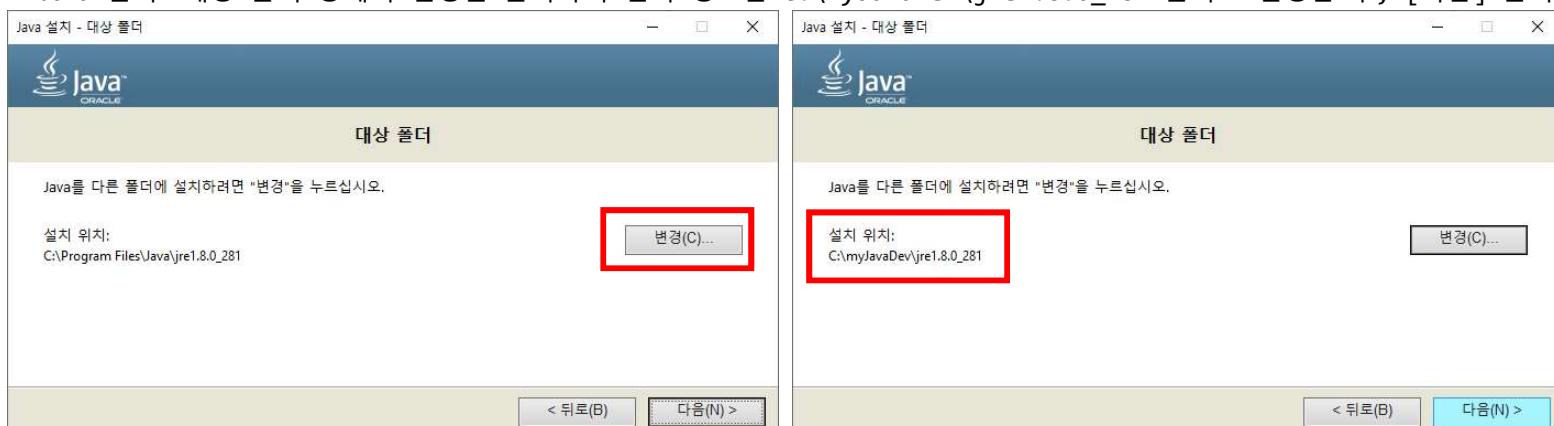
→ 설치 위치 지정 창에서, Change를 클릭하여 생성해 둔 C:\myJavaDev\jdk1.8.0\_281 폴더를 설치 위치로 지정 후 Next 클릭



→ 설치가 진행됩니다.



→ Java 설치- 대상 폴더 창에서 변경을 클릭하여 설치 경로를 c:\myJavaDev\jre1.8.0\_281 폴더로 변경한 후, [다음] 클릭



→ Java RunTime Environment(JRE) 설치가 진행됩니다.



→ Close 클릭하여 설치를 종료합니다.



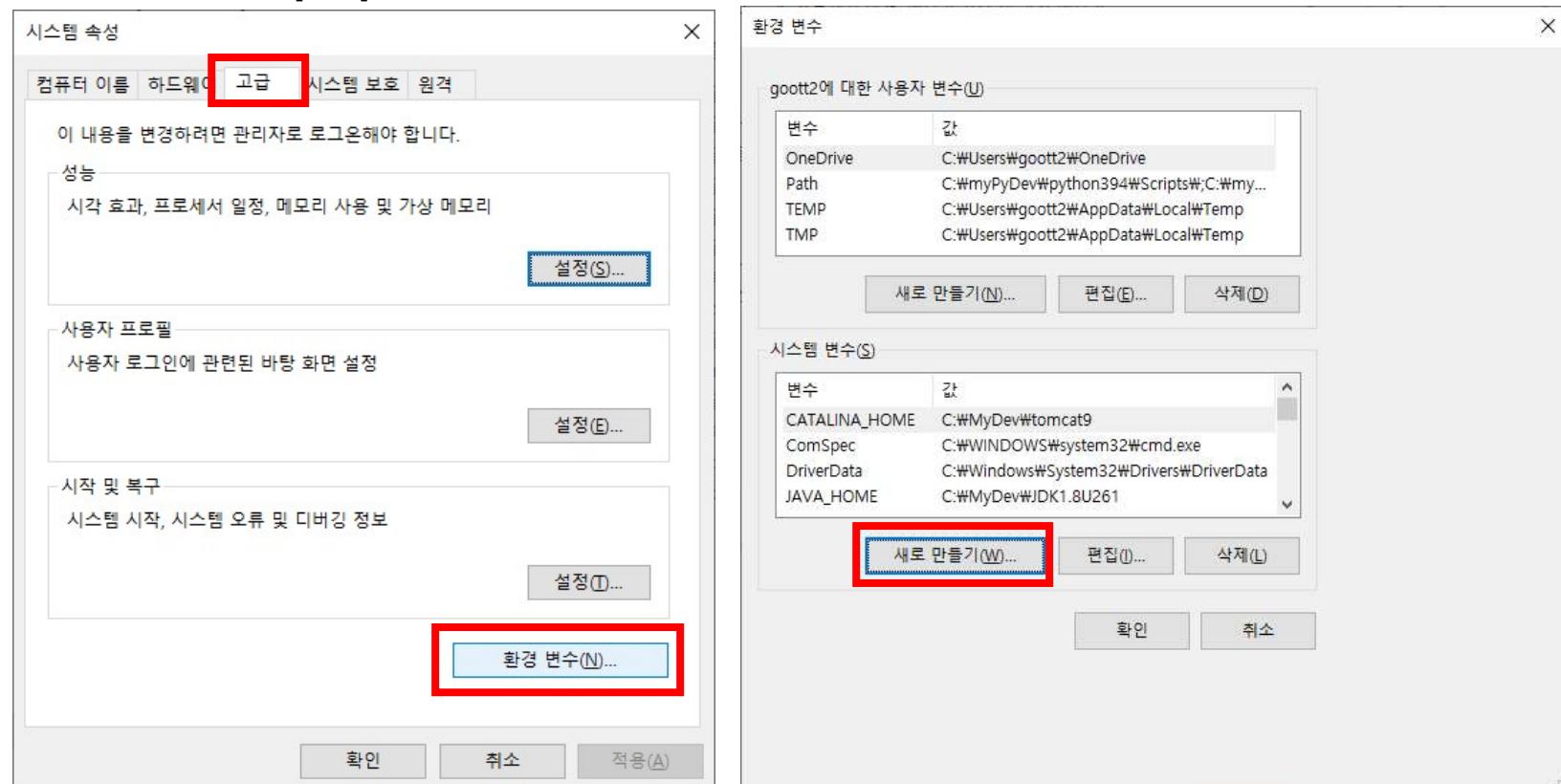
### 3. Java Standard Edition Development Kit 8 (JDK-8) 운영체제 환경변수 구성

☞ JAVA\_HOME 환경변수를 생성하고, Path환경 변수를 수정합니다.

→ 원도우키와 R키를 동시에 눌러 실행 창을 표시 → 열기에 sysdm.cpl 입력 후, 확인을 클릭 → 시스템 속성 창이 표시됩니다

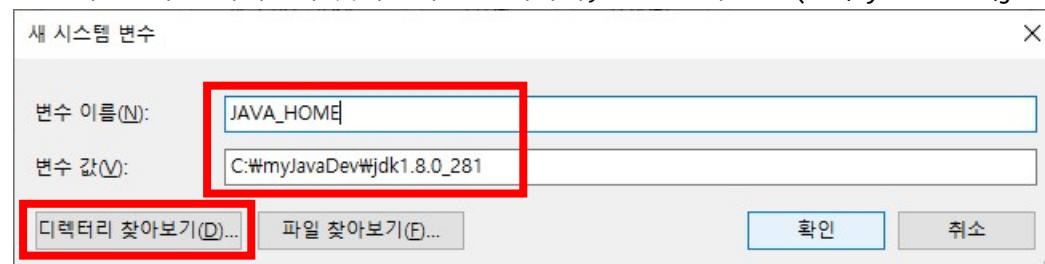


→ 시스템 속성 창에서 [고급] 탭을 클릭 → 고급 탭 페이지의 환경변수 클릭 → 시스템 변수 부분에 있는 새로 만들기를 클릭합니다.



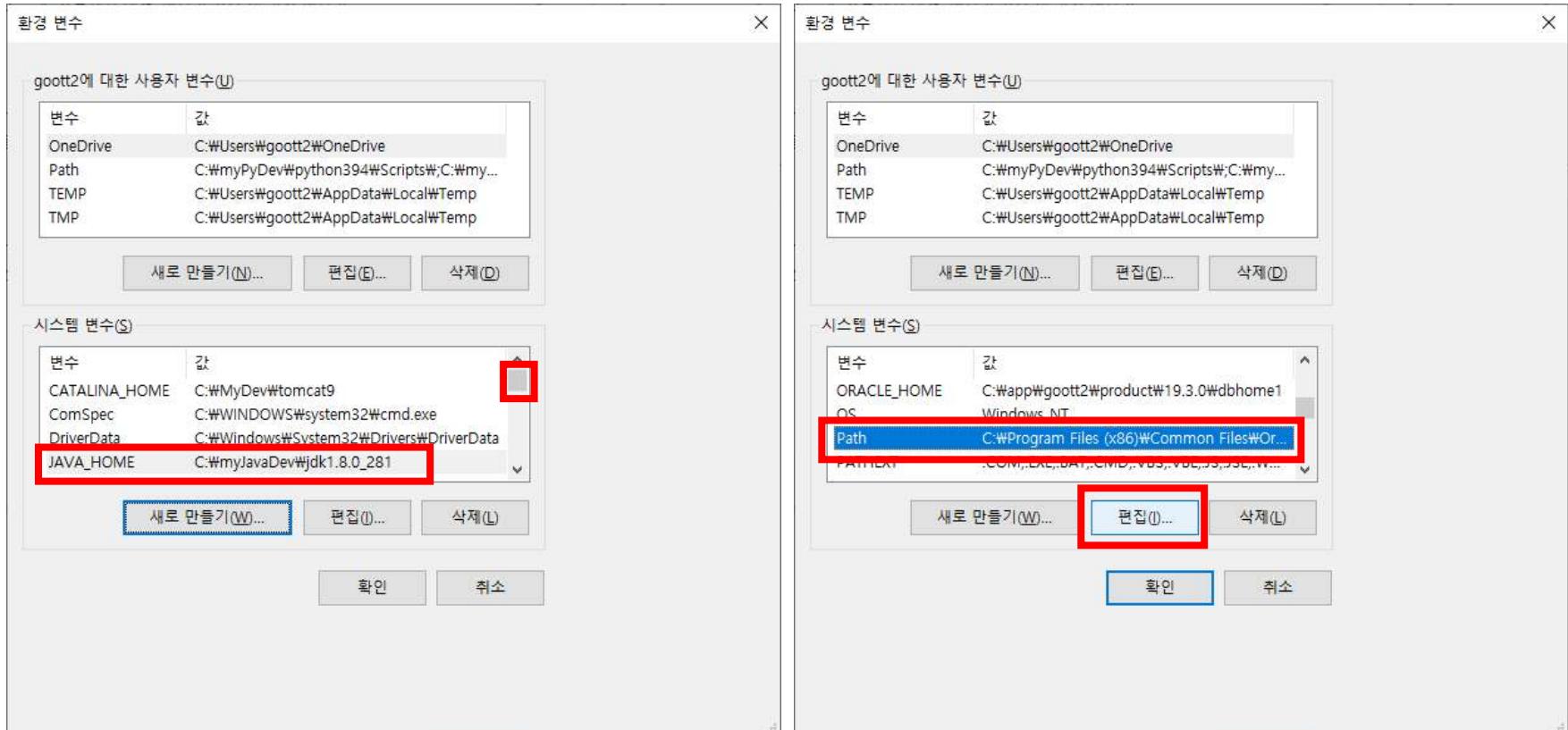
→ 새 시스템 변수창의 변수 이름에 JAVA\_HOME을 입력하고,

변수 값에는 디렉토리 찾아보기를 클릭하여, JDK 설치 경로(C:\myJavaDev\jdk1.8.0\_281)를 복사하여 붙여넣기 → 확인을 클릭합니다.

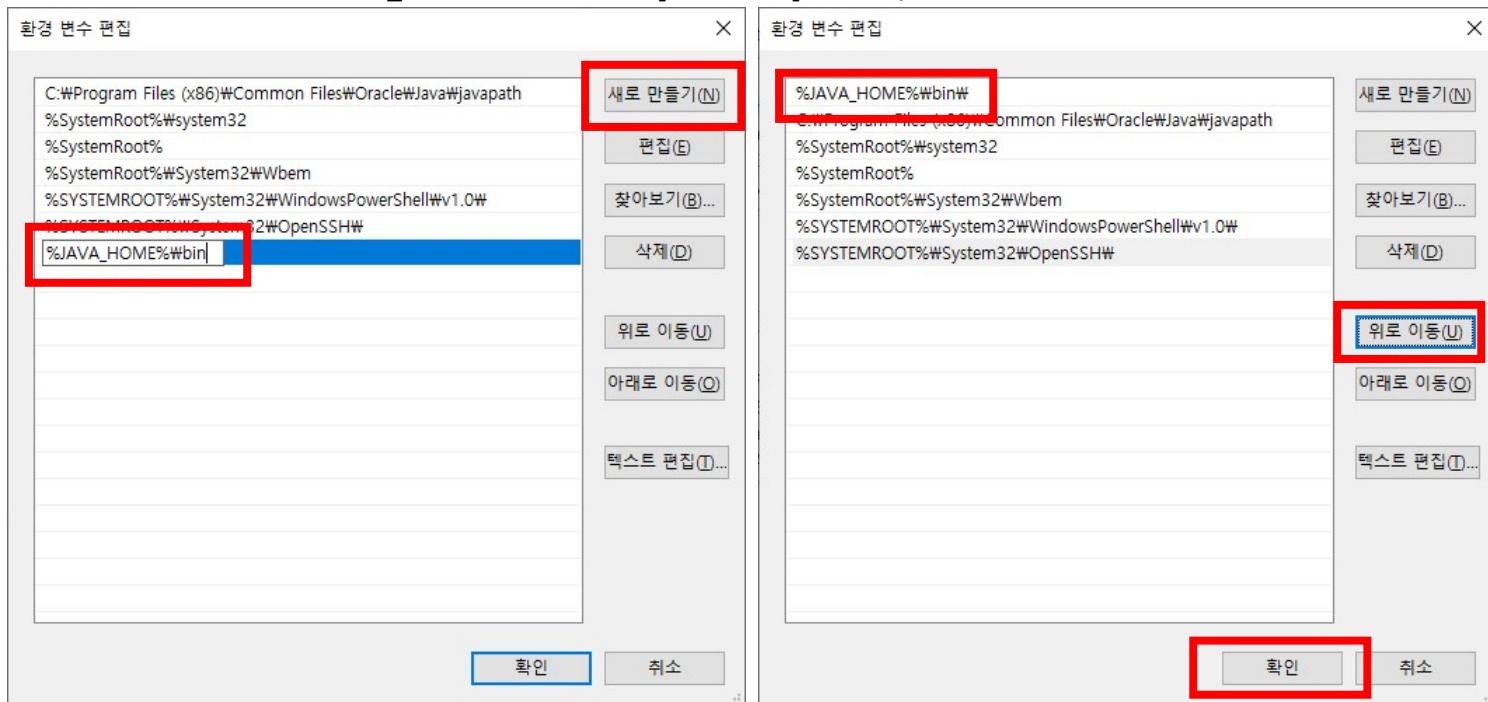


→ 환경 변수 창이 다시 표시되면, 설정한 JAVA\_HOME 환경변수가 표시되는지 확인

→ 환경 변수 창의 시스템 변수의 표시영역을 아래로 스크롤하여 Path 환경변수를 선택 후, [편집]을 클릭



→ 새로 만들기 클릭 → %JAVA\_HOME%\bin 입력 → [위로 이동]을 눌러, 제일 위로 이동 → 확인을 클릭합니다.



→ 환경변수 창에서 [확인]을 클릭하여 환경변수 창 닫음 → 시스템 속성 창에서 [확인]을 클릭하여 시스템 속성 창을 닫습니다.

→ 명령 프롬프트를 새로 실행하여 다음의 명령어를 입력한 후, 실행(Enter키 누름)합니다.

```
C:\Users\goott2>javac -version  
javac 1.8.0_281
```

☞ 위와 같이 설치된 java 컴파일러 (javac) 버전이 표시되면, 설치 및 환경변수 구성이 잘 수행된 것입니다.

☞ 만약, "'javac'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는 배치 파일이 아닙니다."라는 메시지가 표시되면, 환경 변수 설정이 잘못(설정값에 오타가 있거나 또는 대소문자가 틀림)된 경우이므로, 위의 환경 변수 설정 과정을 다시 수행하여 잘못을 수정하십시오.

## [DE-02] 이클립스 IDE(Integrated Development Environment) 설치

- 다음의 작업들이 진행됩니다.
  - 이클립스 다운로드 및 설치

### 1. 이클립스 다운로드

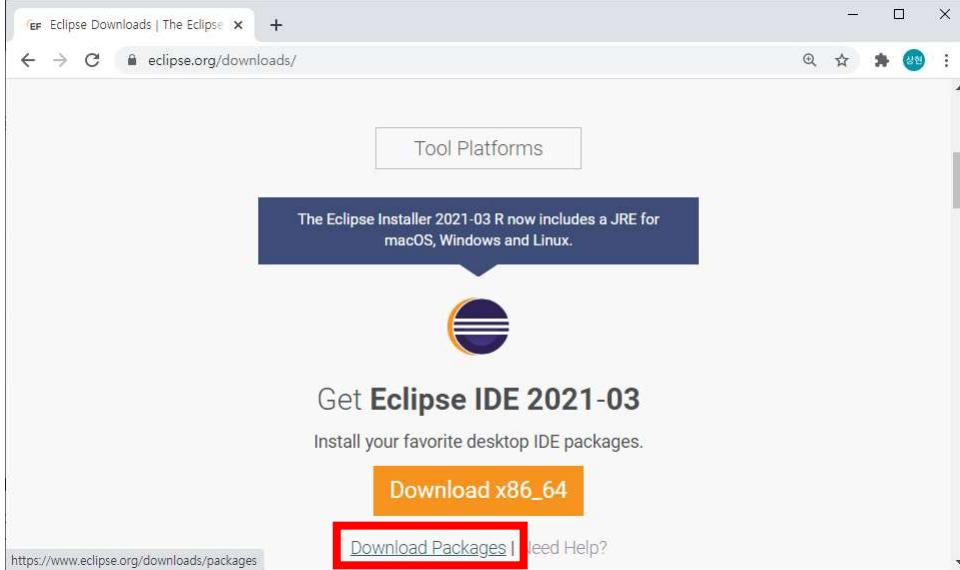
- ☞ 이클립스는 인스톨러 형태와 압축파일 형태의 2가지 형식으로 소프트웨어를 배포합니다.
  - 인스톨러 형태는 앞의 JDK-8과 유사하게 PC에 설치해서 이클립스를 사용할 수 있습니다.
  - 압축파일 형태는 다운로드 받은 파일을 압축해제 하여 이클립스를 사용할 수 있습니다(설치과정이 없음)

- ☞ 문서에서는 압축파일 형태의 이클립스를 다운로드 받아 사용하도록 구성하겠습니다.

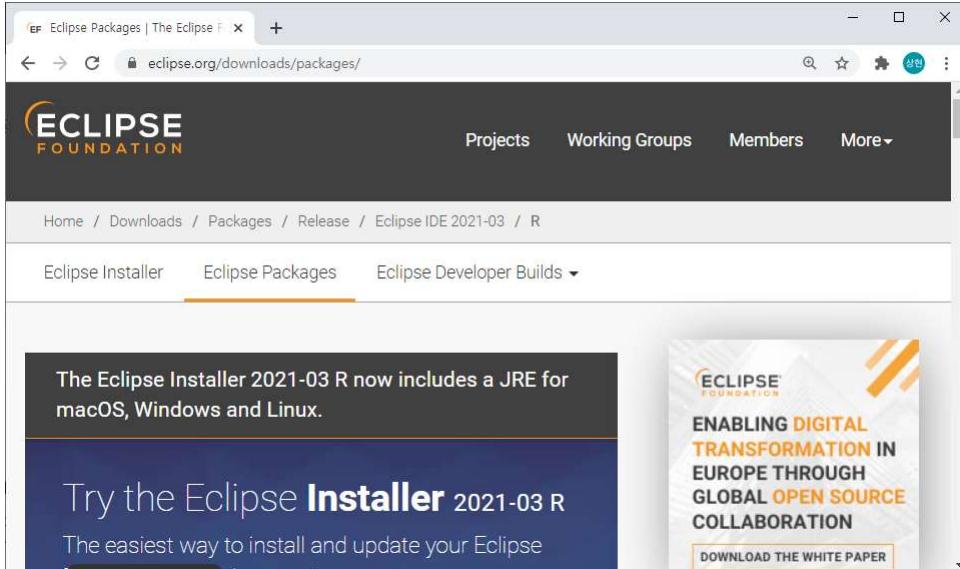
→ 웹 브라우저에서 <https://www.eclipse.org/downloads/> 사이트에 접속합니다.

[참고] Download x86\_64 버튼을 클릭하면, 인스톨러 형태의 이클립스 설치파일을 다운로드 받을 수 있는 페이지로 이동됩니다.  
문서에서는 압축파일 형태의 이클립스 배포판을 다운로드 받으므로, Download x86\_64 버튼을 클릭하지 않습니다.

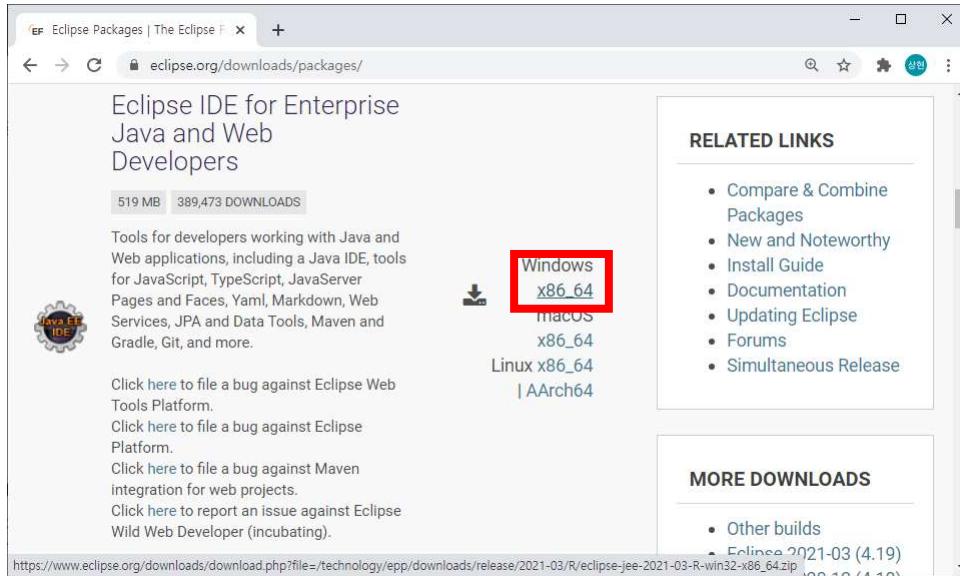
→ Download x86\_64 버튼 아래의 Download Packages 링크를 클릭합니다.



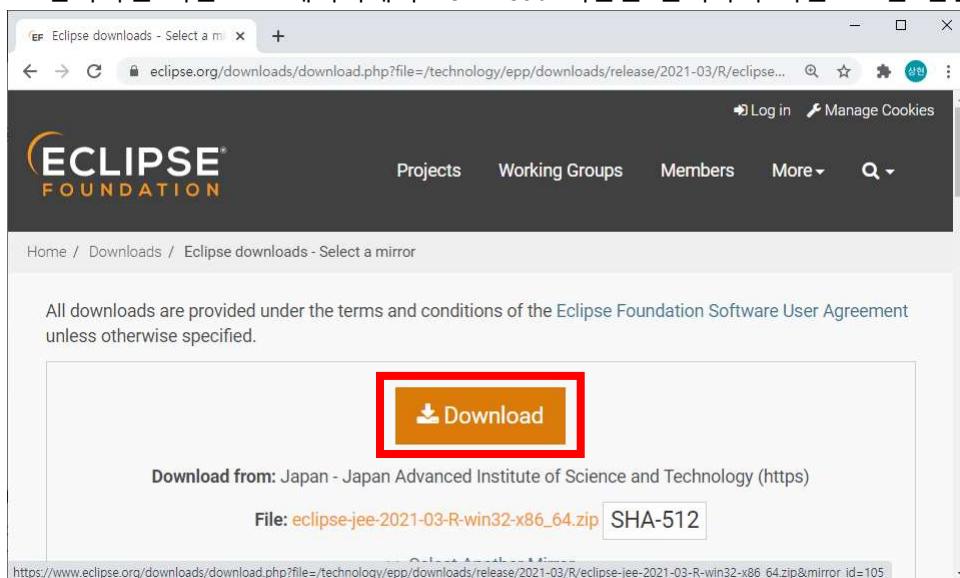
→ 표시된 Eclipse Packages 페이지를 아래로 스크롤하여, Eclipse IDE for Enterprise Java and Web Developers 부분으로 이동



→ Eclipse IDE for Enterprise Java and Web Developers 부분의 Windows x86\_64 링크를 클릭



→ 앱축파일 다운로드 페이지에서 Download 버튼을 클릭하여 다운로드를 받습니다.



## 2. 이클립스 설치

→ 다운로드 된 앱축파일을 C:\myJavaDev 폴더에 복사합니다.

→ 압축해제 프로그램을 이용하여 '여기에 풀기' 옵션으로 압축해제

☞ eclipse 폴더가 생성되고, 압축이 해제됩니다.

→ 파일탐색기를 실행하여 이클립스 설치 폴더인 C:\myJavaDev\eclipse 폴더로 이동합니다.

→ 폴더에 있는 eclipse.exe 파일의 바로가기를 바탕화면에 생성합니다.

☞ 이클립스 설치 완료

## [DE-03] 이클립스에 스프링 프레임워크 소프트웨어 설치

- 다음의 작업들이 진행됩니다.
  - 이클립스에 Spring Tools 3 (Standalone Edition) 소프트웨어 추가 설치

[참고] 스프링 프레임워크 4 와 5에서 JDK 버전 요구사항

스프링 프레임워크 버전	JDK 버전 범위
5.2.x	JDK 8 이상
5.1.x	JDK 8 이상
5.0.x	JDK 8 이상
4.3.x	JDK 6 ~ 8

☞ [출처] <https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-Versions>

[참고] 이클립스에 스프링 프레임워크를 사용하도록 구성하는 것은 3 가지 방법

- (1) 이클립스에 Spring Tool Suite (STS, , 피보탈 사에서 배포하는 IDE 툴) 소프트웨어를 추가로 설치하는 방법
- (2) <https://spring.io/> 사이트에서 STS를 다운로드 받아 설치하는 방법 → 이클립스 대신 사용
- (3) <https://repo.spring.io/release/org/springframework/spring/> 사이트에서 라이브러리를 다운로드 받아 프로젝트에 적용하는 방법

☞ 본 문서에서는 이클립스에 STS 3 를 추가로 설치하여 사용하는 방법을 이용합니다.

☞ 주의할 것은, STS 4 버전부터는 Spring Boot 기반의 개발환경만 구성되며, 스프링 프레임워크를 사용하기 위한 환경 구성이 포함되지 않습니다. 따라서, 스프링 프레임워크 라이브러리를 사용하기 위해서는 STS 3 버전을 설치해야 합니다.

### 1. 이클립스에 STS 3 (스프링 프레임워크 개발 구성 포함) 소프트웨어 설치

→ 바탕화면의 바로가기를 더블클릭하여 이클립스 실행

→ 이클립스 워크스페이스 설정창이 표시되면, Launch를 클릭

☞ 뒤에서 워크스페이스를 새로 생성하므로, 이 단계는 기본 구성으로 진행합니다.

→ 이클립스의 Help 메뉴 클릭

→ Eclipse Marketplace를 실행

→ Eclipse Marketplace 창에서 Find 입력란에 STS 입력 후, [Go] 클릭하여 검색

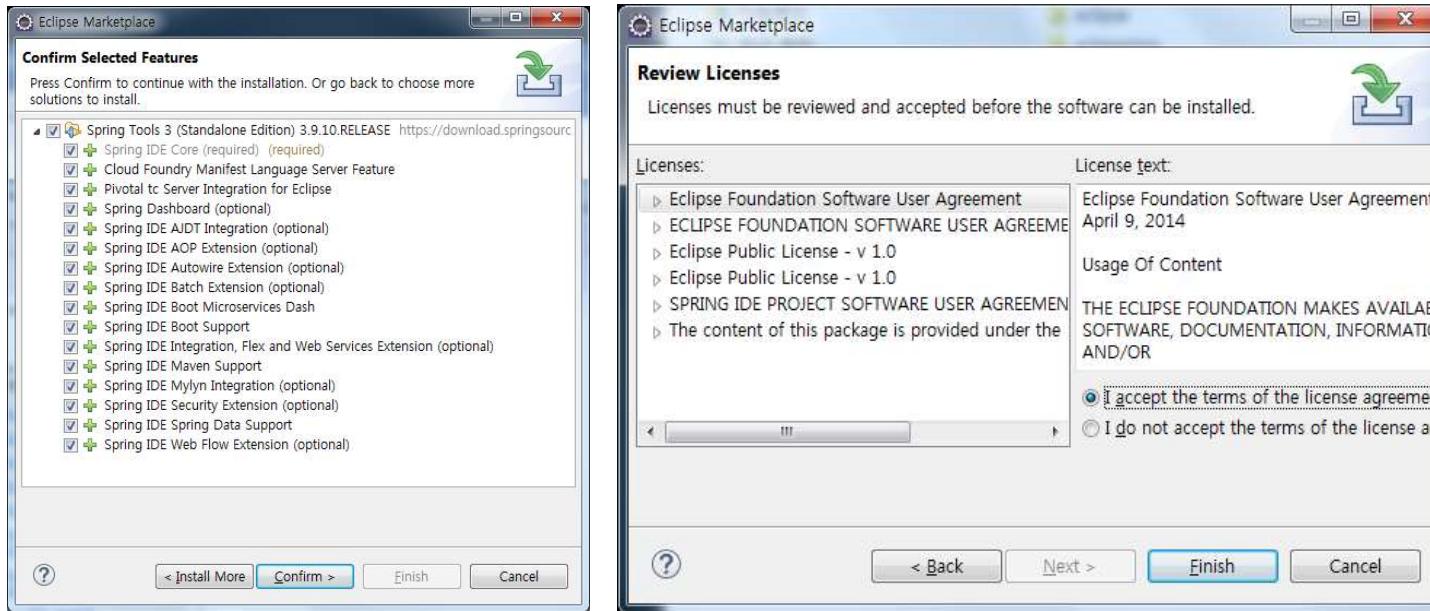
→ 표시된 검색결과에서 Spring Tools 3 (Standalone Edition) 3.9.14.RELEASE를 클릭하여 설치.

☞ [주의] Spring Tools 3 Add-on for Spring Tools 4.3.9.16.RELEASE 를 설치하면 됩니다.

☞ 3 버전의 STS를 설치해야만 스프링 프레임워크 기반 프로젝트를 생성할 수 있는 기능이 설치됩니다.

→ Confirm Selected Features 페이지가 표시되면, 모든 항목이 선택되었는지 확인 후, Confirm 클릭.

→ Review Licenses 페이지에서 [I accept the terms of the license agreements] 선택 후 [Finish]를 클릭합니다.



→ 이클립스 하단 오른쪽에 Installing Software 상태바에 설치 진행 상태가 표시됩니다.

→ 설치가 완료되면, 이클립스를 다시 시작하겠는지를 묻는 메시지 박스가 표시됩니다.



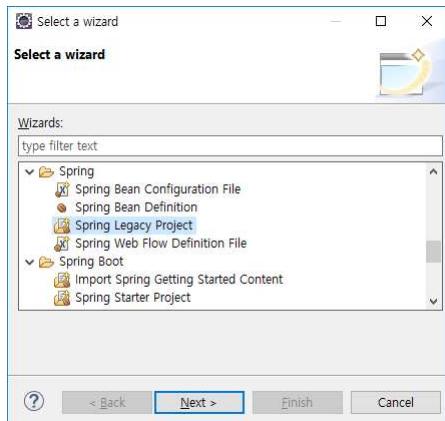
→ [Restart Now]를 클릭하여 이클립스를 다시 시작합니다.

→ 이클립스 상단의 File 메뉴 클릭 → New → Other 클릭

→ Select a wizard 창에서 Spring 폴더와 Spring Boot 폴더가 표시되어야 함

→ Spring 폴더와 Spring Boot 폴더를 각각 더블클릭하여 확장했을 때

다음처럼 Spring Legacy Project와 Spring Starter Project 항목이 표시되어야 함



→ 스프링 프레임워크 기반 기능 설치 완료

☞ 스프링 프레임워크 기반 프로젝트는 Spring 폴더에 있는 Spring Legacy Project 항목을 이용해서 생성합니다.

## [DE-04] 이클립스에 롬복(Lombok) 소프트웨어 설치

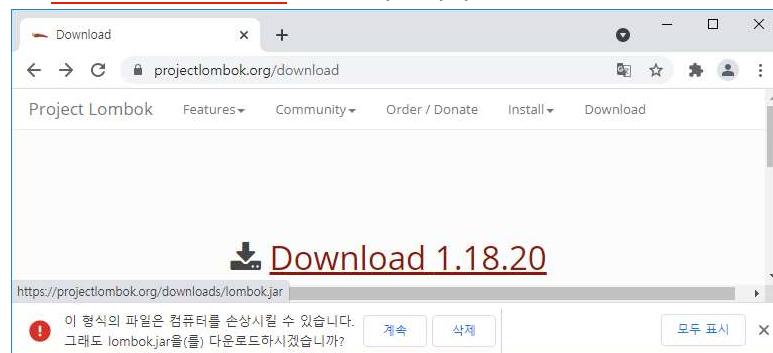
- 다음의 작업들이 진행됩니다.
  - 롬복 소프트웨어 다운로드
  - 이클립스에 롬복 소프트웨어 설치

### 1. 롬복(Lombok) 소프트웨어 다운로드

☞ 롬복 소프트웨어는 자바 소스코드 작성 시에, getter-setter 메소드, `toString()`, `equals()`, `hashCode()` 메소드, 생성자들을 소스 코드에 직접 작성하지 않더라도 컴파일 시에 클래스 파일에 생성해주는 어노테이션을 제공해 주는 소프트웨어입니다.

→ 웹 브라우저에서 <https://projectlombok.org/download> 사이트에 접속합니다.

→ Download 1.18.20 을 클릭합니다.



→ 브라우저에서 위의 그림처럼 다운로드 하겠습니까? 묻는 메시지가 브라우저에 표시되면 [계속]을 클릭합니다.

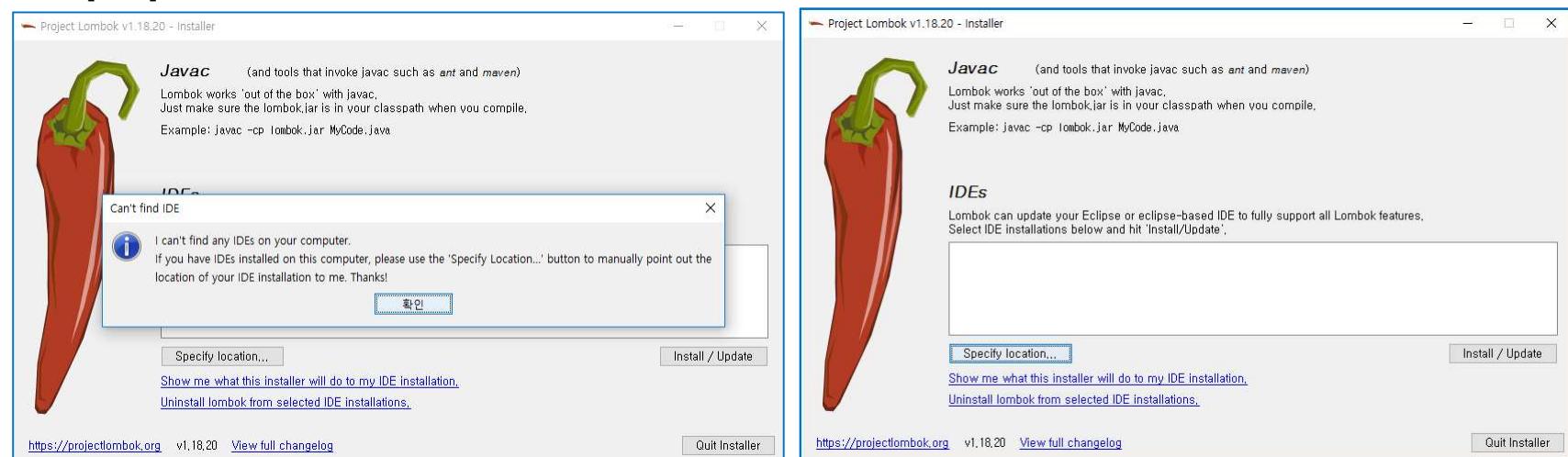
→ `lombok.jar` 파일이 다운로드 됩니다.

### 2. 롬복(Lombok) 소프트웨어 설치

→ 다운로드 된, `lombok.jar` 파일을 C:\ 폴더에 복사합니다.

→ C:\ 폴더에 복사된 `lombok.jar` 파일을 더블 클릭하여 실행합니다. → 다음과 같은 설치화면과 메시지 박스가 표시됩니다.

→ [확인]을 클릭하여 IDE를 찾을 수 없다는 메시지 박스를 닫습니다.



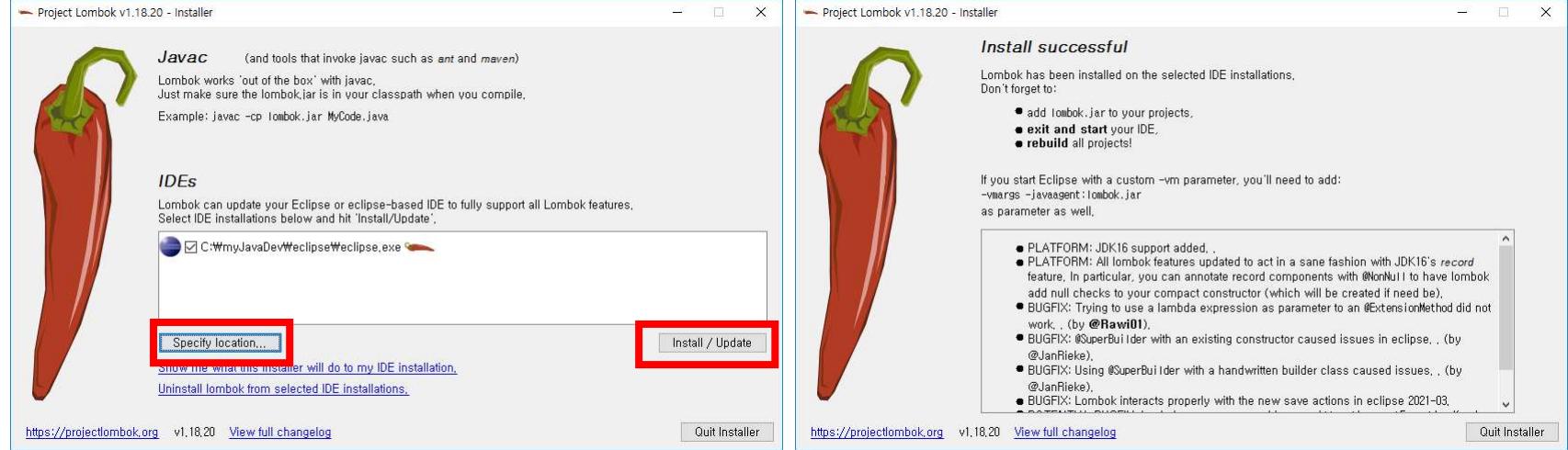
☞ lombok.jar 실행 시, 위의 설치화면이 표시되지 않는 경우, 명령 프롬프트에서 다음을 실행하면, 위의 설치화면이 표시됩니다.

```
C:\Users\goott5>cd C:\
```

```
C:\>java -jar lombok.jar
```

→ Specify location 을 클릭 → 이클립스가 설치된 C:\myJavaDev\eclipse 폴더를 선택 → Install/Update 를 클릭합니다.

→ Quit Installer를 클릭하여 설치를 종료합니다.



→ 파일탐색기에서 이클립스 설치폴더인 C:\myJavaDev\eclipse 폴더로 이동하여,

lombok.jar 파일이 생성되어 있는지 확인합니다. 파일이 존재하면, 설치가 완료된 것입니다.

→ 바탕화면에 있는 eclipse.exe의 바로가기를 삭제한 후,

→ C:\myJavaDev\eclipse 폴더에 있는 eclipse.exe 파일의 바로가기를 바탕화면에 새로 생성합니다.

☞ 이 후, 이클립스에서 프로젝트를 생성할 때마다 설치한 lombok 소프트웨어의 버전과 동일한 라이브러리를 프로젝트에 구성해주면, 룸복의 어노테이션을 이용한 여러가지 기능을 Java 코드 작성 시에 사용할 수 있습니다.

## [DE-05] 이클립스 워크스페이스 생성 및 구성

- 다음의 작업들이 진행됩니다.
  - 이클립스 워크스페이스 폴더 생성
  - 워크스페이스를 JDK 8 사용 구성으로 변경(2021-03(4.19.0) 버전부터 필요)
  - 워크스페이스에 UTF-8 인코딩 적용

### [참고] 워크스페이스와 프로젝트

워크스페이스와 프로젝트는 프로그램 소스를 관리하기 위한 디렉토리(폴더) 체계라고 할 수 있습니다(이 체계는 IDE마다 조금씩 차이가 납니다).

워크스페이스는, 이클립스에서 제작되는 프로그램 소스들이 저장되는 가장 상위의 디렉토리입니다. 워크스페이스는 필요에 따라 여러 개를 생성할 수 있지만, 이클립스가 시작될 때 오직 하나의 워크스페이스만 사용할 수 있습니다.

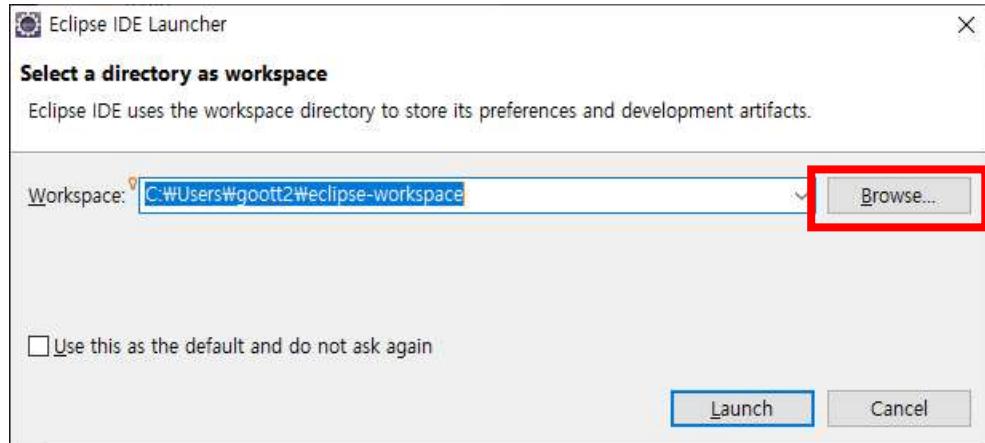
특정 워크스페이스를 이용하여 이클립스가 실행된 후, 사용자들은 애플리케이션을 제작할 때 제일 먼저 프로젝트라는 것을 생성해야만 합니다. 하나의 애플리케이션은 다수의 프로그램들로 구성될 수 있고, 따라서 하나의 애플리케이션과 관련된 프로그램들을 사용자들이 쉽게 구분 지을 수 있도록 이클립스는 프로젝트라는 단위를 사용합니다. 하나의 워크스페이스에는 다수의 프로젝트가 구성될 수 있습니다.

이클립스는 실행 시에, 사용자가 지정한 워크스페이스(최상위 폴더)를 이용하도록 구성되며, 해당 워크스페이스에 사용자가 생성한 프로젝트 단위로 구성된 애플리케이션 소스들을 작업할 수 있도록 구성해 줍니다

### 1. 이클립스 워크스페이스 폴더 생성

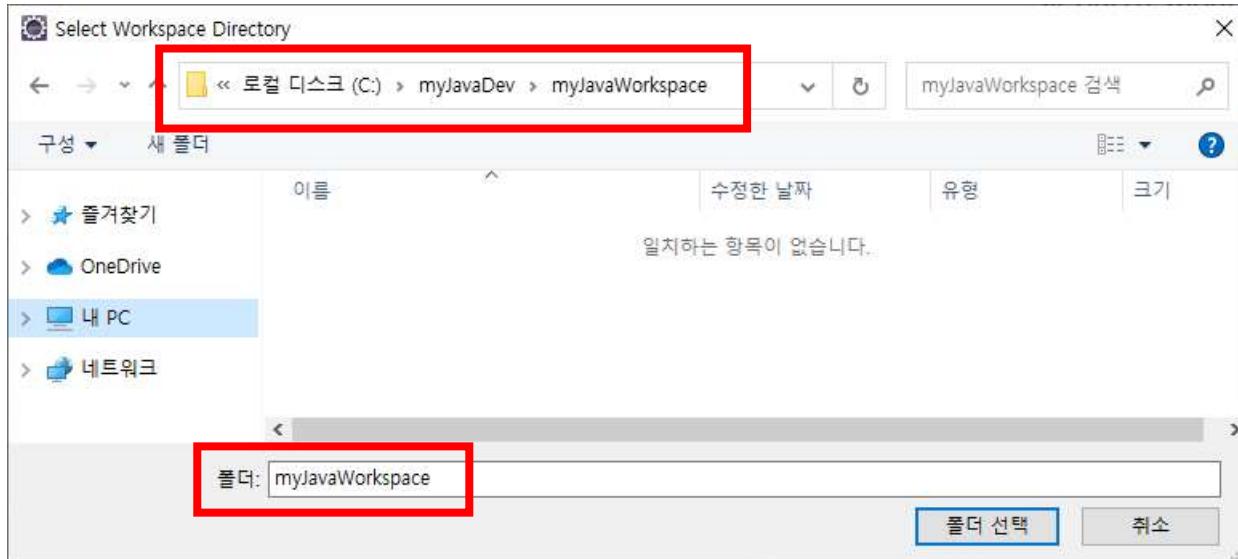
→ 바탕화면의 이클립스 바로가기를 이용하여 이클립스를 실행

→ 실행 직 후, 워크스페이스(workspace) 디렉토리를 지정하는 창이 표시됨 → Browse 버튼을 클릭



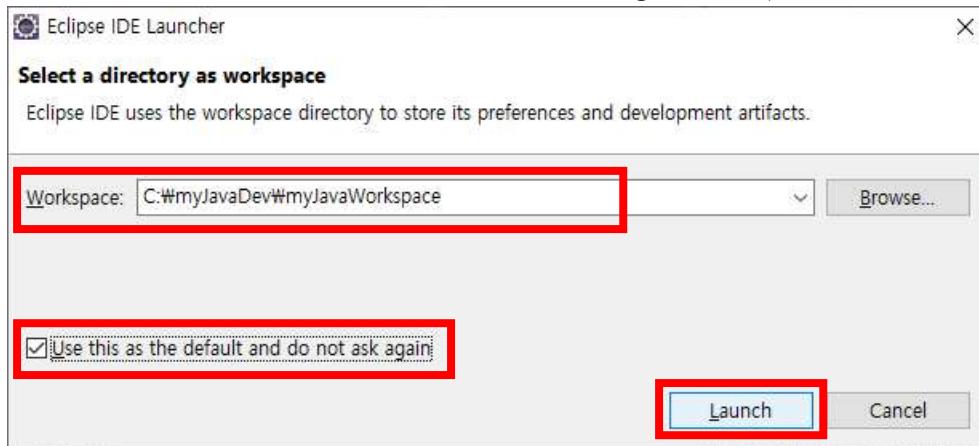
→ C:\myJavaDev로 이동한 후, myXXXXWorkspace 폴더 생성 → myXXXXWorkspace 폴더로 이동 → 폴더 선택 클릭

☞ XXXX는 상황에 따라 Java, Web, Spring 등, 자신이 원하는 것으로 대체됩니다. 실습에서는 Java로 정합니다.

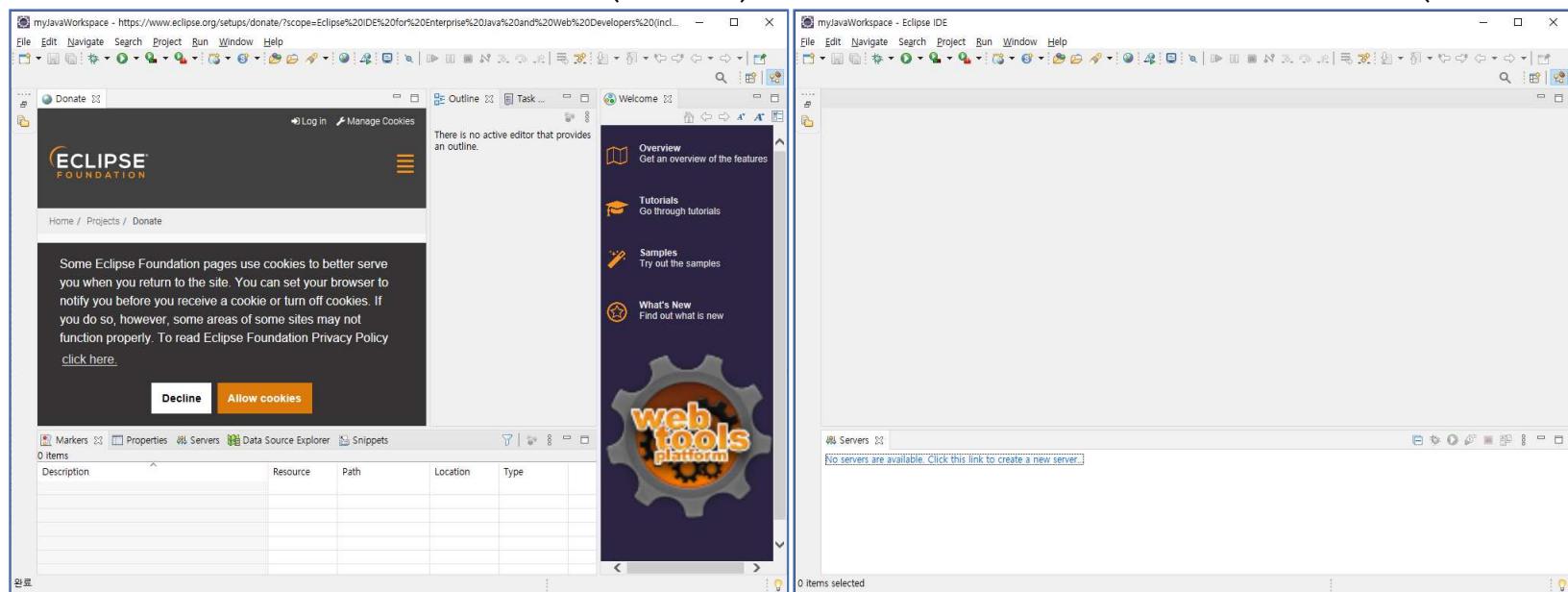


→ 다시 워크스페이스(workspace) 디렉토리를 지정하는 창이 표시됨, 설정한 Workspace 경로 확인

→ Use this as the default and do not ask again 체크(다음 이클립스 실행 시에 이 창이 표시 않됨) 후 Launch 클릭



→ 이클립스가 정상적으로 시작된 창이 표시됩니다(왼쪽 그림). → 개발 시에 사용되지 않는 필요없는 뷰를 닫습니다(오른쪽 그림)



☞ 워크스페이스 생성이 완료되었습니다.

## 2. 워크스페이스를 JDK 8 사용 구성으로 변경(2021-03(버전: 4.19.0) 버전부터 필요)

☞ 이클립스 2021-03 버전부터 이클립스에 JRE15(JRE11 이상)이 기본 포함되어 있습니다.

☞ 문서 실습에서는 현재 자바 웹 애플리케이션 개발 시에 많이 사용하는 JDK8을 기반으로 실습이 진행되므로, 이클립스에 기본 포함된 JRE15 대신, 앞에서 설치 구성한 JDK8에 포함된 JRE8을 사용하도록 변경해야 합니다.

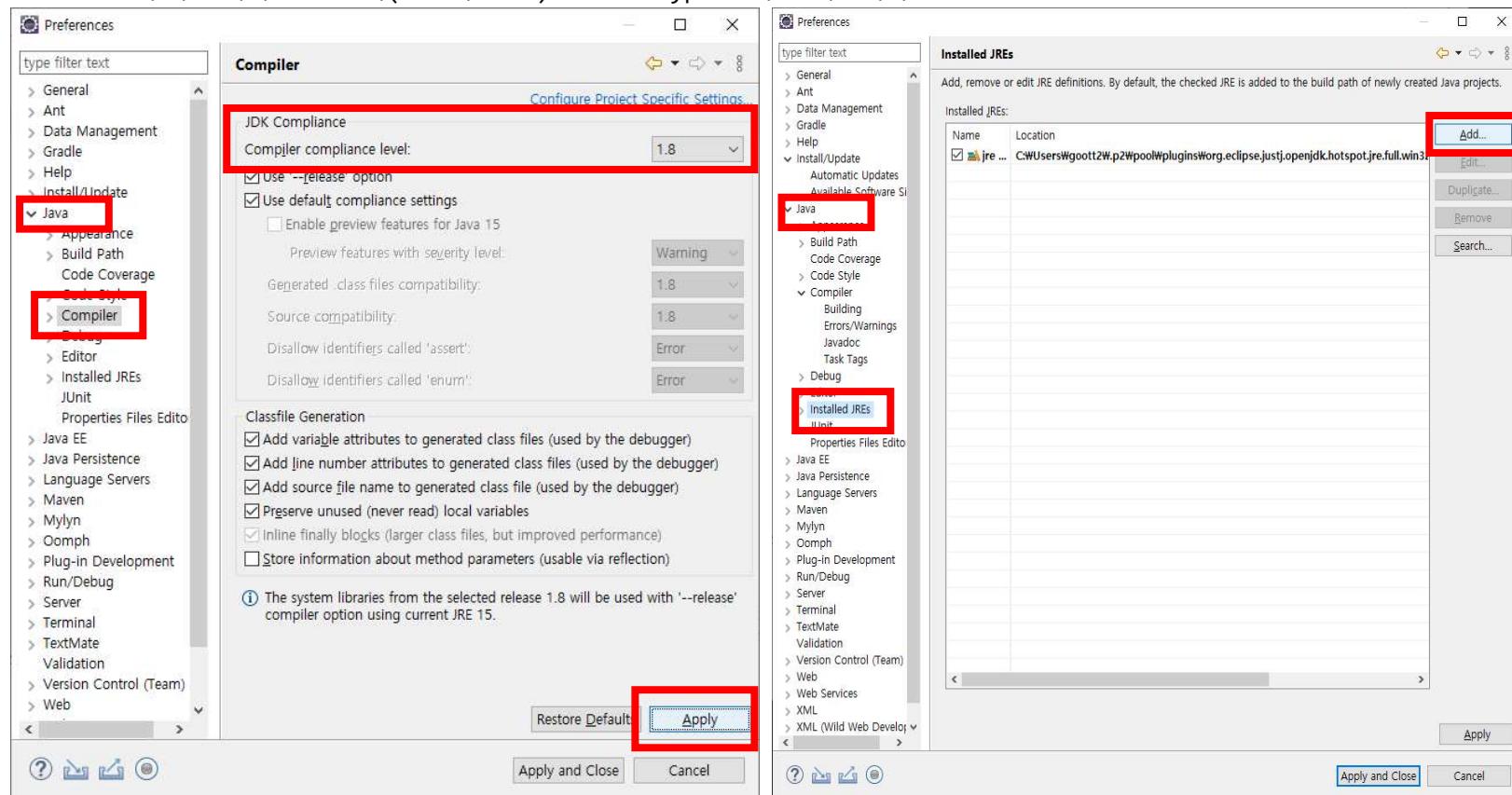
→ 이클립스의 위에 있는 window 메뉴 클릭 → Preferences 항목 클릭 → Preferences 창이 표시됩니다.

→ 왼쪽에서 Java 더블클릭(확장) → Compiler 클릭

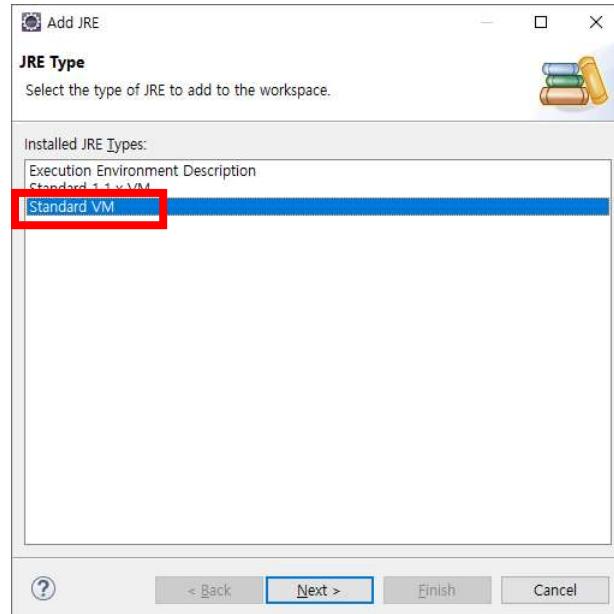
→ 오른쪽 부분에서 Compiler compliance level의 항목값을 15에서 1.8로 변경 → Apply를 클릭합니다(왼쪽 그림).

→ 왼쪽에서 Java 더블클릭(확장) → Installed JREs 클릭

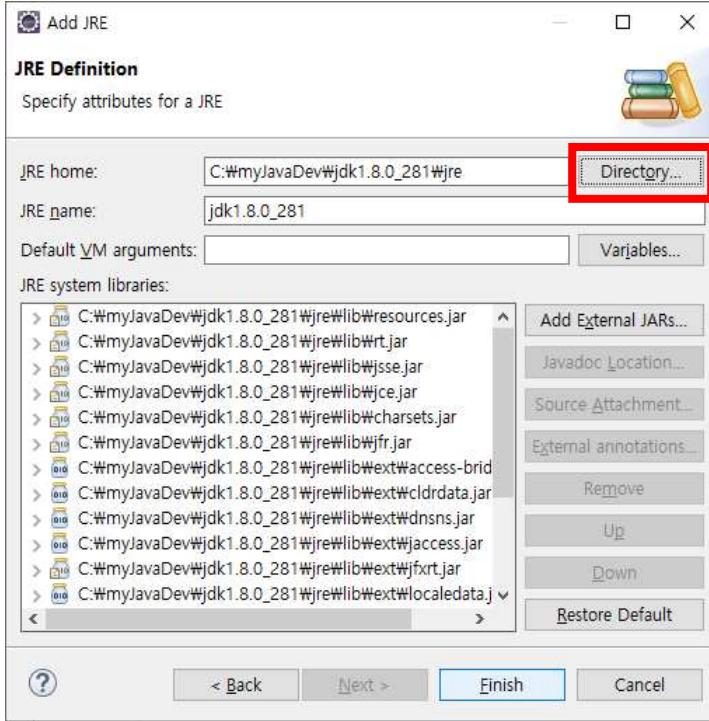
→ 오른쪽 부분에서 Add 클릭(오른쪽 그림) → JRE Type 창이 표시됩니다



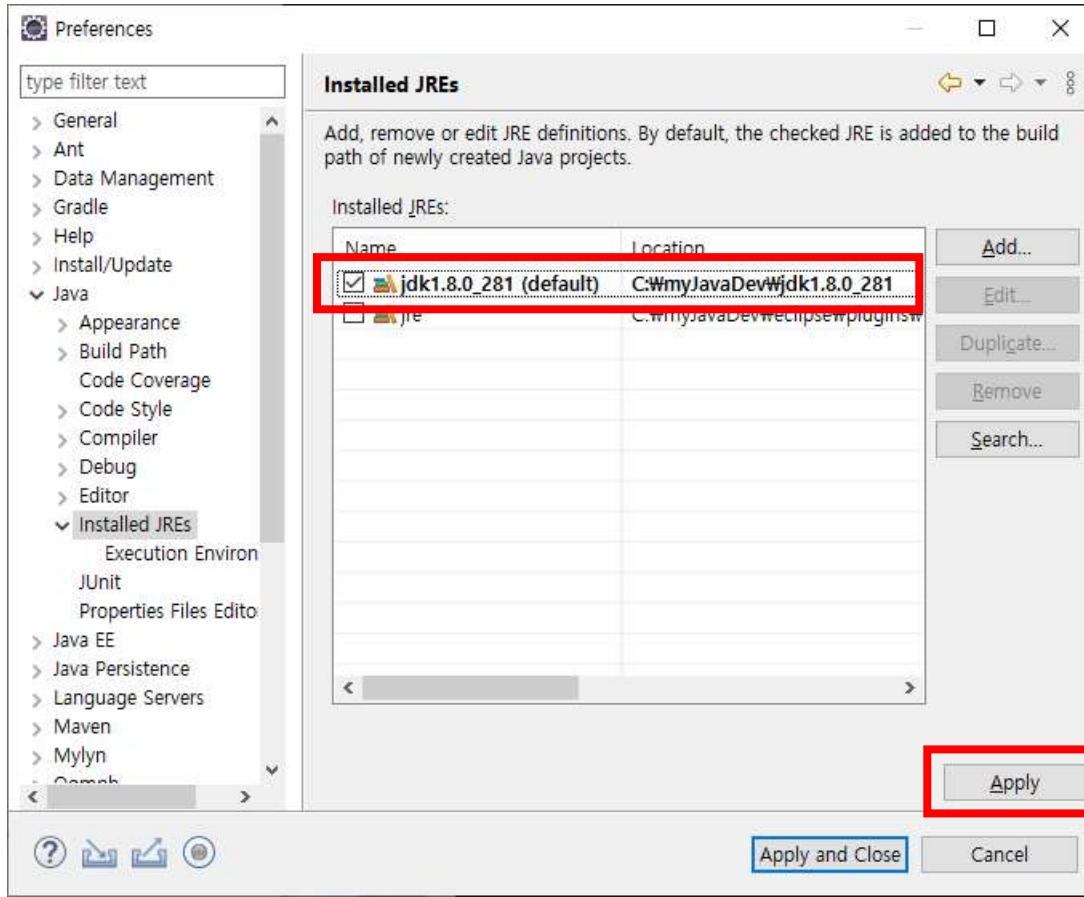
→ JRE Type 창에서 Standard VM 선택 후, Next 클릭 (왼쪽 그림)



- JRE Definition 창 → JRE home 입력란의 Directory 버튼 클릭
- 폴더 선택창에서 JDK-8 이 설치된 디렉토리(C:\myJavaDev\jdk1.8.0\_281)를 지정 후, 폴더선택 클릭
- 다시 JRE Definition 창이 다시 표시되면, JRE home에 설정한 경로를 확인 후, Finish 버튼 클릭

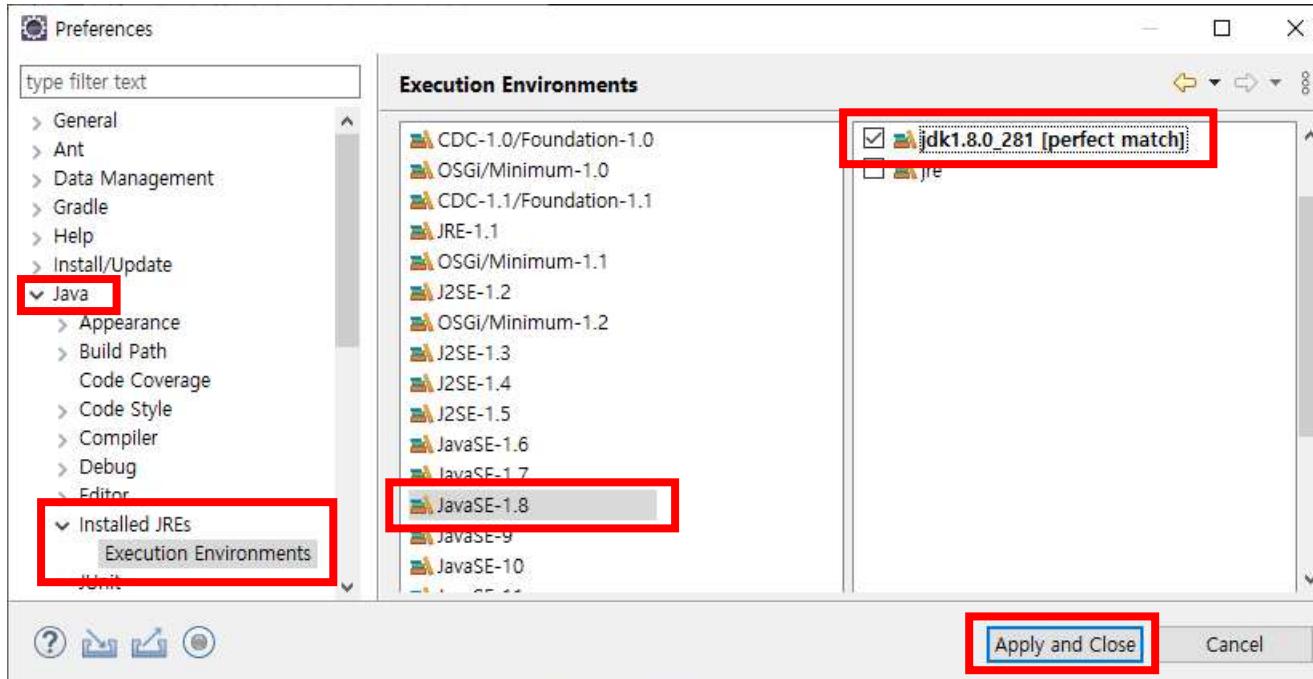


- Installed JRE 창에 추가된 jdk1.8.0\_281(default) 항목을 체크(이 때 기존 항목의 체크가 사라짐) → Apply를 클릭합니다.



→ 왼쪽에서 Java 더블클릭(확장) → Installed JREs 더블클릭(확장) → Execution Environments 클릭

→ 오른쪽의 Execution Environments에서 JavaSE-1.8 클릭(선택) → jdk1.8.0\_281[perfect match] 체크 → Apply and Close 클릭



☞ 워크스페이스를 JDK-8, JRE-8 을 사용하는 구성으로 변경 완료

☞ 새로 생성된 워크스페이스에서 JDK-8, JRE-8을 사용하려면 이 섹션을 워크스페이스 생성 시에 매번 해주어야 합니다.

### 3. 워크스페이스에 UTF-8 인코딩 설정

#### [참고] 문자 인코딩이란

- 문자 인코딩(Encoding)이란, 컴퓨터에서 실행되는 프로그램에서 문자를 처리(저장 및 읽어 옴) 시에 사용하는 형식(코드체계)입니다. 특정 인코딩 형식은, 각 문자마다 일정한 코드값을 지정하고, 컴퓨터에서 실행되는 프로그램들이 해당 인코딩을 사용하는 경우, 문자마다 지정된 고유한 코드값으로 문자를 처리합니다.

이러한 인코딩(Encoding)이라는 형식(코드체계)은 많은 종류(ASCII, MS949, EUC-KR, UTF-8 등등)가 있으며, 이 중에 일반적으로 많이 사용되는 형식이 UTF-8 인코딩입니다. 유니코드라는 UTF-8 인코딩을 사용하면, 한글을 포함하여 여러 국가의 다양한 문자들을 모두 처리할 수 있습니다.

특히 자바의 경우 기본 인코딩 형식으로 UTF-8 인코딩을 사용합니다.

☞ 한글 Windows 운영체제에 설치된 이클립스는, 운영체제에서 사용되는 MS949 인코딩(한글과 영어문자만 처리)을 사용하도록 설정됩니다.

☞ 따라서, 이클립스에서 작성되는 다양한 프로그램 소스(java소스, HTML 소스, JSP 소스 등등)들에 대한 인코딩 형식을 범용적으로 많이 사용되는 UTF-8로 설정하면, 워크스페이스에 작성되는 모든 프로젝트의 소스코드에 한글이 포함될 경우, 한글이 정상적으로 표시 및 저장될 수 있습니다.

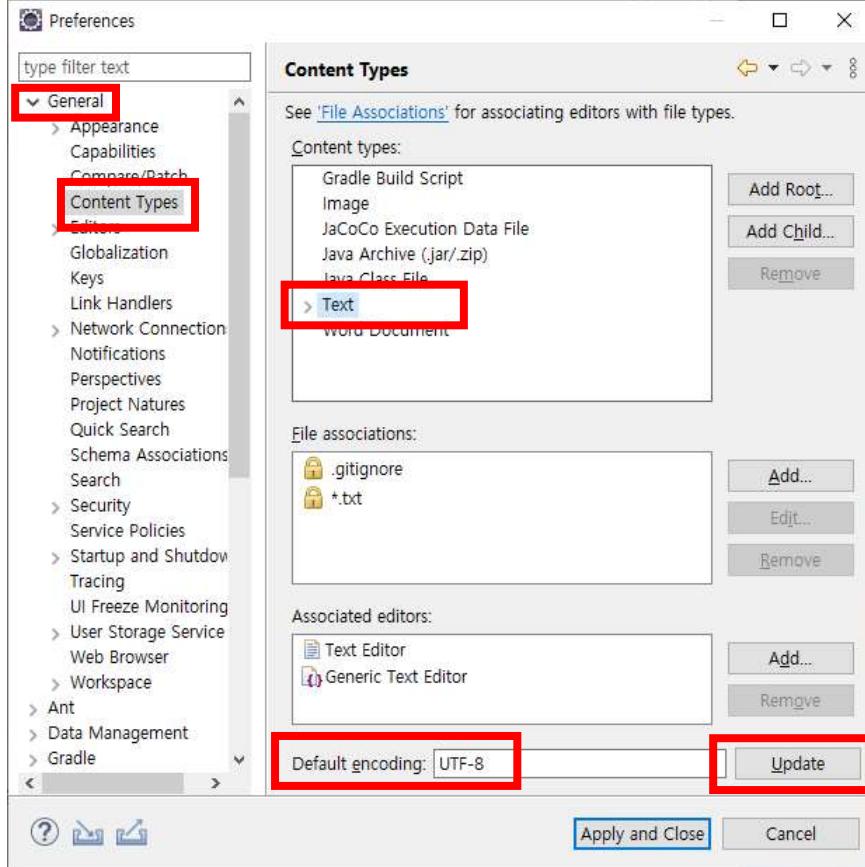
☞ 앞으로 실습에서 생성되는 소스파일들이 작성 및 저장될 때 UTF-8 인코딩을 사용하도록

앞에서 생성한 myJavaWorkspace 워크스페이스에 구성합니다(설정할 항목이 많으니, 주의하시기 바랍니다).

→ 이클립스의 위에 있는 window 메뉴 클릭 → Preferences 항목 클릭 → Preferences 창이 표시됩니다.

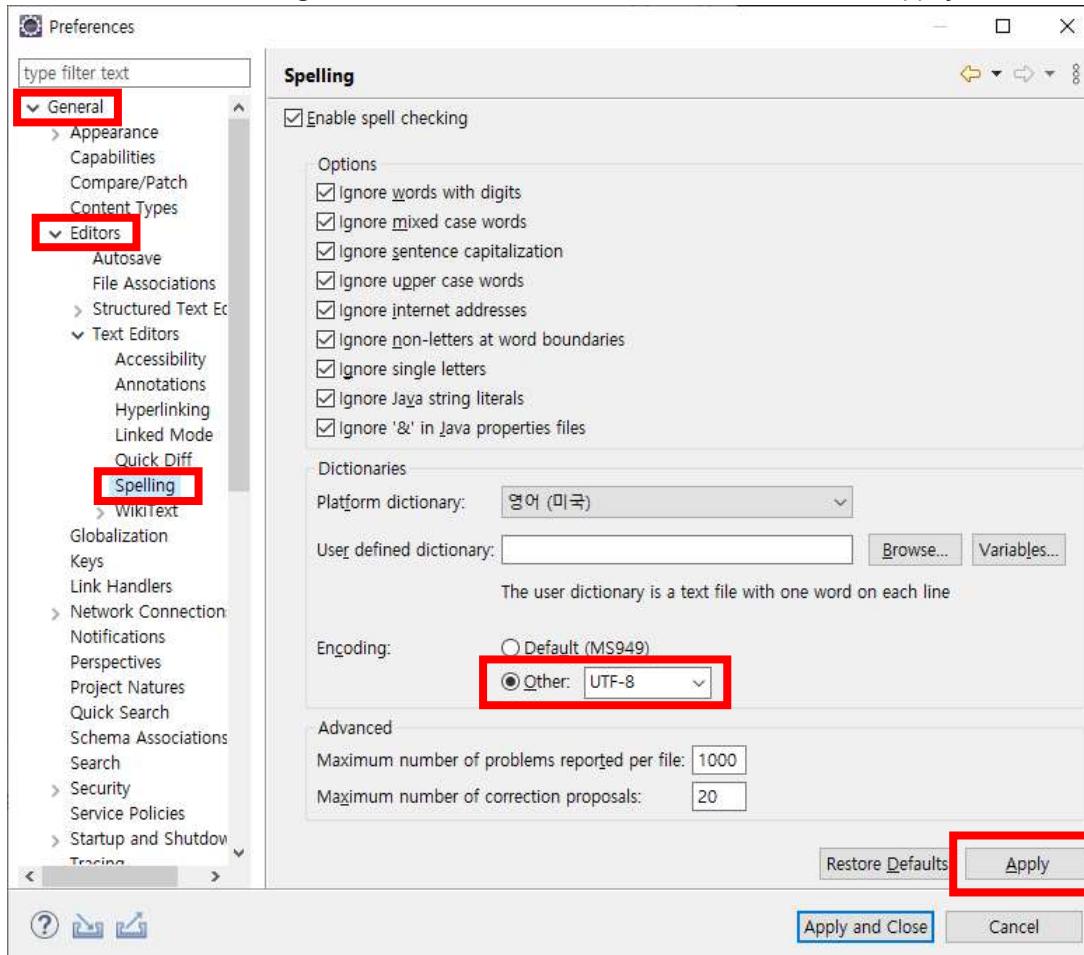
→ 왼쪽에서 General → Content Types 클릭

→ 오른쪽에서 Content types 부분의 Text 클릭 후, 제일 밑의 Default encoding 입력란에 UTF-8 입력 → Update 클릭



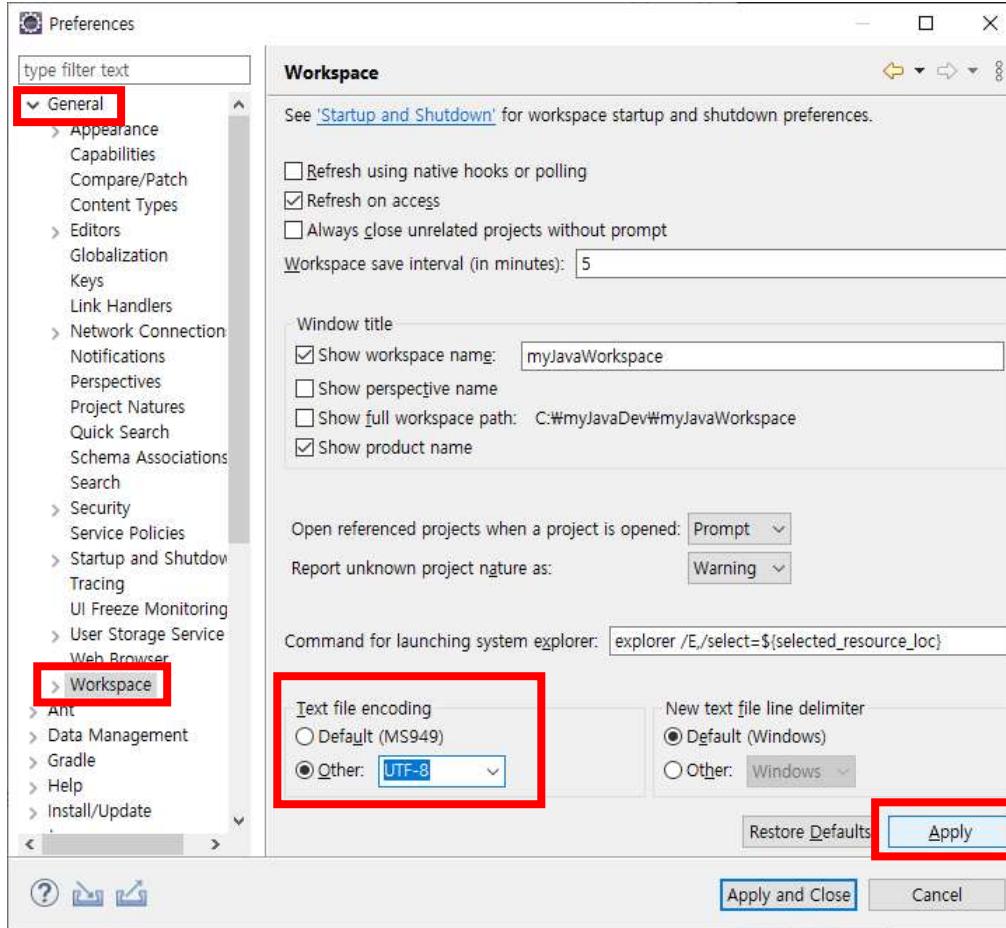
→ 왼쪽에서 General → Editors 더블클릭(확장) → Spelling 클릭

→ 오른쪽에서 Encoding 부분 항목을 Other 선택 후, UTF-8 선택 → Apply 클릭



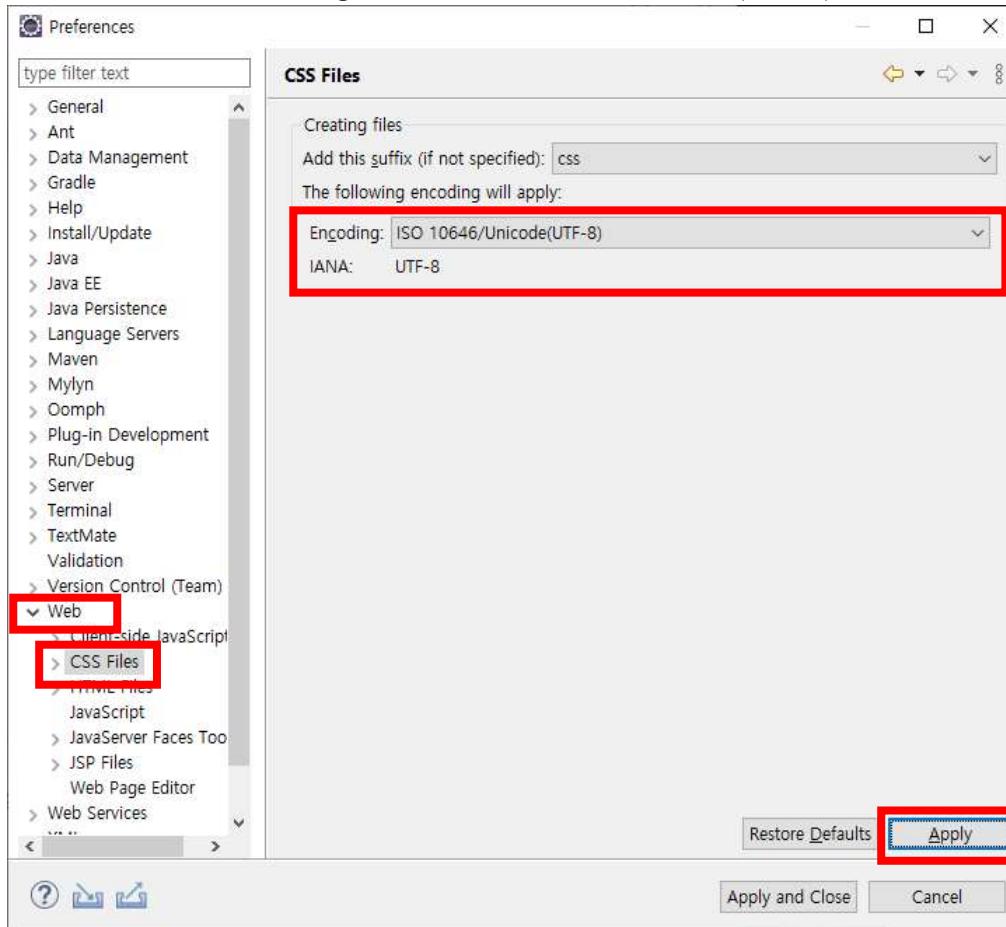
→ 왼쪽에서 General → Workspace 클릭

→ 오른쪽의 밑에 있는 Text file encoding 부분에서 Other 항목을 선택 후, UTF-8 선택 → Apply 클릭



→ 왼쪽에서 Web 더블클릭(확장) → CSS Files 클릭

→ 오른쪽에서 Encoding 항목값을 ISO 10646/Unicode(UTF-8) 값으로 변경 → Apply 클릭(오른쪽 그림)



→ 동일한 방법으로 왼쪽에서 Web 항목에 있는 HTML Files 와 JSP Files 도

Encoding 항목값을 ISO 10646/Unicode(UTF-8) 값으로 변경합니다. (주의) 각각 설정할 때마다 Apply를 클릭해야 합니다.

→ Apply and Close를 클릭하여 변경 사항을 적용 후, Preferences 창을 닫습니다.

☞ UTF-8 인코딩 지정은 여러 방법이 있지만, 이 섹션의 수행 방법이 제일 확실하게 작동합니다.

☞ 워크스페이스에 UTF-8 인코딩 지정 완료

[참고] 코드 작성 시에 사용되는 텍스트 편집기의 글자 크기나 폰트는 Preferences 창의 다음 항목에서 변경할 수 있습니다.

→ 왼쪽에서 General → Appearance 더블클릭(확장) → Colors and Fonts 클릭

→ 오른쪽에서 BASIC 더블클릭(확장) → Text Font 더블클릭 → 글꼴과 크기를 설정하는 창이 표시되면, 취향에 맞게 설정합니다.

폰트 선택 시, 영문 기준으로 대문자 아이(I), 소문자 엘(l), 숫자 일(1) 이 구분되는 폰트를 사용하는 것을 권장합니다.

보통, Consolas, Verdana, Courier New 폰트가 위의 문자를 구분할 수 있어서 많이 사용됩니다.

[참고] 이클립스 자동 업데이트 기능 해제

왼쪽에서 Install/Update 더블클릭(확장) → Automatic Updates 클릭

→ 오른쪽에서 제일 위의 Automatically find new updates and notify me 체크를 없앰 → Apply 클릭 → Apply and Close 클릭

[참고] 이클립스에서 JSP 작성 시, HTML 요소의 속성과 관련된 경고 표시를 없애는 방법(권장하진 않음).

이클립스 개발환경에서 자주 표시되는 "Undefined attribute name" 경로는 HTML에서 지정하지 않은 속성을 사용하고 있다는 경고인데 추가속성을 사용해야 하는 상황이라면 이 경고를 제거할 수 없을 것이므로 경고가 많아 질 수 있다. 이로 인해 더 중요한(심각한 문제를 일으킬 수 있는) 경고를 인지하지 못하게 될 수 있다. 결국 HTML에서 발생하는 경고는 일부 무시할 수밖에 없다.

개발자 본인의 책임이 될 수 있지만, 다음의 방법으로 위의 경고를 표시되지 않도록 설정할 수 있다.

이클립스 메뉴 중 Window → Preferences 클릭 → 왼쪽창에서 Web → HTML files → Validation 클릭

→ 오른쪽 창에서 Attributes 항목을 클릭 → Undefined attribute name 항목 값을 Ignore 로 설정 후, Apply 클릭

→ Apply and Close 클릭

## [DE-06] 톰캣 웹 애플리케이션 서버 설치

- 다음의 작업들이 진행됩니다.
  - 톰캣 9 다운로드(압축파일 형태)
  - 톰캣 9 설치(압축해제)

☞ 톰캣의 경우도 이클립스처럼 인스톨러 형태와 압축파일 형태로 소프트웨어를 제공합니다.

☞ 필요한 경우 프로젝트마다 별도의 톰캣을 구성해서 사용하기도 하므로,  
압축파일 형태의 프로그램을 다운로드 받아 사용할 것을 권장합니다.

☞ 현재 Tomcat 10이 2021-04-06일자로 정식버전으로 발표되었지만, 현재까지 사용된 JavaEE 기반 Servlet/JSP 컨테이너가 아닌 JakartaEE9 기반의 Servlet/JSP 컨테이너만 지원하는 버전입니다.  
따라서, 문서에서는 JavaEE 기반 Servlet/JSP 컨테이너를 지원하는 톰캣 9 버전을 다운로드 받아 사용합니다.

### 1. 톰캣 9 다운로드(압축파일 형태)

→ 웹 브라우저에서 <http://tomcat.apache.org> 사이트에 접속합니다.

→ 사이트 내용 왼쪽에 있는 Which version? 을 클릭하여, 다운로드 받으려는 버전(문서에서는 9.0.x)의 스펙사항을 확인합니다.

Servlet Spec	JSP Spec	EL Spec	WebSocket Spec	Authentication (JASIC) Spec	Apache Tomcat Version	Latest Released Version	Supported Java Versions
5.0	3.0	4.0	2.0	2.0	10.0.x	10.0.5	8 and later
4.0	2.3	3.0	1.1	1.1	9.0.x	9.0.45	8 and later
3.1	2.3	3.0	1.1	1.1	8.5.x	8.5.65	7 and later
3.1	2.3	3.0	1.1	N/A	8.0.x (superseded)	8.0.53 (superseded)	7 and later
3.0	2.2	2.2	1.1	N/A	7.0.x	7.0.108	6 and later (7 and later for WebSocket)

→ 동일 페이지의 왼쪽에서 Tomcat 9 링크를 클릭 → Tomcat 9 Software Downloads 페이지가 표시됨

→ 아래로 스크롤하여, 아래 그림처럼 버전 부분으로 이동 → 64-bit Windows zip 링크를 클릭하여 압축파일을 다운로드 받습니다.

9.0.45

Please see the [README](#) file for packaging information. It explains what every distribution contains.

Binary Distributions

- Core:
  - [zip \(pgp, sha512\)](#)
  - [tar.gz \(pgp, sha512\)](#)
  - [32-bit Windows zip \(pgp, sha512\)](#)
  - [64-bit Windows zip \(pgp, sha512\)](#)
  - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)
- Full documentation:
  - [tar.gz \(pgp, sha512\)](#)
- Deployer:
  - [zip \(pgp, sha512\)](#)
  - [tar.gz \(pgp, sha512\)](#)
- Embedded:
  - [tar.gz \(pgp, sha512\)](#)
  - [zip \(pgp, sha512\)](#)

## 2. 톰캣 9 설치(압축해제)

→ 다운로드 된 apache-tomcat-9.0.45-windows-x64.zip 파일을 C:\myJavaDev 폴더에 복사한 후,  
‘여기에 풀기’ 옵션으로 압축을 해제합니다.

→ 압축해제 된 폴더 이름을 tomcat9로 변경합니다.

☞ 톰캣 9 설치 완료

## [DE-07] 이클립스 워크스페이스에서 톰캣 사용 구성

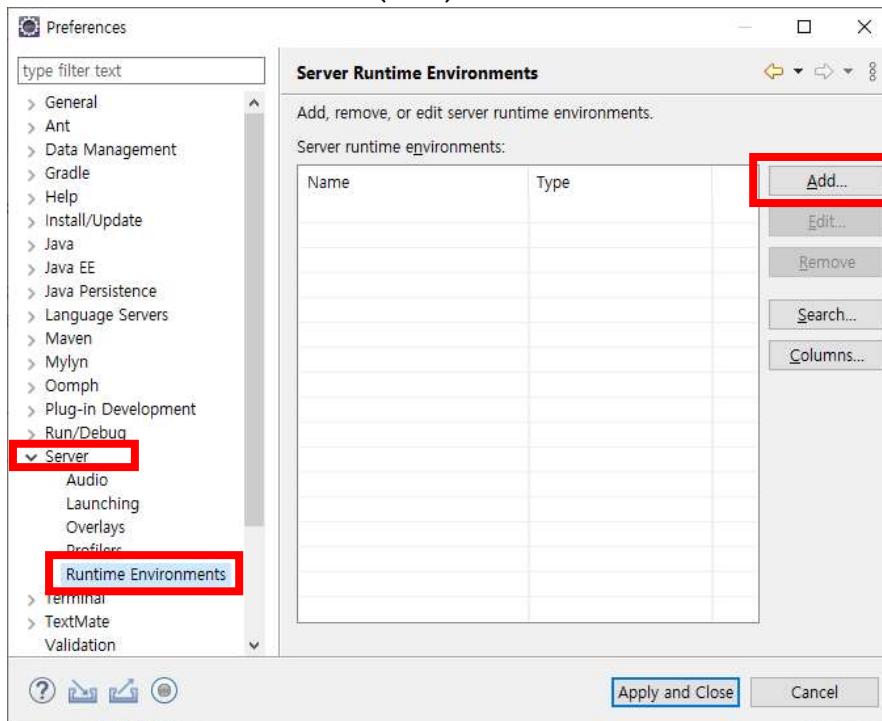
- 다음의 작업들이 진행됩니다.
  - 이클립스 워크스페이스에서 톰캣 9 사용 구성: 이클립스에서 사용되는 서버 자원 추가.
  - 톰캣 9를 서버 프로젝트로 등록: 프로젝트와 톰캣 9 서버 연동 준비.

### 1. 이클립스 워크스페이스에서 톰캣 9 사용 구성: 이클립스에서 사용되는 서버 자원 추가.

☞ 이 단계에서는 이클립스에서 사용할 수 있는 톰캣 9 서버를 워크스페이스에 등록시킵니다.

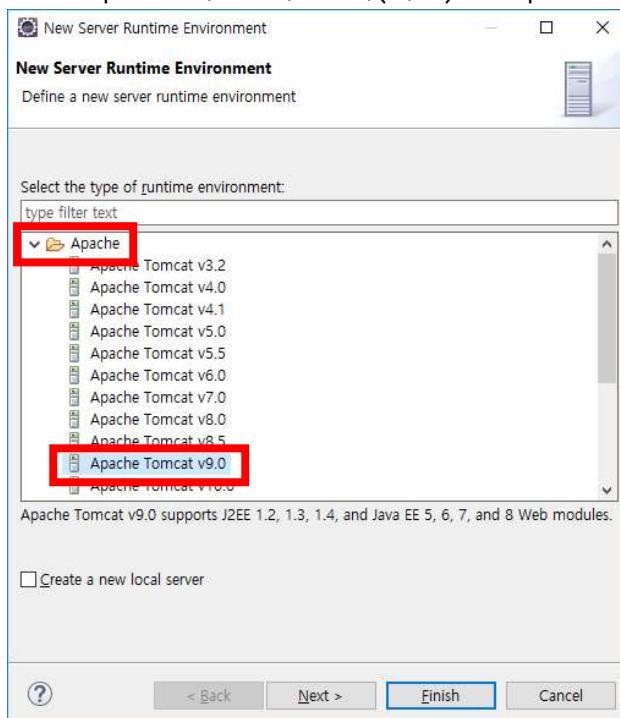
→ 이클립스의 위에 있는 window 메뉴 클릭 → Preferences 항목 클릭 → Preferences 창이 표시됩니다.

→ 왼쪽에서 Server 더블클릭(확장) → Runtime Environments 클릭 → 오른쪽에서 Add 클릭(왼쪽 그림)

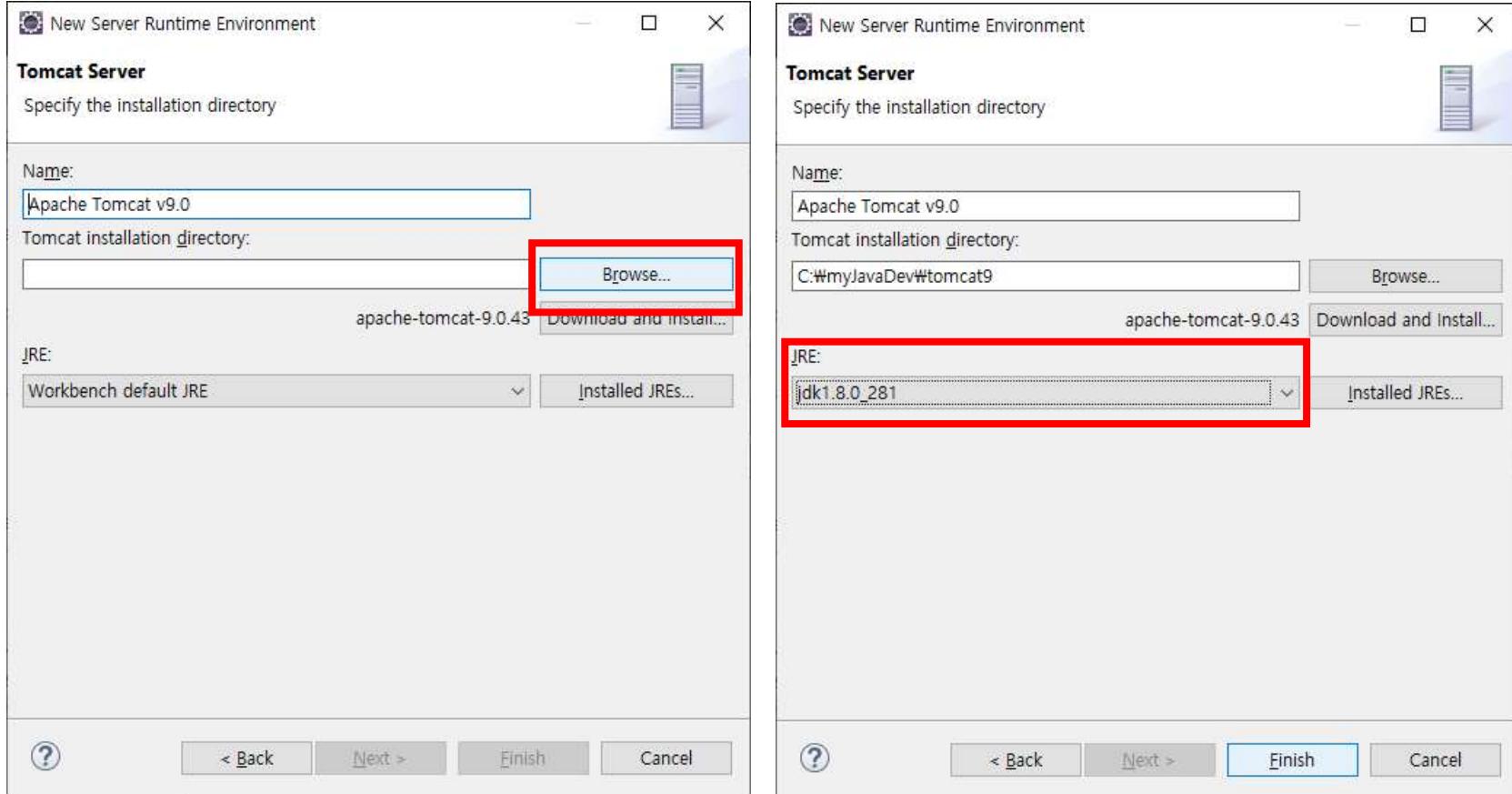


→ New Server Runtime Environments 창 표시됨(오른쪽 그림)

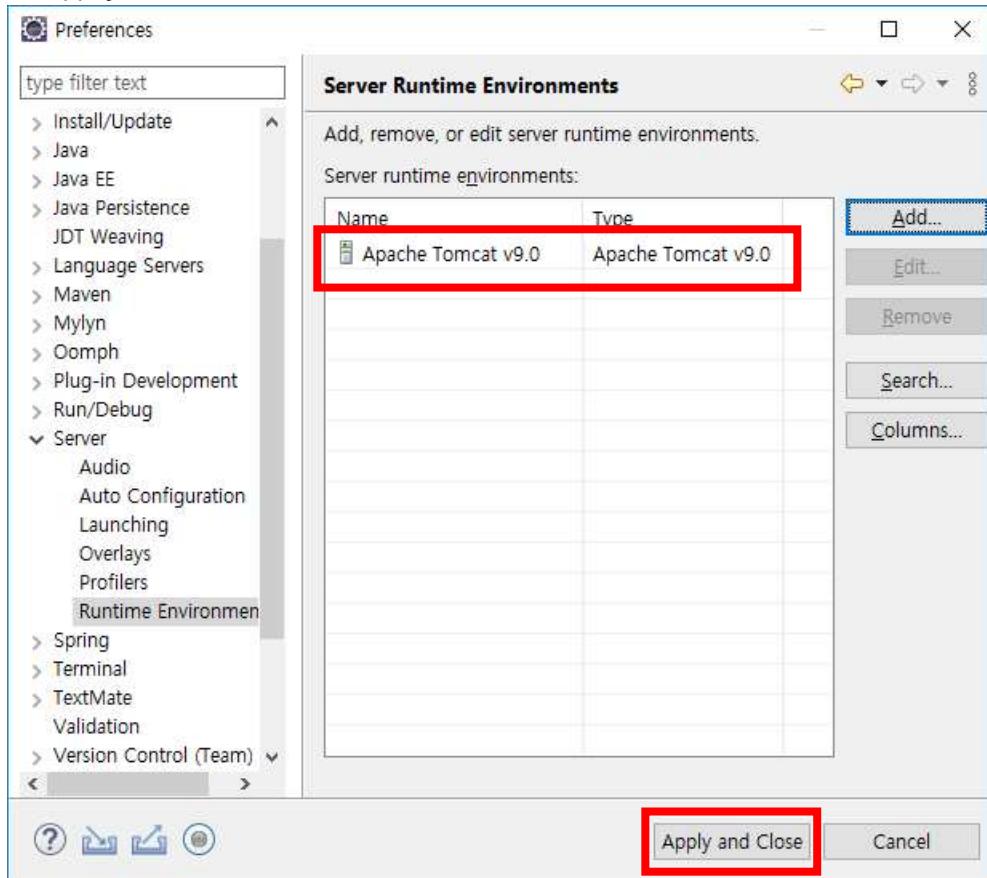
→ Apache폴더 를 더블클릭(확장) → Apache Tomcat v9.0 항목 선택 후, Next 클릭 → Tomcat Server 창이 표시됩니다.



- Tomcat Installation directory 항목의 Browse... 버튼을 클릭(왼쪽그림)
- 폴터 선택 창에서 톰캣 9가 설치된 폴더(C:\myJavaDev\tomcat9)를 선택하여 설정
- JRE 항목을 클릭하여 jdk\_1.8.0\_281을 선택한 후, Finish 클릭(오른쪽 그림)



- Server Runtime Environments 창 표시됨
- 추가된 서버 항목이 화면에 표시되는지 확인
- Apply and Close를 클릭하여 Server Runtime Environments 창 닫음



☞ 이클립스에 톰캣 9 서버 추가가 완료되었습니다.

## 2. 톰캣 9를 서버 프로젝트로 등록: 프로젝트와 톰캣 9 서버 연동 준비.

- ☞ 이 단계에서는 워크스페이스에 등록된 톰캣 9 서버를 프로젝트와 연동 시키기 위하여 톰캣 9 서버 프로젝트를 생성합니다.  
이 단계를 진행해야만, 앞으로 생성되는 프로젝트와 톰캣 서버를 연동시킬 수 있습니다.

→ 이클립스의 Servers 뷰에 [No servers are available. Click this link to create a new server] 링크를 클릭합니다.

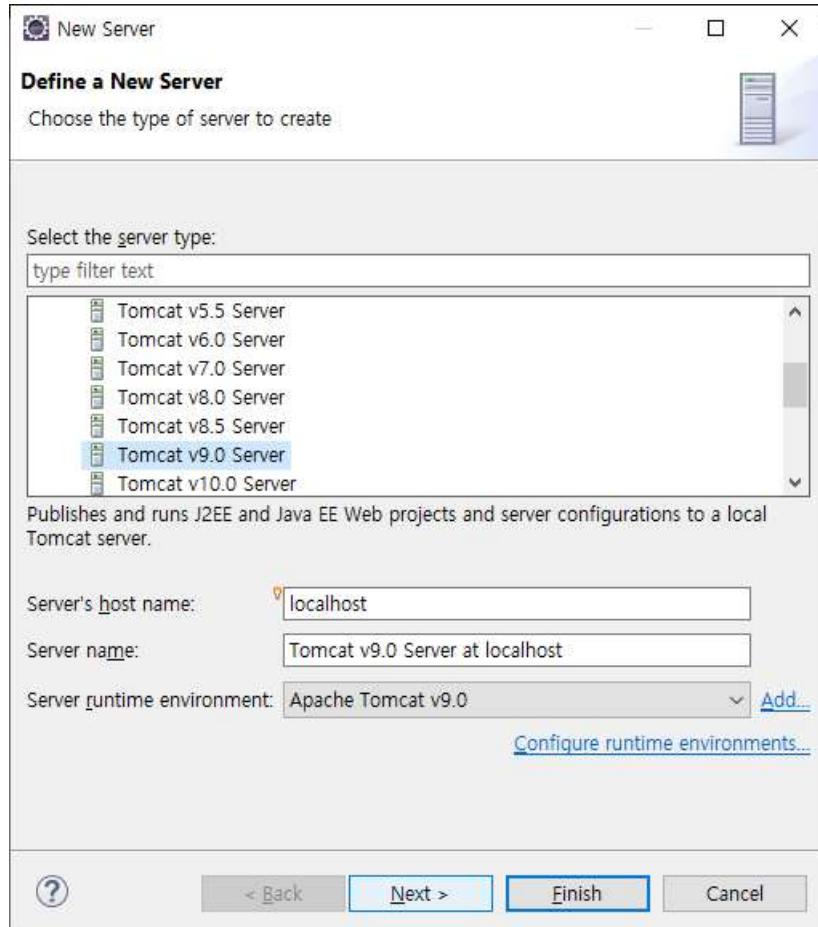
→ Define a New Server 창이 표시됩니다.



→ Define a New Server 창의 입력란에는 다음의 값들이 설정됩니다. 문서에서는 기본값을 그대로 사용합니다.

- Select the server type 에는 표시된 항목들 중 사용하려는 서버의 유형을 지정합니다. 이클립스에 추가된 (앞 섹션에서 추가함) 서버 중 프로젝트와 연동 시킬 톰캣 서버의 버전에 해당하는 유형을 선택합니다. 실습에서는 Tomcat v9.0 Server 를 설정합니다.
- Server's host name 입력란에는 톰캣이 실행 중인 시스템의 hostname 이나 IP 주소를 설정합니다.  
개인 PC의 개발환경에 구성된 톰캣을 사용하는 경우, 대부분 localhost 를 설정합니다.
- Server name 에는 사용되는 톰캣 서버에 대하여 이클립스의 서버 뷰에서 표시되는 이름을 입력합니다.  
보통 기본 입력값을 그대로 사용하는 경우가 많습니다.
- Server runtime environment 에는 이클립스에 추가된 (앞 섹션에서 추가함) 서버 중 프로젝트와 연동 시킬 톰캣 9 서버를 지정합니다.

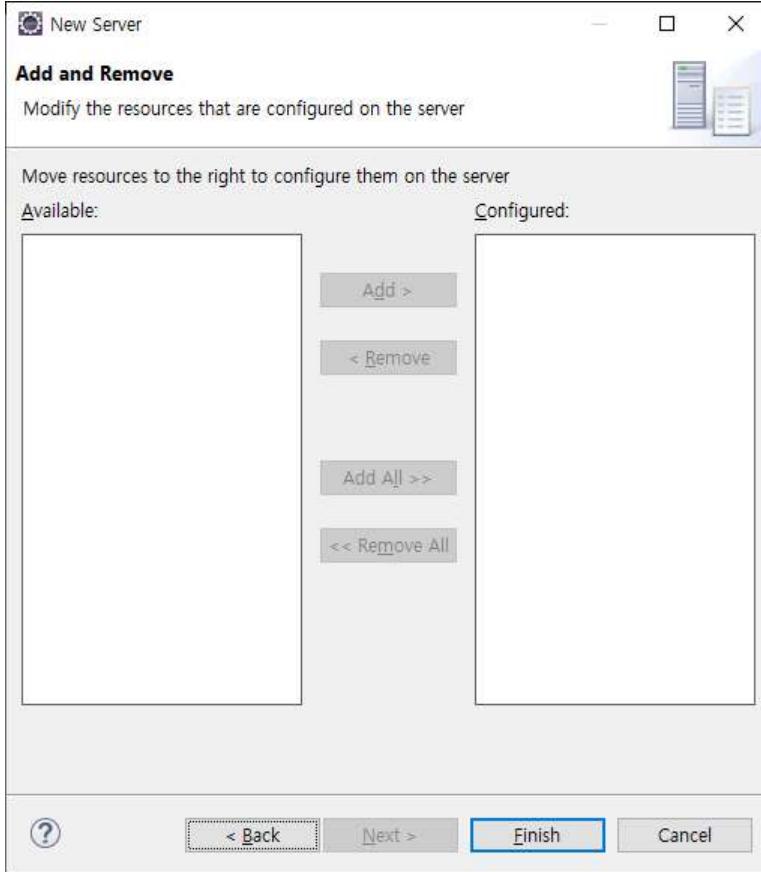
→ 실습에서는 기본값을 그대로 사용하겠습니다. → 입력 값을 확인 후, Next 클릭.



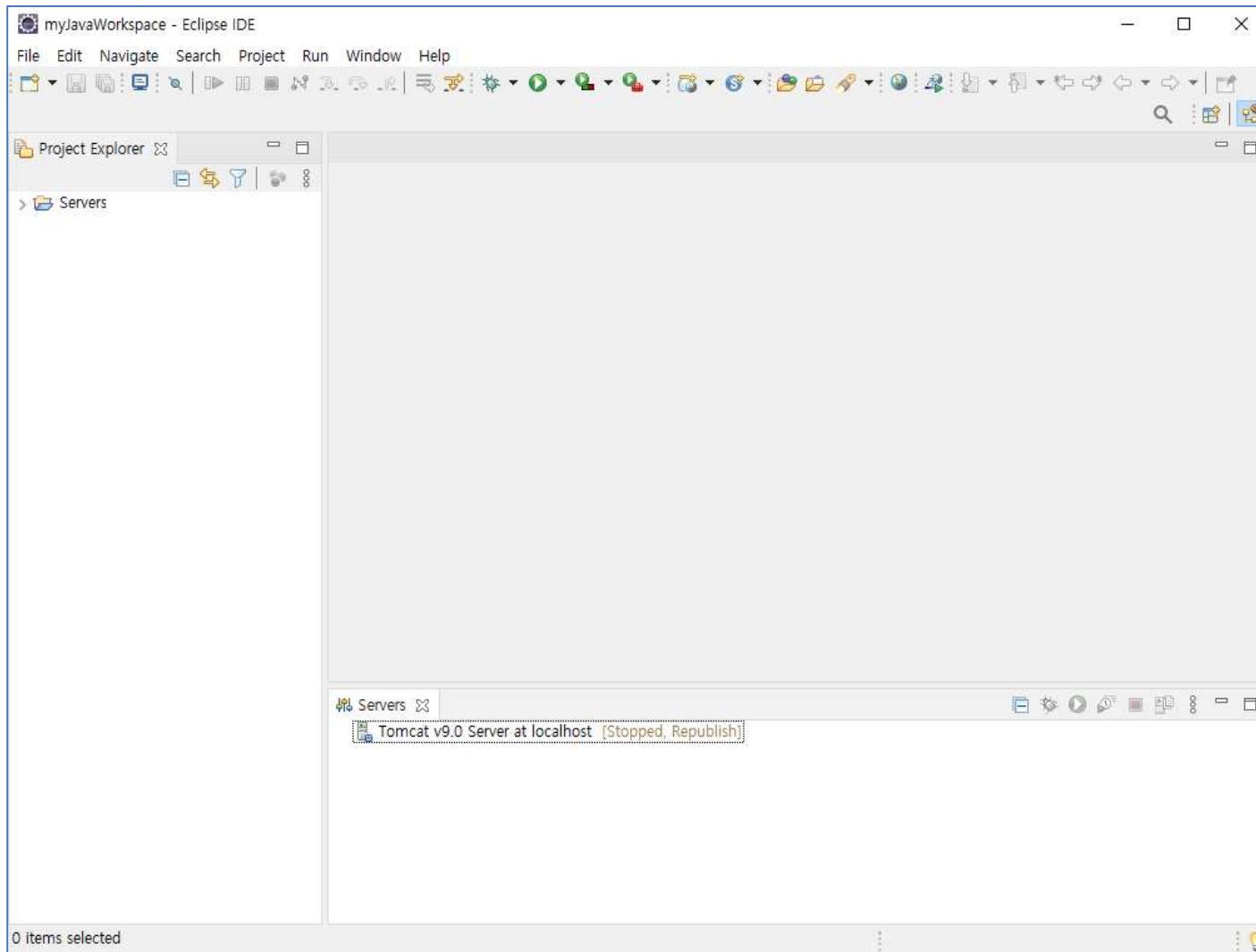
→ Add and Remove 창이 표시됩니다. 서버와 연동시킬 프로젝트를 지정하는 단계입니다.

☞ 현재 서버에 추가할 프로젝트가 없으므로 Available 부분에 아무것도 표시되지 않습니다.

→ Finish를 클릭



→ 설정을 마치면, 아래처럼 Project Explorer 뷰에 Servers 프로젝트가 생성되고 Servers 뷰에 추가한 서버이름이 표시됩니다.



☞ 이 후에 프로젝트 생성 후, 생성된 서버에 프로젝트를 추가하면, 구성된 톰캣 9 서버와 프로젝트가 이를립스에서 연동됩니다.

## [DE-08] 오라클 데이터베이스 11gR2 Express Edition 설치

- 다음의 작업들이 진행됩니다.
  - 오라클 데이터베이스 11gR2 Express Edition 다운로드 및 설치
  - 설치 확인 및 방화벽 예외 프로그램으로 등록
  - 오라클 XDB 서비스 포트 변경(8080 → 0) 및 서비스 사용 메모리 변경

### 1. 오라클 데이터베이스 11gR2 Express Edition 다운로드

☞ 오라클 데이터베이스 Express Edition의 경우 18c 버전이 최신 버전이지만, 실습 데이터 구성이 제외되어 있고, PC 상태에 따라 설치 상에 오류가 많이 발생되므로, 개인적으로는 11gR2 버전을 사용하는 것을 추천합니다. 문서에서는 오라클 데이터베이스 11g R2 Express Edition (64 비트)를 설치하여 사용합니다.

→ 웹 브라우저에서 <https://www.oracle.com/database/technologies/xe-prior-releases.html> 사이트에 접속합니다.

→ Oracle Database XE Prior Release Archive 페이지가 표시됩니다.

[Oracle Database 11gR2 Express Edition for Windows x64](#) 링크를 클릭하여 설치파일을 다운로드 합니다.

→ 다운로드 시에, 로그인을 요청합니다. → 미리 생성해 둔 계정으로 로그인 합니다 → 다운로드가 진행됩니다.

☞ 회원가입을 하지 않은 경우, 표시된 로그인 페이지에서 "계정 만들기"를 클릭하여 회원가입을 수행한 후, 다시 다운로드를 수행합니다.

☞ Oracle Database Express Edition (이하 XE)은, 개발용도로만 사용되는 것을 목적으로 하며, 쉽게 개발환경을 구축할 수 있도록 소프트웨어를 설치할 때, 기본적으로 XE라는 이름의 오라클 데이터베이스 서비스와 LISTENER라는 이름의 오라클 리스너 서비스(사용자 접속포트: 1521)가 자동으로 구성되며, 기본적으로 소프트웨어 설치 중에는 관련 설정을 변경할 수 없습니다.

또한 Enterprise Edition과 Standard Edition과는 달리 새로운 데이터베이스 서비스를 생성할 수 없습니다.

그리고 단일 사용자만 지원하므로, XE가 설치된 PC를 여러 사용자 계정으로 이용하는 경우, XE를 설치한 사용자만 XE 소프트웨어와 오라클 서비스를 사용할 수 있습니다.

## 2. 오라클 데이터베이스 11gR2 Express Edition 설치

→ Windows 운영체제 로그인 사용자 이름이 영문 한 단어로 된 사용자로 접속된 상태인지 확인합니다. 한글이 포함되거나 둘 이상의 단어로 된 사용자 이름으로 로그인 한 경우, 오라클 소프트웨어 설치 시에 불필요한 오류가 발생될 수 있습니다.

☞ 만약 영문 한 단어로 된 Windows 운영체제 사용자로 로그인 되지 않은 경우, 문서 제일 앞의 설명한 Window 10 사용자 추가의 내용을 참고하여, 영문 한 단어로 된 Windows 운영체제 사용자로 로그인하여 오라클 설치를 반드시 진행합니다.

→ 이클립스가 실행 중인 경우, **이클립스를 중지합니다.**

☞ 오라클 XE 설치 중에 구성되는 오라클 서비스에서 8080 포트를 사용하는 기능이 활성화 됩니다.

그런데, 이 8080 포트는 톰캣 서버의 기본 포트로 설정되어 있고, 앞으로 실습에서도 톰캣 접속포트로 사용됩니다.

따라서, **이클립스를 중지하여, 오라클 설치 중에는 8080 포트가 사용되지 않는 상태로 해줍니다.**

오라클 설치 후에, 오라클에서 8080 포트를 사용하지 못하도록 구성합니다.

→ 다운로드 된 OracleXE112\_Win64.zip 파일을 소프트웨어를 C:\ 폴더에 복사합니다.

☞ 압축해제된 오라클 설치파일들 중, 디렉토리를 포함한 파일이름이 매우 긴 것들이 인식되지 않는 경우가 발생될 수 있기 때문에 C:\ 폴더에 오라클 설치 압축파일을 위치시키고 압축을 해제합니다.

→ 오라클 설치 압축파일을 "여기에 풀기" 옵션으로 압축해제 합니다.

☞ 압축해제 시 Windows 운영체제에서 제공하는 압축해제 기능을 사용하지 마십시오. 시간이 매우 오래 걸릴 수 있습니다.

반드시 별도의 압축해제 프로그램을 이용하시기 바랍니다.

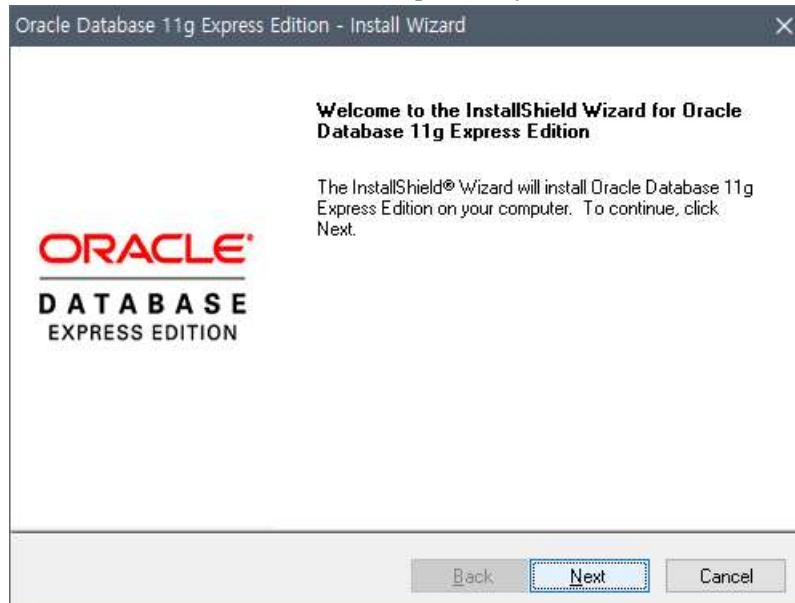
☞ Disk1 이름의 폴더가 생성된 후, Disk1 폴더 안에 설치파일들이 압축해제 됩니다.

→ C:\Disk1 폴더로 이동 → setup.exe 파일을 마우스 우측 버튼으로 클릭 → "관리자 권한으로 실행" 항목을 클릭하여 오라클 XE 인스톨러를 실행.

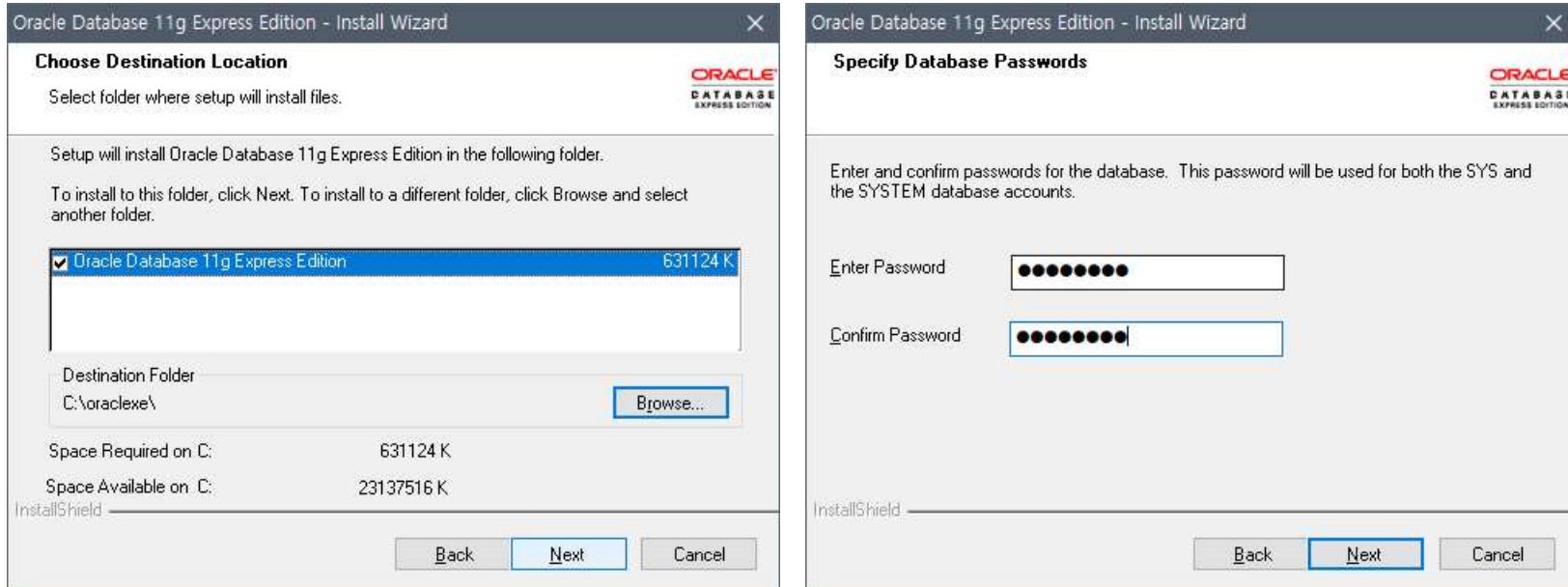
→ 이 앱이 디바이스를 변경할 수 있도록 허용하시겠어요? 물어오는 사용자 계정 컨트롤창이 표시되면, 예 클릭

→ Welcome 창이 표시됨, Next 클릭

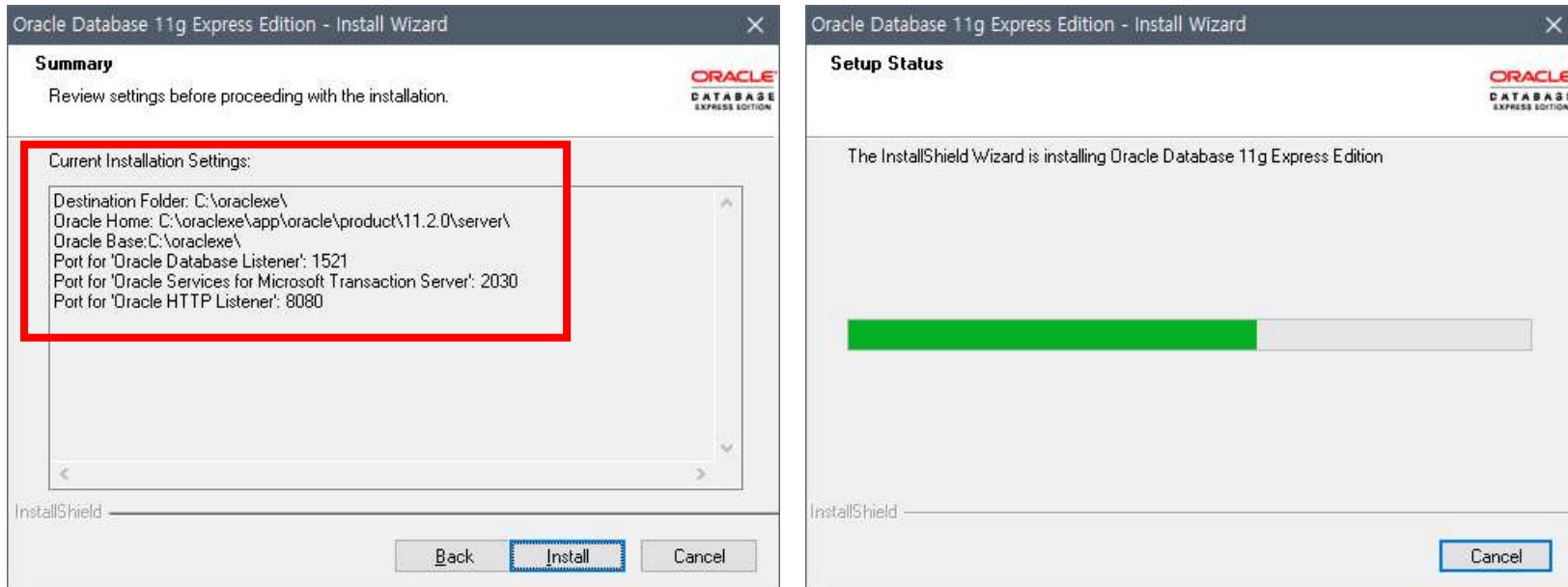
→ 라이선스 동의창이 표시됨, [I accept the terms in the license agreement] 항목을 선택한 후, Next 클릭



- 설치 위치 지정, 기본 설정된 C:\oraclexe를 그대로 사용함, Next 클릭
- 소프트웨어 설치 시에 구성되는 오라클 데이터베이스의 관리자인 SYS와 SYSTEM 계정의 암호를 설정
- oracle4U 를 각각 입력 후, Next 클릭

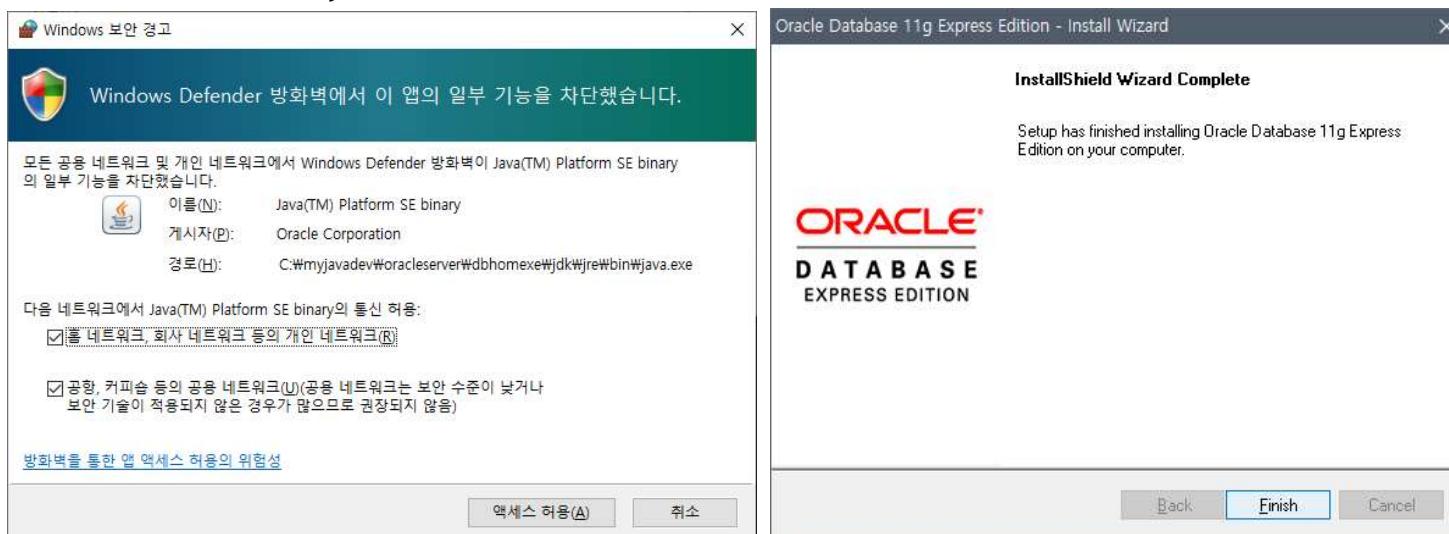


- 설치 요약 창 표시됨, 내용 확인 후, Install 클릭 → 설치가 시작됨



- 설치 진행 중에 Windows 보안 경고창이 표시되면 액세스 허용 클릭(오른쪽 그림)

- 설치 완료창이 표시됨, Finish를 클릭하여 설치 종료



☞ 설치 완료

### 3. 설치 확인

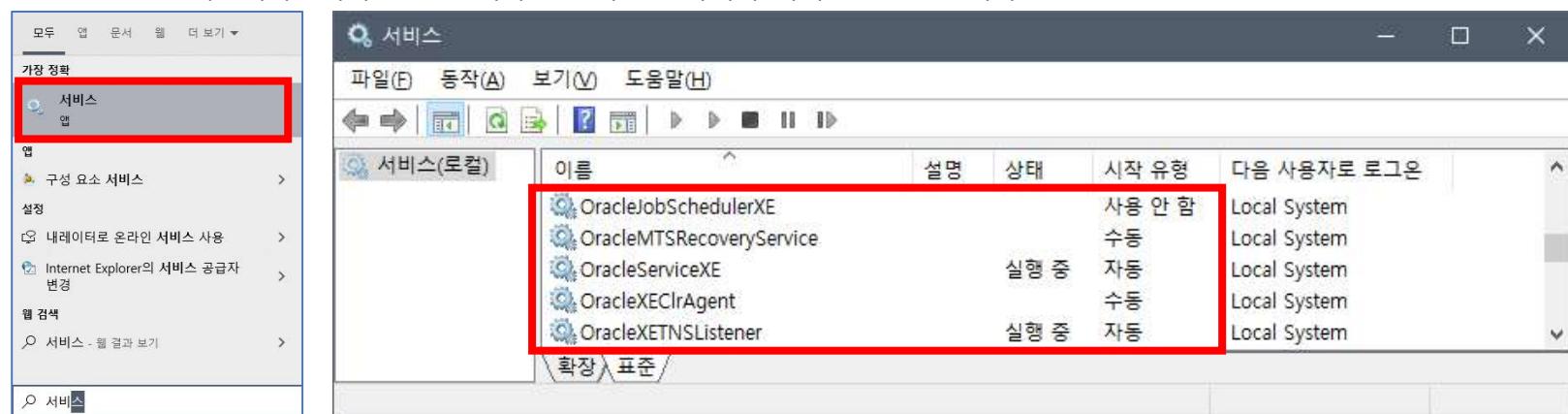
→ 파일탐색기를 열어 C:\oraclexe\app\oracle\oradata\XE 폴더로 이동

→ 아래 그림처럼 오라클 데이터베이스 파일(이 중 일부 파일에 서비스 상의 데이터가 저장됨)들이 존재해야 합니다.



☞ 위의 파일들 중 일부의 파일에, 이 후의 문서 실습에서 제작되는 웹 프로그램들에 의하여 생성되는 데이터가 저장되며, 이들 파일들을 오라클 데이터베이스라고 합니다. 이 파일들의 데이터를 서비스하는 오라클 프로세스를 오라클 데이터베이스 서비스라고 합니다.

→ 프로그램 검색란에서 "서비스"를 입력하고 검색된 결과에서 서비스를 실행합니다.



☞ 오라클 XE 가 정상적으로 설치된 경우, 위의 오른쪽 그림처럼 Windows 서비스에 오라클 서비스 관련 항목들이 생성됩니다.

☞ OracleServiceXE 서비스가 오라클 데이터베이스 서비스(해당 프로그램: oracle.exe)이고, OracleXETNSListener 서비스가 오라클 리스너 서비스(해당 프로그램: TNSLNR.exe)입니다.

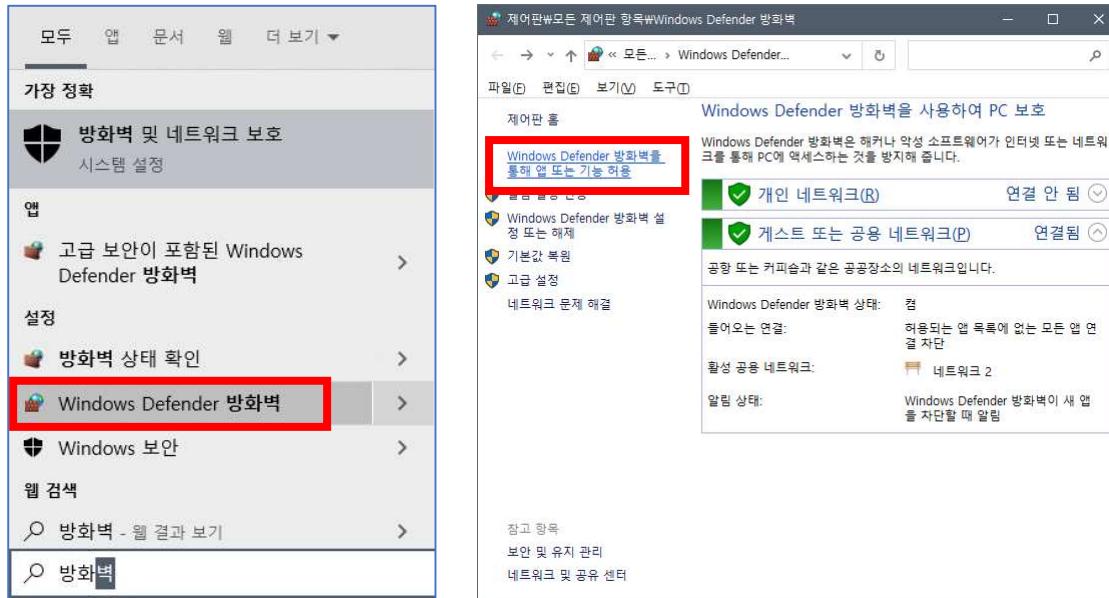
☞ OracleServiceXE와 OracleXETNSListener 서비스의 시작유형이 자동으로 설정되어야 하며, 상태가 실행 중으로 표시되어야 합니다.

### 4. 방화벽 예외 프로그램으로 등록

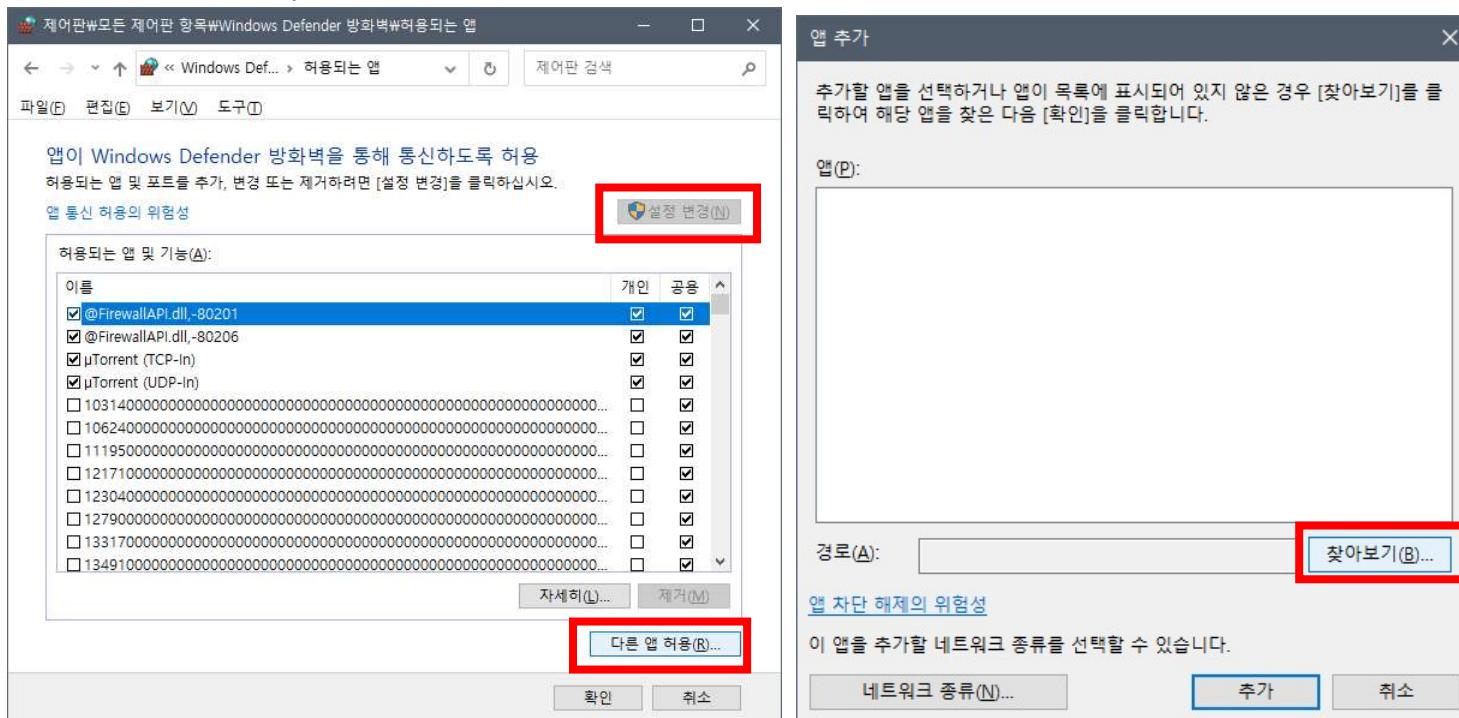
☞ 이 후에 설치되는 오라클 데이터베이스 개발툴인 SQL\*Developer로 원격 접속이 정상적으로 처리되기 위하여 앞에서 확인한 오라클 데이터베이스 서비스와 오라클 리스너 서비스에 해당하는 프로그램들을 Windows 방화벽의 예외 프로그램으로 설정합니다.

→ 프로그램 검색란에서 "방화벽"를 입력하고 검색된 결과에서 "Windows Defender 방화벽" 실행

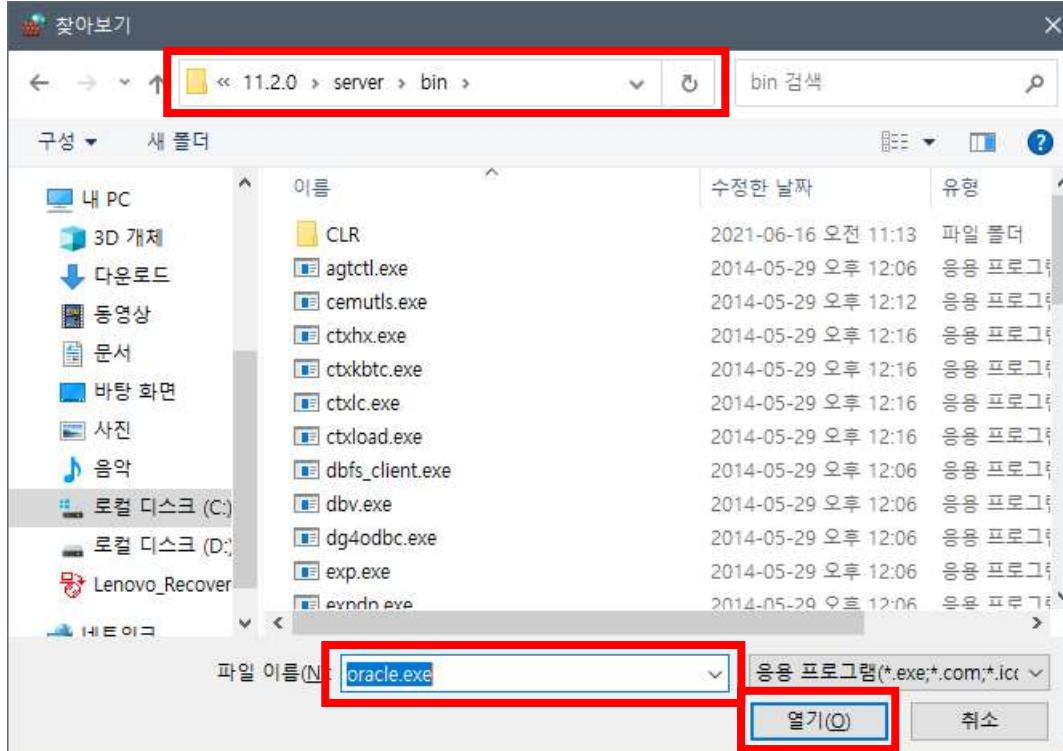
→ "Windows Defender 방화벽을 통해 앱 또는 기능 허용" 링크를 클릭합니다(방화벽 적용 예외로 설정됨).



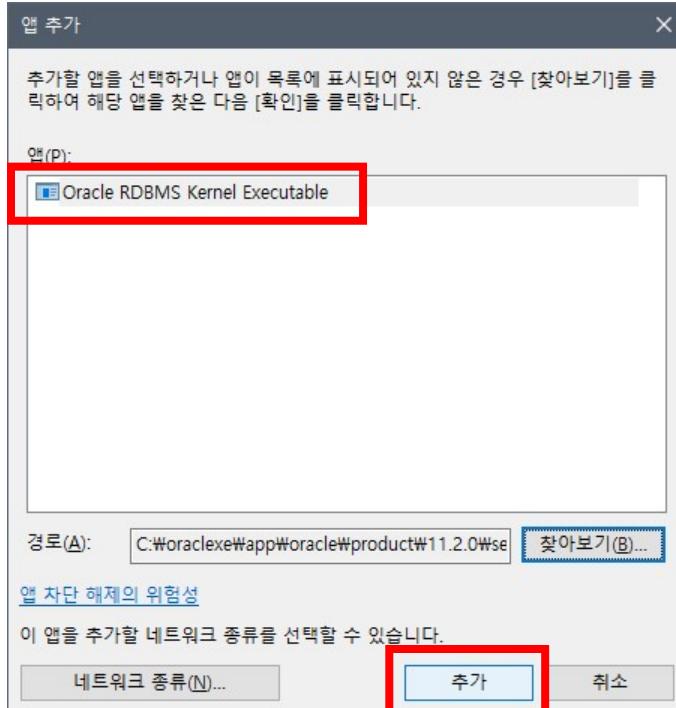
→ "설정변경" 클릭 후, "다른 앱 허용" 클릭 → 앱 추가 창에서 찾아보기 클릭



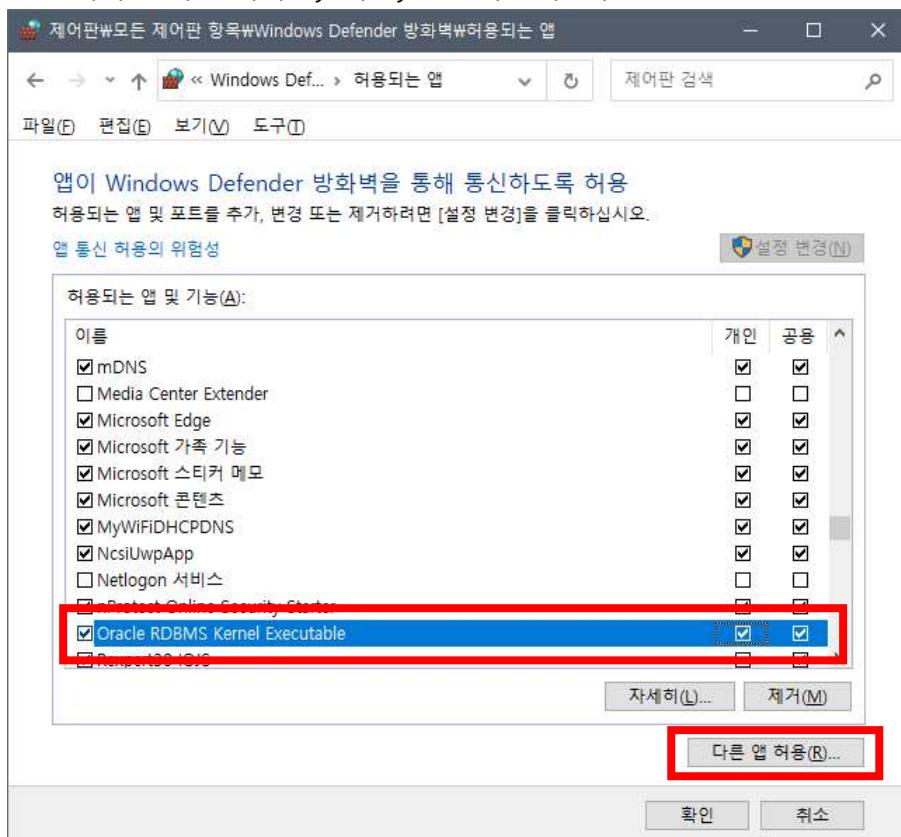
→ 찾아보기 창에서 C:\oracle\app\oracle\product\11.2.0\server\bin 폴더로 이동 후, oracle.exe 파일 선택 → 열기 클릭



→ 앱 추가 창에 Oracle RDBMS Kernel Executable 항목이 표시됨 → 추가 클릭



→ 방화벽 창이 표시되면, 개인, 공용에 모두 체크

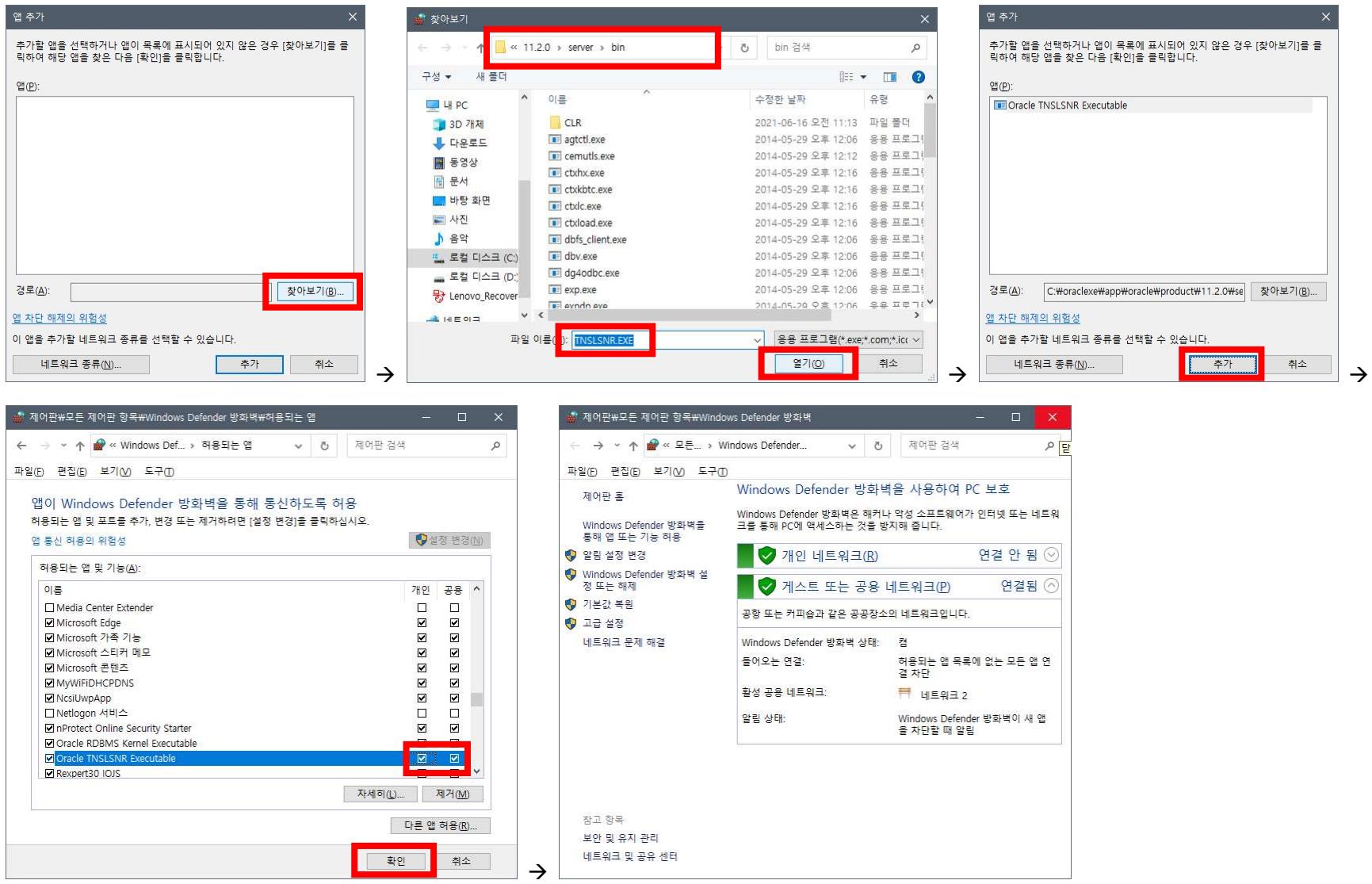


☞ 오라클 데이터베이스 서비스 프로세스(oracle.exe)의 방화벽 예외 설정 완료

→ 다른 앱 허용 클릭

→ 동일한 방법으로, C:\oracle\app\oracle\product\11.2.0\server\bin 폴더에 있는 TNSLSNR.EXE도 등록하여  
오라클 리스너 서비스 프로세스 (TNSLSNR.EXE)의 방화벽 예외 설정도 완료하십시오

☞ 아래의 그림 참고

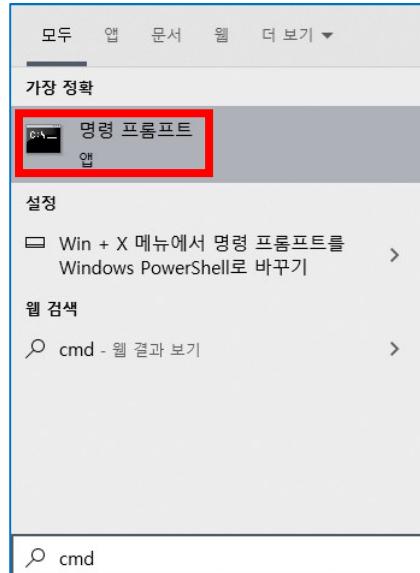


## 5. 오라클 XDB 서비스 포트 변경(8080 → 0) 및 서비스 사용 메모리 변경

☞ 오라클 데이터베이스 서비스에 관련된 XML DB 기능이 사용 중인 8080 포트번호를 0 번으로 변경

☞ 오라클 데이터베이스 서비스 프로세스(oracle.exe)가 사용하는 메모리 크기를 줄임

→ 프로그램 검색란을 이용하여 명령 프롬프트를 실행한 후, 다음의 명령어를 실행합니다.



→ 실행된 명령 프롬프트에서 다음의 명령어(파란색 명령어)를 실행합니다.

```
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.
```

```
C:\Users\shshin>lsnrctl status LISTENER ← 오라클 리스너 서비스 상태 확인 명령어
```

```
LSNRCTL for 64-bit Windows: Version 11.2.0.2.0 - Production on 16-6월 -2021 11:40:28
```

```
Copyright (c) 1991, 2014, Oracle. All rights reserved.
```

```
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC1)))
```

```
STATUS of the LISTENER
```

```
-----  
Alias LISTENER  
Version TNSLSNR for 64-bit Windows: Version 11.2.0.2.0 - Production  
Start Date 16-6월 -2021 11:13:02  
Uptime 0 days 0 hr. 27 min. 25 sec  
Trace Level off  
Security ON: Local OS Authentication  
SNMP OFF  
Default Service XE  
Listener Parameter File C:\Oracle\product\11.2.0\server\network\listener.ora  
Listener Log File C:\Oracle\diag\tnslsnr\X230-SHSHIN\listener\alert\log.xml  
Listening Endpoints Summary...  
(DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(PIPENAME=\.pipe\EXTPROC1ipc)))  
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=X230-SHSHIN)(PORT=1521))) ← 오라클 데이터베이스 서비스에 접속할 수 있는 리스너 포트  
Services Summary...  
Service "CLRExtProc" has 1 instance(s).  
Instance "CLRExtProc", status UNKNOWN, has 1 handler(s) for this service...  
Service "PLSExtProc" has 1 instance(s).  
Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...  
Service "XEXDB" has 1 instance(s).  
Instance "xe", status READY, has 1 handler(s) for this service...  
Service "xe" has 1 instance(s).  
Instance "xe", status READY, has 1 handler(s) for this service... ← 데이터베이스 서비스 항목  
The command completed successfully
```

```
C:\Users\shshin>
```

```
C:\Users\shshin>sqlplus / as sysdba ← SQL*Plus 툴을 이용하여 관리자인 SYS 계정으로 접속(AS SYSDBA 옵션 때문).
```

```
SQL*Plus: Release 11.2.0.2.0 Production on 수 6월 16 11:32:15 2021
```

```
Copyright (c) 1982, 2014, Oracle. All rights reserved.
```

```
Connected to:
```

```
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
```

```
SQL> -- 500MB가 과도하게 작을 경우, 설정 가능한 가장 작은 크기로 자동으로 변경되어 설정됨.
```

```
SQL> ALTER SYSTEM SET sga_max_size = 500M SCOPE=SPFILE ;
```

```
System altered.
```

```
SQL> ALTER USER sys IDENTIFIED BY oracle4U ; ← SYS 계정 암호 재설정
```

```
User altered.
```

```
SQL> ALTER USER system IDENTIFIED BY oracle4U ; ← SYSTEM 계정 암호 재설정
```

```
User altered.
```

```
SQL> ALTER USER hr IDENTIFIED BY oracle4U ACCOUNT unlock ; ← hr 계정 암호 재설정 및 LOCK 해제
```

```
User altered.
```

```
SQL> SELECT dbms_xdb.gethttpport FROM dual ; ← 현재 설정 포트 확인
```

```
GETHTTPPORT
```

-----  
8080

SQL> EXEC dbms\_xdb.sethttpport(0) ← 사용 포트를 0 으로 설정.

PL/SQL procedure successfully completed.

SQL> SHUTDOWN IMMEDIATE ← 오라클 데이터베이스 서비스 종료

Database closed.

Database dismounted.

ORACLE instance shut down.

SQL>

SQL> STARTUP ← 오라클 데이터베이스 서비스 기동

ORACLE instance started.

Total System Global Area 638889984 bytes ← 1GB 에서 600MB 정도로 줄어들었습니다

Fixed Size 2255832 bytes

Variable Size 184550440 bytes

Database Buffers 448790528 bytes

Redo Buffers 3293184 bytes

Database mounted.

Database opened.

SQL>

SQL> SELECT dbms\_xdb.gethttpport FROM dual ; ← 변경 설정 확인: 0 으로 변경됨

GETHTTPPORT

-----  
0

SQL> exit ← SQL\*Plus 접속 종료

Disconnected from Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production

C:\Users\shshin>

## [DE-09] 오라클 SQL\*Developer 설치

- 다음의 작업들이 진행됩니다.
  - 오라클 SQL\*Developer 다운로드 및 설치
  - 오라클 데이터베이스 서비스로의 SYS 계정과 SYSTEM 계정으로의 접속 구성

☞ 오라클 SQL\*Developer는 구성된 오라클 데이터베이스 서비스에 접속하여, SQL문으로 필요한 데이터 처리를 요청하고, 그 결과를 받아서 사용자에게 표시해주는 클라이언트 개발 도구입니다. 데이터는 데이터베이스 서비스에 구성된 데이터베이스에 저장되어 있으므로, 데이터에 대한 실질적인 모든 처리는 오라클 데이터베이스 서비스의 프로세스인 oracle.exe에 의해서 수행됩니다. 즉, SQL\*Developer는 필요한 처리를 오라클 데이터베이스 서비스 프로세스에 요청하고, 오라클 데이터베이스 서비스의 프로세스인 oracle.exe가 요청을 처리한 후, 처리 결과를 SQL\*Developer에게 전달합니다. SQL\*Developer는 전달된 처리 결과를 표시하여 사용자에게 전달합니다. SQL문의 처리 결과로 데이터가 전달될 수도 있고, 처리 상태를 나타내는 메시지가 전달될 수도 있습니다.

### 1. 오라클 SQL\*Developer 다운로드(JDK 포함되지 않은 버전 다운로드)

→ 웹 브라우저에서 <https://www.oracle.com/tools/downloads/sqldev-downloads.html> 사이트에 접속합니다.

SQL Developer 20.4 Downloads  
Version 20.4.0.379.2205 - January 25, 2021

Platform	Download	Notes
Windows 64-bit with JDK 8 included	<a href="#">Download (422 MB)</a>	<ul style="list-style-type: none"><li>MD5: b302cc5eabe6796d11c4c52faf3f748d</li><li>SHA1: 23b49f634ca003ac216d69a33ce629f53f36c18f</li><li><a href="#">Installation Notes</a></li></ul>
Windows 32-bit/64-bit	<a href="#">Download (432 MB)</a>	<ul style="list-style-type: none"><li>MD5: 97129c75de00a35fd4a110b4485a124</li><li>SHA1: a1b45b0a2b20f43f93e250a302c7069ae0667cb9</li><li><a href="#">Installation Notes</a></li><li><b>JDK 8 or 11 required</b></li></ul>

→ 이 버전은 사용 중에 "응답없음"이 자주 발생되어 사용을 권장하지 않습니다.

→ <https://www.oracle.com/tools/downloads/sqldev-downloads-202.html> 사이트로 접속합니다.

SQL Developer 20.2 Downloads  
Version 20.2.0.175.1842 - June 25, 2020

Platform	Download	Notes
Windows 64-bit with JDK 8 included	<a href="#">Download (494 MB)</a>	<ul style="list-style-type: none"><li>MD5: c445a894c09f3b2c039e0fb64b4d41e9</li><li>SHA1: c1ea4ce7c25e6718fa9ee50df970c7d2eacd6bd8</li><li><a href="#">Installation Notes</a></li></ul>
Windows 32-bit/64-bit	<a href="#">Download (413 MB)</a>	<ul style="list-style-type: none"><li>MD5: 457dd75e626695b3262f38ea4f65c350</li><li>SHA1: 79810d37bf8c442f9753215236abffef5a8ab0</li><li><a href="#">Installation Notes</a></li><li><b>JDK 8 or 11 required</b></li></ul>

☞ Windows 64-bit with JDK 8 included는 소프트웨어에 JDK 8이 내장된 버전입니다.

☞ Windows 32-bit/64-bit는 JDK 8이 내장되지 않은 버전으로 SQL\*Developer가 사용되는 PC에 이미 설치되어 있는

JDK 를 이용하여 실행되는 버전입니다. 오라클 사는 설치 가이드를 통해 SQL\*Developer 사용 시에 JDK 8 또는 JDK 11을 지원한다고 명시하고 있습니다.

☞ 문서에서는 Windows 32-bit/64-bit 플랫폼용의 SQL\*Developer를 다운로드 하여 설치합니다.

→ Windows 32-bit/64-bit 플랫폼의 Download 링크를 클릭 → 로그인 수행 → 다운로드

## 2. 오라클 SQL\*Developer 설치(압축해제) 및 실행

→ 다운로드 된 sqldeveloper-20.2.0.175.1842-no-jre.zip 파일을 C:\myJavaDev 폴더에 복사한 후,  
'여기에 풀기' 옵션으로 압축을 해제합니다.  
→ sqldeveloper 폴더가 생성되고, 폴더 안에 압축이 해제됩니다. → 설치 완료(별도의 설치과정이 없습니다).

→ 압축해제 된 sqldeveloper 폴더로 이동 → sqldeveloper.exe 실행파일의 바로가기를 바탕화면에 생성

→ 바탕화면에 생성된 sqldeveloper.exe 파일에 대한 바로가기의 이름을 SQLDeveloper20.2로 변경합니다.

→ 바탕화면에 생성된 SQLDeveloper20.2 바로가기를 더블클릭하여 SQL\*Developer를 실행합니다.



→ 확인 클릭

→ 오류가 발생됨. msvcr100.dll이 지정된 경로에 없기 때문에 발생된 오류입니다.

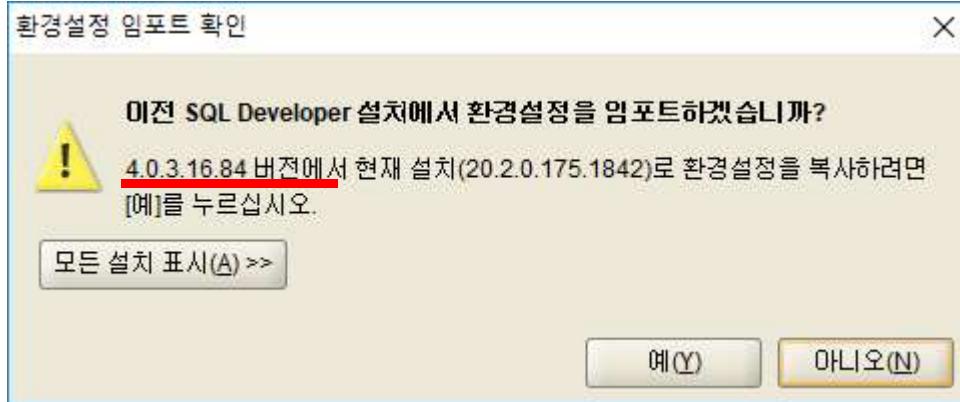
### [오류 해결 방법]

- 파일탐색기에서 JDK 8 이 설치된 폴더에 있는 jre\bin 폴더(문서에서는 C:\myJavaDev\jdk1.8.0\_291\jre\bin)로 이동
- 폴더에 있는 msdp140.dll 파일을 찾아서 복사 → 동일 폴더에 붙여넣기
- 생성된 msdp140 - 복사본.dll 파일의 이름을 msvcr100.dll 으로 변경

☞ 조치 완료

→ 다시 SQL\*Developer 바로가기를 실행합니다.

→ 환경 설정 임포트 확인 창이 표시됩니다.



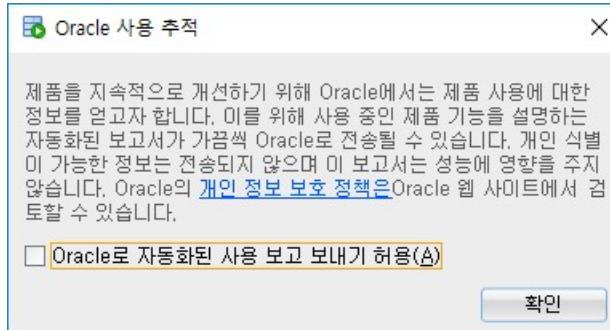
☞ PC에 사용했던 이전 버전의 SQL\*Developer가 있는 경우, 위의 메시지가 포함된 환경 설정 임포트 확인 창이 표시됩니다.

문서에서는 신규로 구성하는 것을 설명하므로 "아니오"를 클릭합니다.

이전에 사용한 버전이 없는 경우에도, "아니오"를 클릭합니다.

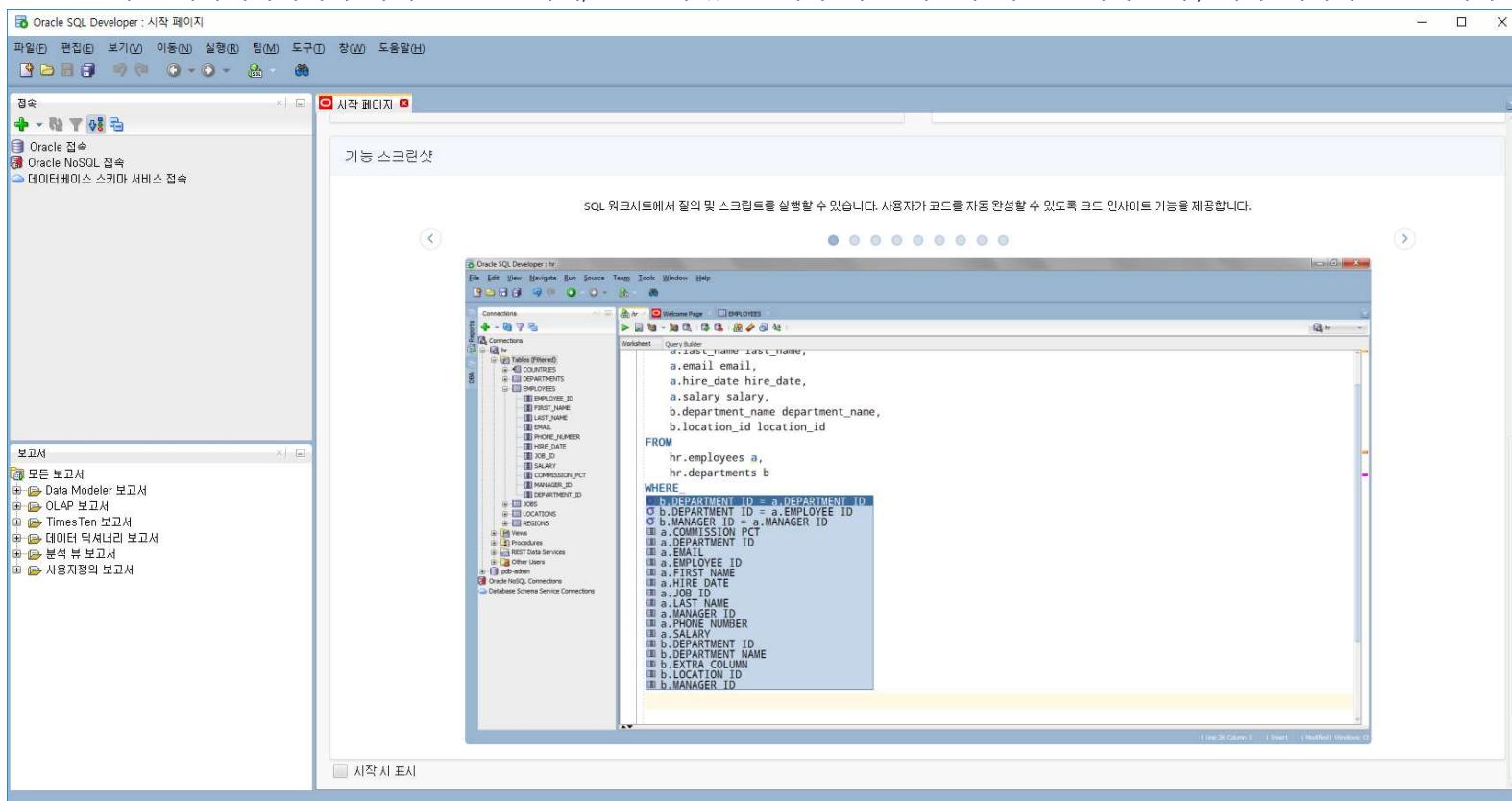
→ 실행 과정이 완료되고, 아래의 Oracle 사용 추적 메시지 박스 표시된 SQL\*Developer가 표시됩니다.

→ 표시된 Oracle 사용 추적 메시지 박스의 체크를 없애고 "확인"을 클릭합니다.



→ 다음은 실행 직 후 SQL\*Developer 모습입니다.

→ 표시된 시작페이지에서 아래로 스크롤하여, 맨 밑에 있는 "시작 시 표시"의 체크를 해제한 후, 시작 페이지를 닫습니다.



### 3. 오라클 데이터베이스 관리자 계정인 SYS 계정과 SYSTEM 계정의 xe 오라클 데이터베이스 서비스 접속 구성

☞ 오라클 데이터베이스 서비스에는 데이터베이스 관리자 계정으로 SYS 계정과 SYSTEM 계정이 기본적으로 생성됩니다.

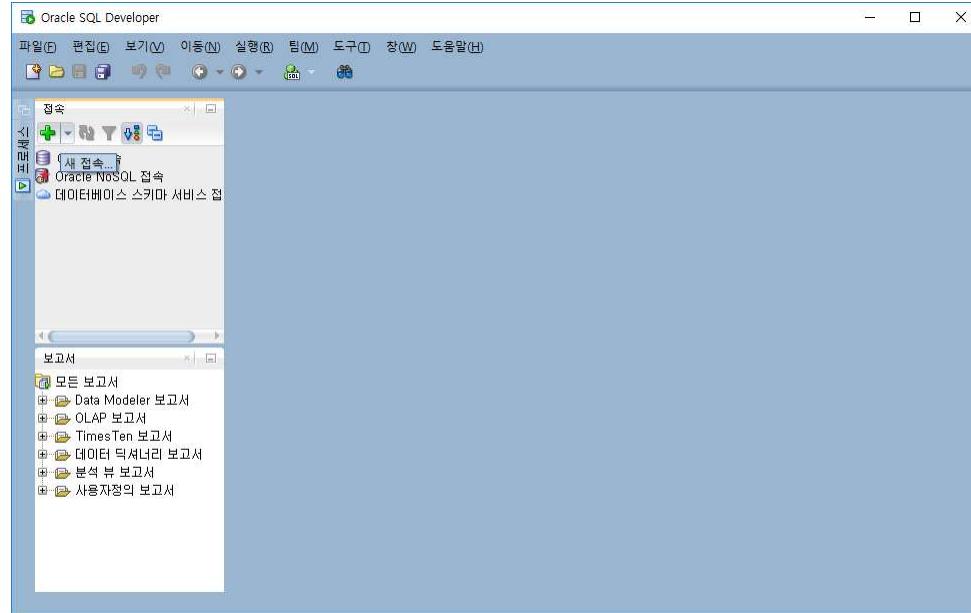
SYS 계정은 데이터베이스 서비스 중지 및 기동을 할 수 있는 권한까지 부여된 오라클 데이터베이스 서비스의 최상위 계정이고, SYSTEM 계정은 기동된 데이터베이스에 접속하여 관리 시에 사용되는 DBA 계정입니다.

주의할 것은, **프로그램 제작 시에, 이 두 계정을 데이터베이스 접속 계정으로 사용하면 않된다는 것입니다. 즉, 프로그램 코드나 설정에 이 두 계정을 사용해서는 않됩니다.**

☞ SYS 계정과 SYSTEM 계정은 새로운 데이터베이스 접속 계정을 생성하고, 생성된 계정에게 데이터베이스 서비스에 대한 액세스부터 데이터베이스에 저장된 데이터에 대한 권한을 부여하는 등의 계정 관리부터, 저장공간 생성과 관리 같은 데이터베이스 서비스 관리 작업 시에 데이터베이스 관리자(DBA)에 의해서 사용됩니다.

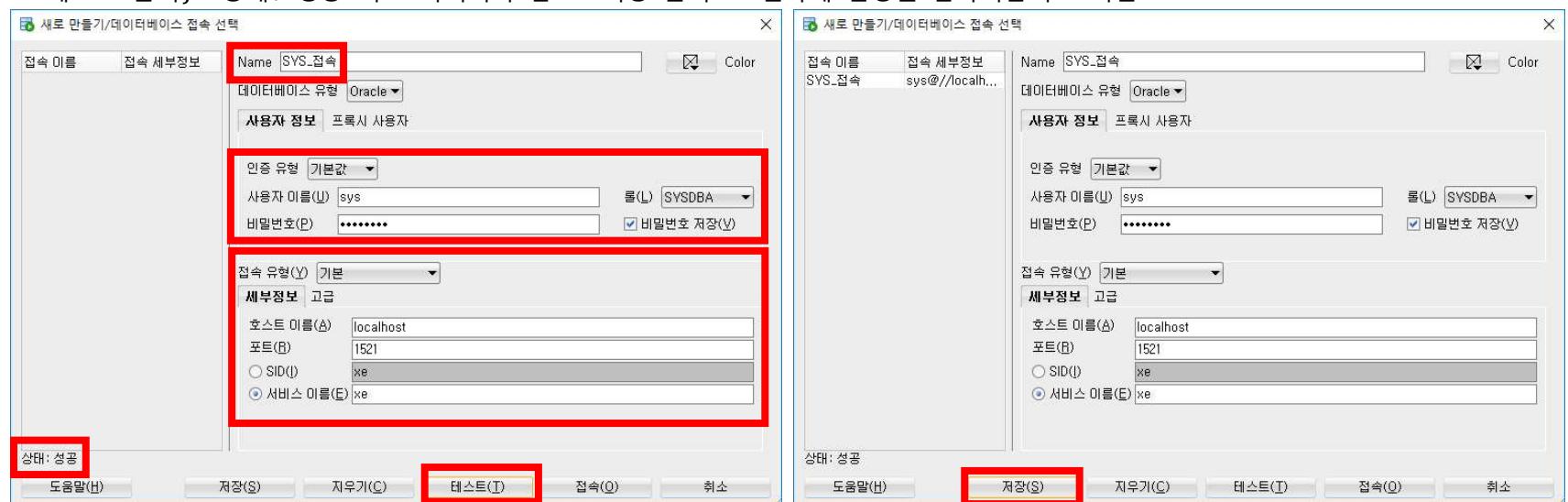
☞ 앞으로 프로젝트 수행 시에 필요한 데이터베이스 계정을 생성하고 권한을 설정하는 작업을 수행해야 할 때 접속해야 하므로, 이 두 계정에 대한 데이터베이스 접속을 SQL\*Developer 에 생성해 놓습니다.

→ 다음은 시작페이지를 닫은 SQL\*Developer의 모습입니다. → 접속창에 있는 [+] 아이콘을 클릭 → 새접속 생성 창이 표시됩니다.



→ 아래 그림처럼 입력(SYS 계정에 대한 접속 생성)

→ 테스트 클릭, "상태: 성공"이 표시되어야 함 → 저장 클릭 → 왼쪽에 설정된 접속이름이 표시됨



→ 입력된 내용을 다음처럼 수정 (SYSTEM 계정에 대한 접속 생성)

→ 테스트 클릭, "상태: 성공"이 표시되어야 함 → 저장 클릭 → 왼쪽에 설정된 접속이름이 표시됨 → 완료 후, 접속 클릭



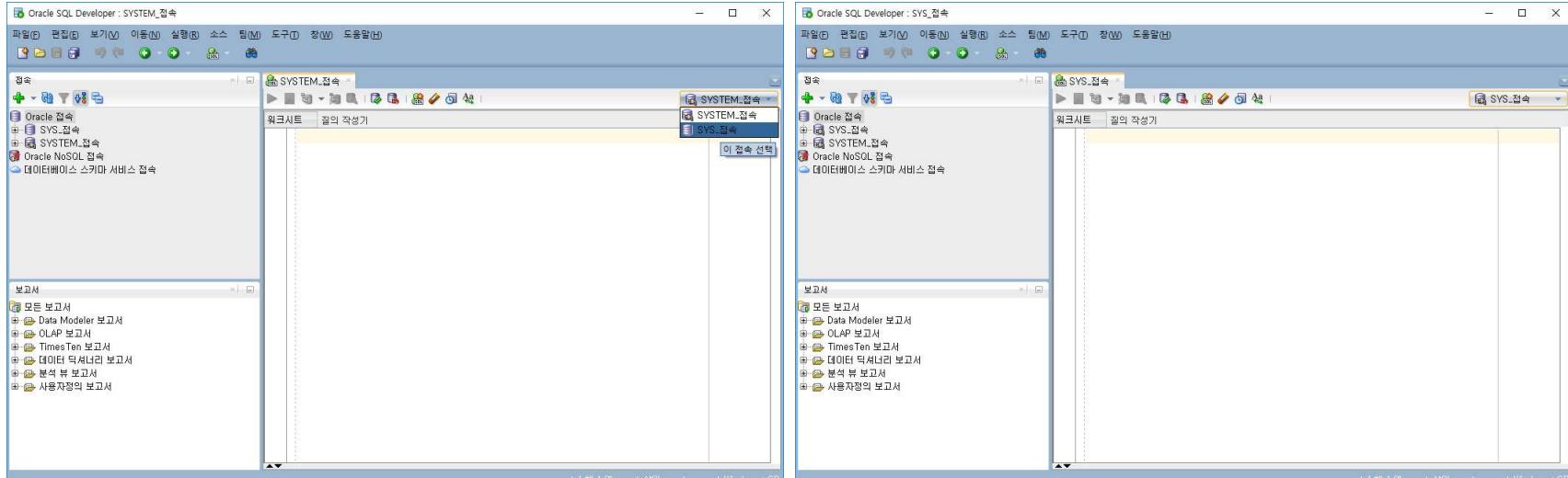
☞ Name, 사용자 이름, 룰(기본값)만 다르고 나머진 sys 계정과 동일합니다.

☞ 다음은 문서 실습에서 sys 계정과 SYSTEM 계정에 대하여 각 항목에 설정된 설정값 및 설정값에 대한 설명을 간단히 표시한 내용입니다.

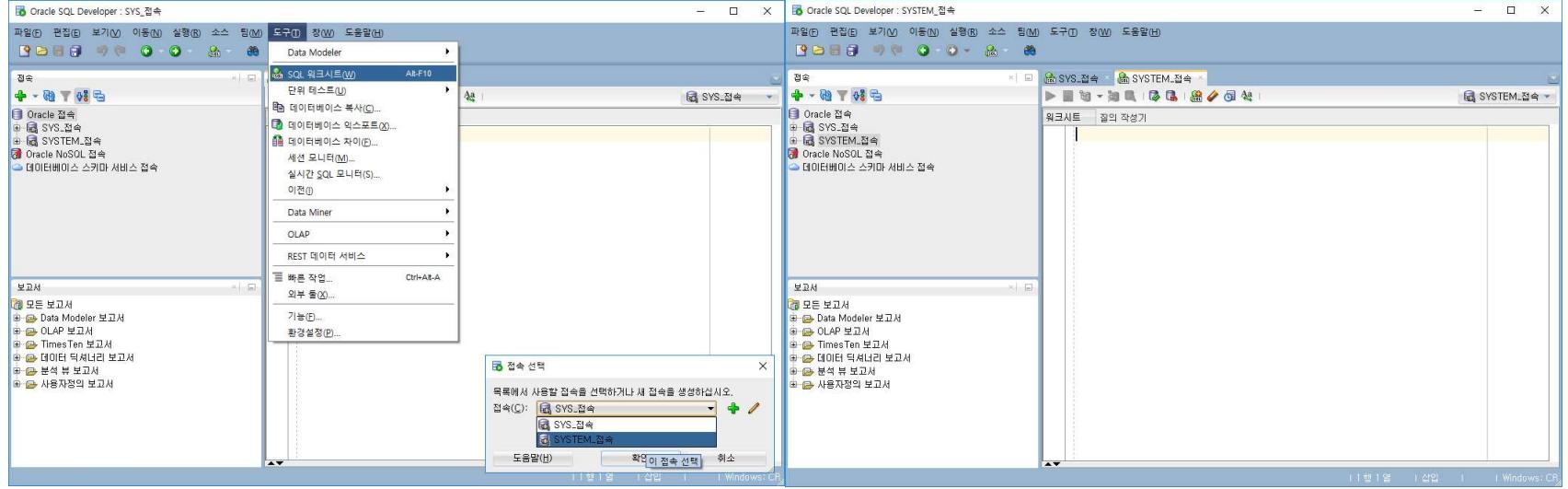
항목이름	설정하는 값	설명
Name	SYS_접속 SYSTEM_접속	SQL*Developer의 접속 창에 표시되는 이름을 설정.
인증유형	기본값	기본값으로 설정하면, 비밀번호에 설정된 값을 데이터베이스에 저장된 계정의 암호와 비교하여 인증
사용자 이름	sys system	오라클 데이터베이스 서비스에 접속하는 데이터베이스 계정 이름을 설정.
비밀번호	oracle4U	접속하는 데이터베이스 사용자 계정의 패스워드를 설정
룰	sys: SYSDBA system: 기본값	사용자 이름 항목에 설정한 데이터베이스 계정에게 SYSDBA 권한이 부여된 경우에만 SYSDBA로 설정함 별도로 추가하지 않은 경우, SYSDBA 권한은 sys 계정에게만 부여되어 있음.
비밀번호 저장	체크 함	SQL*Developer에서 계정의 암호를 저장해 놓을지를 설정. 접속 시에 패스워드 입력이 필요없음.
접속 유형	[기본]	기본으로 설정하면, 오라클 리스너를 통한 원격접속 방식으로 데이터베이스 서비스에 접속함
호스트이름	localhost	접속하는 오라클 데이터베이스 서버의 IP 주소 또는 localhost 를 설정.
포트	1521	오라클 리스너 서비스에 대한 접속 포트를 설정.
서비스이름	xe	접속하려는 오라클 데이터베이스의 서비스 이름을 설정.

→ SQL\*Developer 에 접속 생성 완료 → 오른쪽에 있는 접속 선택을 클릭 → SYS 접속 선택

→ SYS 접속으로 변경됨 → 이 때 이 전의 SYSTEM 접속은 해당 작업창에서는 사용할 수 없습니다.



→ SQL\*Developer 의 도구 메뉴 클릭 → SQL 워크 시트 클릭 → 접속 선택창에서 SYSTEM\_접속 선택 후, 확인



→ 2개의 워크시트에서 각각 접속하여 작업할 수 있습니다.

# [프로젝트 준비]

## [01] Spring Legacy Project 생성 및 기본구성: Eclipse 또는 STS에서 수행

- 다음의 작업들이 진행됩니다.
  - Spring Legacy Project(Spring MVC Project Template 적용) 생성
  - 생성된 프로젝트 기본 구성 변경(Spring 5 기준)
  - 실습을 위해 필요한 추가 라이브러리 구성
  - 스프링 UTF-8 인코딩 필터(한글처리)

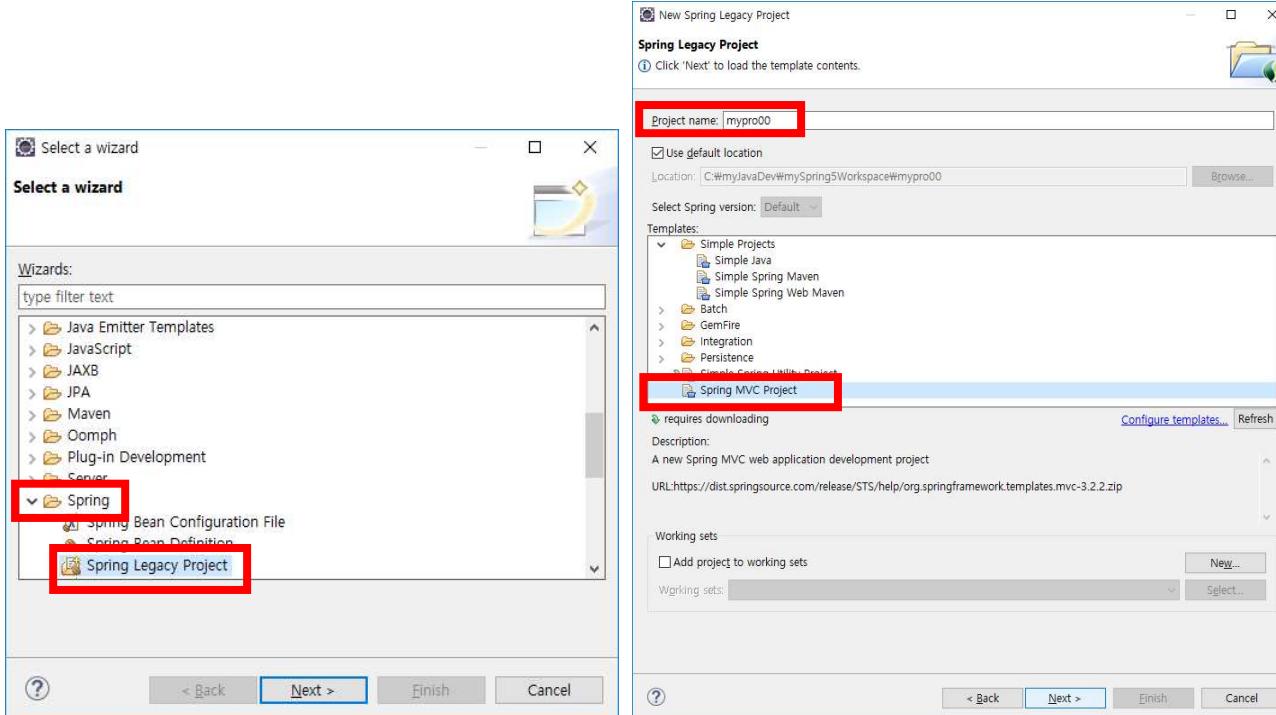
☞ 이클립스에서 스프링 프레임워크 기반 앱 애플리케이션을 제작하기 위한 프로젝트를 생성하고, 생성된 프로젝트를 스프링 프레임워크 5 기반(Servlet 4)으로 사용할 수 있도록 구성하는 방법을 설명합니다.

### 1-1. mypro00 프로젝트 생성(Spring Legacy Project 이용, Spring MVC Projec 템플릿 적용)

→ 이클립스 상단의 File 메뉴 → New → Other 클릭 → Spring 폴더의 Spring Legacy Project 클릭 → Next 클릭(왼쪽 그림)

→ Project Name에, mypro00 프로젝트 이름 입력(프로젝트 이름은 소문자로 자신의 원하는 이름을 하나의 영어 단어로 지정)

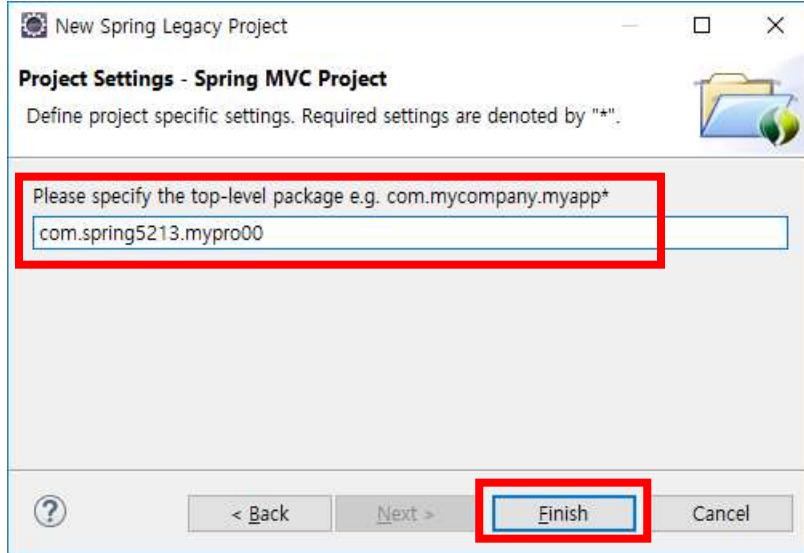
→ Template에서 Spring MVC Project 선택 후, → Next 클릭



→ 패키지 지정, 입력란에 com.spring5213.mypro00 입력

- ☞ 패키지 이름에 정해진 형식은 없지만, 웹 애플리케이션을 사용하는 회사 도메인 이름의 역순으로 된 이름(com.회사명.프로젝트이름 또는 앱명)을 소문자로 지정.

→ Finish 클릭



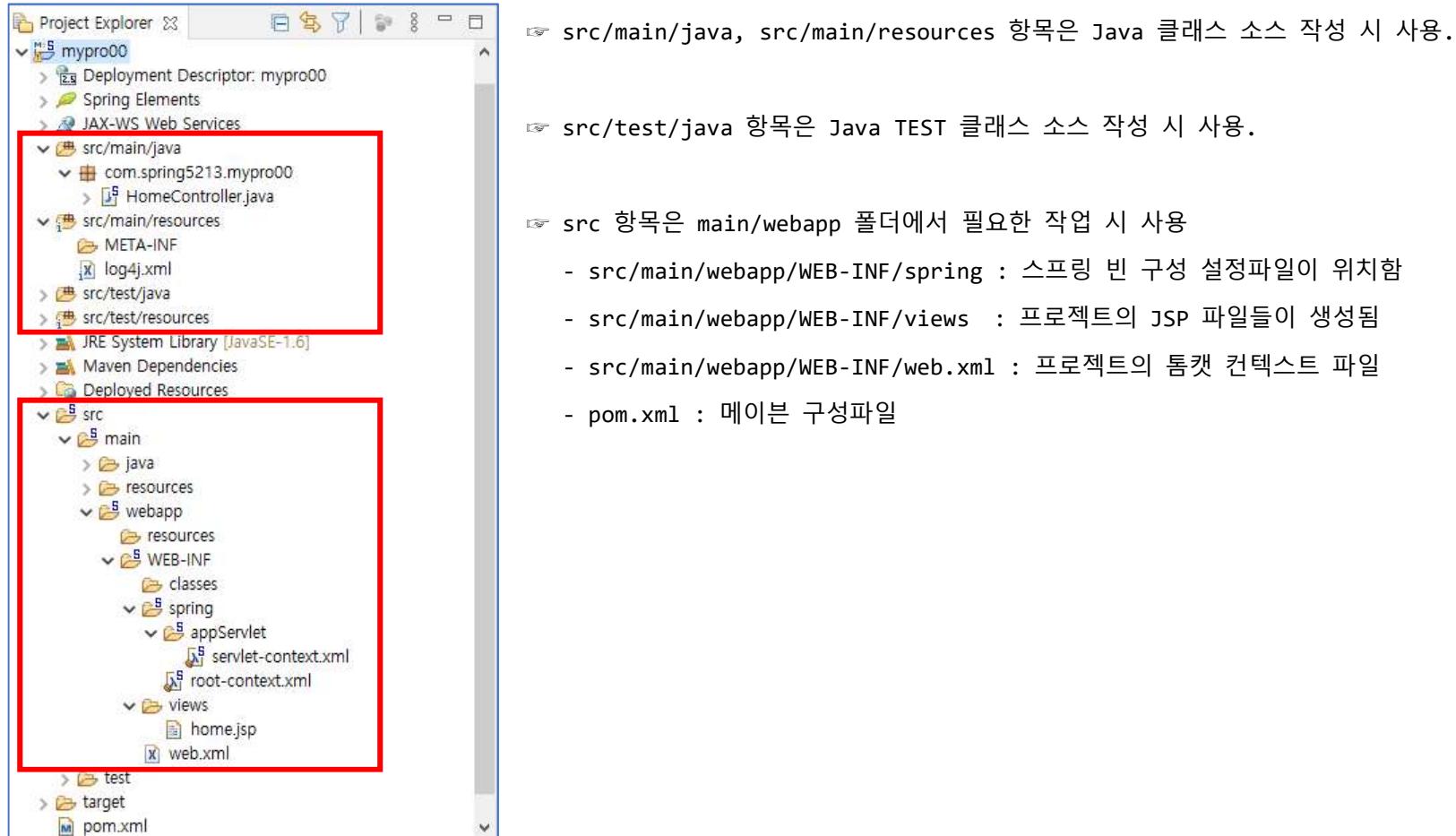
→ 프로젝트 생성이 진행됨

- ☞ 이클립스 오른쪽 하단에 프로젝트 생성 및 구성이 진행되는 녹색 바가 표시됩니다.

→ 녹색바가 사라질 때가 기다립니다.

▶ 프로젝트 생성 완료

☞ 다음은 생성된 프로젝트의 프로젝트 탐색기에서의 모습입니다.



## 1-2. 생성된 mypro00 프로젝트에 대한 기본 구성

### (1) 생성된 mypro00 프로젝트의 속성(Properties) 수정

→ 프로젝트 이름을 마우스 오른쪽 버튼으로 클릭 → 표시된 목록의 제일 밑에 있는 Properties 클릭

☞ 프로젝트 속성 창(Properties) 표시됨

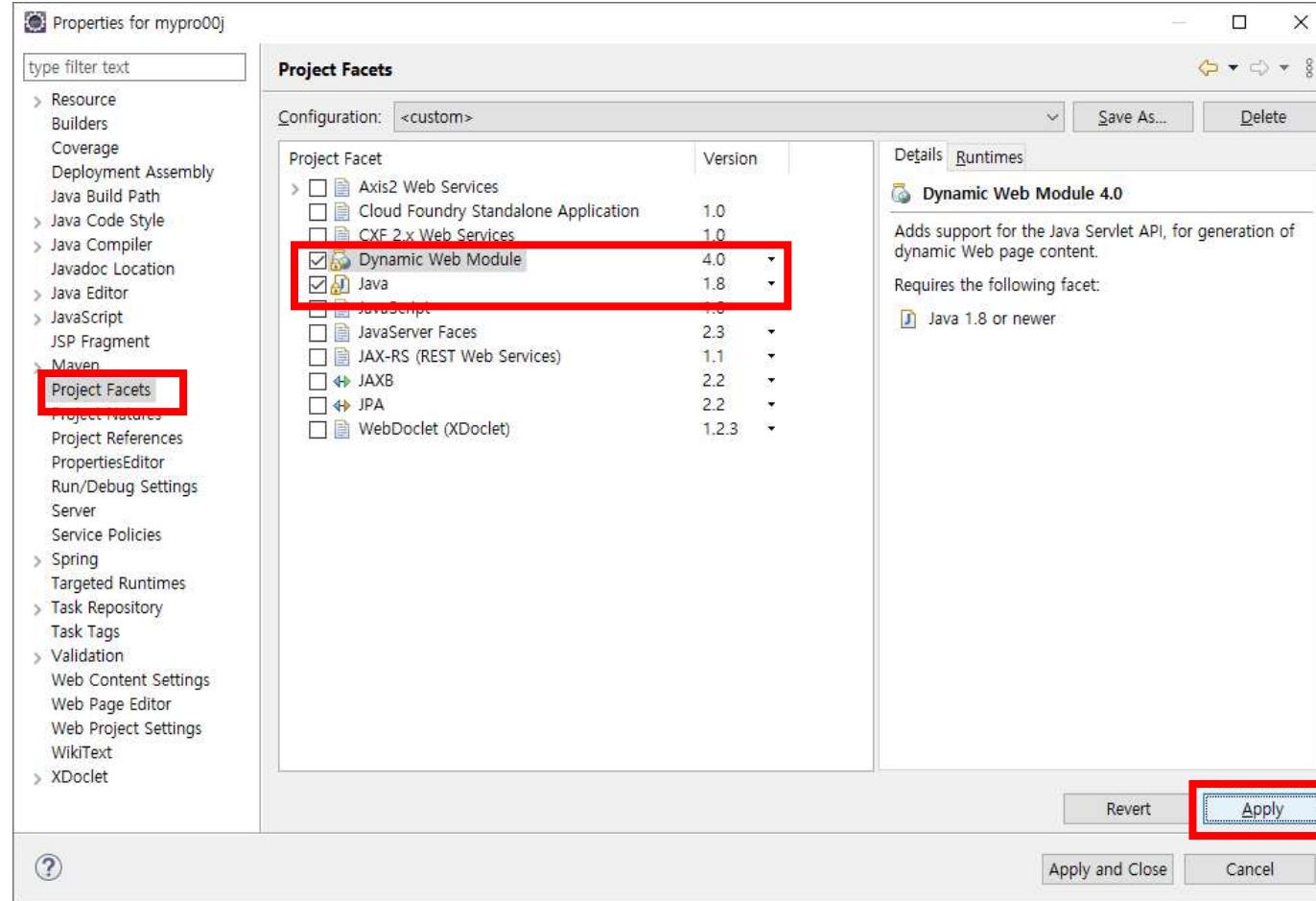
→ 왼쪽에서 Project Facets 항목 선택

→ 오른쪽에서 다음을 수행

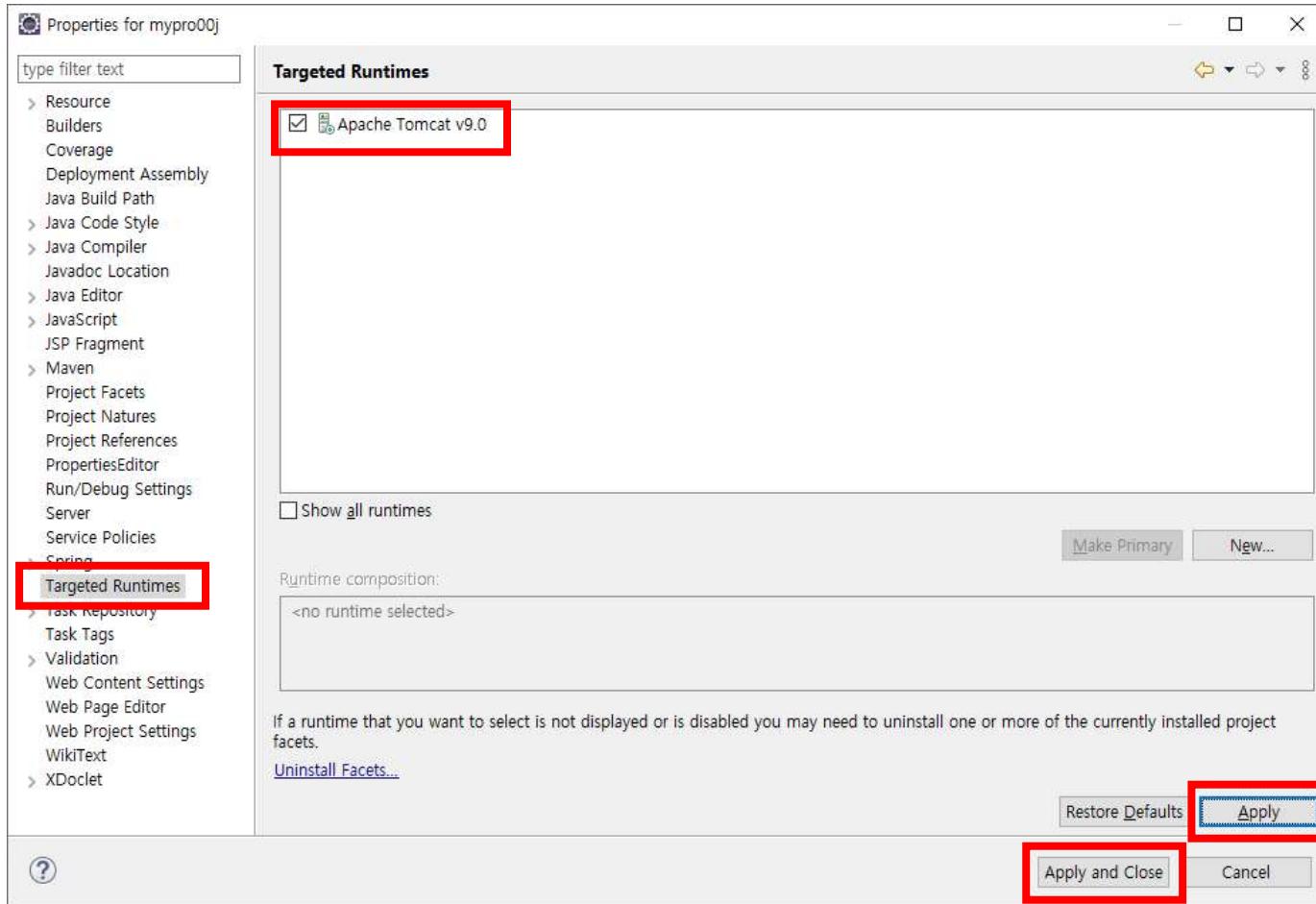
→ Java 항목의 값을 1.8로 변경

→ Dynamic Web Module 항목의 값을 4.0으로 변경

→ Apply 클릭



- 왼쪽에서 Targeted Runtimes 선택
- 오른쪽에서 Apache Tomcat v9.0 앞에 체크 ➔ 프로젝트 실행 시에 사용할 톰캣 서버가 지정됩니다.
- Apply 버튼 클릭
- Apply and Close 버튼 클릭 ➔ 프로젝트 속성 창(Properties) 종료



## (2) pom.xml 파일 수정

- ☞ <https://mvnrepository.com/> 사이트에서 파일에 표시된 라이브러리의 버전을 확인해서 수정.
- ☞ 일반적으로 <https://mvnrepository.com/> 사이트에서 원하는 라이브러리를 검색하여 결과에 표시된 버전 중에 "사용자 수가 제일 많은 버전"을 사용하거나, 신규 개발 환경일 경우에는, 최신 버전이나 사용자 요구에 해당하는 버전으로 설정합니다.
- ☞ 개발팀에서 개발환경이 이미 구성된 경우, 환경 설정을 위하여 사용되는 pom.xml 파일을 제공하기도 합니다.

[참고] 다음은 문서 실습에 사용된 모든 라이브러리들이 포함된 프로젝트의 pom.xml 파일의 내용입니다.

☞ 스프링 프레임워크와 스프링 시큐리티는 RELEASE 버전 중 최신 버전으로 설치, 주석처리 되어있음.

☞ 스프링 프레임워크 버전 5.2.13 기준, 라이브러리들은 2021.3.16 기준 최신 버전의 라이브러리 사용

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.spring5213</groupId>
  <artifactId>mypro00</artifactId>
  <name>mypro00</name>
  <packaging>war</packaging>
  <version>1.0.0-BUILD-SNAPSHOT</version>
  <properties>
    <java-version>1.8</java-version>
    <org.springframework-version>5.2.13.RELEASE</org.springframework-version>
    <org.aspectj-version>1.9.6</org.aspectj-version><!--1.6.10-->1.9.6 -->
    <org.slf4j-version>1.7.30</org.slf4j-version><!-- 1.6.6 -->1.7.30 -->
```

```
</properties>
<dependencies>

<!-- 새로 추가: aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.6</version>
</dependency>

<!-- 새로 추가: spring-security -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>5.3.8.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>5.3.8.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>5.3.8.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>5.3.8.RELEASE</version>
</dependency>

<!-- 새로 추가: 이미지에 대한 썸네일 이미지 생성 라이브러리 thumbnailator -->
<dependency>
    <groupId>net.coobird</groupId>
    <artifactId>thumbnailator</artifactId>
    <version>0.4.14</version>
</dependency>

<!-- 새로 추가: commons-fileupload -->
<!-- <dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.4</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.8.0</version>
</dependency> -->

<!-- 새로 추가: jackson-databind (JSON 처리 라이브러리) -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
```

```
<groupId>org.json</groupId>
<artifactId>json</artifactId>
<version>20210307</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.6</version>
</dependency>

<!-- 새로 추가: log4jdbc-log4j2 -->
<!-- log4jdbc-log4j2-jdbc4 사용 시: Driver does not support get/set network timeout
        for connections. 경고 메시지 표시됨. -->
<!-- log4jdbc-log4j2-jdbc4.1 사용 시: 경고 메시지 표시 안 됨 -->
<dependency>
    <groupId>org.bgee.log4jdbc-log4j2</groupId>
    <artifactId>log4jdbc-log4j2-jdbc4.1</artifactId>
    <version>1.16</version>
</dependency>

<!-- 새로 추가: mybatis, mybatis-spring -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.7</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>2.0.6</version>
</dependency>

<!-- 새로 추가: HikariCP -->
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
    <version>4.0.3</version>
</dependency>

<!-- 새로 추가:ojdbc6: Oracle JDBC Driver compatible with JDB6 ~ 8 -->
<!-- 접속하려는 오라클 데이터베이스의 버전에 해당하는 드라이버를 사용하는 것을 권장 -->
<!--
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0.4</version>
</dependency> -->
<!-- 새로 추가: ojdbc8: Oracle JDBC Driver compatible with JDK8, JDK11 ~ 15 -->
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>21.1.0.0</version>
</dependency>

<!-- 새로 추가: spring-tx, spring-jdbc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>

<!-- 새로 추가: lombok -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.18</version>
</dependency>
```

```

<!-- 새로 추가: spring-test -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-test</artifactId>
<version>${org.springframework-version}</version>
</dependency>

<!-- Spring -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>${org.springframework-version}</version>
<exclusions>
<!-- Exclude Commons Logging in favor of SLF4j -->
<exclusion>
<groupId>commons-logging</groupId>
<artifactId>commons-logging</artifactId>
</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>${org.springframework-version}</version>
</dependency>

<!-- AspectJ -->
<dependency>
<groupId>org.aspectj</groupId>
<artifactId>aspectjrt</artifactId>
<version>${org.aspectj-version}</version>
</dependency>

<!-- Logging -->
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-api</artifactId>
<version>${org.slf4j-version}</version>
</dependency>
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>jcl-over-slf4j</artifactId>
<version>${org.slf4j-version}</version>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
<version>${org.slf4j-version}</version>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.17</version><!-- 1.2.15 ->1.2.17 -->
<exclusions>
<exclusion>
<groupId>javax.mail</groupId>
<artifactId>mail</artifactId>
</exclusion>
<exclusion>
<groupId>javax.jms</groupId>
<artifactId>jms</artifactId>
</exclusion>
<exclusion>
<groupId>com.sun.jdmk</groupId>
<artifactId>jmxtools</artifactId>
</exclusion>
<exclusion>
<groupId>com.sun.jmx</groupId>
<artifactId>jmxri</artifactId>
</exclusion>
</exclusions>
</dependency>

```

```

</exclusions><!--<scope>runtime</scope> -->
</dependency>

<!-- @Inject -->
<dependency>
    <groupId>javax.inject</groupId>
    <artifactId>javax.inject</artifactId>
    <version>1</version>
</dependency>

<!-- Servlet --><!--
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId> servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.2</version>
    <scope>provided</scope>
</dependency> -->

<!-- 위의 Servlet 교체: javax.servlet-api, javax.servlet.jsp-api -->
<!-- 버전 교체: javax.servlet-api, REST기능 사용 -->
<!-- 톰캣9내장 파일업로드 라이브러리 사용을 위해서 servlet 버전이 3.0 이상되어야 함 -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.3</version>
    <scope>provided</scope>
</dependency>
<!-- 교체 끝 -->

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>

<!-- Test -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version><!-- 4.7->4.13.2 -->
    <!-- <scope>test</scope> --><!-- 테스트 코드 작성 시 오류 방지 -->
</dependency>

<!-- Apache 파일업로드/다운로드 libary -->
<!--
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.4</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.7</version>
</dependency>
-->

<!-- Tiles Framework 관련 라이브러리 설치 않함 -->
<!--
<dependency>
    <groupId>org.apache.tiles</groupId>

```

```

<artifactId>tiles-api</artifactId>
<version>3.0.8</version>
</dependency>
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-core</artifactId>
    <version>3.0.8</version>
</dependency>
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-jsp</artifactId>
    <version>3.0.8</version>
</dependency>
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-servlet</artifactId>
    <version>3.0.8</version>
</dependency>
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-template</artifactId>
    <version>3.0.8</version>
</dependency>
-->

<!-- JSP Standard Tag Library (커스텀 JSTL 라이브러리) -->
<!-- 위의 javax.servlet의 jstl, 1.2 가 문제 있을 시 교체 -->
<!--
<dependency>
    <groupId>org.apache.taglibs</groupId>
    <artifactId>taglibs-standard-impl</artifactId>
    <version>1.2.5</version>
</dependency>
<dependency>
    <groupId>org.apache.taglibs</groupId>
    <artifactId>taglibs-standard-spec</artifactId>
    <version>1.2.5</version>
</dependency>
<dependency>
    <groupId>org.apache.taglibs</groupId>
    <artifactId>taglibs-standard-jstl</artifactId>
    <version>1.2.5</version>
</dependency>
<dependency>
    <groupId>org.apache.taglibs</groupId>
    <artifactId>taglibs-standard-compat</artifactId>
    <version>1.2.5</version>
</dependency>
-->

</dependencies>

<build>
    <plugins>
        <plugin>
            <artifactId>maven-eclipse-plugin</artifactId>
            <version>2.10</version><!-- 2.9 ->2.10 -->
            <configuration>
                <additionalProjectnatures>
                    <projectnature>
                        org.springframework.ide.eclipse.core.springnature
                    </projectnature>
                </additionalProjectnatures>
                <additionalBuildcommands>
                    <buildcommand>
                        org.springframework.ide.eclipse.core.springbuilder
                    </buildcommand>
                </additionalBuildcommands>
                <downloadSources>true</downloadSources>
                <downloadJavadocs>true</downloadJavadocs>
            </configuration>
        </plugin>
    </plugins>
</build>

```

```

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version><!-- 2.5.1 -> 3.8.1 -->
    <configuration>
        <source>1.8</source><!-- 1.6 -> 1.8 -->
        <target>1.8</target><!-- 1.6 -> 1.8 -->
        <compilerArgument>-Xlint:all</compilerArgument>
        <showWarnings>true</showWarnings>
        <showDeprecation>true</showDeprecation>
    </configuration>
</plugin>
<!-- 중요!!!!!! -->
<!-- XML 설정파일 대신 자바 구성 설정 클래스를 사용하는 경우 -->
<!-- 메이븐의 정상작동을 위해 아래의 주석부분의 플러그인을 추가합니다(주석제거.) -->
<!--
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>3.3.1</version>
    <configuration>
        <failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>
</plugin>
-->
<!-- XML 설정파일 대신 자바 구성 설정 클래스를 사용하는 경우 END-->
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>3.0.0</version><!-- 1.2.1 -> 3.0.0 -->
    <configuration>
        <mainClass>org.test.int1.Main</mainClass>
    </configuration>
</plugin>
</plugins>
</build>
</project>

```

→ 수정 및 저장 후, 이클립스에서 라이브러리 다운로드 및 구성이 끝날 때까지 기다립니다

- 이클립스 오른쪽 밑에 녹색 진행바가 진행상태를 표시합니다 (Progress 뷰가 있으면, 표시된 내용이 없을 때까지 기다림)
- 라이브러리 다운로드가 완료되면, 프로젝트이름 → 오른쪽 마우스 클릭 → Maven → Update Project 실행

#### ◆ 장애해결

```

<!-- STS3 plug-in 설치 후, 기존 SPRING 5 프로젝트의 pom.xml 파일의 메이븐 xsi:schemaLocation 오류 해결 Library -->
<!-- <dependency>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <version>3.1.0</version>
</dependency> -->

<!-- Maven war Plug in 오류 시, maven-compiler-plugin 오류 해결법
[증상] pom.xml 파일의 <packaging>war</packaging> 태그에 빨간줄이 가면서 아래와 같은
오류메시지가 발생될 때 해결방법
Execution default-testResources of goal org.apache.maven.plugins:
maven-resources-plugin:2.6:testResources failed:
.....
위와 비슷한 오류가 발생되면, 다음의 방법 중 하나를 수행합니다.
1) 프로젝트 우클릭 -> Run As -> Maven install 수행
2) 프로젝트 우클릭 -> Maven -> Project Update 수행
-->

```

### (3) 프로젝트의 web.xml 파일 수정

☞ <http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html#8> 사이트 참고

→ 프로젝트 탐색 창에서 src/main/webapp/WEB-INF 폴더에 있는 web.xml 파일을 열고, 다음의 빨간색 코드를 수정 및 추가합니다.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- 설정된 URL에서 제동되는 Dynamic Web 모듈에 대한 XML 스키마 정의 파일에 오류가 있어 4.0 버전에서는 사용 못함 -->
<!--
<web-app version="4.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  https://java.sun.com/xml/ns/javaee/web-app_4_0.xsd"> --><!-- 사용못함 -->
<!-- 위에서 version을 4.0 값으로 수정,
  xsi:schemaLocation 줄의 제일 마지막에 있는 숫자를 4_0 로 수정 -->

<!-- 4.0 일 경우: 오라클 사의 정의 정의 파일 사용: 2021.05 부터. -->
<web-app version="4.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee
  http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_4_0.xsd"> ← 기존 내용 대신 추가

<!-- 스프링 UTF-8 인코딩 필터(한글처리) --> ← 한글처리를 위한 UTF-8 인코딩 필터를 추가
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>
    org.springframework.web.filter.CharacterEncodingFilter
  </filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
<!--<servlet-name>appServlet</servlet-name> -->
</filter-mapping>

<!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>

<!-- Processes application requests -->
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>
```

#### (4) log4j 설정파일의 정의 구성 변경

→ src/main/resources 폴더와 src/test/resources 폴더에 있는 log4j.xml 파일들의 DTD 정의를 아래의 지시대로 변경.  
각 파일의 위에 있는 <!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd"> 줄에서,  
제일 끝에 있는 "log4j.dtd"를 → "http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/xml/doc-files/log4j.dtd"  
로 변경

## [02] 커넥션 풀, MyBatis 구성 및 오라클 데이터베이스와 접속 구성

- 다음의 작업들이 진행됩니다.
  - 오라클 데이터베이스 접속 계정 생성
  - 커넥션 풀(Connection Pool)을 이용한 오라클 데이터베이스 서비스와의 접속 구성
    - JDBC 접속을 위한 정보가 저장된 properties 파일 생성
  - 스프링 빈 구성 설정파일 생성: mybatis-context.xml 파일
  - Mybatis, mybatis-spring 사용 구성
  - log4jdbc-log4j2 라이브러리를 통한 SQL 처리 로그 표시 구성
  - 오라클 데이터베이스 접속 및 SQL문 처리 관련 클래스들의 빈 설정 구성
  - 구성사항 테스트: 테스트 클래스 작성

### 2-1. 오라클 데이터베이스 접속 계정 생성

☞ 프로젝트에서 사용할 book\_ex 계정을 생성합니다.

→ SQL\*Developer에서 SYS\_접속을 이용하여, 오라클 데이터베이스 서비스에 접속한 후, 다음의 SQL을 실행합니다.

```
--book_ex 접속 계정 생성(암호: book_ex)
CREATE USER book_ex          --접속 사용자 이름으로 book_ex
IDENTIFIED BY book_ex        --접속 암호로 book_ex 설정
DEFAULT TABLESPACE users     --기본 데이터 저장 공간으로 user 테이블스페이스 사용
TEMPORARY TABLESPACE temp    --메모리 부족 시, 임시저장 공간으로 temp 테이블스페이스 사용
QUOTA unlimited ON users;   --데이터 저장공간(users)에 대한 사용량 설정

--book_ex 계정에게 데이터베이스 권한 설정
GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE PROCEDURE,
       CREATE TRIGGER, CREATE VIEW, CREATE SYNONYM, ALTER SESSION
TO book_ex;
```

☞ RDBMS(Relational Database Management System)에서, 테이블을 통해 입력되는 사용자의 데이터는, 실제로 데이터파일에 저장됩니다.

RDBMS 제품마다 이 데이터파일을 구성하는 방식이 차이가 납니다.

오라클 데이터베이스에서는 데이터파일(들)을 "테이블스페이스(Tablespace)"라는 단위로 데이터베이스 관리자(DBA)가 직접 이름을 지정하여(예, users) 구성합니다. 접속 계정(예, book\_ex)이 자신에게 사용량이 할당된 테이블스페이스 공간의 이름을 테이블 생성 시에 지정하면, 오라클 데이터베이스 서비스가 지정된 테이블스페이스 공간에 구성된 데이터파일에 테이블의 데이터가 저장될 공간을 자동으로 구성해줍니다.

오라클과는 달리 MySQL(RDBMS)에서는 사용자가 테이블을 생성하면, 자동으로 데이터파일을 생성하여, 해당 테이블의 데이터만 저장되도록 구성됩니다. 즉 테이블 당 하나의 데이터파일을 사용합니다.

## 2-2. 커넥션 풀(Connection Pool)을 이용한 오라클 데이터베이스 서비스와의 접속 구성

- ☞ 오라클 JDBC 드라이버를 기반한 커넥션 풀을 이용하여 오라클 데이터베이스 서비스에 접속할 수 있도록 구성합니다.
- ☞ 커넥션 풀은 JDBC 를 이용한 데이터베이스 접속을 효율적으로 처리하기 위하여 사용되는 라이브러리입니다.  
프로그램 실행 중에 데이터베이스에 접속하는 것이 아니고, 톰캣 서버가 기동될 때 하나 이상의 커넥션 객체들의 그룹(Pool, 풀)이 하나의 JDBC 객체를 통해 데이터베이스와 연결됩니다. 따라서, 프로그램 실행 중에 데이터베이스에 대한 액세스가 필요할 경우, 톰캣 서버의 메모리에 구성된 다수의 커넥션들 중 하나를 사용하여 데이터베이스를 사용하게 됩니다.

(1) mypro00 프로젝트에 오라클 JDBC 드라이버 및 HikariCP 커넥션 풀 라이브러리 설치 (이미 설치되어 있습니다).

- ☞ 뒤에서 실습할 MyBatis, Mybatis-spring 라이브러리도 같이 설치합니다.

☞ 오라클 JDBC 드라이버의 경우, 사용되는 오라클 RDBMS 소프트웨어 버전에 맞는 것을 사용해야 합니다

JDK 8 개발환경에서 오라클 데이터베이스 11gR2 버전의 RDBMS를 사용하는 경우, ojdbc 6 또는 ojdbc 8 중 하나를 사용합니다. 문서에서는 ojdbc 8을 사용합니다. 단, 2개의 드라이버를 동시에 사용하면 안됩니다.

☞ 커넥션 풀(Connection Pool, 줄여서 CP)은, 많은 라이브러리들이 있으며, 문서에서는 HikariCP 커넥션 풀을 사용합니다.

추가로 문서에는 Mybatis에서 기본 제공하는 커넥션 풀을 사용하는 구성도 포함되어 있습니다.

→ 이를립스에서 pom.xml 파일을 열고, 필요한 라이브러리 정보를 추가합니다(이미 구성되어 있음).

```
<!-- 새로 추가: ojdbc: 접속하려는 오라클 데이터베이스의 버전에 해당하는 드라이버를 사용하는 것을 권장 -->
<!-- ojdbc6: Oracle JDBC Driver compatible with JDB6 ~ 8 -->
<!--
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0.4</version>
</dependency> --}}
<!-- ojdbc8: Oracle JDBC Driver compatible with JDK8, JDK11 ~ 15 -->
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>21.1.0.0</version>
</dependency>

<!-- 새로 추가: HikariCP -->
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
    <version>4.0.3</version>
</dependency>

<!-- 새로 추가: mybatis, mybatis-spring -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.7</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>2.0.6</version><!-- mybatis-spring은 1 버전이 아닌 2 버전을 사용함 -->
</dependency>
```

## (2) 스프링 빈 설정파일 생성

☞ 커넥션 풀을 사용하여 데이터베이스에 접속할 때 필요한 클래스들의 빈이 정의됩니다.

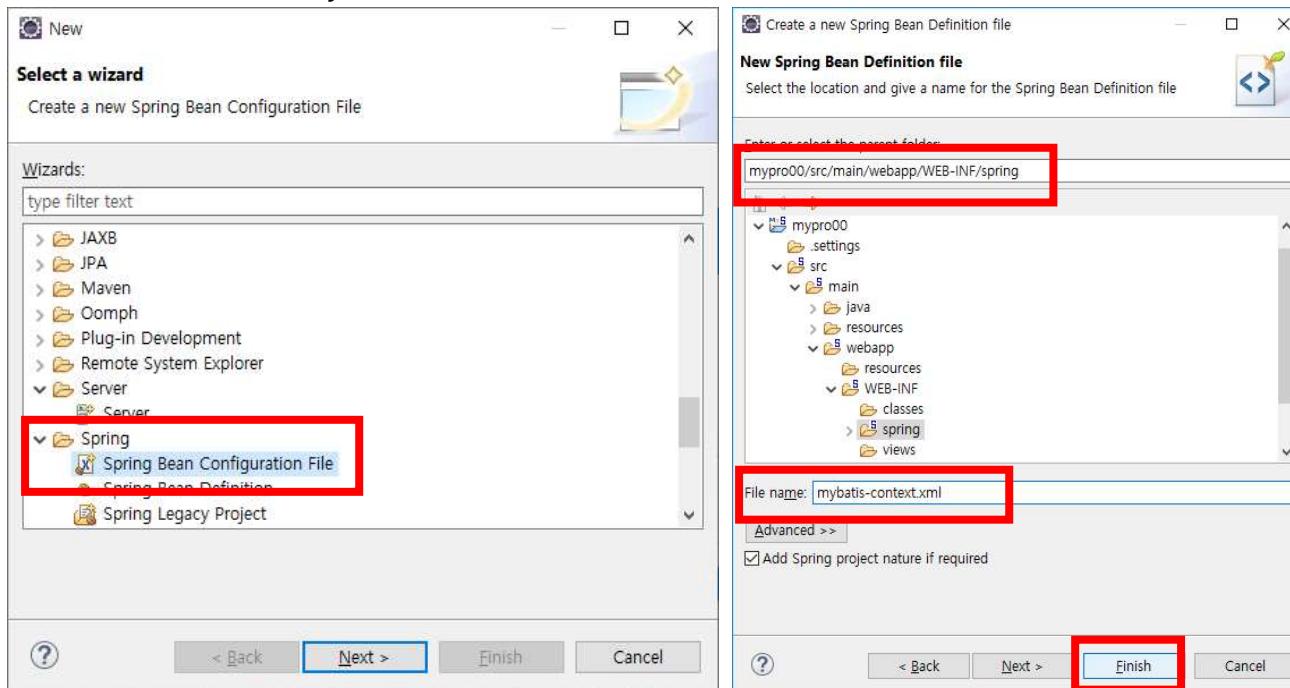
☞ 이 후에 Mybatis 구성 시 필요한 빈들도 등록할 것을 고려하여 별도의 스프링 빈 구성 파일을 생성하도록 합니다.

→ 프로젝트 탐색창에서 `src/main/webapp/WEB-INF/spring` 폴더에서 마우스 오른쪽 버튼 클릭 → New → Other 클릭

→ Spring 폴더의 Spring Bean Configuration File 선택 후, Next

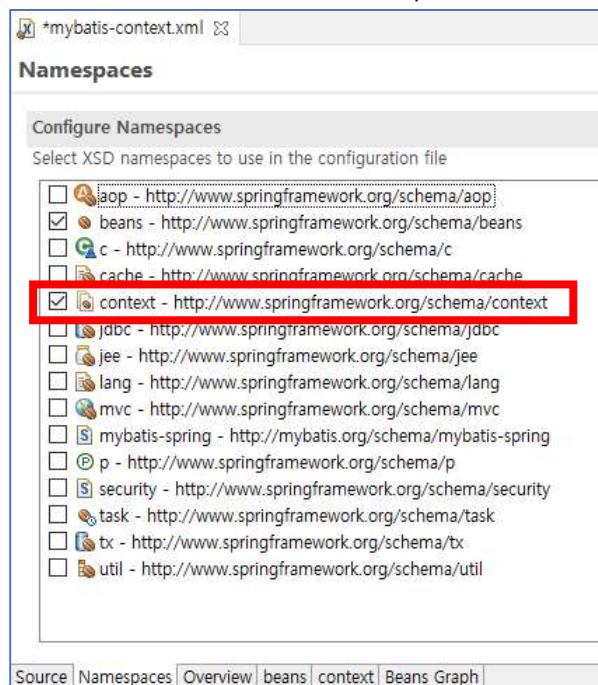
→ 위에 표시된 폴더 경로가 `mypro00/src/main/webapp/WEB-INF/spring` 인지 확인

→ File name 입력란에 `mybatis-context.xml` 입력 후 → Finish 클릭



→ 코드 작성부에 생성된 `mybatis-context.xml` 파일이 표시됩니다.

→ 코드 작성부 하단의 Namespaces 탭을 클릭 → context 네임스페이스를 체크 → 반드시 저장 → 저장 후, Source 탭 클릭



☞ 다음은 코드 작성부에 생성된 mybatis-context.xml 파일의 내용입니다.

```
<?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: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-4.3.xsd">

</beans>
```

(3) JDBC 드라이버와 HikariCP 커넥션 풀을 이용하여 오라클 데이터베이스 서비스로의 접속 정보 구성

☞ 데이터베이스 접속을 구성하는 방법은 2가지가 있습니다.

- 데이터베이스 접속에 필요한 JDBC 정보를 커넥션 풀 빈에 직접 설정하는 방법(단순 구성).
- 데이터베이스 접속에 필요한 JDBC 정보가 저장된 properties 파일을 이용하여 설정하는 방법

(3-1) 데이터베이스 접속에 필요한 JDBC 정보를 커넥션 풀 빈에 직접 설정하는 방법

→ 스프링 빈 구성 파일인 mybatis-context.xml 을 열고, 다음의 빨간색 부분을 오타없이 작성합니다(빨간색/녹색(주석) 코드 부분).

```
<?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: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-4.3.xsd">

<!-- Connection Pool 관련 Bean 구성: HikariCP --&gt;

<!-- 방법1: JDBC 정보를 직접 설정 --&gt;
<!-- 1. JDBC 설정 정보가 설정된 HikariConfig 빈 생성--&gt;
&lt;bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig"&gt;
    &lt;!-- 접속 시 사용되는 JDBC 클래스 전체 이름 --&gt;
    &lt;property name="driverClassName" value="oracle.jdbc.OracleDriver"&gt;&lt;/property&gt;
    &lt;!-- 접속하려는 오라클 서비스의 네트워크 정보 --&gt;
    &lt;property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe"&gt;&lt;/property&gt;
    &lt;property name="username" value="book_ex"&gt;&lt;/property&gt;&lt;!-- 접속 계정 --&gt;
    &lt;property name="password" value="book_ex"&gt;&lt;/property&gt;&lt;!-- 접속 계정의 암호 --&gt;
&lt;/bean&gt;

<!-- 2. HikariDataSource 클래스를 이용한 dataSource 빈 생성: JDBC 설정이 저장된 HikariConfig 빈 주입받음 --&gt;
&lt;bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close"&gt;
    &lt;constructor-arg ref="hikariConfig" /&gt;
&lt;/bean&gt;

&lt;/beans&gt;</pre>
```

→ 구성 내용 테스트: Junit을 이용해 커넥션 풀 구성 테스트

- ☞ 테스트를 위해 필요한 spring-test 라이브러리 설치 및 Junit 라이브러리의 버전 설정은  
앞 단원의 프로젝트 생성 및 사용구성 실습에서 수행되었습니다.

→ mypro00 프로젝트의 src/test/java/com.spring5213.mypro00 패키지에 datasource 패키지를 생성

→ 생성된 패키지에 DataSourceTests 클래스를 생성한 후, 다음의 코드를 작성.

```
package com.spring5213.mypro00.datasource;

import static org.junit.Assert.fail;

import java.sql.Connection;

import javax.sql.DataSource;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/mybatis-context.xml")
@Log4j
public class DataSourceTests {

    @Setter(onMethod_ = { @Autowired })
    private DataSource dataSource;

    @Test
    public void testConnection() {

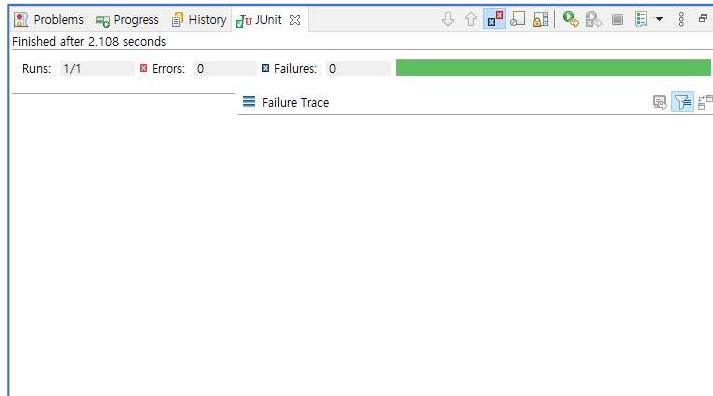
        try (Connection con = dataSource.getConnection()){
            log.info(con);

        } catch(Exception e) {
            fail(e.getMessage());
        }
    }
}
```

→ 테스트 수행: DataSourceTests.java 코드작성 뷰를 코드 작성 뷰에 표시하고, 마우스로 소스를 클릭한 후, 마우스 오른쪽 버튼 클릭

→ Run As → 2. JUnit Test 클릭

→ 테스트가 정상적으로 완료되면, Junit 뷰에 녹색진행표시가 표시되고 콘솔에 INFO 레벨의 기록이 표시됩니다.



[콘솔 표시 내용]

```
...(생략)...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
INFO : com.spring5213.mypro00.datasource.DataSourceTests - HikariProxyConnection@669284403 wrapping oracle.jdbc.driver.T4CConnection@14555e0a
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown completed.
```

### (3-2) properties 파일을 이용한 커넥션 풀의 데이터베이스 접속을 구성하는 방법

#### (3-2-1) properties 파일 생성

☞ properties 파일에는 데이터베이스 접속에 필요한 JDBC 정보가 저장됩니다.

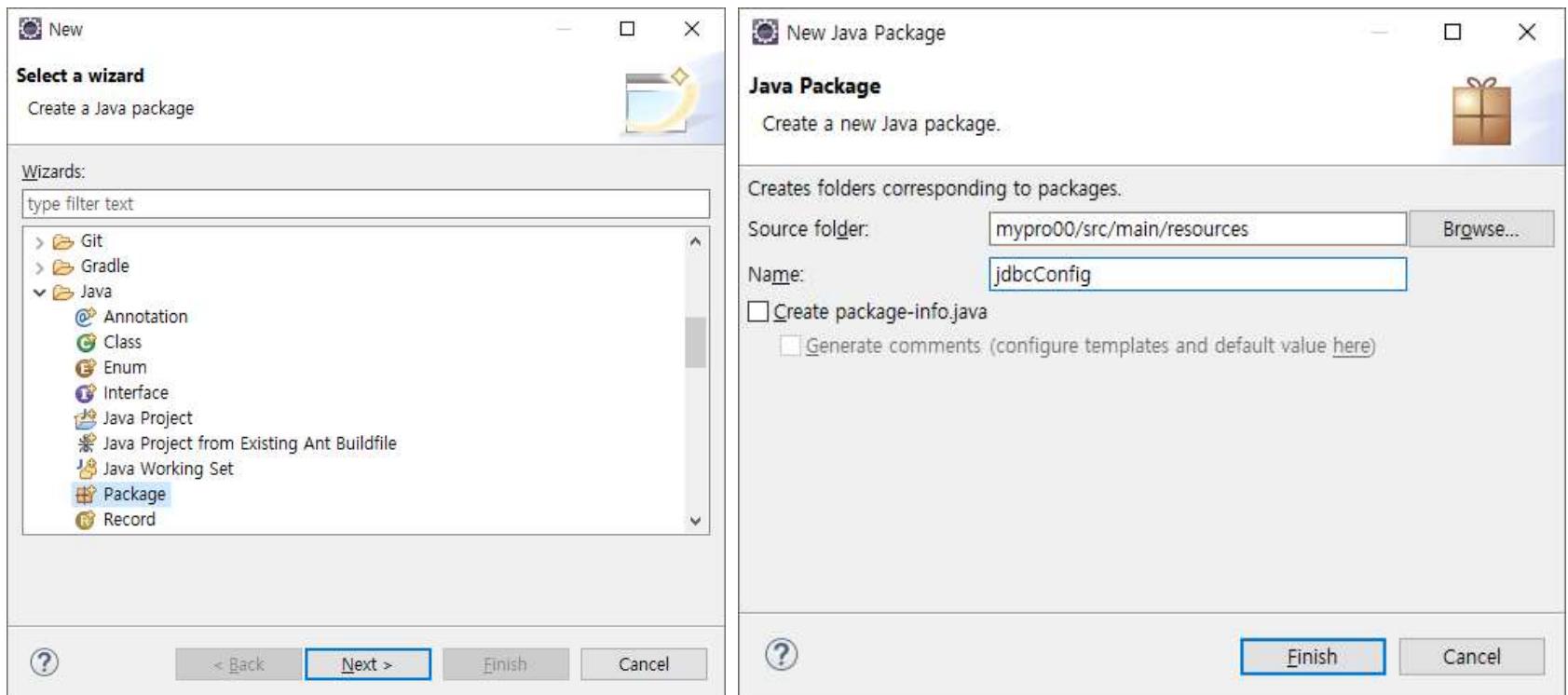
##### (3-2-1-1) jdbc.properties 파일이 저장될 jdbcConfig 패키지 생성.

→ src/main/resources 폴더에서 마우스 오른쪽 버튼 클릭 → New → Other 클릭

→ Java 폴더의 Package 선택 후, Next 클릭(왼쪽 그림)

→ 상단에 표시된 위치가 mypro00/src/main/resources 인지 확인(오른쪽 그림)

→ Name 입력란에 jdbcConfig 입력 → Finish 클릭 → 패키지 생성 완료



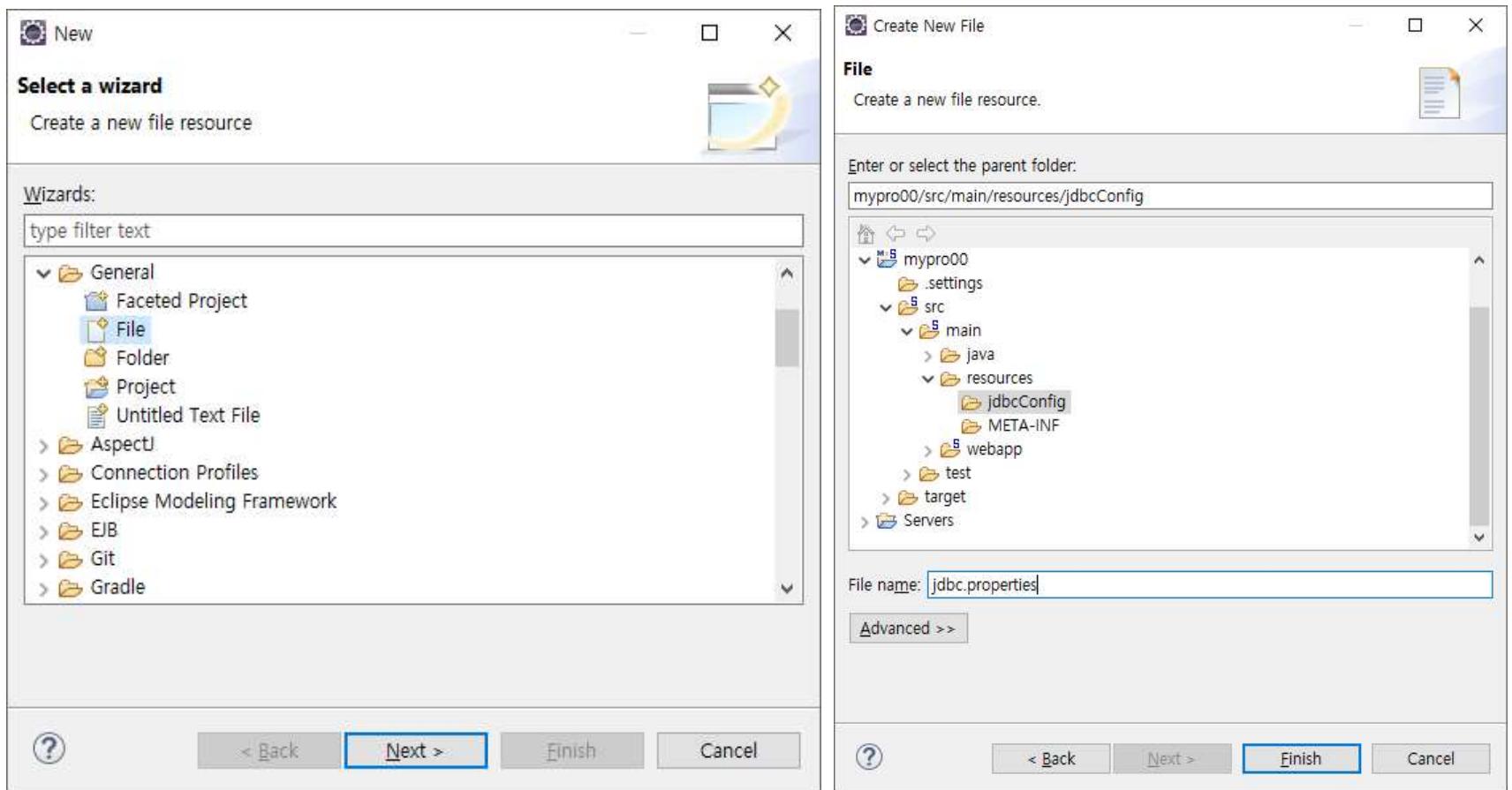
##### (3-2-1-2) 생성된 jdbcConfig 패키지에 jdbc.properties 파일 생성 및 내용 작성.

→ 생성된 jdbcConfig 패키지에서 마우스 오른쪽 버튼 클릭 → New → Other 클릭

→ General 폴더의 File 선택 후, Next 클릭(왼쪽 그림)

→ 위에 표시된 폴더 경로가 mypro00/src/main/resources/jdbcConfig 인지 확인

→ File name 입력란에 jdbc.properties 입력 → Finish 클릭



→ 코드 작성부에 열려진 `jdbc.properties` 파일에 다음의 내용을 작성한 후, 저장(오타 및 대소문자 주의 !!!).

```
#HikariCP: For Deploy
jdbcHikari.driverClassName=oracle.jdbc.OracleDriver          #<- 접속 시 사용되는 JDBC 클래스 전체 이름
jdbcHikari.jdbcUrl=jdbc:oracle:thin:@localhost:1521:xe      #<- 접속하려는 오라클 서비스의 네트워크 정보
jdbcHikari.username=book_ex                                    #<- 접속 계정 이름
jdbcHikari.password=book_ex                                    #<- 접속 계정 암호

#ibatisCP: For Deploy
jdbcIbatis.driverClassName=oracle.jdbc.driver.OracleDriver
jdbcIbatis.url=jdbc:oracle:thin:@localhost:1521:xe
jdbcIbatis.username=book_ex
jdbcIbatis.password=book_ex
```

☞ 작성된 내용에서 `jdbcHikari.` 접두어를 사용한 설정들은 Hikari 커넥션 풀을 이용하여 오라클 데이터베이스 서비스에 접속할 때 사용되는 JDBC 설정들이고, `jdbcIbatis.` 접두어를 사용한 설정들은 Mybatis 커넥션 풀을 이용할 때 사용되는 JDBC 설정들입니다.

(3-2-1-3) 스프링 빈 구성 파일(`mybatis-context.xml`)에 `properties` 파일을 이용한 데이터베이스 접속 구성 설정

☞ 앞에서 실습한 JDBC 정보 직접 설정 실습 시 구성된 내용을 모두 주석처리 후, 아래의 내용을 추가해야 합니다.

→ `mybatis-context.xml` 을 열고, 다음의 빨간색 코드를 작성(오타, 대소문자 주의!!!). 파란색 주석은 작성할 필요 없습니다.

```
...(생략)...
<!-- Connection Pool 관련 Bean 구성: HikariCP -->

<!-- 방법1: JDBC 정보를 직접 설정 -->
<!-- 1. JDBC 설정 정보가 설정된 HikariConfig 빈 생성--> <!--
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <property name="driverClassName" value="oracle.jdbc.OracleDriver"></property>
    <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe"></property>
    <property name="username" value="book_ex"></property>
    <property name="password" value="book_ex"></property>
```

```

</bean> -->

<!-- 2. HikariDataSource 클래스를 이용한 dataSource 빈 생성: JDBC 설정이 저장된 HikariConfig 빈 주입받음 --><!--
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close">
    <constructor-arg ref="hikariConfig" />
</bean> -->

<!-- 방법2: HikariCP - properties 파일을 이용한 접속 구성 -->
<!-- 1. jdbc.properties 파일에서 읽어 들이는 설정: PropertyPlaceholderConfigurer 클래스 빈을 이용 -->
<!-- [주의] PropertyPlaceholderConfigurer 클래스 빈을 이용하는 아래의 방법은 스프링 5에서 Deprecated.
<bean id="propertyPlaceholderConfigurer"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <value>/WEB-INF/config/jdbc.properties</value>
    </property>
</bean> -->

<!-- 1. jdbc.properties 파일에서 읽어 들이는 설정 -->
<!-- properties 파일을 webapp 하위에 구성한 경우: JUnit에서는 사용되지 못함, 문서에서 구성 않함 -->
<!-- <context:property-placeholder location="/WEB-INF/jdbcConfig/jdbc2.properties"/> -->
<!-- properties 파일을 src/main/resources 하위에 구성한 경우 -->
<context:property-placeholder location="classpath:jdbcConfig/jdbc.properties"/>

<!-- 2. HikariConfig 빈을 통해 읽어들인 정보를 설정, 설정값 형식이 JSTL 표현식과 유사 -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <property name="driverClassName" value="${jdbcHikari.driverClassName}"></property>
    <property name="jdbcUrl" value="${jdbcHikari.jdbcUrl}"></property>
    <property name="username" value="${jdbcHikari.username}"></property>
    <property name="password" value="${jdbcHikari.password}"></property>
</bean>

<!-- 3. HikariDataSource 클래스를 이용한 dataSource 빈 생성 -->
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close">
    <constructor-arg ref="hikariConfig" />
</bean>

</beans>

```

→ 구성 내용 테스트: Junit을 이용해 커넥션 풀 구성 테스트

- ☞ mybatis-context.xml 파일에 "단순 구성" 실습코드는 주석처리 해야 합니다.
- ☞ 앞에서 사용한 테스트 코드를 그대로 사용합니다.

→ 코드 작성부에 DataSourceTests.java 소스 코드를 오픈

→ DataSourceTests.java 코드 작성 창에서 마우스 오른쪽 버튼 클릭 → Run As → 2. JUnit Test 클릭

→ 테스트가 정상적으로 완료되면, Junit 뷰에 녹색진행표시가 표시되고 콘솔에 INFO 레벨의 기록이 표시됩니다.

[콘솔 표시 내용]

```

...(생략)...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
INFO : com.spring5213.mypro00.datasource.DataSourceTests - HikariProxyConnection@1581267786 wrapping oracle.jdbc.driver.T4CConnection@2ed2d9cb
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown completed.

```

☞ 문서에서는 properties 파일을 이용한 Hikari 커넥션 풀 접속 구성을 사용합니다.

[참고] 다음은 Hikari 커넥션 풀 대신 Mybatis 커넥션 풀을 사용했을 때 설정입니다. 문서에서는 구성하지 않습니다.

```
<!-- Connection Pool 관련 Bean 구성: MybatisCP -->

<!-- 방법1: MybatisCP - JDBC 정보를 직접 구성 --><!--
<bean id="dataSource" class="org.apache.ibatis.datasource.pooled.PooledDataSource">
    <property name="driver" value="oracle.jdbc.driver.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="username" value="book_ex" />
    <property name="password" value="book_ex" />
</bean> -->

<!-- 방법2: MybatisCP - properties 파일을 이용한 접속 구성 -->
<!-- 1. jdbc.properties 파일에서 읽어 들이는 설정 -->
    <!-- src/main/resources 하위에 구성 --><!--
    <context:property-placeholder location="classpath:jdbcConfig/jdbc.properties"/> -->

<!-- 2. dataSource 빈을 생성 --><!--
<bean id="dataSource" class="org.apache.ibatis.datasource.pooled.PooledDataSource">
    <property name="driver" value="${jdbcIbatis.driverClassName}" />
    <property name="url" value="${jdbcIbatis.url}" />
    <property name="username" value="${jdbcIbatis.username}" />
    <property name="password" value="${jdbcIbatis.password}" />
</bean> -->
```

#### (4) web.xml 파일에 스프링 빈 구성 설정 파일(mybatis-context.xml) 파일 등록

☞ mybatis-context.xml 파일의 내용이 톰캣 기동 시에 읽혀져서, 설정된 빈들이 구성되도록, web.xml 파일에 등록합니다.

→ 프로젝트 탐색 창에서 src/main/webapp/WEB-INF 폴더에 있는 web.xml 을 열고, 다음을 작성합니다(빨간색 부분).

```
...(생략)...
<!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/spring/root-context.xml
        /WEB-INF/spring/mybatis-context.xml
    </param-value>
</context-param>
...(생략)...
```

☞ 스프링 프레임워크에서는 2 종류의 컨텍스트 영역이 사용됩니다.

- root-context.xml 설정파일로 구성되는 ROOT 컨텍스트 영역: 데이터베이스 접속과 관련된 공통 빈들이 구성됨.
- servlet-context.xml 설정파일로 구성되는 APPLICATION 컨텍스트 영역: 웹 애플리케이션의 컨트롤러/서비스/DAO/VO 빈들이 구성됨

☞ 실습으로 생성한 mbatis-context.xml 파일은 데이터베이스 접속과 관련된 빈들이므로,  
web.xml 파일의 ROOT 컨텍스트 설정부분에 구성을 추가합니다.

## 2-3. MyBatis(mybatis-spring 포함) 구성

☞ 자바 소스로부터 SQL문장을 분리하여 개발의 효율을 높이는 Mybatis 프레임워크를 프로젝트에 적용합니다.

(1) MyBatis와 mybatis-spring 라이브러리 설치(이미 설치되어 있습니다).

- 이클립스에서 pom.xml 파일을 열고 다음을 추가합니다.

```
<!-- 새로 추가: mybatis, mybatis-spring -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.7</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>2.0.6</version>
</dependency>
```

☞ 문서에서는 1.3 버전 대신 2 버전의 Mybatis-spring 라이브러리를 사용합니다.

mybatis-spring에 대한 자세한 내용은 <http://mybatis.org/spring/ko/index.html> 사이트를 참고하십시오.

(2) mybatis-spring 라이브러리 빈 설정

☞ 앞에서 구성된 데이터베이스 접속을 처리하는 dataSource 빈을 주입받아, 데이터베이스에 세션을 오픈하고, 데이터베이스에 SQL 요청을 하고 결과를 받아 전달하는 mybatis-spring 라이브러리의 SqlSessionFactoryBean을 mybatis-context.xml 파일에 설정합니다.

→ mybatis-context.xml 파일에, org.mybatis.spring.SqlSessionFactoryBean 빈을 등록합니다. 아래의 빨간색 코드를 추가합니다.

```
<!-- 4. mybatis-spring 연동 (dataSource 빈 주입 받음) -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
<!--
    <property name="configLocation"
        value="classpath:com/spring5213/mypro00/mapper/mapper/mybatisConfig.xml"/> --
<!-- mybatis-spring <property name="mapperLocations" value="classpath:경로명/*Mapper.xml"> 처럼
    value 속성에 하나의 값만 설정해야 했습니다. 버전2 부터 mapperLocations에 여러 개의 매퍼파일들을
    아래처럼 정의할 수 있습니다. --><!--
    <property name="mapperLocations">
        <list>
            <value>classpath:com/spring5213/mypro00/mapper/MyBoardMapper.xml</value>
            <value>classpath:com/spring5213/mypro00/mapper/MyReplyMapper.xml</value>
        </list>
    </property> --
</bean>

<!-- 5. mybatis-spring 연동 (SqlSessionFactoryBean 빈 주입 받음) -->
<!-- SqlSessionFactory 보다 쓰레드에 안전 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>

</beans>
```

☞ Mybatis를 사용할 때, SQL-매퍼(mapper) 라고 불리는 XML 파일에, 데이터베이스로 전달되는 SQL문을 작성하여 구성합니다.

위의 파란색 주석 내용 중에 name 속성에 mapperLocations 로 설정된 property 요소에

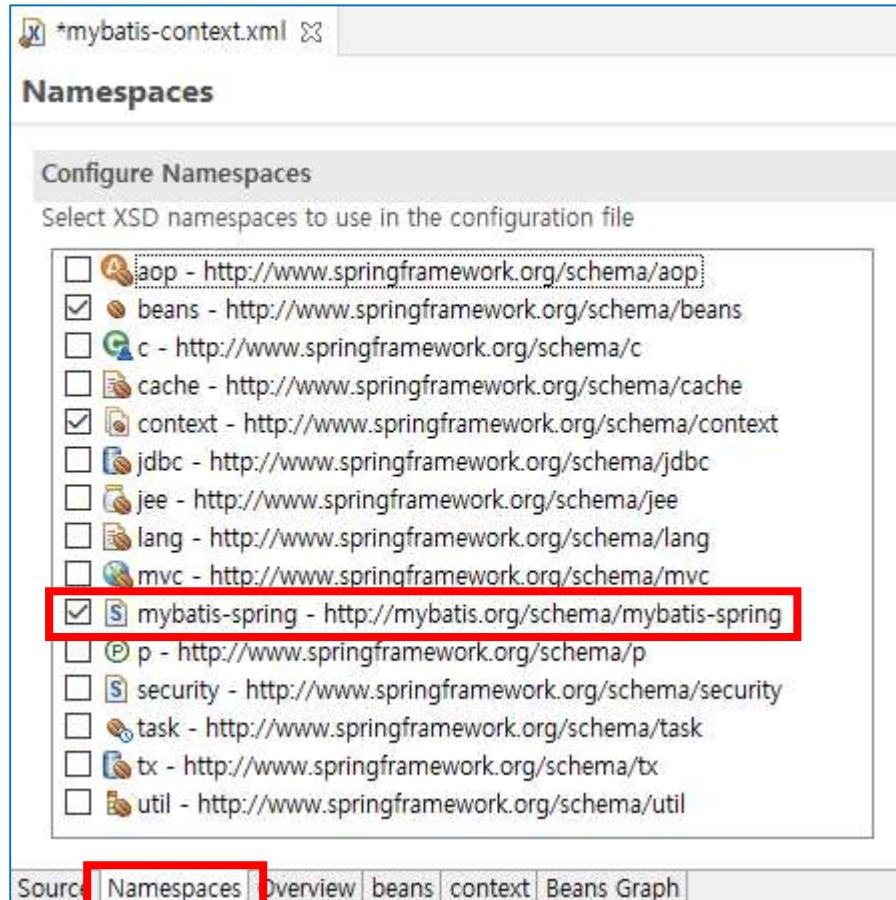
웹 애플리케이션에서 사용되는 매퍼파일의 위치와 파일이름을 설정합니다. 뒤에서 실습합니다.

☞ SQL-매퍼 파일에 저장된 SQL문을 호출하기 위하여 기존에는 DAO (Data Access Object)라는 자바 클래스를 사용했습니다.

최근에는 DAO 대신 인터페이스를 이용하여 SQL-매퍼에 있는 문장을 호출합니다. 문서에서도 인터페이스를 이용하여 SQL문을 호출하도록 구성합니다. 인터페이스를 이용하여 SQL-매퍼에 있는 SQL문을 호출하려면, 스프링 빈 구성 설정파일(mybatis-context.xml) 파일에 mybatis-spring 네임스페이스를 추가로 설정해야 합니다.

→ 코드 작성 뷰에 mybatis-context.xml 파일을 열고, 코드 작성 뷰 하단의 Namespaces 탭 클릭 → mybatis-spring 항목을 체크

→ 저장 !!! → Source 탭을 클릭.



→ mybatis-context.xml 파일 상단의 내용에, 다음처럼 네임스페이스 설정이 자동으로 추가됩니다.

→ 이 때, 새로 추가된 내용(빨간색 코드)에서, mybatis-spring-1.2.xsd를 mybatis-spring.xsd로 수정합니다.

→ 다음은 수정까지 마친 mybatis-context.xml 파일의 네임스페이스 설정 부분입니다.

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

☞ 문서에서 설치한 mybatis-spring 라이브러리 버전이 2.x.x 이므로, 이에 대응되는 스키마 정의 파일을 사용하기 위하여, 1.x.x 버전에

해당하는 1.2를 삭제합니다.

## 2-4. log4jdbc-log4j2 라이브러리를 이용한 로그기록 구성

☞ log4jdbc-log4j2 라이브러리를 사용하여, 개발 시에 Mybatis 통해서 처리되는 SQL문과 SQL문에 바인딩되는 값들을 콘솔에 로그로 표시할 수 있습니다. 단, 개발시에만 활용하면 좋습니다. 운영환경에서는 제외하는 것이 좋습니다.

(1) log4jdbc-log4j2 라이브러리 설치(이미 설치되어 있습니다).

→ 이클립스에서 pom.xml 파일을 열고 다음을 추가합니다.

```
<!-- 새로 추가: log4jdbc-log4j2 -->
<!-- log4jdbc-log4j2-jdbc4 사용 시:
    HikariPool-1 - Driver does not support get/set network timeout for connections. 경고 표시됨 -->
<!-- log4jdbc-log4j2-jdbc4.1 사용 시: 위의 메시지 표시 안 됨 -->
<dependency>
    <groupId>org.bgee.log4jdbc-log4j2</groupId>
    <artifactId>log4jdbc-log4j2-jdbc4.1</artifactId>
    <version>1.16</version>
</dependency>
```

(2) 설치된 log4jdbc-log4j2 라이브러리를 사용하여 로그가 기록되도록 다음을 구성합니다.

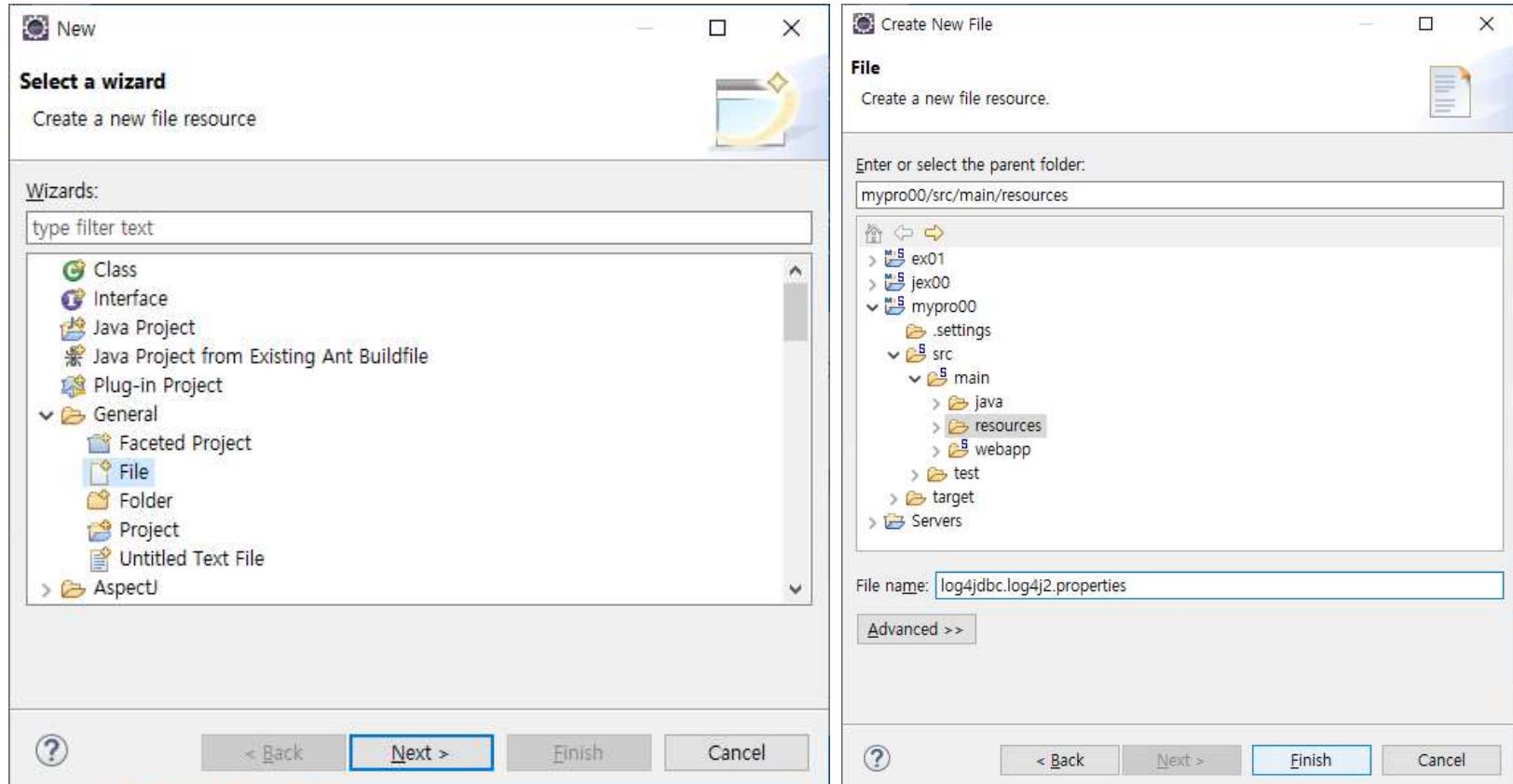
(2-1) src/main/resources 폴더에 log4jdbc.log4j2.properties 파일 생성.

→ src/main/resources 폴더에서 마우스 우측 버튼 클릭 후, New → Other 클릭

→ General 폴더의 File 선택 후, Next 클릭

→ 위에 표시된 폴더 경로가 mypro00/src/main/resources 인지 확인

→ File name 입력란에 log4jdbc.log4j2.properties 입력(오타, 대소문자 주의!!) → Finish 클릭.



→ 생성된 log4jdbc.log4j2.properties 파일에 다음을 입력하고 저장합니다(오타 및 대소문자 주의!!!).

```
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
```

(2-2) jdbc.properties 및 mybatis-context.xml 파일에 설정된 JDBC 정보 수정

☞ Mybatis의 SQL문 요청 처리가 log4jdbc-log4j2를 통해서, 커넥션 풀 → Oracle JDBC → 오라클 DB에 전달될 수 있도록, mybatis-context.xml 파일과 jdbc.properties 파일에 필요한 내용을 추가 및 수정합니다.

☞ 이 때 mybatis-context.xml 파일에서는, 수정되는 기존 내용(서버 배포용 드라이버 구성)은 주석으로 처리한 후, 내용을 작성하세요.

→ 코드 작성 뷰에 src/main/resources/jdbcConfig 폴더에 있는 jdbc.properties 파일에 다음의 코드를 기존 내용 밑에 추가

☞ 기존 코드를 복사해서 밑에 붙여넣고, 굵은 부분만 수정하시면 편합니다(오타, 대소문자 주의!!)

```
#HikariCP-log4jdbc.log4j2: For Development
jdbcHikariLog.driverClassName=net.sf.log4jdbc.sql.jdbcapi.DriverSpy
jdbcHikariLog.jdbcUrl=jdbc:log4jdbc:oracle:thin:@localhost:1521:xe
jdbcHikariLog.username=book_ex
jdbcHikariLog.password=book_ex

#ibatisCP-log4jdbc.log4j2: For Development
jdbcIbatisLog.driverClassName=net.sf.log4jdbc.sql.jdbcapi.DriverSpy
jdbcIbatisLog.url=jdbc:log4jdbc:oracle:thin:@localhost:1521:xe
jdbcIbatisLog.username=book_ex
jdbcIbatisLog.password=book_ex
```

→ 코드 작성뷰에 mybatis-context.xml 파일을 열고 다음처럼 수정합니다

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

    <!-- Connection Pool 관련 Bean 구성: HikariCP -->

    <!-- 방법1: HikariCP - JDBC 정보를 직접 설정 -->
    <!-- 1. JDBC 설정 정보를 구성하는 HikariConfig 빈 생성 --><!--
        <bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig"> -->
            <!-- for Server Deploy --><!--
            <property name="driverClassName" value="oracle.jdbc.OracleDriver"></property>
            <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe"></property> -->

            <!-- for Development, log4jdbc-log4j2 라이브러리 --><!--
            <property name="driverClassName" value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
            <property name="jdbcUrl" value="jdbc:log4jdbc:oracle:thin:@localhost:1521:xe"></property> -->

            <!-- 접속 계정 및 암호 --><!--
            <property name="username" value="book_ex"></property>
            <property name="password" value="book_ex"></property>
```

```

</bean> -->

<!-- 2. HikariDataSource 클래스를 이용한 dataSource 빈 생성: JDBC 설정이 저장된 HikariConfig 빈 주입받음 --><!--
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close">
    <constructor-arg ref="hikariConfig" />
</bean> -->

<!-- 방법2: HikariCP - properties 파일을 이용한 접속 구성 -->
<!-- 1. jdbc.properties 파일에서 읽어 들이는 설정 -->
    <!-- properties 파일을 src/main/resources 하위에 구성한 경우 -->
    <context:property-placeholder location="classpath:jdbcConfig/jdbc.properties"/>

<!-- 2. HikariConfig 빈을 통해 읽어들인 정보를 설정: 설정값 형식이 JSTL 표현식과 유사 -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <!-- for Server Deploy: 기존 아래의 설정 4개 주석처리 후, 밑에 추가 --><!--
    <property name="driverClassName" value="${jdbcHikari.driverClassName}"></property>
    <property name="jdbcUrl" value="${jdbcHikari.jdbcUrl}"></property>
    <property name="username" value="${jdbcHikari.username}"></property>
    <property name="password" value="${jdbcHikari.password}"></property> -->

    <!-- for Development - log4jdbc-log4j2 라이브러리-->
    <property name="driverClassName" value="${jdbcHikariLog.driverClassName}"></property>
    <property name="jdbcUrl" value="${jdbcHikariLog.jdbcUrl}"></property>
    <property name="username" value="${jdbcHikariLog.username}"></property>
    <property name="password" value="${jdbcHikariLog.password}"></property>
</bean>

<!-- 3. HikariDataSource 클래스를 이용한 dataSource 빈 생성 -->
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close">
    <constructor-arg ref="hikariConfig" />
</bean>

<!-- 4. mybatis-spring 연동 (dataSource 빈 주입 받음) -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
<!--
    <property name="configLocation"
        value="classpath:com/spring5213/mypro00/mapper/mapper/mybatisConfig.xml"/>
    <property name="mapperLocations">
        <list>
            <value>classpath:com/spring5213/mypro00/mapper/MyBoardMapper.xml</value>
            <value>classpath:com/spring5213/mypro00/mapper/MyReplyMapper.xml</value>
        </list>
    </property> -->
</bean>

<!-- 5. mybatis-spring 연동 (SqlSessionFactoryBean 빈 주입 받음) -->
<!-- SqlSessionFactory 보다 쓰래드에 안전 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>

</beans>

```

### (3) 콘솔 표시 로그양 줄이는 구성

☞ 콘솔에 표시되는 log4j 로그의 로그 레벨을 info로 변경하고, 이 때 표시되는 log4j-콘솔 로그의 양을 줄이기 위하여, src/main/resources 폴더와 src/test/resource 폴더에 있는 log4j.xml 파일에 다음의 빨간색 내용을 추가합니다.

... (생략)...

<logger name="org.springframework.web">

```

<level value="info" />
</logger>

<!-- JDBC 로그기록 양을 줄이기 위하여 추가된 내용 -->
<logger name="jdbc.audit">
<level value="warn" />
</logger>

<logger name="jdbc.resultset">
<level value="warn" />
</logger>

<logger name="jdbc.connection">
<level value="warn" />
</logger>
<!-- Root Logger -->
<root>
<priority value="info" /><!-- warn을 info로 변경 -->
<appender-ref ref="console" />
</root>

</log4j:configuration>

```

#### [구성 내용 Junit 테스트]

- 앞에서 작성한 src/test/java 폴더의 com.spring5213.mypro00.datasource 패키지에 작성한 DataSourceTests.java 파일에 편집  
 → 기존 JDBC 테스트 메서드는 주석처리 후, 다음의 빨간색 테스트 메서드를 기존 주석처리된 메서드 밑에 추가.

```

package com.spring5213.mypro00.datasource;

import static org.junit.Assert.fail;

import java.sql.Connection;

//import javax.sql.DataSource;

//import org.apache.ibatis.session.SqlSession;
//import org.apache.ibatis.session.SqlSessionFactory;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/mybatis-context.xml")
@Log4j
public class DataSourceTests {

//    @Setter(onMethod_ = { @Autowired })
//    private DataSource dataSource;

//    @Test
//    public void testConnection() {
//        try (Connection conn = dataSource.getConnection()){
//            log.info(conn);
//        } catch(Exception e) {
//            fail(e.getMessage());
//        }
}

```

```

//      }
//    }

//    @Setter(onMethod_ = { @Autowired })
//    private SqlSessionFactory sqlSessionFactory;

    @Setter(onMethod_ = { @Autowired })
    private SqlSessionTemplate sqlSession;

    @Test
    public void testMyBatis() {
        try ( //SqlSession sqlSession = sqlSessionFactory.openSession();
              Connection conn = sqlSession.getConnection(); ) {
            log.info(conn);
            log.info(sqlSession);

        } catch (Exception e) {
            fail(e.getMessage());
        }
    }
}

```

☞ 작성된 테스트 코드는 지금까지 구성정보를 이용하여, 데이터베이스에 접속 및 Mybatis의 SqlSession 객체 생성까지 테스트합니다.

#### → 테스트 수행

→ 코드 작성 뷰에서 마우스 오른쪽 버튼 클릭 → Run As → Junit Test를 클릭하여 테스트 실행

☞ 테스트가 정상적으로 완료되면, Junit 뷰에 녹색 진행 표시가 표시되고 콘솔에 INFO 레벨의 기록이 표시됩니다.

☞ 다음은 src/main/resources/jdbcConfig/jdbc.properties 파일에 있는 HikariCP와 log4jdbc-log4j2 드라이버 정보를 이용하여 오라클 JDBC 드라이버로 오라클 데이터베이스에 접속하는 설정을 이용한 경우를 테스트했을 때 콘솔에 표시된 로그입니다(테스트 환경 구성 관련 로그는 제외됨).

[SqlSessionFactory를 이용하여 테스트 한 로그]

```

...(생략)...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
INFO : com.spring5213.mypro00.datasource.DataSourceTests - org.apache.ibatis.session.defaults.DefaultSqlSession@3d1848cc
INFO : com.spring5213.mypro00.datasource.DataSourceTests - HikariProxyConnection@877612522 wrapping
net.sf.log4jdbc.sql.jdbcapi.ConnectionSpy@1b7c473a
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown completed.

```

[SqlSessionTemplate를 이용하여 테스트 한 로그]

```

...(생략)...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
INFO : com.spring5213.mypro00.datasource.DataSourceTests - org.mybatis.spring.SqlSessionTemplate@19835e64
INFO : com.spring5213.mypro00.datasource.DataSourceTests - HikariProxyConnection@1280429864 wrapping
com.zaxxer.hikari.pool.ProxyConnection.ClosedConnection
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown completed.

```

[참고] HikariCP 사용 시 driverClassName에 oracle.jdbc.driver.OracleDriver를 사용하는 경우,  
아래와 같은 경고 메시지가 표시됩니다.

```
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
WARN : com.zaxxer.hikari.util.DriverDataSource - Registered driver with driverClassName=oracle.jdbc.driver.OracleDriver
was not found, trying direct instantiation.
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
```

☞ oracle.jdbc.OracleDriver를 사용하는 경우에는 위의 경고 메시지가 표시되지 않습니다.

[참고]

Junit 테스트 시에만 jdbc.properties 파일이 WEB-INF 폴더 하위에 있는 경우에는  
테스트 시에 오류가 발생됩니다(운영 시에는 문제가 없습니다).  
따라서, 문서에서는 테스트를 원활하게 진행하려고, jdbc.properties 파일을  
java/source/resource의 jdbcConfig 패키지에 생성하여 구성하였습니다.

- 단원 [2] 끝

# [프로젝트 수행: 게시판 작성]

## [03] 필요한 패키지 생성

- 다음의 작업이 진행됩니다.
  - 대표적인 프로젝트 구성 클래스들이 저장될 패키지 생성

→ mypro00 프로젝트의 src/main/java 폴더에 구성된 com.spring5213.mypro00 패키지에

다음의 하위 패키지(빨간색 패키지)를 생성합니다.

전체 패키지 이름	용도
com.spring5213.mypro00.common	공통적으로 쓰일 수 있는 기능 정의 클래스(파일 업로드/다운로드/페이징 등)가 저장됨
com.spring5213.mypro00.controller	사용자의 URI 요청을 처리하는 제어계층의 컨트롤러 클래스가 저장됨
com.spring5213.mypro00.domain	값을 저장하는 VO 클래스가 저장됨
com.spring5213.mypro00.mapper	Mybatis 매퍼-인터페이스가 저장됨
com.spring5213.mypro00.service	비즈니스 계층의 서비스 클래스가 저장됨.

- 단원 [3] 끝

## [04] 영속 계층(Persistence-Layer) 구현

- 다음의 작업들이 진행됩니다.
  - 데이터베이스에 게시물 저장 구조(테이블, 인덱스, 시퀀스 등) 생성 및 테스트 데이터 입력
  - 도메인(VO) 클래스 생성
  - 매퍼-인터페이스 및 매퍼-XML 파일 생성 및 매퍼 사용 구성
  - 게시물-목록표시/게시물-등록/게시물-조회/게시물-수정/게시물-삭제 관련 구성
  - 테스트 수행

### 4-1. 데이터베이스 게시물 저장 구조 생성

→ SQL\*Developer를 이용하여 book\_ex 계정(암호: book\_ex)으로 접속 후, 다음의 SQL문을 하나씩 실행합니다.

--게시물 번호(bno)로 사용될 숫자를 생성하는 오라클 데이터베이스 객체 생성

```
CREATE SEQUENCE book_ex.seq_myboard;
```

--게시물 데이터들이 저장될 테이블 생성

```
CREATE TABLE book_ex.tbl_myboard (
    bno NUMBER(10,0),          --게시물번호
    btitle VARCHAR2(200) NOT NULL, --게시물제목
    bcontent VARCHAR2(2000) NOT NULL, --게시물내용
    bwriter VARCHAR2(50) NOT NULL, --게시물작성자 ID
    bregDate DATE DEFAULT sysdate, --게시물등록날짜시간
    bmodDate TIMESTAMP(0) DEFAULT systimestamp(0), --게시물 최종 수정날짜시간
    bviewsCnt NUMBER(10) DEFAULT 0, --게시물 조회수
    breplyCnt NUMBER(10) DEFAULT 0, --게시물의 댓글/답글 수
    bdelFlag NUMBER(1) DEFAULT 0   --게시물 삭제가부 - 1: 삭제요청됨(Y), 0.유지(N)
) TABLESPACE users;
```

--게시물 기본키 추가

```
ALTER TABLE book_ex.tbl_myboard
ADD CONSTRAINT pk_myboard PRIMARY KEY (bno);
```

--테스트 데이터 입력: 10회 수행

```
INSERT INTO book_ex.tbl_myboard
VALUES (book_ex.seq_myboard.nextval,
        '테스트제목' || book_ex.seq_myboard.currval ,
        '테스트내용' || book_ex.seq_myboard.currval,
        'user' || book_ex.seq_myboard.currval,
        DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT); --10회 수행
```

--트랜잭션 커밋

```
COMMIT ;
```

--저장된 데이터 조회

```
SELECT * FROM book_ex.tbl_myboard ORDER BY bno DESC;
```

[SELECT문 실행 결과]

BNO	BTITLE	BCONTENT	BWRITER	BREGDATE	BMODDATE	BVIEWSCNT	BREPLYCNT	BDELFLAG
10	테스트제목10	테스트내용10	user10	2021/02/24 15:38:23	2021/02/24 15:38:24.000000000	0	0	0
9	테스트제목9	테스트내용9	user9	2021/02/24 15:38:23	2021/02/24 15:38:23.000000000	0	0	0

8 테스트제목8	테스트내용8	user8	2021/02/24 15:38:22	2021/02/24 15:38:22.000000000	0	0	0
7 테스트제목7	테스트내용7	user7	2021/02/24 15:38:21	2021/02/24 15:38:21.000000000	0	0	0
6 테스트제목6	테스트내용6	user6	2021/02/24 15:38:20	2021/02/24 15:38:21.000000000	0	0	0
5 테스트제목5	테스트내용5	user5	2021/02/24 15:38:19	2021/02/24 15:38:20.000000000	0	0	0
4 테스트제목4	테스트내용4	user4	2021/02/24 15:38:19	2021/02/24 15:38:19.000000000	0	0	0
3 테스트제목3	테스트내용3	user3	2021/02/24 15:38:18	2021/02/24 15:38:18.000000000	0	0	0
2 테스트제목2	테스트내용2	user2	2021/02/24 15:38:17	2021/02/24 15:38:17.000000000	0	0	0
1 테스트제목1	테스트내용1	user1	2021/02/24 15:38:16	2021/02/24 15:38:17.000000000	0	0	0

10개의 행이 선택됨

[참고] 잘못 정의된 테이블이름, 컬럼이름, 제약조건이름 변경

--테이블이름을 변경

**ALTER TABLE** 기존\_테이블이름 **RENAME TO** 새로운\_테이블\_이름;

--컬럼 이름 변경

**ALTER TABLE** 테이블이름 **RENAME** 기존\_컬럼이름 **TO** 새로운\_컬럼이름;

--제약조건 이름 변경

**ALTER TABLE** 테이블이름 **RENAME CONSTRAINT** 기존\_제약조건이름 **TO** 새로운\_제약조건이름;

## 4-2. 도메인(VO) 클래스 생성

- ☞ 데이터베이스 테이블의 데이터를 프로그램으로 전달하기 위해 사용되는 데이터 저장 클래스를 도메인(Domain) 클래스라고 부르며, 도메인 클래스의 객체를 Value Object 라고 합니다.
- ☞ 게시물 관련 데이터를 저장할 MyBoardVO 이름의 도메인 클래스를 생성합니다.
- ☞ 도메인 클래스의 필드이름은 일반적으로 데이터베이스 테이블의 컬럼명과 동일하게 지정합니다.

→ src/main/java 폴더의 com.spring5213.mypro00.domain 패키지에 MyBoardVO 클래스를 생성합니다.

```
package com.spring5213.mypro00.domain;

import java.sql.Timestamp;
import java.util.Date;
import lombok.Data;

@Data
public class MyBoardVO {
    private long bno ;
    private String btitle ;
    private String bcontent ;
    private String bwriter ;
    private int bviewsCnt ;
    private int breplyCnt ;
    private int bdelFlag ; //1: 삭제요청됨, 0: 유지
    private Date bregDate ;
    private Timestamp bmodDate ;
}
```

- ☞ `@Data` 룰복 어노테이션에 의해 컴파일 된 클래스에는 필드에 대한 getter-setter, 기본생성자, `equals()`, `hashCode()`, `toString()` 메소드가 재정의되어 자동으로 추가됩니다.
- ☞ `Date`를 `java.util.Date`로 임포트 하면, `bregDate` 필드의 값이 `Fri Feb 19 14:44:33 KST 2021`로 저장됩니다.
- ☞ `Date`를 `java.sql.Date`로 임포트 하면, `bregDate` 필드의 값이 `2021-02-19`로 저장됩니다.

#### 4-3. Mybatis 매퍼 인터페이스와 SQL-매퍼 파일 생성 및 Mybatis 매퍼 사용 구성

- ☞ Mybatis 매퍼는 Mybatis 매퍼 인터페이스와 SQL-매퍼 파일을 의미합니다.
- ☞ TBL\_MYBOARD 테이블에 대한 Mybatis 매퍼를 구성합니다.

##### (1) 데이터베이스의 TBL\_MYBOARD 테이블에 대한 매퍼-인터페이스 생성

→ src/main/java 폴더의 `com.spring5213.mypro00.mapper` 패키지에 `MyBoardMapper` 인터페이스를 생성하고, 다음의 내용을 작성합니다.

```
package com.spring5213.mypro00.mapper;

import java.util.List;

import com.spring5213.mypro00.domain.MyBoardVO;

public interface MyBoardMapper {
    //게시물 조회 - 목록
    public List<MyBoardVO> selectMyBoardList();

    //게시물 등록1 - selectKey 이용 않함
    public Integer insertMyBoard(MyBoardVO myBoard);

    //게시물 등록2 - selectKey 이용
    public Integer insertMyBoardSelectKey(MyBoardVO myBoard);

    //특정 게시물 조회
    public MyBoardVO selectMyBoard(Long bno);

    //특정 게시물 삭제 요청 - 해당 글의 bdelFlag을 1로 수정
    public int updateBdelFlag(Long bno);

    //특정 게시물 삭제 - 실제 삭제
    public int deleteMyBoard(Long bno);

    //게시물 삭제(관리자) - 사용자 삭제 요청된 게시물(bdelFlag = 1) 전체 삭제
    public int deleteAllBoardSetDeleted();

    //특정 게시물 수정
    public int updateMyBoard(MyBoardVO myBoard);

    //특정 게시물 조회수 증가
    public void updateBviewsCnt(Long bno);
}
```

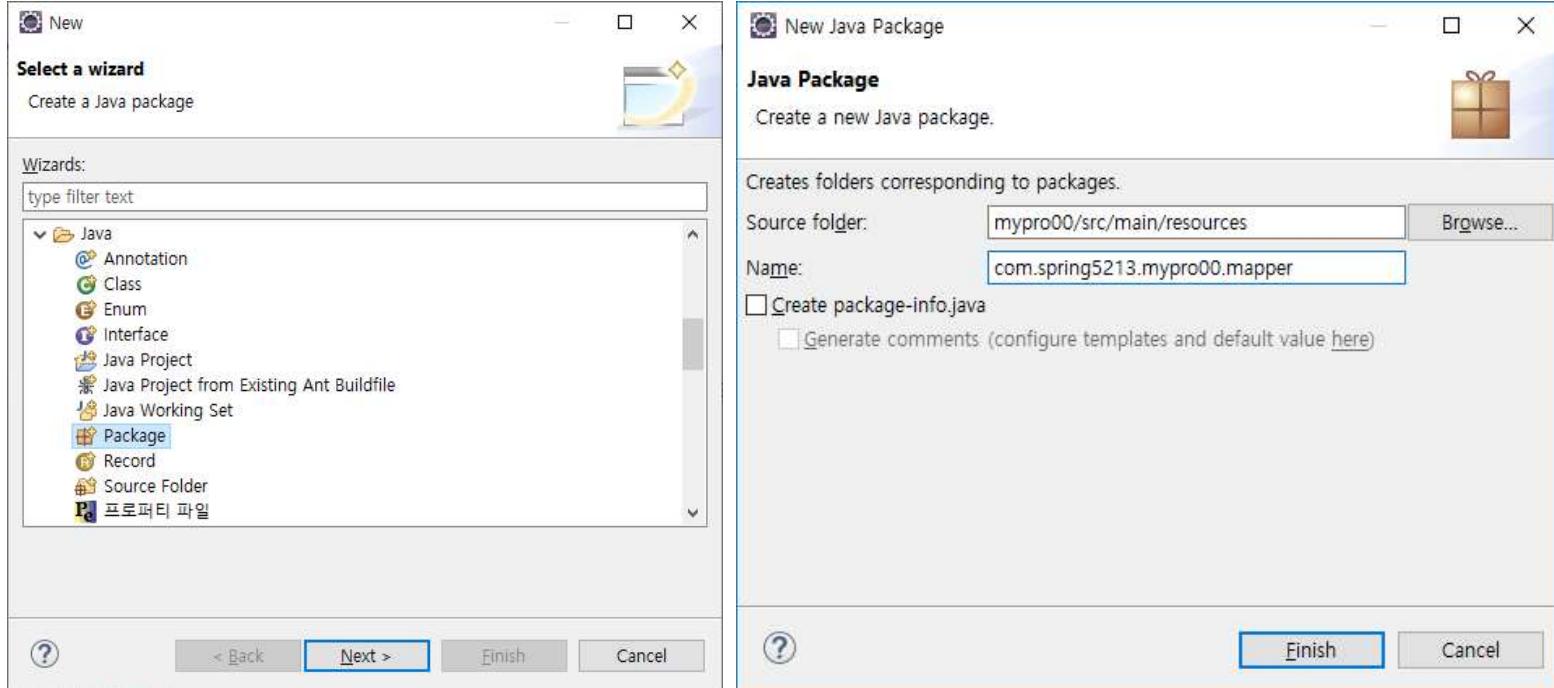
## (2) MyBoard SQL 매퍼 파일 생성

### (2-1) src/main/resources 폴더에 com.spring5213.mypro00.mapper 패키지 생성

→ src/main/resources 폴더에서 마우스 오른쪽 버튼 클릭 → new → Package 클릭

→ Source folder에 mypro00/src/main/resources 표시 확인 후, Name 입력란에 com.spring5213.mypro00.mapper 입력

→ Finish 클릭



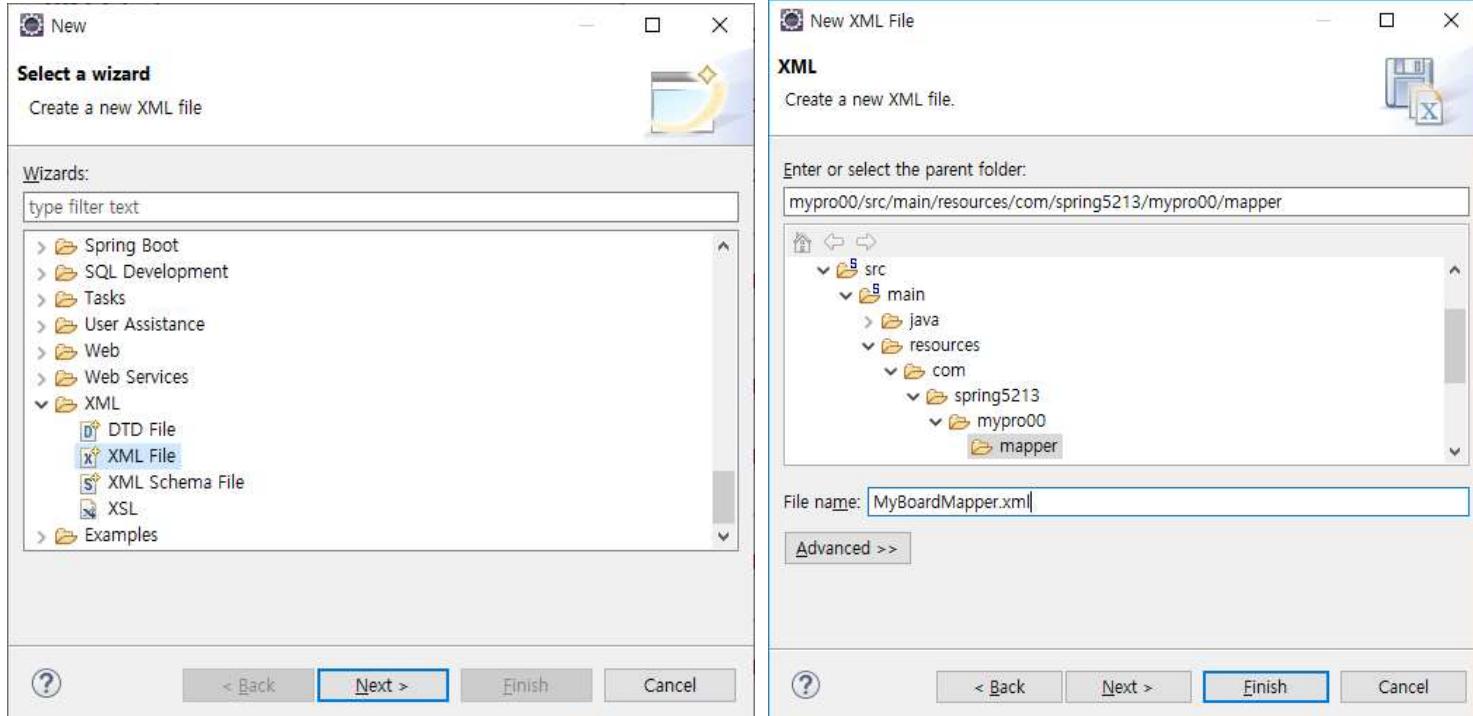
### (2-2) MyBoardMapper.xml 파일 생성

→ 생성된 src/main/resources 폴더의 com.spring5213.mypro00.mapper 패키지에서 마우스 오른쪽 버튼 클릭

→ New → Other 클릭 → XML 폴더에서 XML File 선택 후, Next 클릭

→ 상단의 폴더이름이 mypro00/src/main/resources/com/spring5213/mypro00/mapper 인지 확인 후,

→ File Name 입력란에 MyBoardMapper.xml 입력 후, Finish 클릭.



### (2-3) MyBoardMapper.xml 파일 내용 작성

☞ 생성된 MyBoardMapper.xml 파일에 MyBoardMapper 인터페이스의 각 메서드에 대응되는 SQL문들을 작성합니다.

→ src/main/resources 폴더의 com.spring5213.mypro00.mapper 패키지에 생성된 MyBoardMapper.xml 파일에 다음의 내용을 작성합니다. 내용이 많습니다. 오타없이 작성하시기 바랍니다.

☞ SQL 매퍼파일의 경우, 아래에서 !DOCTYPE 다음에 mapper를 설정하고, XML 정의파일을 mapper로 정의해야 합니다.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!-- 문서 유형에 mapper를 설정해야 하고, XML 정의 파일은 Mapper의 정의를 가져오도록 설정함 -->
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<!-- SQL문들은 <mapper>와 </mapper> 사이에 해당 문장 유형의 요소를 사용하여 정의함 -->
<!-- mapper 요소의 namespace 속성에는 매퍼 인터페이스의 패키지이름이 포함된 전체이름을 설정함 -->
<mapper namespace="com.spring5213.mypro00.mapper.MyBoardMapper">

<!-- 각 문장 유형 요소의 id 속성의 값은, 해당 문장을 호출하는 매퍼-인터페이스의 메소드 이름과 동일해야 함 -->

<!-- 게시물 조회 - 목록 -->
<select id="selectMyBoardList" resultType="com.spring5213.mypro00.domain.MyBoardVO">
    SELECT * FROM book_ex.tbl_myboard
    WHERE bno > 0 <!-- AND bDelFlag = 0 -->
    ORDER BY bno DESC
</select>

<!-- 게시물 등록1 - selectKey 이용않함 -->
<insert id="insertMyBoard">
<![CDATA[
    INSERT INTO book_ex.tbl_myboard
    VALUES (seq_myboard.nextval, #{btitle}, #{bcontent}, #{bwriter},
            DEFAULT, DEFAULT, DEFAULT, DEFAULT)
]]>
</insert>

<!-- 게시물 등록2 - selectKey 이용 -->
<insert id="insertMyBoardSelectKey">
    <selectKey keyProperty="bno" order="BEFORE" resultType="long">
        SELECT book_ex.seq_myboard.nextval FROM dual
    </selectKey>

    INSERT INTO book_ex.tbl_myboard
    VALUES (#{bno}, #{btitle}, #{bcontent}, #{bwriter},
            DEFAULT, DEFAULT, DEFAULT, DEFAULT)
</insert>

<!-- 특정 게시물 조회 -->
<select id="selectMyBoard" resultType="com.spring5213.mypro00.domain.MyBoardVO">
    SELECT * FROM book_ex.tbl_myboard
    WHERE bno = #{bno} <!-- AND bDelFlag = 0 -->
</select>

<!-- 특정 게시물 수정 -->
<update id="updateMyBoard">
    UPDATE book_ex.tbl_myboard
    SET btitle = #{btitle},
        bcontent = #{bcontent},
        bmodDate = DEFAULT
    WHERE bno = #{bno} <!-- AND bdelFlag = 0 -->
</update>
```

```

<!-- 특정 게시물 삭제 요청 - 해당 글의 bdelFlag을 1로 수정 -->
<update id="updateBdelFlag">
    UPDATE book_ex.tbl_myboard
    SET bdelFlag = 1
    WHERE bno = #{bno}
</update>

<!-- 특정 게시물 삭제 - 실제 삭제 -->
<delete id="deleteMyBoard">
    DELETE book_ex.tbl_myboard WHERE bno = #{bno} <!-- AND bDelFlag = 1 -->
</delete>

<!-- 게시물 삭제 - 삭제 요청된 전체 게시물 삭제 -->
<delete id="deleteAllBoardSetDeleted">
    DELETE book_ex.tbl_myboard WHERE bDelFlag = 1
</delete>

<!-- 특정 게시물 조회수 증가 : 게시물 조회 시 + 1 -->
<update id="updateBviewsCnt">
    UPDATE book_ex.tbl_myboard
    SET bviewsCnt = bviewsCnt + 1
    WHERE bno = #{bno} <!-- AND bdelFlag = 0 -->
</update>

</mapper>

```

### (3) Mybatis 매퍼 사용 구성

☞ 구성된 MyBoardMapper 인터페이스와 MyBoardMapper.xml 파일을 mybatis-spring 라이브러리를 통해 Mybatis에서 사용될 수 있도록 구성합니다.

→ src/main/webapp/WEB-INF/spring 폴더에 있는 mybatis-context.xml 파일을 열고, mybatis-spring 네임스페이스가 아래처럼 정의되어 있는지 확인합니다. 아래의 빨간색 코드가 없는 경우, 코드 작업 뷰 하단의 Namespaces 탭을 이용하여 mybatis-spring 네임스페이스를 추가하고 저장합니다.

```

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

```

☞ 문서에서 설치한 mybatis-spring 라이브러리 버전이 2.x.x 이므로, 이에 대응되는 스키마 정의 파일에 버전이 명시되면 않습니다.

→ mybatis-context.xml 파일의 아래에, 다음의 코드(빨간색 코드)를 추가합니다.

```

<!-- 4. mybatis-spring 연동 (HikariDataSource 빈 주입 받음) -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>

    <!-- <property name="configLocation" value="classpath:com/spring5213/mypro00/mapper/mybatisConfig.xml"/>
        <property name="mapperLocations" value="classpath:com/spring5213/mypro00/mapper/*Mapper.xml"/> -->

    <!-- sqlSessionFactory에, configLocation 프라퍼티를 사용하여 마이바티스 XML 설정파일의 위치를 지정할 수 있습니다.
        또한, mapperLocations 프라퍼티를 사용하여 마이바티스 SQL 매퍼파일의 위치를 지정할 수 있습니다 -->

```

실습에서는 마이바티스 XML 설정 파일을 구성하지 않습니다. `mapperLocations` 프로퍼티를 다음처럼 설정합니다. 이 때, 매퍼파일들의 위치는 `src/main/resources` 폴더에 `com/spring5213/mypro00/mapper` 폴더구조가 있고 매퍼파일이 위치해야 합니다.-->

```
<property name="mapperLocations">
    <list><!-- 매퍼파일이 여러 개일 경우, list 요소 내에 value 하부요소를 이용하여 파일 위치를 지정합니다.>
        <value>classpath:com/spring5213/mypro00/mapper/MyBoardMapper.xml</value>
        <!-- <value>classpath:com/spring5213/mypro00/mapper/MyReplyMapper.xml</value> -->
    </list>
</property>
</bean>
```

```
<!-- 5. mybatis-spring 연동 (SqlSessionFactoryBean 빈 주입 받음) -->
<!-- SqlSessionFactory 보다 쓰레드에 안전 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>

<!-- 매퍼인터페이스 자동 빈 생성 -->
<!-- 설정된 패키지에 DAO클래스를 대신하는 매퍼-인터페이스가 있습니다. -->
<mybatis-spring:scan base-package="com.spring5213.mypro00.mapper"/>

</beans>
```

☞ 다음은 `mybatis-spring`의 공식문서에 명시된 `configLocation`과 `mapperLocations`의 설명입니다.

하나의 공통적인 프로퍼티는 마이바티스 XML 설정파일의 위치를 지정하기 위해 사용되는 `configLocation`이다. 이 프로퍼티를 설정하는 것은 디폴트 설정을 가진 마이바티스 설정을 변경해야 할 경우 뿐이다. 대개는 `<settings>`과 `<typeAliases>` 섹션을 변경하는 경우이다.

설정파일이 마이바티스 설정을 완전히 다룰 필요는 없다. 어떤 환경, 어떤 데이터소스 그리고 마이바티스 트랜잭션 관리자가 무시될수도 있다. `SqlSessionFactoryBean`는 필요에 따라 이 값을 설정하여 자체적인 MyBatis `Environment`를 만든다.

설정파일이 필요한 다른 이유는 마이바티스 XML 파일이 매퍼 클래스와 동일한 클래스패스에 있지 않은 경우이다. 이 설정을 사용하면 두 가지 옵션이 있다. 첫 번째는 마이바티스 설정파일에 `<mappers>` 섹션을 사용해서 XML 파일의 클래스패스를 지정하는 것이다. 두 번째는 팩토리 빈의 `mapperLocations` 프로퍼티를 사용하는 것이다.

`mapperLocations` 프로퍼티는 매퍼에 관련된 자원의 위치를 나열한다. 이 프로퍼티는 마이바티스의 XML 매퍼 파일들의 위치를 지정하기 위해 사용될 수 있다. 디렉터리 아래 모든 파일을 로드하기 위해 앤트(Ant) 스타일의 패턴을 사용할 수도 있고 가장 상위 위치를 지정하는 것으로 재귀적으로 하위 경로를 찾도록 할 수도 있다. 예를 들어보면 다음과 같다.

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mapperLocations" value="classpath*:sample/config/mappers/**/*.xml" />
</bean>
```

#### 4-4. 테스트 매퍼 구성 테스트

(1) `MyBoardMapper` 인터페이스와 SQL 매퍼파일 테스트를 위한 기본 테스트 클래스 준비

→ `src/test/java` 폴더의 `com.spring5213.mypro00` 패키지에 `mapper` 패키지 생성

→ 생성된 com.spring5213.mypro00.mapper 패키지에 MyBoardMapperTest 클래스를 생성한 후, 다음의 코드 작성

```
package com.spring5213.mypro00.mapper;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/mybatis-context.xml")
@Log4j
public class MyBoardMapperTests {

    @Setter(onMethod_ = @Autowired)
    private MyBoardMapper myBoardMapper;

}
```

☞ 위의 클래스에 테스트 메서드를 하나씩 추가하면서 테스트를 수행합니다.

☞ 테스트 시에는 하나의 메서드씩 실행합니다.

따라서, 테스트가 완료된 메서드는 주석처리 후, 새로운 메서드를 하나씩 추가하면서 테스트를 수행합니다.

☞ 문서에 명시된 콘솔 로그 표시 내용에, 테스트 환경 구성 로그와 HikariDataSource 로그는 생략합니다.

## (2) 게시물 목록 조회 테스트

→ MyBoardMapperTests 클래스에 다음의 메서드를 클래스 내부의 기존 내용 밑에 추가

```
//게시물 목록 조회 테스트 → 테스트 후 메서드 주석처리
@Test
public void testSelectBoardList() {
    myBoardMapper.selectMyBoardList().forEach(myBoard -> log.info(myBoard));
}
```

→ 저장 후 → 마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트가 수행 → 이클립스 콘솔의 로그 확인.

### [콘솔 표시 내용]

```
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno > 0 AND bdelFlag = 0 ORDER BY bno DESC
INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno > 0 AND bdelFlag = 0 ORDER BY bno DESC
{executed in 6 msec}
INFO : jdbc.resultsettable -
+-----+-----+-----+-----+-----+-----+-----+
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
+-----+-----+-----+-----+-----+-----+-----+
|10 |테스트제목 10 |테스트내용 10 |user10 |2021-02-24 15:38:23.0 |2021-02-24 15:38:24.0 |0 |0 |0
|9 |테스트제목 9 |테스트내용 9 |user9 |2021-02-24 15:38:23.0 |2021-02-24 15:38:23.0 |0 |0 |0
|8 |테스트제목 8 |테스트내용 8 |user8 |2021-02-24 15:38:22.0 |2021-02-24 15:38:22.0 |0 |0 |0
|7 |테스트제목 7 |테스트내용 7 |user7 |2021-02-24 15:38:21.0 |2021-02-24 15:38:21.0 |0 |0 |0
|6 |테스트제목 6 |테스트내용 6 |user6 |2021-02-24 15:38:20.0 |2021-02-24 15:38:21.0 |0 |0 |0
|5 |테스트제목 5 |테스트내용 5 |user5 |2021-02-24 15:38:19.0 |2021-02-24 15:38:20.0 |0 |0 |0
|4 |테스트제목 4 |테스트내용 4 |user4 |2021-02-24 15:38:19.0 |2021-02-24 15:38:19.0 |0 |0 |0
|3 |테스트제목 3 |테스트내용 3 |user3 |2021-02-24 15:38:18.0 |2021-02-24 15:38:18.0 |0 |0 |0
|2 |테스트제목 2 |테스트내용 2 |user2 |2021-02-24 15:38:17.0 |2021-02-24 15:38:17.0 |0 |0 |0
|1 |테스트제목 1 |테스트내용 1 |user1 |2021-02-24 15:38:16.0 |2021-02-24 15:38:17.0 |1 |0 |0
+-----+-----+-----+-----+-----+-----+-----+
MyBoardVO(bno=10, btitle=테스트제목 10, bcontent=테스트내용 10, bwriter=user10, bviewsCnt=0, breplyCn
KST 2021, bmodDate=2021-02-24 15:38:24.0)
```

```

MyBoardVO(bno=9, btitle=테스트제목 9, bcontent=테스트내용 9, bwriter=user9, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:23 KST
2021, bmodDate=2021-02-24 15:38:23.0)
MyBoardVO(bno=8, btitle=테스트제목 8, bcontent=테스트내용 8, bwriter=user8, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:22 KST
2021, bmodDate=2021-02-24 15:38:22.0)
MyBoardVO(bno=7, btitle=테스트제목 7, bcontent=테스트내용 7, bwriter=user7, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:21 KST
2021, bmodDate=2021-02-24 15:38:21.0)
MyBoardVO(bno=6, btitle=테스트제목 6, bcontent=테스트내용 6, bwriter=user6, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:20 KST
2021, bmodDate=2021-02-24 15:38:21.0)
MyBoardVO(bno=5, btitle=테스트제목 5, bcontent=테스트내용 5, bwriter=user5, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:19 KST
2021, bmodDate=2021-02-24 15:38:20.0)
MyBoardVO(bno=4, btitle=테스트제목 4, bcontent=테스트내용 4, bwriter=user4, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:19 KST
2021, bmodDate=2021-02-24 15:38:19.0)
MyBoardVO(bno=3, btitle=테스트제목 3, bcontent=테스트내용 3, bwriter=user3, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:18 KST
2021, bmodDate=2021-02-24 15:38:18.0)
MyBoardVO(bno=2, btitle=테스트제목 2, bcontent=테스트내용 2, bwriter=user2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:17 KST
2021, bmodDate=2021-02-24 15:38:17.0)
MyBoardVO(bno=1, btitle=테스트제목 1, bcontent=테스트내용 1, bwriter=user1, bviewsCnt=1, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:16 KST
2021, bmodDate=2021-02-24 15:38:17.0)

```

### (3) 게시물 입력 테스트

→ MyBoardMapperTests 클래스에 다음의 메서드를 클래스 내부의 기존 내용 밑에 추가

```

import com.spring5213.mypro00.domain.MyBoardVO;
...(생략)...
//    @Test
//    public void testSelectBoardList() {
//        myBoardMapper.selectMyBoardList().forEach(myBoard -> log.info(myBoard));
//    }

//게시물 등록 테스트 - selectKey 사용 안함
@Test
public void testInsertMyBoard() {

    MyBoardVO myBoard = new MyBoardVO();
    myBoard.setBtitle("매퍼 테스트-입력제목");
    myBoard.setBcontent("매퍼 테스트-입력내용");
    myBoard.setBwriter("test");

    myBoardMapper.insertMyBoard(myBoard);
    log.info(myBoard);
}

//게시물 등록 테스트 - selectKey 사용
@Test
public void testInsertMyBoardSelectKey() {

    MyBoardVO myBoard = new MyBoardVO();
    myBoard.setBtitle("매퍼 테스트-select key");
    myBoard.setBcontent("매퍼 테스트-select key");
    myBoard.setBwriter("test");

    myBoardMapper.insertMyBoardSelectKey(myBoard);
    log.info(myBoard);
}

```

→ 저장 후 → 마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트가 수행 → 이클립스 콘솔의 로그 확인.

#### [콘솔 표시 내용]

```

INFO : jdbc.sqlonly - SELECT book_ex.seq_myboard.nextval FROM dual      ← select key insert Test Log 시작
INFO : jdbc.sqltiming - SELECT book_ex.seq_myboard.nextval FROM dual
{executed in 80 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|11      |
|-----|
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myboard VALUES (11, '매퍼 테스트-select key', '매퍼 테스트-select key', 'test01',
DEFAULT, DEFAULT, DEFAULT, DEFAULT)

```

```
INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myboard VALUES (11, '매퍼 테스트-select key', '매퍼 테스트-select key', 'test01',  
DEFAULT, DEFAULT, DEFAULT, DEFAULT)  
{executed in 10 msec}  
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - MyBoardVO(bno=11, btitle=매퍼 테스트-select key, bcontent=매퍼 테스트-select key,  
bwriter=test01, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null)  
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myboard VALUES (seq_myboard.nextval, '매퍼 테스트-입력제목', '매퍼 테스트-입력내용', ← select key 사용  
없는 insert Test Log 시작  
'test', DEFAULT, DEFAULT, DEFAULT, DEFAULT)  
  
INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myboard VALUES (seq_myboard.nextval, '매퍼 테스트-입력제목', '매퍼 테스트-입력내용',  
'test', DEFAULT, DEFAULT, DEFAULT, DEFAULT)  
{executed in 7 msec}  
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - MyBoardVO(bno=0, btitle=매퍼 테스트-입력제목, bcontent=매퍼 테스트-입력내용, bwriter=test,  
bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null)
```

- ☞ `select key`를 사용한 경우, 테스트 메서드에 사용된 `myBoard` 객체에 `bno` 값이 자동으로 전달되어 지정되어 있는 것이 확인됩니다.  
해당 값은 따라서, 필요한 경우, 서비스 또는 컨트롤러 클래스에서 `bno` 값이 필요할 경우, `getter`를 이용하여 값을 사용할 수 있습니다.

#### (4) 게시물 조회 테스트

→ MyBoardMapperTests 클래스에 다음의 메서드를 클래스 내부의 기존 내용 밑에 추가

//게시물 조회 테스트(by bno) → 테스트 후 메서드 주석처리

```
@Test  
public void testSelectMyBoard() {  
    // 존재하는 게시물 번호로 테스트  
    log.info(myBoardMapper.selectMyBoard(1L));  
}
```

→ 저장 후 → 마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트가 수행 → 이클립스 콘솔의 로그 확인.

## [콘솔 표시 내용]

```
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
{executed in 99 msec}
INFO : jdbc.resultsettable -
+-----+-----+-----+-----+-----+-----+-----+-----+
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
+-----+-----+-----+-----+-----+-----+-----+-----+
|1   |테스트제목1 |테스트내용1 |user1 |2021-02-24 15:38:16.0 |2021-02-24 15:38:17.0 |0          |0          |0          |
+-----+-----+-----+-----+-----+-----+-----+-----+
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - MyBoardVO(bno=1, btitle=테스트제목1, bcontent=테스트내용1, bwritter=user1, bviewsCnt=2,
breplyCnt=0, bdelflag=0, bregDate=Wed Feb 24 15:38:16 KST 2021, bmodDate=2021-02-24 15:38:17.0)
```

## (5) 게시물 삭제 테스트

→ MyBoardMapperTests 클래스에 다음의 메서드를 클래스 내부의 기존 내용 밑에 추가

//게시물 삭제 요청 테스트 - bdelFlag 컬럼값이 0 -> 1로 수정만 됨

```
@Test  
public void testUpdateBdelFlag() {  
    myBoardMapper.updateBdelFlag(10L);  
    log.info(myBoardMapper.selectMyBoard(10L));  
}
```

//게시물 삭제 테스트 - 실제 특정 게시물이 삭제됨

```
@Test  
public void testDeleteMyBoard() {  
    log.info("DELETE COUNT: " + myBoardMapper.deleteMyBoard(9L));  
    log.info(myBoardMapper.selectMyBoard(9L));  
}
```

→ 저장 후 → 마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트가 수행 → 이클립스 콘솔의 로그 확인.

#### [콘솔 표시 내용]

```
INFO : jdbc.sqlonly - DELETE book_ex.tbl_myboard WHERE bno = 9 ← 관리자 삭제
INFO : jdbc.sqltiming - DELETE book_ex.tbl_myboard WHERE bno = 9
{executed in 15 msec}
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - DELETE COUNT: 1
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 9

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 9
{executed in 83 msec}
INFO : jdbc.resultsettable - ← 표시되는 데이터 없음
+-----+-----+-----+-----+-----+-----+-----+
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
+-----+-----+-----+-----+-----+-----+-----+
|-----|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - ← 사용자 삭제
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bdelFlag = 1 where bno = 10

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bdelFlag = 1 where bno = 10
{executed in 1 msec}
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 10

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 10
{executed in 2 msec}
INFO : jdbc.resultsettable - ← 실제 삭제가 아니므로 데이터가 조회됩니다.
+-----+-----+-----+-----+-----+-----+-----+
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
+-----+-----+-----+-----+-----+-----+-----+
|10 |테스트제목10 |테스트내용10 |user10 |2021-02-22 13:20:07.0 |2021-02-22 13:20:08.0 |0 |0 |1 |
+-----+-----+-----+-----+-----+-----+-----+
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - MyBoardVO(bno=10, btitle=테스트제목10, bcontent=테스트내용10, bwriter=user10, bviewsCnt=0,
breplyCnt=0, bdelFlag=1, bregDate=Wed Feb 24 15:38:23 KST 2021, bmodDate=2021-02-24 15:38:24.0) ← 실제 삭제가 아니므로 데이터가 유지 됩니다.
```

#### (6) 게시물 삭제 테스트 - bdelFlag 컬럼값이 1 인 모든 행 삭제

→ MyBoardMapperTests 클래스에 다음의 메서드를 클래스 내부의 기존 내용 밑에 추가

```
//게시물 삭제 테스트 - 삭제 요청된 게시물을 전체 삭제 (관리자)
@Test
public void testDeleteAllBoard() {
    log.info("DELETE COUNT: " + myBoardMapper.deleteAllBoardSetDeleted());
    log.info(myBoardMapper.selectMyBoard(10L));
}
```

→ 저장 후 → 마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트가 수행 → 이클립스 콘솔의 로그 확인.

#### [콘솔 표시 내용]

```
INFO : jdbc.sqlonly - DELETE book_ex.tbl_myboard WHERE bDelFlag = 1
INFO : jdbc.sqltiming - DELETE book_ex.tbl_myboard WHERE bDelFlag = 1
{executed in 19 msec}
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - DELETE COUNT: 1
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 10

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 10
{executed in 93 msec}
INFO : jdbc.resultsettable -
+-----+-----+-----+-----+-----+-----+-----+
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
+-----+-----+-----+-----+-----+-----+-----+
|-----|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests -
```

☞ 앞의 테스트에서 사용자의 삭제 요청된 행(bno: 10)이 삭제되어 조회되지 않습니다.

## (7) 게시물 수정 테스트

→ MyBoardMapperTests 클래스에 다음의 메서드를 클래스 내부의 기존 내용 밑에 추가

```
//게시물 수정 테스트
@Test
public void testUpdateMyBoard() {
    MyBoardVO myBoard = new MyBoardVO();
    myBoard.setBno(1L); //실행 전 존재하는 번호인지 확인할 것
    myBoard.setTitle("수정된 제목");
    myBoard.setContent("수정된 내용");

    log.info("UPDATE COUNT: " + myBoardMapper.updateMyBoard(myBoard));
    myBoard = myBoardMapper.selectMyBoard(1L);
    System.out.println("수정된 행 값: " + myBoard.toString());
}
```

→ 저장 후 → 마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트가 수행 → 이클립스 콘솔의 로그 확인.

### [콘솔 표시 내용]

```
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET btitle = '수정된 제목', bcontent = '수정된 내용', bmodDate = DEFAULT WHERE
bno = 1

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET btitle = '수정된 제목', bcontent = '수정된 내용', bmodDate = DEFAULT WHERE
bno = 1
{executed in 14 msec}
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - UPDATE COUNT: 1
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
{executed in 80 msec}
INFO : jdbc.resultsettable -
+-----+-----+-----+-----+-----+-----+-----+-----+
| bno | btitle | bcontent | bwritter | bregdate | bmoddate | bviewscnt | breplycnt | bdelflag |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1   | 수정된 제목 | 수정된 내용 | user1 | 2021-02-24 15:38:16.0 | 2021-02-24 15:58:44.0 | 0          | 0          | 0          |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

## (8) 게시물 조회수 증가 테스트

→ MyBoardMapperTests 클래스에 다음의 메서드를 클래스 내부의 기존 내용 밑에 추가

```
//게시물 조회수 증가 테스트: 3번 수행
@Test
public void testUpdateBviewsCnt() {
    myBoardMapper.updateBviewsCnt(1L);
    System.out.println("수정된 행 값: " + myBoardMapper.selectMyBoard(1L).toString());
}
```

→ 저장 후 → 마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트가 수행 → 이클립스 콘솔의 로그 확인.

[콘솔 표시 내용 - 3번 실행 후 결과]

```
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1
INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1
{executed in 13 msec}
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
{executed in 80 msec}
INFO : jdbc.resultsettable -
+-----+-----+-----+-----+-----+-----+-----+
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
+-----+-----+-----+-----+-----+-----+-----+
```

1	수정된 제목	수정된 내용	user1	2021-02-24 15:38:16.0	2021-02-24 15:58:44.0	3	0	0	
수정된 행 값: MyBoardVO(bno=1, btitle=수정된 제목, bcontent=수정된 내용, bwriter=user1, bviewsCnt=3, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:16 KST 2021, bmodDate=2021-02-24 15:58:44.0)									

단원 [04] 끝

## [05] 비즈니스 계층(Business-Layer) 구현: MyBoardService

- 다음의 작업들이 진행됩니다.
  - 비즈니스 계층은 인터페이스와 구현클래스 형태로 MyBoardService를 구현합니다
  - 게시물-목록표시/게시물-등록/게시물-조회/게시물-수정/게시물-삭제 비즈니스 처리 메서드를 구현합니다.

☞ 사용자의 비즈니스(업무) 처리가 구현되는 계층으로 서비스 계층이라고도 합니다.

☞ 현장에서 빈번히 추가 또는 수정되는 업무를 프로그램에서 적용하기 위한 계층으로,  
비즈니스 로직이 빈번히 추가/수정되는 경우를 대응해야 하므로, 인터페이스와 구현 객체 구조로 구현합니다.

☞ 비즈니스 처리가 하나 이상의 데이터처리 작업을 포함하는 경우, 이를 비즈니스 계층에 구현합니다.

### 5-1. MyBoardService 인터페이스와 MyBoardServiceImpl 구현 클래스 생성

→ src/main/java 폴더에 구성된 com.spring5213.mypro00.service 패키지에 MyBoardService 인터페이스를 생성하고,  
다음의 내용을 작성합니다.

```
package com.spring5213.mypro00.service;

import java.util.List;

import com.spring5213.mypro00.domain.MyBoardVO;

public interface MyBoardService {

    //게시물 목록 조회 서비스1
    public List<MyBoardVO> getBoardList();

    //게시물 등록: selectKey이용
    public long registerBoard(MyBoardVO myBoard);

    //게시물 조회: by bno + 조회수 증가 고려
    public MyBoardVO getBoard(Long bno);

    //게시물 수정
    public boolean modifyBoard(MyBoardVO myBoard);

    //게시물 삭제 - bdelFlag 컬럼만 1로 수정
    public boolean setBoardDeleted(Long bno);

    //게시물 삭제 - 실제 삭제 발생
    public boolean removeBoard(Long bno);

    //게시물 삭제(관리자) - 사용자 삭제 요청된 게시물 전체 삭제: bdelFlag = 1
    public int removeAllDeletedBoard();
}
```

→ src/main/java 폴더에 구성된 com.spring5213.mypro00.service 패키지에 MyBoardServiceImpl 클래스를 생성하고, 다음의 내용을 작성합니다.

```
package com.spring5213.mypro00.service;

import java.util.List;

import org.springframework.stereotype.Service;

import com.spring5213.mypro00.domain.MyBoardVO;
import com.spring5213.mypro00.mapper.MyBoardMapper;

import lombok.AllArgsConstructor;
import lombok.extern.log4j.Log4j;

@Log4j
@Service
@AllArgsConstructor
public class MyBoardServiceImpl implements MyBoardService {

    //매퍼 인터페이스 주입: 생성자를 이용한 자동 주입
    private MyBoardMapper myBoardMapper;

    //    //MyBoardMapper를 주입받기 위한 생성자 ← @AllArgsConstructor로 대체됨
    //    public MyBoardServiceImpl(MyBoardMapper myBoardMapper) {
    //        this.myBoardMapper = myBoardMapper;
    //    }

    //게시물 목록 조회 서비스1
    @Override
    public List<MyBoardVO> getBoardList(){
        log.info("MyBoardService.getBoardList() 실행");
        return myBoardMapper.selectMyBoardList();
    }

    //게시물 등록:selectKey이용
    @Override
    public long registerBoard(MyBoardVO myBoard) {
        log.info("MyBoardService.registerBoard()에 전달된 MyBoardVO: " + myBoard);

        myBoardMapper.insertMyBoardSelectKey(myBoard);
        System.out.println("MyBoardService에서 등록된 게시물의 bno: " + myBoard.getBno());

        return myBoard.getBno();
    }

    //게시물 조회: by bno + 조회수 증가 고려
    @Override
    public MyBoardVO getBoard(Long bno) {
        log.info("MyBoardService.getBoard()에 전달된 bno: " + bno);

        //조회수 증가 후, bno 게시물 데이터 반환
        myBoardMapper.updateBviewsCnt(bno);
        return myBoardMapper.selectMyBoard(bno);
    }

    //게시물 수정 처리
    @Override
    public boolean modifyBoard(MyBoardVO myBoard){
        log.info("서비스에서의 게시물 수정 메소드(modify): " + myBoard);

        //게시물 정보 수정, 수정된 행수가 1 이면 True.
        return myBoardMapper.updateMyBoard(myBoard) == 1;
    }

    //게시물 삭제 - bdelFlag 컬럼만 1로 수정
    @Override
    public boolean setBoardDeleted(Long bno){
        log.info("MyBoardService.setBoardDeleted()에 전달된 bno: " + bno);
        return myBoardMapper.updateBdelFlag(bno) == 1;
    }
}
```

```

//게시물 삭제 - 실제 삭제 발생
@Override
public boolean removeBoard(Long bno){
    log.info("MyBoardService.removeBoard()에 전달된 bno: " + bno);
    return myBoardMapper.deleteMyBoard(bno) == 1;
}

//게시물 삭제(관리자) - 사용자 삭제 요청된 게시물 전체 삭제: bdelFlag = 1
@Override
public int removeAllDeletedBoard(){
    log.info("MyBoardService.removeAllDeletedBoard() 실행");
    //삭제된 총 행수 반환
    return myBoardMapper.deleteAllBoardSetDeleted();
}
}

```

## 5-2. MyBoardService 서비스 객체의 빈 자동 등록 구성

- ☞ MyBoardService 서비스 객체가 자동으로 실행되어 빈으로 사용되도록 설정합니다.
- ☞ 웹 애플리케이션 객체의 빈(Bean)들이 스프링의 Dispatcher Servlet의 APPLICATION 컨텍스트(ROOT 컨텍스트가 아닌)의 빈으로 생성되도록, servlet-context.xml 파일에 MyBoardService 빈들의 자동 등록 구성을 설정합니다.

→ src/main/webapp/WEB-INF/spring/appServlet/ 폴더에 있는 servlet-context.xml 파일을 열고 아래처럼, 기존 내용을 주석처리 후, 빨간색 코드를 추가 작성합니다.

```

<!-- 자동 빈 등록 -->
<!--<context:component-scan base-package="com.spring5213.mypro00" /> --><!-- 기존 내용 주석 처리 -->
<!-- 명시된 패키지의 클래스들 중, @Service 어노테이션이 적용된 클래스만 자동으로 빈으로 등록됩니다. -->
<context:component-scan base-package="com.spring5213.mypro00.service" />

```

## 5-3. MyBoardService 서비스 테스트

→ src/test/java 폴더에 com.spring5213.mypro00.service 패키지 생성

→ 생성된 패키지에 MyBoardServiceTests 클래스를 생성한 후, 다음의 내용을 작성합니다.

```

package com.spring5213.mypro00.service;

import static org.junit.Assert.assertNotNull;

import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;

```

```

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@WebAppConfiguration //테스트 시, DispatcherServlet의 servlet-context.xml 설정 구성파일(들)을 사용하기 위한 어노테이션
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({ "file:src/main/webapp/WEB-INF/spring/mybatis-context.xml",
                      "file:src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml" })
@Log4j
public class MyBoardServiceTests {

    @Setter(onMethod_ = { @Autowired })
    private MyBoardService myBoardService;

    //// JUnit은 테스트 클래스에서 생성자 주입을 사용하면 오류가 발생됩니다.(JUnit 특징)
    //// private MyBoardService myBoardService;
    //// public BoardServiceTests(MyBoardService myBoardService) {
    ////     this.myBoardService = myBoardService;
    //// } → 발생 오류: java.lang.Exception: Test class should have exactly one public zero-argument constructor
    ////
    //// 따라서, JUnit에서는 아래의 setter 주입방식을 이용해야 합니다.
    ////
    //// private MyBoardService myBoardService;
    //// @Autowired
    //// public void setMyBoardService(MyBoardService myBoardService) {
    ////     this.myBoardService = myBoardService;
    //// } ← 필드 선언 및 setter에 의한 자동 주입 코드가 @Setter(onMethod_ = { @Autowired }) 어노테이션으로 대체됨

}

```

☞ 각 테스트는 다음의 순서로 진행합니다.

- 기존 테스트 메서드 주석처리
- 테스트할 메소드를 MyBoardServiceTests 클래스의 기존 메서드 밑에 추가
- 코드 작성 뷰에 표시된 MyBoardServiceTests.java 코드에서 마우스 우측버튼 클릭 → Run As → 2 JUnit Test 클릭
- 테스트 실행 완료 시 JUnit 테스트 뷰에 녹색 진행 바가 표시됨
  - 녹색이 아닌 자주색 바가 표시되면, 테스트 실패 → 원인을 찾아 해결한 후, 다시 JUnit 테스트 수행
- 콘솔에 표시된 로그 확인

☞ 메소드 추가 시, 필요한 클래스에 대한 import 문도 포함시켰습니다. 메서드 추가 후, Ctrl + Shift + O 키를 동시에 누르면 필요한 클래스가 자동으로 import 되어 코드가 완성되므로, import는 직접 작성하지 않습니다.

☞ 위의 순서에 따라, 각 테스트를 스스로 진행한 후, 결과를 확인합니다.

문서의 콘솔 로그는 테스트 관련 빈 구성 로그와 HikariDataSource 빈 관련 로그는 생략합니다.

### (1) MyBoardService 빈 생성 테스트

☞ MyBoardServiceImpl 클래스의 빈이 생성되어 MyBoardService 유형 변수에 주입되어야 합니다.

→ 다음의 테스트 메서드를 MyBoardServiceTests 클래스에 추가합니다.

```

import static org.junit.Assert.assertNotNull;
import org.junit.Test;

```

```
//MyBoardService 빈 주입 확인 테스트
@Test
public void testMyBoardServiceExist() {

    log.info(myBoardService);
    assertNotNull(myBoardService); //MyBoardService 필드가 null 이면, AssertionError 예외 발생
}
```

☞ 실행 시, `AssertionError` 예외가 발생되지 않으면, 빈 주입이 되었다고 판단합니다.

#### [콘솔 로그 내용 일부]

```
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - com.spring5213.mypro00.service.MyBoardServiceImpl@3569edd5
```

#### (2) 게시물 전체 목록 조회 서비스 테스트

→ 다음의 테스트 메서드를 `MyBoardServiceTests` 클래스에 추가합니다.

```
//게시물 목록 조회 서비스 테스트
@Test
public void testGetBoardList() {
    //페이지 고려 안함
    myBoardService.getBoardList().forEach(myBoard -> log.info(myBoard));
}
```

☞ 조회 결과가 표시되면, 테스트 성공입니다.

#### [콘솔 로그 내용 일부]

```
//게시물 목록 조회 서비스 테스트 결과
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoardList() 실행
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno > 0 AND bdelFlag = 0 ORDER BY bno DESC

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno > 0 AND bdelFlag = 0 ORDER BY bno DESC
{executed in 82 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewsCnt |breplyCnt |bdelFlag |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|12 |매퍼 테스트-입력제목 |매퍼 테스트-입력내용 |test |2021-02-24 15:48:00.0 |2021-02-24 15:48:01.0 |0 |0 |0 |
|11 |매퍼 테스트-select key |매퍼 테스트-select key |test01 |2021-02-24 15:45:00.0 |2021-02-24 15:45:01.0 |0 |0 |0 |
|8 |테스트제목8 |테스트내용8 |user8 |2021-02-24 15:38:22.0 |2021-02-24 15:38:22.0 |0 |0 |0 |
|7 |테스트제목7 |테스트내용7 |user7 |2021-02-24 15:38:21.0 |2021-02-24 15:38:21.0 |0 |0 |0 |
|6 |테스트제목6 |테스트내용6 |user6 |2021-02-24 15:38:20.0 |2021-02-24 15:38:21.0 |0 |0 |0 |
|5 |테스트제목5 |테스트내용5 |user5 |2021-02-24 15:38:19.0 |2021-02-24 15:38:20.0 |0 |0 |0 |
|4 |테스트제목4 |테스트내용4 |user4 |2021-02-24 15:38:19.0 |2021-02-24 15:38:19.0 |0 |0 |0 |
|3 |테스트제목3 |테스트내용3 |user3 |2021-02-24 15:38:18.0 |2021-02-24 15:38:18.0 |0 |0 |0 |
|2 |테스트제목2 |테스트내용2 |user2 |2021-02-24 15:38:17.0 |2021-02-24 15:38:17.0 |0 |0 |0 |
|1 |수정된 제목 |수정된 내용 |user1 |2021-02-24 15:38:16.0 |2021-02-24 15:58:44.0 |3 |0 |0 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=12, btitle=매퍼 테스트-입력제목, bcontent=매퍼 테스트-입력내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:48:00 KST 2021, bmodDate=2021-02-24 15:48:01.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=11, btitle=매퍼 테스트-select key, bcontent=매퍼 테스트-select key, bwriter=test01, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:45:00 KST 2021, bmodDate=2021-02-24 15:45:01.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=8, btitle=테스트제목8, bcontent=테스트내용8, bwriter=user8, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:22 KST 2021, bmodDate=2021-02-24 15:38:22.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=7, btitle=테스트제목7, bcontent=테스트내용7, bwriter=user7, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:21 KST 2021, bmodDate=2021-02-24 15:38:21.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=6, btitle=테스트제목6, bcontent=테스트내용6, bwriter=user6, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:20 KST 2021, bmodDate=2021-02-24 15:38:21.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=5, btitle=테스트제목5, bcontent=테스트내용5, bwriter=user5, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:19 KST 2021, bmodDate=2021-02-24 15:38:20.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=4, btitle=테스트제목4, bcontent=테스트내용4, bwriter=user4, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:19 KST 2021, bmodDate=2021-02-24 15:38:19.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=3, btitle=테스트제목3, bcontent=테스트내용3, bwriter=user3, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:18 KST 2021, bmodDate=2021-02-24 15:38:18.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=2, btitle=테스트제목2, bcontent=테스트내용2, bwriter=user2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:17 KST 2021, bmodDate=2021-02-24 15:38:17.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=1, btitle=수정된 제목, bcontent=수정된 내용, bwriter=user1, bviewsCnt=3, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:16 KST 2021, bmodDate=2021-02-24 15:58:44.0)
```

### (3) 게시물 등록 서비스 테스트

☞ SQL 매퍼에 selectKey가 사용된 것만 사용할 것이므로, selectKey 가 있는 것만 테스트합니다.

→ 다음의 테스트 메서드를 MyBoardServiceTests 클래스에 추가합니다.

```
import com.spring5213.mypro00.domain.MyBoardVO;

//게시물 등록(selectKey 이용) 테스트
@Test
public void testRegisterBoard() {
    MyBoardVO myBoard = new MyBoardVO();
    myBoard.setBtitle("서비스 새글 입력 테스트 제목");
    myBoard.setBcontent("서비스 새글 입력 테스트 내용");
    myBoard.setBwriter("test");

    myBoardService.registerBoard(myBoard);
    log.info("등록된 게시물의 Bno: " + myBoard.getBno());
}
```

#### [콘솔 로그 내용 일부]

```
//게시물 등록: selectKey 이용 테스트 결과
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.registerBoard()에 전달된 MyBoardVO: MyBoardVO(bno=0, btitle=서비스 새글 입력 테스트 제목, bcontent=서비스 새글 입력 테스트 내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null)
INFO : jdbc.sqlonly - SELECT book_ex.seq_myboard.nextval FROM dual
INFO : jdbc.sqltiming - SELECT book_ex.seq_myboard.nextval FROM dual
{executed in 82 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|13      |
|-----|
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myboard VALUES (13, '서비스 새글 입력 테스트 제목', '서비스 새글 입력 테스트 내용', 'test', DEFAULT, DEFAULT, DEFAULT, DEFAULT)
INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myboard VALUES (13, '서비스 새글 입력 테스트 제목', '서비스 새글 입력 테스트 내용', 'test', DEFAULT, DEFAULT, DEFAULT, DEFAULT)
{executed in 10 msec}
MyBoardService에서 등록된 게시물의 bno: 13
```

### (4) 특정 게시물 수정 서비스 테스트

→ 다음의 테스트 메서드를 MyBoardServiceTests 클래스에 추가합니다.

```
//게시물 수정 테스트
@Test
public void testModifyBoard() {
    MyBoardVO myBoard = myBoardService.getBoard(1L);

    if (myBoard == null) {
        return;
    }

    myBoard.setBtitle("제목 수정:서비스 테스트");
    log.info("수정된 게시물 조회 결과(boolean): " + myBoardService.modifyBoard(myBoard));
}
```

#### [콘솔 로그 내용 일부]

```
//게시물 수정 테스트 결과
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoard()에 전달된 bno: 1
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1
INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1
```

#### (5) 특정 게시물 조회 서비스 테스트

☞ 조회 테스트를 여러 번 수행하여 조회수(bviewCnt) 값이 1씩 증가되는지도 확인합니다.

→ 다음의 테스트 메서드를 MyBoardServiceTests 클래스에 추가합니다.

```
//게시물 조회 테스트: by bno + 조회수 증가 고려
@Test
public void testGetBoard() {
    log.info(myBoardService.getBoard(1L));
}
```

### [콘솔 로그 내용 일부]

```
//게시물 조회: by bno + 조회수 증가 고려 테스트 결과: 여러 번 수행함.

INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoard()에 전달된 bno: 1
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1
{executed in 14 msec}
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
{executed in 75 msec}
INFO : jdbc.resultsettable -
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|bno |btitle      |bcontent |bwriter |bregdate           |bmoddate          |bviewscnt |breplycnt |bdelflag |
+---+-----+-----+-----+-----+-----+-----+-----+-----+
|1  |제목 수정:서비스 테스트 |수정된 내용 |user1   |2021-02-24 15:38:16.0 |2021-02-24 16:10:13.0 |7             |0         |0         |
+---+-----+-----+-----+-----+-----+-----+-----+-----+

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=1, btitle=제목 수정:서비스 테스트, bcontent=수정된 내용, bwritter=user1,
bviewsCnt=7, breplyCnnt=0, bdelfFlag=0, bregDate=Wed Feb 24 15:38:16 KST 2021, bmodDate=2021-02-24 16:10:13.0)
```

#### (6) 특정 게시물 삭제 서비스 테스트

→ 다음의 테스트 메서드를 MyBoardServiceTests 클래스에 추가합니다.

```
//특정 게시물 삭제 테스트 - 실제 삭제  
@Test  
public void testRemoveBoard() {  
    // 게시물 번호의 존재 여부를 확인하고 테스트할 것  
    log.info("서비스: 특정 게시물 삭제 테스트: " + myBoardService.removeBoard(6L));  
}
```

## [콘솔 로그 내용 일부]

//게시물 삭제 - 실제 삭제 테스트 결과

```
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.removeBoard()에 전달된 bno: 6
INFO : jdbc.sqlonly - DELETE book_ex.tbl_myboard WHERE bno = 6
INFO : jdbc.sqltiming - DELETE book_ex.tbl_myboard WHERE bno = 6
{executed in 15 msec}
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - 서비스: 특정 게시물 삭제 테스트: true ←삭제된 행이 없으면, false 가 표시됨.
```

## (7) 특정 게시물 삭제 요청 서비스 테스트

☞ bdelflag 컬럼의 값이 1로 설정되는지 확인합니다.

→ 다음의 테스트 메서드를 MyBoardServiceTests 클래스에 추가합니다.

```
//게시물 삭제 요청 처리 서비스 테스트 - bdelflag 컬럼만 1로 수정
@Test
public void testSetBoardDeleted() {
    // 게시물 번호의 존재 여부를 확인하고 테스트할 것
    log.info("수행결과: " + myBoardService.setBoardDeleted(7L));
    log.info("수행결과: " + myBoardService.setBoardDeleted(8L));
    log.info(myBoardService.getBoard(7L));
    log.info(myBoardService.getBoard(8L));
}
```

## [콘솔 로그 내용 일부]

//게시물 삭제 - bdelflag 컬럼만 1로 수정 테스트 결과

```
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.setBoardDeleted()에 전달된 bno: 7
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bdelflag = 1 where bno = 7
INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bdelflag = 1 where bno = 7
{executed in 14 msec}
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - 수행결과: true
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.setBoardDeleted()에 전달된 bno: 8
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bdelflag = 1 where bno = 8
INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bdelflag = 1 where bno = 8
{executed in 1 msec}
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - 수행결과: true
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoard()에 전달된 bno: 7
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 7
INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 7
{executed in 0 msec}
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 7
INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 7
{executed in 75 msec}
INFO : jdbc.resultsettable -
|----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewsCnt |breplyCnt |bdelflag |
|----|-----|-----|-----|-----|-----|-----|-----|
| 7  |테스트제목7 |테스트내용7 |user7 |2021-02-24 15:38:21.0 |2021-02-24 15:38:21.0 | 1        | 0        | 1      |
|----|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=7, btitle=테스트제목7, bcontent=테스트내용7, bwriter=user7, bviewsCnt=1, breplyCnt=0, bdelflag=1, bregDate=Wed Feb 24 15:38:21 KST 2021, bmodDate=2021-02-24 15:38:21.0)
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoard()에 전달된 bno: 8
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 8
INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 8
{executed in 1 msec}
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 8
INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 8
{executed in 1 msec}
INFO : jdbc.resultsettable -
|----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewsCnt |breplyCnt |bdelflag |
|----|-----|-----|-----|-----|-----|-----|-----|
| 8  |테스트제목8 |테스트내용8 |user8 |2021-02-24 15:38:22.0 |2021-02-24 15:38:22.0 | 1        | 0        | 1      |
|----|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=8, btitle=테스트제목8, bcontent=테스트내용8, bwriter=user8, bviewsCnt=1, breplyCnt=0, bdelflag=1, bregDate=Wed Feb 24 15:38:22 KST 2021, bmodDate=2021-02-24 15:38:22.0)
```

## (8) 삭제 요청된 모든 게시물 삭제 테스트

→ 다음의 테스트 메서드를 MyBoardServiceTests 클래스에 추가합니다.

```
//게시물 삭제 테스트 - 사용자 삭제 요청된 게시물(bdelFlag = 1) 전체 삭제
@Test
public void testRemoveAllDeletedBoard() {
    log.info("서비스: 삭제 설정된 모든 게시물 삭제로 삭제된 행 수: " + myBoardService.removeAllDeletedBoard());
}
```

[콘솔 로그 내용 일부]

```
//게시물 삭제 - 삭제 요청된 게시물(bdelFlag = 1) 전체 삭제 테스트 결과
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.removeAllDeletedBoard() 실행
INFO : jdbc.sqlonly - DELETE book_ex.tbl_myboard WHERE bDelFlag = 1
INFO : jdbc.sqltiming - DELETE book_ex.tbl_myboard WHERE bDelFlag = 1
{executed in 4 msec}
INFO : com.spring5213.mypro00.service.MyBoardServiceTests - 서비스: 삭제 설정된 모든 게시물 삭제로 삭제된 행 수: 2
```

단원 [05] 끝

## [06] 제어 계층 구현: Controller 구현

- 다음의 작업들이 진행됩니다.
  - 컨트롤러 클래스 생성
    - 게시물 목록 표시/게시물 등록/게시물 조회/게시물 수정/게시물 삭제 요청 처리 메서드
  - 테스트 수행

☞ 제어계층은 요청 받아들이고, 이를 처리할 비즈니스 로직이 정의된 서비스 계층으로 전달하는 역할과 서비스 계층으로부터 처리된 결과를 받아 프레젠테이션 계층으로 전달하는 역할을 합니다.

☞ 제어계층은 스프링 프레임워크 상에서 컨트롤러라는 역할이 주어진, 자바 클래스 또는 인터페이스와 구현클래스 형태로 구현됩니다. 제어계층에 구현된 클래스 이름에는 보통 Controller라는 접두어나 접미어를 포함시켜 이름을 지정합니다.

☞ 스프링 프레임워크에서 컨트롤러 역할의 클래스에 구현된 각 메소드는, 사용자의 요청에 매핑되는 요청방식과 URL 패턴이 지정됩니다.

☞ 사용자 브라우저로부터의 요청 URL은 톰캣 엔진을 거쳐, 스프링의 Dispatcher 서블릿에게 전달되고, Dispatcher 서블릿은, 스프링의 내부 요소와 URL 매핑 정보를 이용하여, 요청을 처리할 컨트롤러의 특정 메서드를 호출하여 사용자의 요청을 처리시킵니다. 컨트롤러의 메서드는, 서버에 개발된 요소를 이용하여 사용자 요청을 처리한 후, 처리결과를 DispatcherServlet에게 전달합니다. DispatcherServlet은 최종적으로 JSP를 통해 처리결과가 표시된 HTML 내용을 생성시키고, 생성된 HTML을 사용자 브라우저로 전송합니다.

☞ 따라서, 제어계층의 컨트롤러 및 그 메서드들을 구현하려면, 다음의 표와 같이, 사용자 요청에 대하여 처리를 담당하는 메소드에 매핑 URL과 요청방식을 지정해야 하며, 요청 처리 후, 결과가 표시될 페이지(즉, 이동할 페이지)의 URL에 대하여 계획해야 합니다.

게시물에 대한 요청 작업 유형	요청방식과 컨트롤러 매핑 URL	비교
목록 조회	GET /myboard/list	-
등록 페이지 호출	GET //myboard/register	목록 페이지 → 등록 페이지로 이동
등록 처리	POST /myboard/register	등록 처리(서버 저장) 후 → 목록페이지로 이동
특정 게시물 조회	GET /myboard/detail	목록 페이지 → 조회 페이지 호출(내용 표시 및 조회수 1 증가)
특정 게시물 수정삭제 페이지 호출	GET /myboard/modify	조회 페이지 → 수정-삭제 페이지 이동
특정 게시물 수정	POST /myboard/modify	수정 처리(서버 저장) 후 → 조회 페이지 이동
특정 게시물 삭제의뢰	POST /myboard/delete	삭제의뢰 처리(서버 저장) 후 → 목록 페이지 이동
특정 게시물 삭제	POST /myboard/remove	삭제 처리(서버 저장) 후 → 목록 페이지 이동

☞ 실습에서는 인터페이스와 구현클래스 형태가 아닌 단일 클래스 형태로 컨트롤러 클래스를 생성합니다.

## 6-1. MyBoardController 클래스 생성

→ src/main/java/com.spring5213.mypro00.controller 패키지에 MyBoardController 클래스를 생성하고, 다음의 내용을 작성합니다.

```
package com.spring5213.mypro00.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.spring5213.mypro00.domain.MyBoardVO;
import com.spring5213.mypro00.service.MyBoardService;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@Controller //스프링 프레임워크에서 컨트롤러 역할을 부여하는 어노테이션.
@Log4j
@RequestMapping("/myboard") //클래스에 URL 매핑 지정, 메서드의 매핑 패턴의 앞에 자동으로 추가됨
public class MyBoardController {

    @Setter(onMethod_ = @Autowired) //롬복 어노테이션을 이용하여 setter 방식으로 MyBoardService 의 빈이 주입되도록 지정
    private MyBoardService myBoardService;

    //빈 주입 방법1: 생성자 주입방법(권장), 단일 생성자만 존재해야 함.
    //주의: 기본 생성자가 존재 시에 오류 발생
    //    private MyBoardService myBoardService;
    //    public MyBoardController(MyBoardService myBoardService) {
    //        this.myBoardService = myBoardService;
    //    }
    //

    //빈 주입 방법2: @Autowired(내부적으로 setter를 이용)
    //    @Autowired
    //    private MyBoardService myBoardService;
    //

    //빈 주입 방법3: setter 주입방법: lombok에서의 @Setter(onMethod_ = @Autowired) 이용하는 방법과 동일
    //    private MyBoardService myBoardService;
    //    @Autowired
    //    public void setMyBoardService(MyBoardService myBoardService) {
    //        this.myBoardService = myBoardService;
    //    }

    //게시물 전체 목록 조회
    @GetMapping("/list")
    public void showBoardList(Model model) { //반환타입이 없음 → WEB-INF/views/myboard/list.jsp 가 결과 HTML 생성
        log.info("컨트롤러 - 게시물 목록 조회.....");
        model.addAttribute("boardList", myBoardService.getBoardList());
    }

    //게시물 등록 - 페이지 호출
    @GetMapping("/register")
    public void showBoardRegisterPage() {
        log.info("컨트롤러 - 게시물 등록 페이지 호출");
    }

    //게시물 등록 - 처리
    @PostMapping("/register")
    public String registerNewBoard(MyBoardVO myBoard, RedirectAttributes redirectAttr) {
        log.info("컨트롤러 - 게시물 등록: " + myBoard);

        long bno = myBoardService.registerBoard(myBoard);
```

```

log.info("등록된 게시물의 bno: " + bno );

redirectAttr.addFlashAttribute("result", bno);

return "redirect:/myboard/list"; //jsp 대신, 리다이렉트 방식으로 사용자 브라우저가 게시물 목록을 호출하게 함.
}

//특정 게시물 조회-수정 페이지 호출
@GetMapping={"/detail", "/modify"})
public void showBoardDetail(@RequestParam("bno") Long bno, Model model) {
    log.info("컨트롤러 - 게시물 조회-수정 페이지 호출: "+ bno);

    model.addAttribute("board", myBoardService.getBoard(bno));
    log.info("컨트롤러 - 화면으로 전달할 model: "+ model);
}

//특정 게시물 수정 처리
@PostMapping("/modify")
public String modifyBoard(MyBoardVO myBoard){
    log.info("컨트롤러 - 게시물 수정 전달된 myBoard 값: " + myBoard);

    if (myBoardService.modifyBoard(myBoard)) {
        redirectAttr.addFlashAttribute("result", "successModify");
    }
    return "redirect:/myboard/detail?bno=" + myBoard.getBno();
}

//특정 게시물 삭제 요청
@PostMapping("/delete")
public String setBoardDeleted(@RequestParam("bno") Long bno, RedirectAttributes redirectAttr){
    log.info("컨트롤러 - 게시물 삭제(bdelFlag값변경 글번호): " + bno);

    if (myBoardService.setBoardDeleted(bno)) {
        redirectAttr.addFlashAttribute("result", "successRemove");
    }
    return "redirect:/myboard/list";
}

//특정 게시물 실제 삭제
@PostMapping("/remove")
public String removeBoard(@RequestParam("bno") Long bno, RedirectAttributes redirectAttr) {
    log.info("컨트롤러 - 게시물 삭제: 삭제되는 글번호: " + bno);

    if (myBoardService.removeBoard(bno)) {
        redirectAttr.addFlashAttribute("result", "successRemove");
    }
    return "redirect:/myboard/list";
}

//게시물 삭제 - 삭제 요청된 모든 게시물 삭제
@PostMapping("/removeAll")
public String removeAllDeletedBoard(Model model, RedirectAttributes redirectAttr) {

    model.addAttribute("removedRowCnt", myBoardService.removeAllDeletedBoard());
    log.info("관리자에 의해 삭제된 총 행수: " + model.getAttribute("removedRowCnt"));

    redirectAttr.addFlashAttribute("result", "successRemoveAll");
    return "redirect:/myboard/list";
}
}

```

## 6-2. MyBoardController 객체의 빈 자동 등록 구성

☞ MyBoardController 객체가 Dispatcher Servlet의 APPLICATION 컨텍스트의 빈으로 생성되도록 설정합니다.

→ src/main/webapp/WEB-INF/spring/appServlet/ 폴더에 있는 servlet-context.xml 파일을 열고 빨간색 코드를 추가합니다.

```
<!-- 자동 빈 등록 -->
<!--<context:component-scan base-package="com.spring5213.mypro00" /> --><!-- 기존 내용 주석 처리 -->
<!-- 명시된 패키지의 클래스들 중, @Service 어노테이션이 적용된 클래스만 자동으로 빈으로 등록됩니다. -->
<context:component-scan base-package="com.spring5213.mypro00.service" />
<context:component-scan base-package="com.spring5213.mypro00.controller" />
```

## 6-3. MyBoardController 제어계층 테스트

→ src/test/java 폴더에 com.spring5213.mypro00.controller 패키지 생성

→ 생성된 패키지에 MyBoardControllerTests 클래스를 생성한 후, 다음의 내용을 작성합니다.

```
package com.spring5213.mypro00.controller;

import org.junit.Before;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration //DispatcherServlet의 구성설정파일을 사용하여 컨트롤러를 테스트하기 위해 필요한 설정
@ContextConfiguration({ "file:src/main/webapp/WEB-INF/spring/mybatis-context.xml",
                      "file:src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml" })
@Log4j
public class MyBoardControllerTests {

    //테스트 환경 구성 시작, 컨트롤러 테스트를 위해서는 WebApplicationContext 를 객체로 주입 받아야 합니다.
    @Setter(onMethod_ = { @Autowired })
    private WebApplicationContext ctx;

    private MockMvc mockMvc;

    @Before
    public void setup() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(ctx).build();
    } //테스트 환경 구성-끝

}
```

☞ 각 테스트는 다음의 순서로 진행합니다(`MyBoardService` 테스트와 동일한 순서).

- 기존 테스트 메서드 주석처리
  - 테스트할 메소드를 MyBoardControllerTests 클래스의 기존 메서드 밑에 추가
  - 코드 작성 뷰에 표시된 MyBoardControllerTests.java 코드에서 마우스 우측버튼 클릭 → Run As → 2 JUnit Test 클릭
  - 테스트 실행 완료 시 JUnit 테스트 뷰에 녹색 진행 바가 표시됨
    - 녹색이 아닌 자주색 바가 표시되면, 테스트 실패 → 원인을 찾아 해결한 후, 다시 JUnit 테스트 수행
  - 콘솔에 표시된 로그 확인

☞ 메소드 추가 시, 필요한 클래스에 대한 import 문도 포함시켰습니다. 메서드 추가 후, Ctrl + Shift + O 키를 동시에 누르면 필요한 클래스가 자동으로 import 되어 코드가 완성되므로, import는 직접 작성하지 않습니다.

☞ 위의 순서에 따라, 각 테스트를 스스로 진행한 후, 결과를 확인합니다.

문서의 콘솔 로그는 테스트 관련 빈 구성 로그와 HikariDataSource 빈 관련 로그는 생략합니다.

### (1) 게시물 전체 목록 조회 테스트

- 다음의 테스트 메서드를 MyBoardControllerTests 클래스에 추가합니다.

```
import org.junit.Test;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

//게시물 목록 조회 테스트
@Test
public void testShowBoardList() throws Exception {
    log.info(mockMvc.perform(MockMvcRequestBuilders.get("/myboard/list")
        .andReturn()
        .getModelAndView()
        .getModelMap());
}

}
```

## [콘솔 로그 내용 일부]

INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - {boardList=[MyBoardVO(bno=13, btitle=서비스 새글 입력 테스트 제목, bcontent=서비스 새글 입력 테스트 내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 16:08:09 KST 2021, bmodDate=2021-02-24 16:08:10.0), MyBoardVO(bno=12, btitle=매퍼 테스트-입력제목, bcontent=매퍼 테스트-입력내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:48:00 KST 2021, bmodDate=2021-02-24 15:48:01.0), MyBoardVO(bno=11, btitle=매퍼 테스트-select key, bcontent=매퍼 테스트-select key, bwriter=test01, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:45:00 KST 2021, bmodDate=2021-02-24 15:45:01.0), MyBoardVO(bno=5, btitle=테스트제목5, bcontent=테스트내용5, bwriter=user5, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:19 KST 2021, bmodDate=2021-02-24 15:38:20.0), MyBoardVO(bno=4, btitle=테스트제목4, bcontent=테스트내용4, bwriter=user4, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:19 KST 2021, bmodDate=2021-02-24 15:38:19.0), MyBoardVO(bno=3, btitle=테스트제목3, bcontent=테스트내용3, bwriter=user3, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:18 KST 2021, bmodDate=2021-02-24 15:38:18.0), MyBoardVO(bno=2, btitle=테스트제목2, bcontent=테스트내용2, bwriter=user2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:17 KST 2021, bmodDate=2021-02-24 15:38:17.0), MyBoardVO(bno=1, btitle=제목 수정:서비스 테스트, bcontent=수정된 내용, bwriter=user1, bviewsCnt=7, breplyCnt=0, bdelFlag=0, bregDate=Wed Feb 24 15:38:16 KST 2021, bmodDate=2021-02-24 16:10:13.0)]]}

## (2) 게시물 등록 페이지 호출 테스트

→ 다음의 테스트 메서드를 MyBoardControllerTests 클래스에 추가합니다.

```
//게시물 등록-페이지 호출 테스트
@Test
public void testShowBoardRegisterPage() throws Exception {
    log.info(mockMvc.perform(MockMvcRequestBuilders.get("/myboard/register"))
        .andReturn()
        .getModelAndView()
        .getModelMap());
}
```

### [콘솔 로그 내용 일부]

```
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 등록 페이지 호출
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - {}
```

## (3) 게시물 등록 처리 테스트

→ 다음의 테스트 메서드를 MyBoardControllerTests 클래스에 추가합니다.

```
//게시물 등록 처리 테스트
@Test
public void testRegisterNewBoard() throws Exception {
    String resultPage =
        mockMvc.perform(MockMvcRequestBuilders.post("/myboard/register")
            .param("btitle", "게시물 등록 -컨트롤러 테스트 제목")
            .param("bcontent", "게시물 등록 -컨트롤러 테스트 내용")
            .param("bwriter", "test"))
            .andReturn().getModelAndView().getViewName();
    log.info(resultPage);
}
```

### [콘솔 로그 내용 일부]

```
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 등록: MyBoardVO(bno=0, btitle=게시물 등록 -컨트롤러 테스트 제목,
bcontent=게시물 등록 -컨트롤러 테스트 내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null)
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.registerBoard()에 전달된 MyBoardVO: MyBoardVO(bno=0, btitle=게시물 등록 -
컨트롤러 테스트 제목, bcontent=게시물 등록 -컨트롤러 테스트 내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null)
INFO : jdbc.sqlonly - SELECT book_ex.seq_myboard.nextval FROM dual
INFO : jdbc.sqltiming - SELECT book_ex.seq_myboard.nextval FROM dual
{executed in 76 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|14      |
|-----|
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myboard VALUES (14, '게시물 등록 -컨트롤러 테스트 제목', '게시물 등록 -컨트롤러 테스트 내용', 'test',
DEFAULT, DEFAULT, DEFAULT, DEFAULT)
INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myboard VALUES (14, '게시물 등록 -컨트롤러 테스트 제목', '게시물 등록 -컨트롤러 테스트 내용', 'test',
DEFAULT, DEFAULT, DEFAULT, DEFAULT)
{executed in 11 msec}
MyBoardService에서 등록된 게시물의 bno: 14
INFO : com.spring5213.mypro00.controller.MyBoardController - 등록된 게시물의 bno: 14
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - redirect:/myboard/list
```

#### (4) 특정 게시물 조회-수정 페이지 호출 테스트

☞ 테스트 코드를 조금 수정하여 /myboard/detail, /myboard/modify URL을 모두 테스트합니다.

→ 다음의 테스트 메서드를 MyBoardControllerTests 클래스에 추가합니다.

```
//게시물 조회-수정 페이지 호출 테스트: /myboard/detail, /myboard/modify 2개 모두 테스트도 수행
@Test
public void testshowBoardDetail() throws Exception {
    log.info(mockMvc.perform(MockMvcRequestBuilders
        .get("/myboard/detail") // /myboard/detail로 수정한 후, 다시 테스트
        .param("bno", "1"))
        .andReturn()
        .getModelAndView()
        .getModelMap());
}
```

#### [콘솔 로그 내용 일부]

//게시물 조회 페이지 호출 테스트 결과

```
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 조회-수정 페이지 호출: 1
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoard()에 전달된 bno: 1
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1
{executed in 15 msec}
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
{executed in 94 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle      |bcontent      |bwriter |bregdate      |bmoddate      |bviewsCnt |breplyCnt |bdelflag |
|-----|-----|-----|-----|-----|-----|-----|-----|
|1   |게시물 수정-컨트롤러 테스트 제목 |게시물 수정-컨트롤러 테스트 내용 |user1   |2021-06-18 21:42:36.0 |2021-06-19 13:44:04.0 |9          |0          |0
|-----|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 화면으로 전달할 model: {board=MyBoardVO(bno=1, btitle=게시물 수정-컨트롤러 테스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=user1, bviewsCnt=9, breplyCnt=0, bdelflag=0, bregDate=Fri Jun 18 21:42:36 KST 2021, bmodDate=2021-06-19 13:44:04.0)}
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - {board=MyBoardVO(bno=1, btitle=게시물 수정-컨트롤러 테스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=user1, bviewsCnt=9, breplyCnt=0, bdelflag=0, bregDate=Fri Jun 18 21:42:36 KST 2021, bmodDate=2021-06-19 13:44:04.0), org.springframework.validation.BindingResult.board=org.springframework.validation.BeanPropertyBindingResult: 0 errors}
INFO :
```

//게시물 수정 페이지 호출 테스트 결과

```
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 조회-수정 페이지 호출: 1
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoard()에 전달된 bno: 1
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1
{executed in 15 msec}
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
{executed in 93 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle      |bcontent      |bwriter |bregdate      |bmoddate      |bviewsCnt |breplyCnt |bdelflag |
|-----|-----|-----|-----|-----|-----|-----|-----|
|1   |게시물 수정-컨트롤러 테스트 제목 |게시물 수정-컨트롤러 테스트 내용 |user1   |2021-06-18 21:42:36.0 |2021-06-19 13:44:04.0 |10         |0          |0
|-----|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 화면으로 전달할 model: {board=MyBoardVO(bno=1, btitle=게시물 수정-컨트롤러 테스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=user1, bviewsCnt=10, breplyCnt=0, bdelflag=0, bregDate=Fri Jun 18 21:42:36 KST 2021, bmodDate=2021-06-19 13:44:04.0)}
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - {board=MyBoardVO(bno=1, btitle=게시물 수정-컨트롤러 테스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=user1, bviewsCnt=10, breplyCnt=0, bdelflag=0, bregDate=Fri Jun 18 21:42:36 KST 2021, bmodDate=2021-06-19 13:44:04.0), org.springframework.validation.BindingResult.board=org.springframework.validation.BeanPropertyBindingResult: 0 errors}
INFO :
```

☞ 특정 게시물의 조회 페이지 호출 테스트 시 해당 게시물의 조회수가 1 증가되는 것은 올바른 것인지, 수정 페이지 호출 시에도 조회수가 1 증가되는 것은 잘못된 것입니다. 이를 테스트 후에 수정하겠습니다.

## (5) 특정 게시물 수정 처리 테스트

→ 다음의 테스트 메서드를 MyBoardControllerTests 클래스에 추가합니다.

```
//게시물 수정 처리
@Test
public void testModifyBoard() throws Exception {
    String resultPage = mockMvc.perform(MockMvcRequestBuilders
        .post("/myboard/modify")
        .param("bno", "1")
        .param("btitle", "게시물 수정-컨트롤러 테스트 제목")
        .param("bcontent", "게시물 수정-컨트롤러 테스트 내용")
        .param("bwriter", "test"))
        .andReturn()
        .getModelAndView()
        .getViewName();

    log.info(resultPage);
}
```

### [콘솔 로그 내용 일부]

```
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 수정 전달된 myBoard 값: MyBoardVO(bno=1, btitle=게시물 수정-컨트롤러 테스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null)
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - 서비스에서의 게시물 수정 메소드(modify): MyBoardVO(bno=1, btitle=게시물 수정-컨트롤러 테스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null)
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET btitle = '게시물 수정-컨트롤러 테스트 제목', bcontent = '게시물 수정-컨트롤러 테스트 내용', bmodDate = DEFAULT WHERE bno = 1
INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET btitle = '게시물 수정-컨트롤러 테스트 제목', bcontent = '게시물 수정-컨트롤러 테스트 내용', bmodDate = DEFAULT WHERE bno = 1
{executed in 12 msec}
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - redirect:/myboard/detail?bno=1
```

## (6) 특정 게시물 삭제 요청 테스트

☞ 테스트 메서드의 bno 값을 수정하여, 4 와 5로 두 번 테스트를 수행합니다.

→ 다음의 테스트 메서드를 MyBoardControllerTests 클래스에 추가합니다.

```
//게시물 삭제 요청(bdelflag를 1로 수정) 테스트 - bno를 "4", "5"로 두 번 수행
@Test
public void testSetBoardDeleted() throws Exception {
    String resultPage = mockMvc.perform(MockMvcRequestBuilders
        .post("/myboard/delete")
        .param("bno", "4") //5로 변경 후, 한 번 더 테스트 수행
        .param("bdelflag", "1"))

        .andReturn()
        .getModelAndView()
        .getViewName();

    log.info(resultPage);
}
```

### [콘솔 로그 내용 일부]

```
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 삭제(bdelflag값변경 글번호): 5
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.setBoardDeleted()에 전달된 bno: 5
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bdelflag = 1 where bno = 5
INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bdelflag = 1 where bno = 5
{executed in 26 msec}
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - redirect:/myboard/list
```

## (7) 특정 게시물 삭제 테스트 - 실제 삭제

→ 다음의 테스트 메서드를 MyBoardControllerTests 클래스에 추가합니다.

```
//특정 게시물 삭제 테스트 - 실제 삭제
@Test
public void testRemoveBoard() throws Exception {
    //삭제전 데이터베이스에 게시물 번호 확인할 것
    String resultPage = mockMvc.perform(MockMvcRequestBuilders.post("/myboard/remove")
        .param("bno", "3"))
        .andReturn()
        .getModelAndView()
        .getViewName();

    log.info(resultPage);
}
```

### [콘솔 로그 내용 일부]

```
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 삭제: 삭제되는 글번호: 3
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.removeBoard()에 전달된 bno: 3
INFO : jdbc.sqlonly - DELETE book_ex.tbl_myboard WHERE bno = 3

INFO : jdbc.sqltiming - DELETE book_ex.tbl_myboard WHERE bno = 3
{executed in 17 msec}
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - redirect:/myboard/list
```

## (8) 삭제 요청된 모든 게시물 삭제 테스트 - 실제 삭제

→ 다음의 테스트 메서드를 MyBoardControllerTests 클래스에 추가합니다.

```
//삭제 요청된 모든 게시물 삭제 테스트 - 실제 삭제
@Test
public void testRemoveAllDeletedBoard() throws Exception {
    String resultPage = mockMvc.perform(MockMvcRequestBuilders.post("/myboard/removeAll"))
        .andReturn()
        .getModelAndView()
        .getViewName();

    log.info(resultPage);
}
```

### [콘솔 로그 내용 일부]

```
INFO : com.spring5213.mypro00.service.MyBoardService - MyBoardService.removeAllDeletedBoard() 실행
INFO : jdbc.sqlonly - DELETE book_ex.tbl_myboard WHERE bDelFlag = 1

INFO : jdbc.sqltiming - DELETE book_ex.tbl_myboard WHERE bDelFlag = 1
{executed in 5 msec}
INFO : com.spring5213.mypro00.controller.MyBoardController - 관리자에 의해 삭제된 총 행수: 2
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - redirect:/myboard/list
```

## 6-4. MyBoardController 제어계층 테스트 시, 발견된 처리 로직 오류 수정

☞ 컨트롤러 테스트 시에, 게시물 수정 후에 조회 페이지가 호출될 때, 조회수가 1 증가되는 문제가 확인되었습니다.

컨트롤러의 게시물 수정을 처리하는 메서드가 게시물 수정 후에 조회 페이지가 호출 시에도 조회수를 1 증가 시키는 동일한 서비스의 메서드를 호출하기 때문에 발생된 문제입니다.

☞ MyBoardService에 조회수 증가가 제외된 조회 페이지 호출 메서드를 하나 추가한 후, 컨트롤러의 게시물 수정 처리 메서드를 수정합니다.

→ src/main/java/com.spring5213.mypro00.service.MyBoardService.java 인터페이스에 다음의 메서드(빨간색 코드)를 추가합니다.

```
//게시물 조회: 게시물 목록 페이지 -> 게시물 조회 페이지 호출(by bno), 조회수 +1 증가
public MyBoardVO getBoard(Long bno);

//게시물 조회: 게시물 조회 페이지 -> 게시물 수정 페이지 호출(by bno), 조회수 변화 없음
//게시물 조회: 게시물 수정 후 -> 게시물 조회 페이지 호출(by bno), 조회수 증가 없음
public MyBoardVO getBoardDetailModify(Long bno);
```

→ src/main/java/com.spring5213.mypro00.service.MyBoardServiceImpl.java 구현클래스에 다음의 빨간색 메서드를 추가합니다.

```
//게시물 조회: 게시물 목록 페이지 -> 게시물 조회 페이지 호출(by bno), 조회수 +1 증가
@Override
public MyBoardVO getBoard(Long bno) {
    log.info("MyBoardService.getBoard()에 전달된 bno: " + bno);

    //조회수 증가 후, bno 게시물 데이터 반환
    myBoardMapper.updateBviewsCnt(bno);
    return myBoardMapper.selectMyBoard(bno);
}

//게시물 조회: 게시물 조회 페이지 -> 게시물 수정 페이지 호출(by bno), 조회수 증가 없음
//게시물 조회: 게시물 수정 후 -> 게시물 조회 페이지 호출(by bno), 조회수 증가 없음
@Override
public MyBoardVO getBoardDetailModify(Long bno) {
    log.info("MyBoardService.getBoard()에 전달된 bno: " + bno);
    return myBoardMapper.selectMyBoard(bno);
}
```

→ src/main/java/com.spring5213.mypro00.controller.MyBoardController.java 클래스에 다음의 빨간색 코드를 추가 및 수정합니다.

```
//특정 게시물 조회 페이지 호출: 목록페이지에서 호출
@GetMapping("/detail")
public void showBoardDetail(@RequestParam("bno") Long bno, Model model) {
    log.info("컨트롤러 - 게시물 조회 페이지 호출: " + bno);

    model.addAttribute("board", myBoardService.getBoard(bno));
    log.info("컨트롤러 - 화면으로 전달할 model: " + model);
}

//게시물 조회페이지 -> 수정페이지 호출(/modify)
@GetMapping("/modify")
public void showBoardModify(@RequestParam("bno") Long bno, Model model) {
    log.info("컨트롤러 - 게시물 수정 페이지 호출: " + bno);

    model.addAttribute("board", myBoardService.getBoardDetailModify(bno));
    log.info("컨트롤러 - 화면으로 전달할 model: " + model);
}
```

```
//게시물 수정 후 -> 조회페이지 호출 시(/detailmod)
@GetMapping("/detailmod")
public String showBoardDetailMod(@RequestParam("bno") Long bno, Model model) {
    log.info("컨트롤러 - 게시물 수정 페이지 호출: " + bno);

    model.addAttribute("board", myBoardService.getBoardDetailModify(bno));
    log.info("컨트롤러 - 화면으로 전달할 model: " + model);

    return "myboard/detail" ;
}

//특정 게시물 수정 처리
@PostMapping("/modify")
public String modifyBoard(MyBoardVO myBoard, RedirectAttributes redirectAttr){
    log.info("컨트롤러 - 게시물 수정 전달된 myBoard 값: " + myBoard);

    if (myBoardService.modifyBoard(myBoard)) {
        redirectAttr.addFlashAttribute("result", "successModify");
    }

    return "redirect:/myboard/detailmod?bno=" + myBoard.getBno();
}
```

→ 특정 게시물 조회-수정 페이지 호출 테스트를 다시 수행합니다.

→ src/test/java/com.spring5213.mypro00.controller.MyBoardControllerTests.java 파일을 코드 작성부에 표시한 후,  
다른 모든 메서드는 주석 처리 후, testshowBoardDetail() 메서드의 URL을 수정하면서, /myboard/detail,  
/myboard/modify, /myboard/detailmod URL 순서로 각각 테스트합니다.

```
//게시물 조회-수정 페이지 호출 테스트: /myboard/detail, /myboard/modify, /myboard/detailmod 각각 테스트
@Test
public void testshowBoardDetail() throws Exception {
    log.info(mockMvc.perform(MockMvcRequestBuilders
        .get("/myboard/detail")) // /myboard/detail, /myboard/modify, /myboard/detailmod 수정
        .param("bno", "1"))
        .andReturn()
        .getModelAndView()
        .getModelMap());
}
```

☞ 예상, /myboard/detail 요청 시, 조회수 1 증가,

/myboard/modify, /myboard/detailmod 요청 시에는, /myboard/detail 요청 시의 조회수가 동일하게 표시되어야 합니다.

## [콘솔 로그 내용 일부]

```
//게시물 조회 페이지 호출 테스트 결과(/myboard/detail 요청 결과)

INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 조회 페이지 호출: 1
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoard()에 전달된 bno: 1
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1
{executed in 15 msec}
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
{executed in 94 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle      |bcontent      |bwritter     |bregdate      |bmoddate      |bviewscnt    |breplycnt    |bdelflag   |
|-----|-----|-----|-----|-----|-----|-----|-----|
|1  |게시물 수정-컨트롤러 테스트 제목 |게시물 수정-컨트롤러 테스트 내용 |user1       |2021-06-18 21:42:36.0 |2021-06-19 13:44:04.0 |11           |0           |
|-----|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 화면으로 전달할 model: {board=MvBoardVO(bno=1, btitle=게시물 수정-컨트롤러 테스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwritter=user1, bregdate=2021-06-18 21:42:36.0, bmoddate=2021-06-19 13:44:04.0, bviewscnt=11, breplycnt=0, bdelflag=0)}
```

```

테스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=user1, bviewsCnt=9, breplyCnt=0, bdelFlag=0, bregDate=Fri Jun 18 21:42:36 KST 2021,
bmodDate=2021-06-19 13:44:04.0)}
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - {board=MyBoardVO(bno=1, bttitle=게시물 수정-컨트롤러 테스트 제목, bcontent=게시물
수정-컨트롤러 테스트 내용, bwriter=user1, bviewsCnt=9, breplyCnt=0, bdelFlag=0, bregDate=Fri Jun 18 21:42:36 KST 2021, bmodDate=2021-06-19
13:44:04.0), org.springframework.validation.BindingResult.board=org.springframework.validation.BeanPropertyBindingResult: 0 errors}
INFO

//게시물 조회 페이지 호출 테스트 결과(/myboard/modify 요청 결과)
//게시물 수정 후, 조회 페이지 호출 테스트 결과(/myboard/detailmod 요청 결과, 동일)
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 수정 페이지 호출: 1
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoard()에 전달된 bno: 1
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bviewsCnt = bviewsCnt + 1 WHERE bno = 1
{executed in 15 msec}
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myboard WHERE bno = 1
{executed in 93 msec}
INFO : jdbc.resultsettable -
+-----+-----+-----+-----+-----+-----+-----+-----+
|bno |bttitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
+-----+-----+-----+-----+-----+-----+-----+-----+
|1 |게시물 수정-컨트롤러 테스트 제목 |게시물 수정-컨트롤러 테스트 내용 |user1 |2021-06-18 21:42:36.0 |2021-06-19 13:44:04.0 |11 |0 |
+-----+-----+-----+-----+-----+-----+-----+-----+
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 화면으로 전달할 model: {board=MyBoardVO(bno=1, bttitle=게시물 수정-컨트롤러
테스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=user1, bviewsCnt=10, breplyCnt=0, bdelFlag=0, bregDate=Fri Jun 18 21:42:36 KST 2021,
bmodDate=2021-06-19 13:44:04.0)}
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - {board=MyBoardVO(bno=1, bttitle=게시물 수정-컨트롤러 테스트 제목, bcontent=게시물
수정-컨트롤러 테스트 내용, bwriter=user1, bviewsCnt=10, breplyCnt=0, bdelFlag=0, bregDate=Fri Jun 18 21:42:36 KST 2021, bmodDate=2021-06-19
13:44:04.0), org.springframework.validation.BindingResult.board=org.springframework.validation.BeanPropertyBindingResult: 0 errors}
INFO :

```

→ 특정 게시물 수정 처리도 테스트합니다.

```

//게시물 수정 처리
@Test
public void testModifyBoard() throws Exception {
    String resultPage = mockMvc.perform(MockMvcRequestBuilders
        .post("/myboard/modify")
        .param("bno", "1")
        .param("bttitle", "게시물 수정-컨트롤러 테스트 제목")
        .param("bcontent", "게시물 수정-컨트롤러 테스트 내용")
        .param("bwriter", "test"))
        .andReturn()
        .getModelAndView()
        .getViewName();

    log.info(resultPage);
}

```

[콘솔 로그 내용 일부]

```

INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 수정 전달된 myBoard 값: MyBoardVO(bno=1, bttitle=게시물 수정-컨트롤러 테
스트 제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null)
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - 서비스에서의 게시물 수정 메소드(modify): MyBoardVO(bno=1, bttitle=게시물 수정-컨트롤러 테스트
제목, bcontent=게시물 수정-컨트롤러 테스트 내용, bwriter=test, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null)
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bttitle = '게시물 수정-컨트롤러 테스트 제목', bcontent = '게시물 수정-컨트롤러 테스트 내용',
bmodDate = DEFAULT WHERE bno = 1

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bttitle = '게시물 수정-컨트롤러 테스트 제목', bcontent = '게시물 수정-컨트롤러 테스트 내용',
bmodDate = DEFAULT WHERE bno = 1
{executed in 12 msec}
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 수정 처리 결과(boolean): true
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - redirect:/myboard/detailmod?bno=1

```

단원 [06] 끝

## [07] 프레젠테이션 계층 구현: 화면 처리(JSP-뷰 페이지)

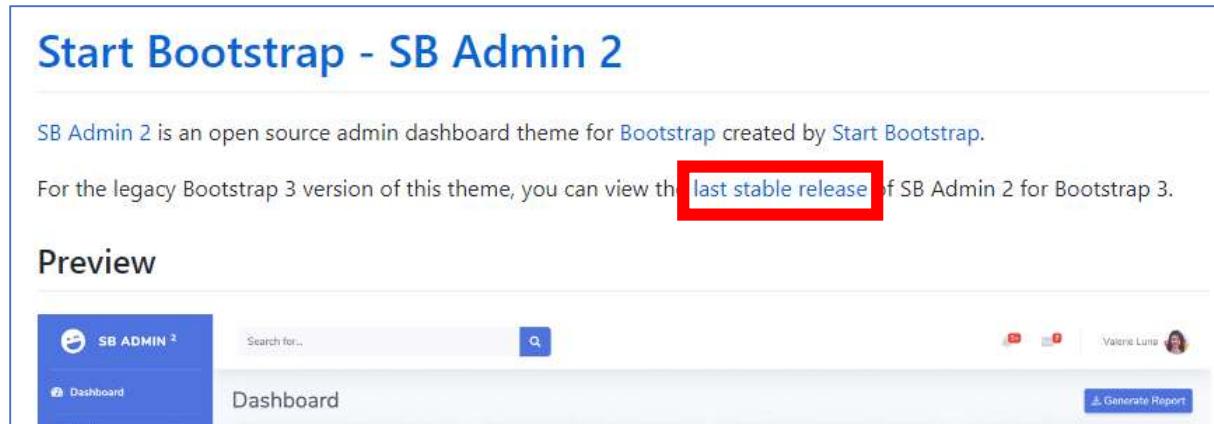
- 다음의 작업들이 진행됩니다.
    - 공통 디자인 Bootstrap Template 구성
    - header/footer JSP 파일을 구성(각 JSP 파일에서 @include 지시자를 이용하여 설정)
    - JSP 페이지 생성: 게시물-목록표시/게시물-등록/게시물-세부정보 표시/게시물-수정 및 삭제
    - 기본 브라우저 화면에서의 페이지 이동 구성:
      - 게시물 목록 페이지 → 게시물 등록 페이지 → 게시물 목록 페이지
      - 게시물 목록 페이지 → 특정 게시물 세부 정보 페이지 → 특정 게시물 수정-삭제 페이지
        - 게시물 수정 후 → 동일 게시물 조회 페이지
        - 게시물 삭제 후 → 게시물 목록 페이지
      - 게시물 등록 페이지 → 등록 취소 시 → 게시물 목록 페이지
      - 특정 게시물 수정-삭제 페이지 → 취소 시 → 동일 게시물 조회 페이지
- ☞ 프레젠테이션 계층에서의 구현 작업은 사용자의 웹 브라우저에 표시되는 화면을 생성하는 JSP-페이지들을 서버에 생성하는 작업입니다.
- ☞ 사용자가 웹 브라우저에 표시된 화면에서, 원하는 동작 실행이나 페이지 이동을 위해, 표시된 버튼이나 링크를 클릭했을 때, 처리를 담당하는 서버 상의 프로그램이 호출되도록 요청하는 기능을 구현해야 합니다.
- ☞ 화면에서 처리되는 기능은, 주로 웹 브라우저에서 실행될 수 있는 JavaScript나 jQuery를 이용하여 HTML의 <script>와 </script> 태그 사이에 작성되거나, HTML의 태그들의 속성(form 태그의 경우)에서 정의됩니다.

### 7-1. BootStrap을 이용하여 무료 디자인을 이용한 기본 화면 구성 준비

☞ BootStrap 4를 이용하는 Template도 있지만, plugin 들 때문에 개발 시 작성되는 Javascript나 jQuery 코드가 제대로 작동되지 않는 문제가 있어서, 문서의 실습에서는 BootStrap 3을 이용하는 최신 버전의 SB-Admin2를 사용합니다.

→ 웹-브라우저에서 <https://github.com/StartBootstrap/startbootstrap-sb-admin-2> 사이트 이동.

→ 페이지를 아래로 스크롤하여 아래의 부분으로 이동.



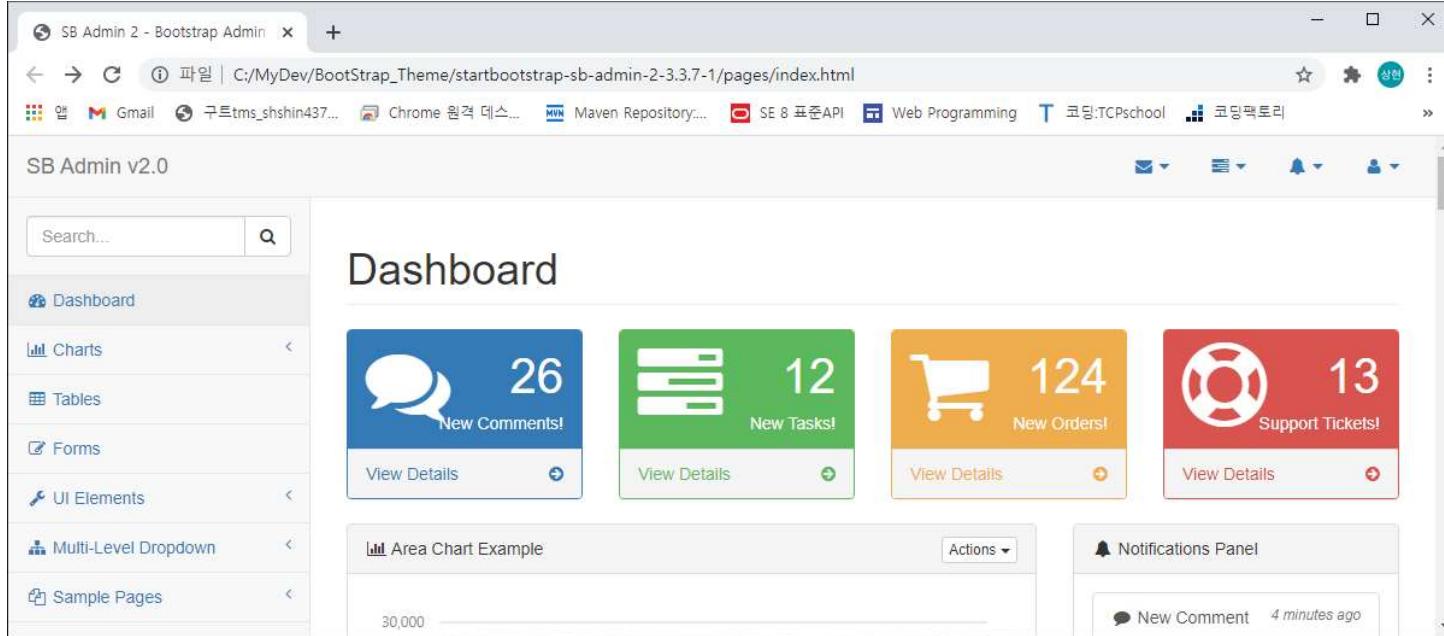
The screenshot shows the homepage of the Start Bootstrap - SB Admin 2 website. At the top, it says "Start Bootstrap - SB Admin 2". Below that, there's a brief description: "SB Admin 2 is an open source admin dashboard theme for Bootstrap created by Start Bootstrap." Underneath, it says "For the legacy Bootstrap 3 version of this theme, you can view the [last stable release](#) of SB Admin 2 for Bootstrap 3." A red box highlights the "last stable release" link. Below this, there's a "Preview" section showing a screenshot of the admin dashboard with a sidebar menu, a search bar, and various dashboard metrics. The sidebar menu includes "Dashboard" and "Generate Report".

→ 표시된 부분에 있는 [last stable release](#) 링크를 클릭 → 다운로드 페이지로 이동.

→ 이동한 페이지에서 [Source code\(zip\)](#) 링크를 클릭 → Bootstrap-3 기반 Sb Admin 2 Template을 다운로드.

→ 다운로드 된 startbootstrap-sb-admin-2-3.3.7-1.zip 파일을 여기에 풀기 옵션으로 압축 해제.

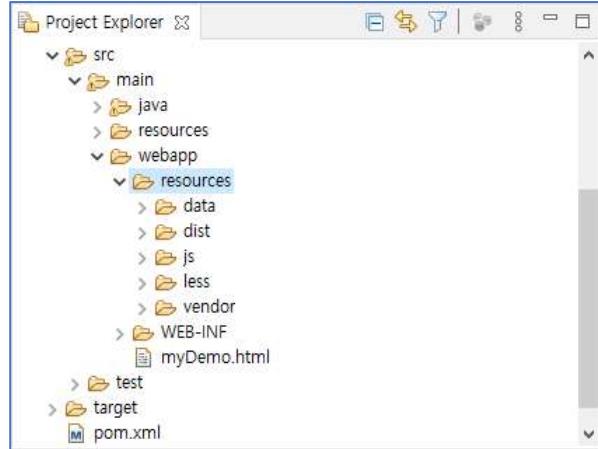
→ 압축해제 된 폴더의 index.html 클릭 → 웹 브라우저에서 화면 디자인 구성 확인.



☞ JSP 페이지 작성 시에, 압축해제 된 원본 startbootstrap-sb-admin-2-3.3.7-1 폴더의 pages 폴더에 있는 HTML 문서들을 먼저 실행하여 웹 브라우저에서 디자인을 확인 후, 해당 HTML 문서의 코드를 복사해서 JSP 파일에 붙여 넣은 후에 적절히 수정해 사용하면, 일관된 페이지 디자인 구성이 유지되며 작업이 쉬워집니다.

→ startbootstrap-sb-admin-2-3.3.7-1 폴더의 pages 폴더를 제외(경고가 너무 많이 발생됩니다)한, 모든 폴더들만 복사하여

→ mypro00 프로젝트의 src/main/webapp/resources 폴더에 붙여넣기합니다.



## 7-2. header/footer JSP-페이지 생성 및 include JSP �렉티브 태그를 이용한 header/footer JSP-페이지 적용

☞ 작성되는 여러 JSP 페이지에서, 공통된 HTML 부분(상단 메뉴 바, 좌측 사이드바 등)을 header/footer JSP-페이지로 작성하고, 이를 <%@include%> JSP �렉티브 태그를 이용하여 각각의 JSP페이지에 포함시켜, 일관된 레이아웃 디자인이 적용되도록 구성합니다.

☞ 이 실습은 게시물 목록이 표시되는 JSP 페이지를 작성하면서 수행합니다.

### (1) myboard 폴더와 myinclude 폴더를 생성

→ Project Explorer 뷰에서 src/main/webapp/WEB-INF/views 폴더 선택 → 마우스 오른쪽 버튼 클릭 → New → Folder 클릭

→ 위에 표시된 경로명이 mypro00/src/main/webapp/WEB-INF/views 인지 확인

→ Folder name 입력란에 myboard 입력 후 → Finish 클릭

→ 동일한 방법으로 myinclude 폴더도 생성.

### (2) myheader.jsp, myfooter.jsp 및 list.jsp 파일 생성

→ src/main/webapp/WEB-INF/views/myboard 폴더 선택 → 마우스 오른쪽 버튼 클릭 → New → JSP File 클릭

→ 위에 표시된 경로명이 mypro00/src/main/webapp/WEB-INF/views/myboard 인지 확인

→ File name 입력란에 list.jsp 입력 후 → Finish 클릭 ➡ 앞으로 게시물 목록 JSP 페이지로 사용.

→ 같은 방법으로 myinclude 폴더에 myheader.jsp 와 myfooter.jsp 파일 생성 ➡ 앞으로 각각 header/footer JSP 페이지로 사용

### (3) myheader.jsp, myfooter.jsp 및 list.jsp 파일 내용 작성

➡ 생성된 파일들(list.jsp, myheader.jsp, myfooter.jsp)을 다음의 지시대로 내용 추가 및 수정합니다(작업이 많습니다).

→ <!DOCTYPE html> 부터 </html>까지를 모두 삭제 후, 아래의 빨간색 코드를 모두 추가

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" /><!-- 컨텍스트 패스 변수 선언 -->
```

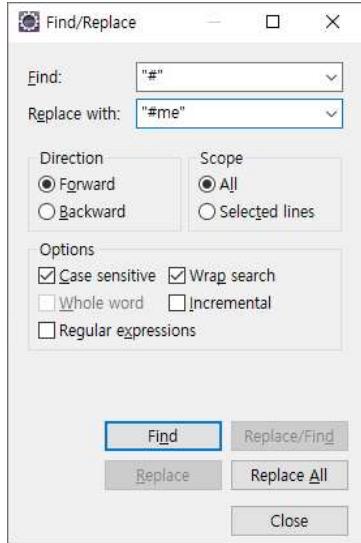
→ 압축해제 된 bootstrap Template이 있는 디렉토리로 이동 → pages 폴더의 tables.html 파일을 워드패드나 이클립스에서 오픈

→ 내용 전체를 복사 → list.jsp 파일의 기존 내용(위에서 작성된 내용) 밑에 붙여넣기

→ 붙여넣기 된 내용에서, <body>를 찾아 <body id="me">로 수정.

→ Ctrl키와 F키를 눌러, 이클립스의 Find/Replace를 실행

→ "#" 검색(href 속성이 "#"로 설정된 모든 <a> 태그를 하나씩 찾아서) → 검색된 href 속성의 "#" 을 "#me"로 수정



☞ 변경 이유: Bootstrap Template의 기능 정상 작동(사이드바 펼침 방지)과 2번의 서버 호출 발생 방지.

→ list.jsp 파일에서 이클립스의 Find/Replace 창을 이용 page-wrapper를 검색 → <div id="page-wrapper"> 줄 표시됨.

→ 검색으로 찾아낸 <div id="page-wrapper"> 줄의 윗줄(</nav>)부터 ~ 제일 위의 <!DOCTYPE html> 까지 잘라내기.

→ 잘라낸 내용을 myheader.jsp 파일의 기존 내용 밑에 붙여넣기.

→ list.jsp 파일에서 다시 page-wrapper를 검색 → </div><!-- /#page-wrapper --> 줄 표시됨(page-wrapper div의 종료 태그).

→ 검색으로 찾아낸 </div><!-- /#page-wrapper --> 줄의 다음 줄(</div><!-- /#wrapper -->)부터 마지막까지 잘라내기

→ 잘라낸 내용을 myfooter.jsp 파일의 기존 내용 밑에 붙여넣기.

→ myheader.jsp 파일 상단의 <head>와 </head> 사이에 있는 <link> 태그들의 href 속성값에서

시작부분에 있는 ../ 를 \${contextPath}/resources/ 로 수정. ← 참조 경로를 프로젝트의 경로 기준으로 변경

→ myfooter.jsp 파일의 모든 <script> 태그의 src 속성값에서

시작부분에 있는 ../ 를 \${contextPath}/resources/ 로 수정. ← 참조 경로를 프로젝트의 경로 기준으로 변경

→ myfooter.jsp 파일에 수정된 값으로 변경된 src 속성이 있는 모든 <script> 요소들을 잘라내기

→ myheader.jsp 파일의 </head> 종료태그 위에 붙여넣기.

→ 붙여넣기 된 내용 중, 기존 jQuery 라이브러리 주소만 삭제한 후, 3.x 의 최신 버전의 CDN 주소로 변경

☞ 문서에서는 <https://developers.google.com/speed/libraries#jquery> 사이트를 이용하여 아래의 CDN으로 수정함

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

☞ 이 후의 실습에서 작성되는 jQuery 코드가 정상적으로 동작됩니다.

→ list.jsp 파일에서, <div id="page-wrapper"> 줄 위에 <%@ include file="..../myinclude/myheader.jsp" %> 를 입력하고,

제일 밑에는 <%@ include file="..../myinclude/myfooter.jsp" %> 를 입력.

→ 다음의 코드를 참고하여 myheader.jsp, myfooter.jsp 및 list.jsp, 파일을 수정합니다.

☞ 빨간색으로 앞에서 수정한 부분과 추가로 변경할 부분(이탤릭체로 굵게 표시)을 표시했으니 주의해서 수정합니다.

→ src/main/webapp/WEB-INF/views/myinclude/myheader.jsp 파일 추가 수정

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<!DOCTYPE html>
<html lang="ko"> ← ko로 수정

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content=""> ← 삭제
    <meta name="author" content=""> ← 삭제

    <title>My Admin Board</title> ← 내용 수정

    <!-- Bootstrap Core CSS -->
    <link href="${contextPath}/resources/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">

    <!-- MetisMenu CSS -->
    <link href="${contextPath}/resources/vendor/metisMenu/metisMenu.min.css" rel="stylesheet">

    <!-- DataTables CSS --><!-- ← 주석처리
    <link href="${contextPath}/resources/vendor/datatables-plugins/dataTables.bootstrap.css" rel="stylesheet">-->

    <!-- DataTables Responsive CSS --><!-- ← 주석처리
    <link href="${contextPath}/resources/vendor/datatables-responsive/dataTables.responsive.css" rel="stylesheet">-->

    <!-- Custom CSS -->
    <link href="${contextPath}/resources/dist/css/sb-admin-2.css" rel="stylesheet">

    <!-- Custom Fonts -->
    <link href="${contextPath}/resources/vendor/font-awesome/css/font-awesome.min.css" rel="stylesheet" type="text/css">

    <!-- HTML5 Shim and Respond.js IE8 support of HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
        <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
        <script src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js"></script>
    <![endif]-->

    <!-- jQuery -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <!-- Bootstrap Core JavaScript -->
    <script src="${contextPath}/resources/vendor/bootstrap/js/bootstrap.min.js"></script>

    <!-- Metis Menu Plugin JavaScript -->
    <script src="${contextPath}/resources/vendor/metisMenu/metisMenu.min.js"></script>

    <!-- DataTables JavaScript --><!-- ← 3줄 모두 주석처리
    <script src="${contextPath}/resources/vendor/datatables/js/jquery.dataTables.min.js"></script>
    <script src="${contextPath}/resources/vendor/datatables-plugins/dataTables.bootstrap.min.js"></script>
    <script src="${contextPath}/resources/vendor/datatables-responsive/dataTables.responsive.js"></script>-->

    <!-- Custom Theme JavaScript -->
    <script src="${contextPath}/resources/dist/js/sb-admin-2.js"></script>
```

```

</head>

<body id="me">
    <div id="wrapper"><%-- 경고 표시는 종료태그가 myheader.jsp 파일에 없기 때문(footer.jsp에 있음, 실행은 문제없음)--%
        <%-- Navigation --%>
        <nav class="navbar navbar-default navbar-static-top" role="navigation" style="margin-bottom: 0">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="index.html">My Admin Board</a> ←내용 수정
            ...(생략)... <%-- 생략된 내용에 있는 모든 <a href="#">가 <a href="#me"> 로 수정됨 --%>
        </div>
    </nav>

```

→ src/main/webapp/WEB-INF/views/myinclude/myfooter.jsp 파일 추가 수정

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

</div><!-- /#wrapper -->

<!-- Page-Level Demo Scripts - Tables - Use for reference -->
<%-- 기본 설정되어 있는데, 문서에서는 아래처럼 주석처리 함 --%>

<%-- ← 아래 <script> 요소 전체 주석처리
<script>
$(document).ready(function() {
    //Bootstrap의 컬럼 정렬, 페이징 기능 등 DataTable 반응기능 활성화
    $('#dataTables-example').DataTable({
        responsive: true
    });
}</script>--%>
</body>
</html>

```

→ src/main/webapp/WEB-INF/views/myboard/list.jsp 파일 일부 추가 수정

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<%@ include file="../myinclude/myheader.jsp" %>

<div id="page-wrapper">
    <div class="row">
        <div class="col-lg-12">
            <h3 class="page-header">Board - List</h3> ←태그 및 내용 수정
        </div><!-- /.col-lg-12 -->
    </div><!-- /.row -->

    <div class="row">

```

```

<div class="col-lg-12">
    <div class="panel panel-default">
        <div class="panel-heading"><h4>게시글 목록</h4></div> ← 내용 수정
        <div class="panel-body">
            <table class="table table-striped table-bordered table-hover" id="dataTables-example"
                style="width:100%;text-align:center;" > ← width="100%"를 style 속성으로 교체
            ...생략...
            </table><!-- /.table-responsive -->
        </div><!-- /.panel-body -->
    </div><!-- /.panel -->
</div><!-- /.col-lg-12 -->
</div><!-- /.row -->
</div><!-- #page-wrapper -->

<%@ include file="../myinclude/myfooter.jsp" %>

```

### 7-3. 게시물 목록 JSP 페이지(list.jsp) 구성 - 데이터베이스의 데이터 표시

- list.jsp 파일에 의해 생성되는 HTML에 컨트롤러로부터 전달받은 게시물 목록 데이터가 테이블 형태로 표시될 수 있도록 다음의 지시대로 list.jsp 파일을 수정합니다.
- 현재 list.jsp 파일에는 여러 개의 테이블이 있으며, 각 <table>~</table>은 <div class="row"> 와 </div><!-- /.row --> 사이에 존재합니다.
- id 가 dataTables-example 인 <table> 요소만 남기고 다른 테이블이 포함된 <div class="row"> 요소를 삭제합니다.

→ 코드 작성 뷰에 src/main/webapp/WEB-INF/views/myboard/list.jsp 파일을 오픈 후, 다음의 지시대로 수정

- id가 dataTables-example인 <table>~</table>이 포함된 <div class="row"> 와 </div><!-- /.row -->만 남기고,
- 다른 테이블이 포함된 <div class="row"> 와 </div><!-- /.row --> 들은 모두 삭제
- <div class="well"> 와 </div>도 삭제
- id가 dataTables-example인 <table>~</table>에서 id 속성만 삭제(id="dataTables-example").  
<tbody>와 </tbody> 사이에 있는 모든 <tr>~</tr> 태그들을 삭제.

- 다음은 여기까지 수정을 마친 list.jsp 파일의 상태입니다(참고만 합니다).

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<%@ include file="../myinclude/myheader.jsp" %>

<div id="page-wrapper">
    <div class="row">
        <div class="col-lg-12">
            <h3 class="page-header">Board - List</h3>

```

```

</div><!-- /.col-lg-12 -->
</div><!-- /.row -->

<div class="row">
  <div class="col-lg-12">
    <div class="panel panel-default">
      <div class="panel-heading"><h4>게시글 목록</h4></div>
      <div class="panel-body">
        <table class="table table-striped table-bordered table-hover"
              style="width:100%;text-align:center;" >

          <thead>
            <tr>
              <th>Rendering engine</th>
              <th>Browser</th>
              <th>Platform(s)</th>
              <th>Engine version</th>
              <th>CSS grade</th>
            </tr>
          </thead>
          <tbody>

        </tbody>
      </table><!-- /.table-responsive -->
    </div><!-- /.panel-body -->
  </div><!-- /.panel -->
</div><!-- /.col-lg-12 -->
</div><!-- /.row -->
</div><!--/#page-wrapper -->

<%@ include file="../myinclude/myfooter.jsp" %>

```

→ <table> ~</table> 사이에 다음의 빨간색 코드를 작성합니다(오타, 대소문자 주의!!!).

```

<table class="table table-striped table-bordered table-hover"
       style="width:100%;text-align:center;" >
  <thead>
    <tr>
      <th style="text-align:center;">번호</th>
      <th style="text-align:center;">제목</th>
      <th style="text-align:center;">작성자</th>
      <th style="text-align:center;">작성일</th>
      <th style="text-align:center;">수정일</th>
      <th style="text-align:center;">조회수</th>
    </tr>
  </thead>
  <tbody>

<c:forEach items="${boardList}" var="board"><%-- 컨트롤러에서 보낸 목록객체 이름: boardList --%>
  <c:if test="${board.bdelFlag == 1}">
    <tr style="background-color:Moccasin; text-align:center">
      <td><c:out value="${board.bno}" /></td>
      <td colspan="5"><em>작성자에 의하여 삭제된 게시글입니다.</em></td>
    </tr>
  </c:if>
  <c:if test="${board.bdelFlag == 0}">
    <tr>
      <td><c:out value="${board.bno}" /></td>
      <td style="text-align:left;" ><c:out value="${board.btitle}" /></td>
      <td><c:out value="${board.bwriter}" /></td>
      <td><fmt:formatDate pattern="yyyy/MM/dd" value="${board.bregDate}" /><br>
          ${board.bregDate}</td>
      <td><fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="${board.bmodDate}" /><br>
          ${board.bmodDate}</td>
    </tr>
  </c:if>

```

```

<td>${board.bviewsCnt}</td>
</tr>
</c:if>
</c:forEach>

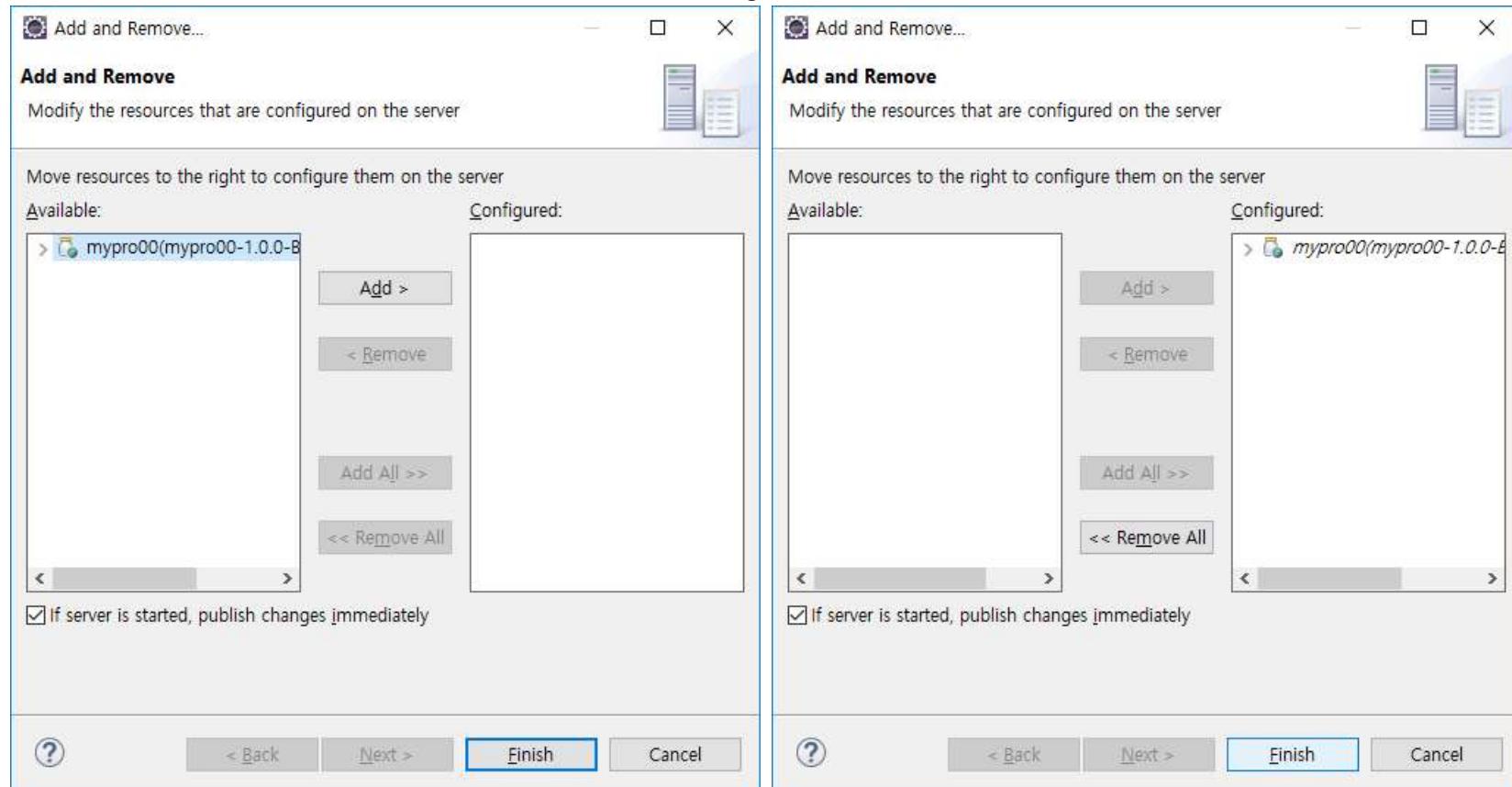
</tbody>
</table><!-- /.table-responsive -->

```

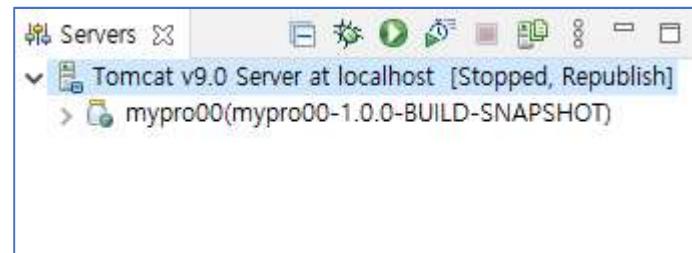
#### 7-4. 톰캣 서버와 mypro00 프로젝트 연동

→ Servers 뷰에 있는 Tomcat v9.0 Server at localhost 항목을 마우스 오른쪽 버튼 클릭 → Add and Remove 항목 선택

→ Available 부분의 프로젝트를 클릭 → Add > 클릭 → Configured로 이동됨 확인 → Finish 클릭



→ Servers 뷰 → Tomcat v9.0 Server at localhost 앞에 [>] 를 클릭 → 프로젝트 항목이 표시되는지 확인



→ 위의 그림에서 녹색 버튼 클릭 → 프로젝트가 연동된 상태의 톰캣서버가 이를립스에 의해서 기동됩니다.

콘솔 창에 톰캣 기동 로그가 표시됩니다.

## [테스트]

☞ 톰캣 기동 후, 웹 브라우저에서 지금까지의 구성 내용을 확인합니다. 설명에서 "표시"는 웹 브라우저에 표시되는 것을 의미합니다.

→ 웹 브라우저에서 `http://localhost:8080/mypro00/myboard/list` URL로 접속 → 다음의 내용이 표시되어야 함.

The screenshot shows a web browser window titled 'My Admin Board'. The address bar displays 'localhost:8080/mypro00/myboard/list'. The main content area is titled 'Board - List' and contains a table with the following data:

번호	제목	작성자	작성일	수정일	조회수
16	게시물 등록 - 컨트롤러 테스트 제목	test	2021/06/19 Sat Jun 19 13:37:18 KST 2021	2021/06/19 13:37:18.0	0
15	서비스 새글 입력 테스트 제목	test	2021/06/19 Sat Jun 19 11:51:21 KST 2021	2021/06/19 11:51:22.0	0
14	메퍼 테스트-입력제목	test	2021/06/18 Fri Jun 18 21:46:22 KST 2021	2021/06/18 21:46:23.0	0
13	메퍼 테스트-select key	test	2021/06/18 Fri Jun 18 21:46:22 KST 2021	2021/06/18 21:46:23.0	0
12	메퍼 테스트-입력제목	test	2021/06/18	2021/06/18 21:43:03	0

☞ 웹 브라우저에서 URL 요청 → 서버처리(최종적으로 게시물 목록 화면 생성) → 표시할 HTML 내용이 웹 브라우저로 전달됨

→ 전달된 HTML 내용을 웹 브라우저가 처리하여 브라우저에 표시된 것입니다.

## 7-5. 게시물 등록 JSP 페이지(register.jsp 파일) 생성 및 버튼 처리 기능 구현

☞ 새로운 게시물을 등록할 수 있는 register.jsp 파일을 생성합니다. 그리고 버튼을 클릭하여 게시물 등록 후, 게시물 목록 화면이 표시되도록 구현합니다.

→ src/main/webapp/WEB-INF/views/myboard 폴더에 있는 list.jsp 파일을 복사하여, 같은 폴더에 붙여넣기

→ 이 때 파일 이름을 register.jsp 로 변경

→ src/main/webapp/WEB-INF/views/myboard 폴더에 생성된 register.jsp 파일을 코드작성 뷰에 오픈한 후,

→ <table>부터 </table> 까지의 내용을 삭제

→ 다음의 빨간색 코드를 수정 및 추가

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" /><!-- 컨텍스트 패스 변수 선언 -->

<%@ include file="../myinclude/myheader.jsp" %>

<div id="page-wrapper">
    <div class="row">
        <div class="col-lg-12">
```

```

        <h1 class="page-header">Board - Register</h1> ←수정
    </div><!-- /.col-lg-12 -->
</div><!-- /.row -->

<div class="row">
    <div class="col-lg-12">
        <div class="panel panel-default">
            <div class="panel-heading"><h4>게시글 등록</h4></div> ← 수정
            <div class="panel-body">
                <!-- 아래의 form 요소 내용 추가 -->
                <form role="form" action="${contextPath}/myboard/register" method="post" >
                    <div class="form-group">
                        <label>제목</label> <input class="form-control" name="btitle">
                    </div>

                    <div class="form-group">
                        <label>내용</label> <textarea class="form-control" rows="3" name="bcontent"></textarea>
                    </div>

                    <div class="form-group">
                        <label>작성자</label> <input class="form-control" name="bwriter">
                    </div>

                    <button type="submit" class="btn btn-primary">등록</button>
                    <button type="button" data-oper="list" class="btn btn-warning"
                           onClick="location.href='${contextPath}/myboard/list'">취소
                    </button>
                </form>
            </div><!-- /.panel-body -->
        </div><!-- /.panel -->
    </div><!-- /.col-lg-12 -->
</div><!-- /.row -->
</div><!-- /#page-wrapper -->

<%@ include file="../myinclude/myfooter.jsp" %>

```

[테스트 - 브라우저에서 확인]

- 템켓 서버를 기동 → 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/register> URL로 접속
- 다음의 화면이 표시되어 함(왼쪽 그림) → 브라우저의 사이즈를 줄이면 아래처럼 바뀌어 표시됩니다.

→ 가운데 그림처럼 내용 입력 → 등록 버튼 클릭 → 게시물 목록 페이지가 표시되고 등록된 글이 표시되어야 합니다(오른쪽 그림).

The figure consists of three side-by-side screenshots of a web browser window titled "My Admin Board".

- Left Screenshot (Board - Register):** This shows the registration form. The title input field contains "화면 등록 테스트 제목", the content area contains "화면 등록 테스트 내용", and the writer input field contains "test". The "등록" button is highlighted in blue.
- Middle Screenshot (Board - Register):** This shows the same registration form after the "등록" button was clicked. The fields remain the same, but the "등록" button has turned orange.
- Right Screenshot (Board - List):** This shows the list of registered posts. It displays a table with columns: 번호 (Number), 제목 (Title), 작성자 (Writer), 작성일 (Created Date), 수정일 (Modified Date), and 조회수 (View Count). The table contains three entries:
 

번호	제목	작성자	작성일	수정일	조회수
17	화면 등록 테스트 제목	test	2021/06/20 Sun Jun 20 14:32:21 KST 2021	2021/06/20 14:32:21	0
16	게시물 등록 - 컨트롤러 테스트 제목	test	2021/06/19 Sat Jun 19 13:37:18 KST 2021	2021/06/19 13:37:18	0
15	서비스 새글 입력 테스트 제목	test	2021/06/19 Sat Jun 19	2021/06/19 11:51:22	0

→ 웹 브라우저에서 `http://localhost:8080/mypro00/myboard/register` URL로 접속 → 다음의 화면이 표시되어 함(왼쪽 그림)

→ 취소 버튼 클릭 → 게시물 목록화면이 표시되어야 함(오른쪽 그림\_

The left screenshot shows the 'Board - Register' page. It has fields for '제목' (Title), '내용' (Content), and '작성자' (Writer). Below these are two buttons: '등록' (Register) in blue and '취소' (Cancel) in orange. The right screenshot shows the 'Board - List' page, which displays a table of posts. The table columns are '번호' (Number), '제목' (Title), '작성자' (Writer), '작성일' (Created Date), '수정일' (Modified Date), and '조회수' (View Count). The table contains three rows of data.

번호	제목	작성자	작성일	수정일	조회수
17	화면 등록 테스트 제목	test	2021/06/20 Sun Jun 20 14:32:21 KST 2021	2021/06/20 14:32:21 2021-06-20 14:32:21.0	0
16	게시물 등록 - 컨트롤러 테스트 제목	test	2021/06/19 Sat Jun 19 13:37:18 KST 2021	2021/06/19 13:37:18 2021-06-19 13:37:18.0	0
15	서비스 새글 입력 테스트 제목	test	2021/06/19 Sat Jun 19	2021/06/19 11:51:22	0

☞ 위의 실습은 간단히 다음의 과정을 거쳐 처리됩니다.

- 게시물 등록 화면 요청 → 서버처리(게시물 등록 화면 생성) → 브라우저로 전달됨 → 게시물 등록 화면 표시
- 값을 입력 후, 등록 버튼 클릭 → 서버처리(게시물 등록처리, 게시물 목록 화면 생성) → 브라우저로 전달됨  
→ 게시물 목록 화면 표시 → 새로 등록된 게시물 표시됨.
- 게시물 등록 화면 요청 → 서버처리(게시물 등록 화면 생성) → 브라우저로 전달됨 → 게시물 등록 화면 표시
- 게시물 등록 화면에서 취소 버튼 클릭 → 서버처리(게시물 목록 화면 생성) → 브라우저로 전달됨 → 게시물 목록 화면 표시

## 7-6. 게시물 조회 JSP 페이지(detail.jsp 파일) 생성 및 버튼 처리 기능 구현

☞ 기존 게시물의 세부 정보를 표시하는 `detail.jsp` 파일을 생성합니다. 그리고, 버튼을 통해 게시물 수정-삭제 화면이나 게시물 목록 화면으로 이동될 수 있도록 구성합니다.

→ `src/main/webapp/WEB-INF/views/myboard` 폴더에 있는 `register.jsp` 파일을 복사하여, 같은 폴더에 붙여넣기

→ 이 때 파일 이름을 `detail.jsp`로 변경

→ `src/main/webapp/WEB-INF/views/myboard` 폴더에 생성된 `detail.jsp` 파일을 코드작성 뷰에 오픈한 후,

→ `<form>`부터 `</form>` 까지의 내용을 삭제

→ 다음의 빨간색 코드를 수정 및 추가, 버튼 클릭 이벤트를 처리하는 jQuery 실행문도 있으니 주의할 것!!

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" /><!-- 컨텍스트 패스 변수 선언 -->

<%@ include file="../myinclude/myheader.jsp" %>

<div id="page-wrapper">
    <div class="row">
        <div class="col-lg-12">
            <h1 class="page-header">Board - Detail</h1> ← 수정
        </div><!-- /.col-lg-12 -->
    </div><!-- /.row -->

    <div class="row">
        <div class="col-lg-12">
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h4>게시글 상세 - <small>조회수: <c:out value="${board.bviewsCnt}"/></small></h4> ← 수정 및 추가
                </div><!-- /.panel-heading -->
                <div class="panel-body">

                    <!-- 아래의 내용 전체 추가 -->

                    <div class="form-group">
                        <label>글번호</label>
                        <input class="form-control" name="bno" value='<c:out value="${board.bno}"/>' readonly="readonly" />
                    </div>
                    <div class="form-group">
                        <label>글제목</label>
                        <input class="form-control" name="btitle" value='<c:out value="${board.btitle}"/>' readonly="readonly" />
                    </div>
                    <div class="form-group">
                        <label>글내용</label>
                        <!-- <textarea>와 </textarea>는 사이에 공백이 없어야
                            데이터베이스 저장 시에 필요 없는 공백이 포함되지 않음 -->
                        <textarea class="form-control" rows="3" name="bcontent" readonly="readonly"><c:out value="${board.bcontent}"/></textarea>
                    </div>
                    <div class="form-group">
                        <label>작성자</label>
                        <input class="form-control" name="bwriter" value='<c:out value="${board.bwriter}"/>' readonly="readonly" />
                    </div>

                    <div class="form-group">
                        <label>최종수정일</label> [등록일시: <fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="${board.bregDate}"/>]
                        <input class="form-control" name="bmodDate" value='<fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="${board.bmodDate}"/>' readonly="readonly" />
                    </div>

                    <button type="button" class="btn btn-default" id="BtnMoveModify" data-oper="modify">수정</button>
                    <button type="button" class="btn btn-info" id="BtnMoveList" data-oper="list">목록</button>

                </div><!-- /.panel-body -->
            </div><!-- /.panel -->
        </div><!-- /.col-lg-12 -->
    </div><!-- /.row -->
</div><!-- #page-wrapper -->
```

```

<script>
//게시물 수정 페이지로 이동
$("#BtnMoveModify").on("click", function(){
    location.href='${contextPath}/myboard/modify?bno=<c:out value="${board.bno}" />';
})

//게시물 목록 페이지로 이동
$("#BtnMoveList").on("click", function(){
    location.href='${contextPath}/myboard/list';
})

</script>

<%@ include file="../myinclude/myfooter.jsp" %>

```

[테스트 - 브라우저에서 확인]

→ 템켓 서버를 기동

→ 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/detail?bno=1> URL로 접속 → 다음의 화면이 표시되어 함(왼쪽 그림)  
→ 브라우저의 사이즈를 줄이면 아래처럼 바뀌어 표시됩니다.

→ 목록 버튼 클릭 → 게시물 목록 페이지로 이동되어 표시됨

번호	제목	작성자	작성일	수정일	조회수
17	화면 등록 테스트 제목	test	2021/06/20 Sun Jun 20 14:32:21 KST 2021	2021/06/20 14:32:21 2021-06-20 14:32:21.0	0
16	게시물 등록 -컨트롤러 테스트 제목	test	2021/06/19 Sat Jun 19 13:37:18 KST 2021	2021/06/19 13:37:18 2021-06-19 13:37:18.0	0
15	서비스 새글 입력 테스트 제목	test	2021/06/19 Sat Jun 19 11:51:22 KST 2021	2021/06/19 11:51:22 2021-06-19 11:51:22.0	0
14	메퍼 테스트-입력 제목	test	2021/06/18 Fri Jun 18 21:46:22 KST 2021	2021/06/18 21:46:23 2021-06-18 21:46:23.0	0

☞ 위의 실습은 간단히 다음의 과정을 거쳐 처리됩니다.

- 글번호 1 인 게시물의 상세 화면 요청 → 서버처리(게시물 상세 화면 생성) → 브라우저로 전달됨 → 게시물 상세 화면 표시
  - 목록 버튼 클릭 → 서버처리(게시물 목록 화면 생성) → 브라우저로 전달됨 → 게시물 목록 화면 표시
- ☞ 수정버튼 기능은 게시물 수정 페이지 구현 후, 테스트를 수행합니다.

## 7-7. 게시물 수정-삭제 JSP 페이지(modify.jsp 파일) 생성 및 버튼 처리 기능 구현

☞ 기존 게시물을 수정 또는 삭제할 수 있는 modify.jsp 파일을 생성합니다. 그리고, 버튼을 통해 게시물 수정, 삭제, 및 게시물 목록 화면으로 이동될 수 있도록 구성합니다.

→ src/main/webapp/WEB-INF/views/myboard 폴더에 있는 detail.jsp 파일을 복사하여, 같은 폴더에 붙여넣기

→ 이 때 파일 이름을 modify.jsp 로 변경

→ src/main/webapp/WEB-INF/views/myboard 폴더에 생성된 modify.jsp 파일을 코드작성 뷰에 오픈한 후,

→ 다음의 빨간색 코드를 수정 및 추가, 버튼 클릭 이벤트를 처리하는 jQuery 실행문 추가(오타, 대소문자 주의!!!)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" /><!-- 컨텍스트 패스 변수 선언 -->

<%@ include file="../myinclude/myheader.jsp" %>

<div id="page-wrapper">
    <div class="row">
        <div class="col-lg-12">
            <h1 class="page-header">Board - Modify</h1> ←수정
        </div><!-- /.col-lg-12 -->
    </div><!-- /.row -->

    <div class="row">
        <div class="col-lg-12">
            <div class="panel panel-default">
                <div class="panel-heading"><h4>게시글 수정-삭제</h4></div><!-- /.panel-heading --> ←수정
                <div class="panel-body">

                    <form role="form" id="frmModify" method="post"> ← 추가

                        <div class="form-group">
                            <label>글번호</label>
                            <input class="form-control" name="bno" value='<c:out value="${board.bno}" />' readonly="readonly" />
                        </div>
                        <div class="form-group">
                            <label>글제목</label> ← input 요소의 readonly 속성 삭제
                            <input class="form-control" name="btitle" value='<c:out value="${board.btitle}" />' />
                        </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

<div class="form-group">
    <label>글내용</label> ← textarea 요소의 readonly 속성 삭제
    <!-- <textarea>와 </textarea>는 사이에 공백이 없어야
        데이터베이스 저장 시에 필요 없는 공백이 포함되지 않음 -->
    <textarea class="form-control" rows="3" name="bcontent"
              ><c:out value="${board.bcontent}" /></textarea>
</div>
<div class="form-group">
    <label>작성자</label>
    <input class="form-control" name="bwriter" value='<c:out value="${board.bwriter}" />'
           readonly="readonly"/>
</div>

<div class="form-group">
    <label>최종수정일</label> [등록일시: <fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss"
                                                               value="${board.bregDate}" />]
    <input class="form-control" name="bmodDate"
           value='<fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="${board.bmodDate}" />'
           disabled="disabled" /> ← input 요소의 readonly 속성을 disabled 속성으로 교체
</div>

<button type="button" class="btn btn-default" id="btnModify" data-oper="modify">수정</button>
<button type="button" class="btn btn-danger" id="btnRemove" data-oper="remove">삭제</button>
<button type="button" class="btn btn-info" id="btnList" data-oper="list">취소</button>

</form> ← 버튼과 form 종료 태그 추가
</div><!-- /.panel-body -->
</div><!-- /.panel -->
</div><!-- /.col-lg-12 -->
</div><!-- /.row -->
</div><!-- #page-wrapper -->

<script> ← 기존 내용 삭제 후, 아래의 내용 추가 작성(오타, 대소문자 주의!!!!)

//form의 수정/삭제/목록보기 버튼 클릭 이벤트 처리
var frmModify = $("#frmModify");

$('button').on("click", function(e){

    //e.preventDefault(); //버튼 유형이 submit가 아니므로 설정할 필요 없음

    var operation = $(this).data("oper"); //각 버튼의 data-xxx 속성에 설정된 값을 저장
    alert("operation: " + operation);

    if(operation == "modify"){ //게시물 수정 요청
        frmModify.attr("action", "${contextPath}/myboard/modify");

    } else if(operation == "remove"){ //게시물 삭제 요청
        frmModify.attr("action", "${contextPath}/myboard/delete");

    } else if(operation == "list"){ //게시물 목록 화면 요청
        frmModify.attr("action", "${contextPath}/myboard/list").attr("method", "get");
        frmModify.empty();
    }

    frmModify.submit() ; //요청 전송
});

</script>

<%@ include file="../myinclude/myfooter.jsp" %>

```

☞ form에 의해서 서버에 전송되는 날짜시간(bmodDate Input) 값은 날짜시간 형식이 yyyy-MM-dd HH:mm:ss 일 경우, 정상적으로 처리됩니다. 그런데 실습처럼 최종 수정시간의 날짜시간 값이 yyyy/MM/dd HH:mm:ss 형식으로 전달되면 서버에서 오류가 발생됩니다.

따라서, 화면에 표시되는 형식을 yyyy-MM-dd HH:mm:ss 로 변경하거나 날짜시간 값이 전송되지 못하도록 합니다.  
화면에서 최종 수정 날짜시간 값이 전달되지 않더라도 최종 변경 날짜시간의 수정은 데이터베이스에서 컬럼에 설정된 기본값으로 처리되므로, 정상적으로 수정날짜시간이 변경되어 저장됩니다.

#### [테스트 - 브라우저에서 확인]

→ 톰캣 서버를 기동

→ 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/detail?bno=1> URL로 접속

→ 서버처리(게시물 상세 화면 생성) 후, 게시물 번호 1 번인 게시물의 상세 화면이 브라우저에 표시됩니다.

→ 글번호 1 번의 게시물의 상세 화면에서 수정 버튼 클릭 → 서버처리(게시물 수정-삭제 화면 생성)

→ 글번호 1 번의 게시물 수정-삭제 화면 표시됨 → 글 제목과 내용을 수정 후, 수정 버튼 클릭

The image shows two side-by-side screenshots of a web browser window titled "My Admin Board".

**Left Screenshot (Board - Detail):**

- URL: localhost:8080/mypro00/myboard/detail?bno=1
- Header: 게시글 상세 - 조회수: 14
- Form Fields:
  - 글번호: 1
  - 글제목: 게시글 수정-컨트롤러 테스트 제목
  - 글내용:

```
게시글 수정-컨트롤러 테스트 내용
```
  - 작성자: user1
  - 최종수정일 [등록일시: 2021/06/18 21:42:36]: 2021/06/19 17:46:02
- Buttons: 수정 (gray), 목록 (blue)

**Right Screenshot (Board - Modify):**

- URL: localhost:8080/mypro00/myboard/modify?bno=1
- Header: 게시글 수정-삭제
- Form Fields:
  - 글번호: 1
  - 글제목: 게시글 수정-컨트롤러 테스트 제목 수정
  - 글내용:

```
게시글 수정-컨트롤러 테스트 내용 수정
```
  - 작성자: user1
  - 최종수정일 [등록일시: 2021/06/18 21:42:36]: 2021/06/19 17:46:02
- Buttons: 수정 (gray), 삭제 (red), 취소 (blue)

→ 서버처리(게시글 수정, 게시물 상세 화면 생성) → 글번호 1 번의 게시물 상세 화면 표시

The image shows two side-by-side browser windows. Both windows have a header "My Admin Board" and a URL starting with "localhost:8080/mypro00/myboard/".

**Left Window (Modification View):**

- Header: "Board - Modify"
- Section: "게시글 수정-삭제"
- Form fields:
  - 글번호: 1
  - 글제목: 게시글 수정-컨트롤러 테스트 제목 수정
  - 글내용:  
게시글 수정-컨트롤러 테스트 내용 수정
  - 작성자: user1
  - 최종수정일 [등록일시: 2021/06/18 21:42:36]  
2021/06/19 17:46:02
- Buttons: "수정" (highlighted), "삭제", "취소"

**Right Window (Detail View):**

- Header: "Board - Detail"
- Section: "게시글 상세 - 조회수: 14"
- Form fields:
  - 글번호: 1
  - 글제목: 게시글 수정-컨트롤러 테스트 제목 수정
  - 글내용:  
게시글 수정-컨트롤러 테스트 내용 수정
  - 작성자: user1
  - 최종수정일 [등록일시: 2021/06/18 21:42:36]  
2021/06/20 16:10:30
- Buttons: "수정", "목록"

→ 게시물의 1번의 상세 화면에서 삭제 테스트를 위해 수정 버튼 다시 클릭 → 서버처리(게시글 수정-삭제 화면 생성)

→ 게시물 수정-삭제 화면에서 → 삭제 버튼 클릭 → 서버처리(게시글 삭제 수행, 게시물 목록 화면 생성)

→ 게시물 목록 화면 표시(화면을 아래로 클릭하면 글번호 1번인 글이 삭제된 글이라고 표시됨)

The image shows two side-by-side browser windows. Both windows have a header "My Admin Board" and a URL starting with "localhost:8080/mypro00/myboard/".

**Left Window (Detail View):**

- Header: "Board - Detail"
- Section: "게시글 상세 - 조회수: 14"
- Form fields:
  - 글번호: 1
  - 글제목: 게시글 수정-컨트롤러 테스트 제목 수정
  - 글내용:  
게시글 수정-컨트롤러 테스트 내용 수정
  - 작성자: user1
  - 최종수정일 [등록일시: 2021/06/18 21:42:36]  
2021/06/20 16:10:30
- Buttons: "수정", "목록" (highlighted)

**Right Window (Modification View):**

- Header: "Board - Modify"
- Section: "게시글 수정-삭제"
- Form fields:
  - 글번호: 1
  - 글제목: 게시글 수정-컨트롤러 테스트 제목 수정
  - 글내용:  
게시글 수정-컨트롤러 테스트 내용 수정
  - 작성자: user1
  - 최종수정일 [등록일시: 2021/06/18 21:42:36]  
2021/06/20 16:10:30
- Buttons: "수정", "삭제" (highlighted), "취소"

The left screenshot shows the 'Board - Modify' page with a modal dialog asking for confirmation to delete ('삭제'). The right screenshot shows the 'Board - List' page where the item with ID 2 has been successfully deleted, indicated by a yellow message at the bottom.

**Left Screenshot (Board - Modify):**

- Modal Dialog: My Admin localhost:8080 내용: operation: remove
- Buttons: 확인 (Confirm), 취소 (Cancel)

**Right Screenshot (Board - List):**

번호	제목	작성자	작성일	수정일	조회수
16	게시물 등록 - 컨트롤러 테스트 제목	test	2021/06/19 Sat Jun 19 13:37:18 KST 2021	2021/06/19 13:37:18 2021-06-19 13:37:18.0	0
15	서비스 새글 입력 테스트 제목	test	2021/06/19 Sat Jun 19 11:51:21 KST 2021	2021/06/19 11:51:22 2021-06-19 11:51:22.0	0
14	메퍼 테스트-입력 제목	test	2021/06/18 Fri Jun 18 21:46:22 KST 2021	2021/06/18 21:46:23 2021-06-18 21:46:23.0	0
13	메퍼 테스트-select key	test	2021/06/18 Fri Jun 18 21:46:22 KST 2021	2021/06/18 21:46:23 2021-06-18 21:46:23.0	0
12	메퍼 테스트-입력 제목	test	2021/06/18 Fri Jun 18 21:43:02 KST 2021	2021/06/18 21:43:03 2021-06-18 21:43:03.0	0
11	메퍼 테스트-select key	test	2021/06/18 Fri Jun 18 21:43:02 KST 2021	2021/06/18 21:43:03 2021-06-18 21:43:03.0	0
2	테스트제목2	user2	2021/06/18 Fri Jun 18 21:42:37 KST 2021	2021/06/18 21:42:38 2021-06-18 21:42:38.0	0

Message: 1 작성자에 의하여 삭제된 게시글입니다!

→ 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/modify?bno=2> URL로 접속 → 서버처리(게시물 수정-삭제 화면 생성)

→ 게시물 수정-삭제 화면에서, 취소 버튼 클릭 → 서버처리(게시물 목록 화면 생성) → 게시물 목록 화면 표시

The left screenshot shows the 'Board - Modify' page with a modal dialog asking for confirmation to list ('목록'). The middle screenshot shows the 'Board - Modify' page with the same dialog. The right screenshot shows the 'Board - List' page displaying the updated list of posts.

**Left Screenshot (Board - Modify):**

- Modal Dialog: My Admin localhost:8080 내용: operation: list
- Buttons: 확인 (Confirm), 취소 (Cancel)

**Middle Screenshot (Board - Modify):**

- Modal Dialog: My Admin localhost:8080 내용: operation: list
- Buttons: 확인 (Confirm), 취소 (Cancel)

**Right Screenshot (Board - List):**

번호	제목	작성자	작성일	수정일	조회수
17	화면 등록 테스트 제목	test	2021/06/20 Sun Jun 20 14:32:21 KST 2021	2021/06/20 14:32:21 2021-06-20 14:32:21.0	0
16	게시물 등록 - 컨트롤러 테스트 제목	test	2021/06/19 Sat Jun 19 13:37:18 KST 2021	2021/06/19 13:37:18 2021-06-19 13:37:18.0	0
15	서비스 새글 입력 테스트 제목	test	2021/06/19 Sat Jun 19 11:51:21 KST 2021	2021/06/19 11:51:22 2021-06-19 11:51:22.0	0
14	메퍼 테스트-입력 제목	test	2021/06/18 Fri Jun 18 21:46:22 KST 2021	2021/06/18 21:46:23 2021-06-18 21:46:23.0	0

## 7-8. 웹 브라우저에서의 각 화면 간 이동 구현

☞ 지금까지의 구현작업으로 다음의 화면 이동은 구성되어 있습니다.

- 게시물 등록 화면에서 버튼클릭 → (서버처리) → 게시물 목록 화면
- 게시물 상세 화면에서 버튼클릭 → (서버처리) → 게시물 목록 화면
- 게시물 상세 화면 버튼클릭 → (서버처리) → 게시물 수정-삭제 화면 이동, 화면에서 버튼 클릭  
→ (서버처리) → 게시물 상세 또는 게시물 목록 화면

☞ 일반적으로, 새 게시물의 등록화면이나 특정 게시물의 상세정보 화면은, 게시물 목록 화면에서 해당 화면으로 이동되도록 구현됩니다.

☞ 게시물 목록 화면을 생성하는 list.jsp 파일에 다음을 구현합니다.

- 게시물 등록 화면으로 이동할 수 있는 버튼
- 게시물 목록에서 특정 게시물의 제목을 클릭했을 때, 게시물 상세화면으로 이동할 수 있는 링크

☞ 게시물 상세 화면 이동의 경우, 다음의 방법들이 사용될 수 있습니다(2, 3 방법은 유사합니다).

- 방법1: 제목에 <a> 태그를 이용하여, href 속성에 호출할 URI에 get 방식으로 전달할 값을 명시  
예, href='\${contextPath}/myboard/detail?bno=<c:out value="\${board.bno}" />'  
→ 이 방법은 구현은 쉽지만, 화면이동 간 전달할 데이터 양이 많아지면, 불편합니다.
- 방법2: 제목에 <a> 태그를 이용하여, href 속성에 href='<c:out value="\${board.bno}" />'를 이용하고  
값을 전달할 비어있는 form을 생성한 후, jQuery를 이용하여 form에 전달할 값을 추가하여 전송하도록 구현  
→ 이 방법은 a 태그의 기능을 방지해야 정상 작동됩니다.
- 방법3: <tr> 태그에 data-bno 속성으로 글 번호를 지정한 후,  
값을 전달할 비어있는 form을 생성한 후, jQuery를 이용하여 form에 전달할 값을 추가하여 전송하도록 구현

☞ 방법2와 방법3은 행이 클릭될 때 게시물의 bno 값이 저장되어 하므로 jQuery를 이용하여 form에 bno 값이 추가되도록 구현합니다.  
또한 클릭 시에 jQuery로 tr 또는 a 태그가 각각 선택될 수 있도록, css 클래스 이름 선택자를 지정합니다.

☞ 실습에서는 방법3을 구현합니다. 방법 1과 방법2는 설명없이 코드만 별도로 추가합니다.

(1) 게시물 목록 화면에서 게시물 등록 화면 호출 구현

→ src/main/webapp/WEB-INF/views/myboard 폴더에 있는 list.jsp 파일을 코드작성 뷰에 오픈한 후,  
panel-heading 클래스인 div를 다음처럼 수정하고, 버튼 클릭 이벤트 jQueyr를 page-wrapper 종료 태그 밑에 추가합니다.  
추가 위치를 잘 확인합니다.

```
<div class="panel-heading">
    <div class="row">
        <div class="col-md-6" style="font-size:20px; height: 45px; padding-top:10px;">게시글 목록</div>
        <div class="col-md-6" style="padding-top:8px;">
            <button type="button" id="btnMoveRegister" class="btn btn-primary btn-sm pull-right">새글 등록</button>
        </div>
    </div><!-- /.panel-heading -->
    ...
</div><!-- /#page-wrapper -->

<script>
<%-- //새글 등록 버튼 클릭 이벤트 처리: 게시물 등록 화면 이동//////////////////////////// --%>
```

```

$( "#btnMoveRegister" ).on("click", function(){
    self.location = "${contextPath}/myboard/register";
})
</script>

```

```
<%@ include file="../myinclude/myfooter.jsp" %>
```

(2) 게시물 목록 화면에서 게시물 상세 화면 호출 구현(설명에서의 방법3으로 구현)

→ list.jsp 파일에 다음의 빨간색 코드를 추가합니다.

```

...(생략)...

<tbody>

<c:forEach items="${boardList}" var="board"><%-- 컨트롤러에서 보낸 목록객체 이름: boardList --%>
    <c:if test="${board.bdelFlag == 1}">
        <tr style="background-color:Moccasin; text-align:center">
            <td><c:out value="${board.bno}" /></td>
            <td colspan="5"><em>작성자에 의하여 삭제된 게시글입니다.</em></td>
        </tr>
    </c:if>
    <c:if test="${board.bdelFlag == 0}">
        <tr class="moveDetail" data-bno='<c:out value="${board.bno}" />' ><%-- tr 요소에 data-bno 속성으로 bno 지정 --%>
            <td><c:out value="${board.bno}" /></td>
            <td style="text-align:left;"><c:out value="${board.btitle}" /></td>
            <td><c:out value="${board.bwriter}" /></td>
            <td><fmt:formatDate pattern="yyyy/MM/dd" value="${board.bregDate}" /><br>
                ${board.bregDate}</td>
            <td><fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="${board.bmodDate}" /><br>
                ${board.bmodDate}</td>
            <td>${board.bviewsCnt}</td>
        </tr>
    </c:if>
</c:forEach>

    </tbody>
</table><%-- /.table-responsive --%>
<form id="frmSendValue"><%-- 전달할 hidden 유형의 input 요소들이 추가되어 값들이 전달될 비어있는 form --%>
</form>

...(생략)...

<%--게시물 행(제목) 클릭 이벤트 처리: 게시물 상세 화면 이동//////////////////////////// --%>
<script type="text/javascript">
var frmSendValue = $("#frmSendValue");

<%--tr 태그 클릭 시 form의 데이터를 전달하고 detail 화면 요청 --%>
$(".moveDetail").on( "click", function(e) {

    <%-- bno 값이 값이 설정된 hidden 유형의 input 요소를 form에 추가 --%>
    <%-- tr 태그의 data-bno 속성의 값을 data() 함수로 값을 읽어와 value 속성에 지정 --%>
    frmSendValue.append("<input type='hidden' name='bno' value='" + $(this).data("bno") + "'/>");

    frmSendValue.attr("action", "${contextPath}/myboard/detail");<%-- form에 action 속성 지정 --%>
    frmSendValue.attr("method", "get");<%-- form에 method 속성 지정 --%>
    frmSendValue.submit();<%-- form 전송 --%>
});

</script>

<%@ include file="../myinclude/myfooter.jsp" %>
```

## ☞ 방법1로 구현한 경우

```
...(생략)...
<c:forEach items="${boardList}" var="board"><%-- 컨트롤러에서 보낸 목록객체 이름: boardList --%>
    <c:if test="${board.bdelFlag == 1}">
        <tr style="background-color:Moccasin; text-align:center">
            <td><c:out value="${board.bno}" /></td>
            <td colspan="5"><em>작성자에 의하여 삭제된 게시글입니다.</em></td>
        </tr>
    </c:if>
    <c:if test="${board.bdelFlag == 0}">
        <tr>
            <td><c:out value="${board.bno}" /></td>
            <td style="text-align:left;">
                <%-- 상세 페이지 이동-방법1: a 태그만 이용. --%>
                <a target="_blank" href='${contextPath}/myboard/detail?bno=<c:out value="${board.bno}" />'>
                    <c:out value="${board.btitle}" />
                </a>
            </td>
            <td><c:out value="${board.bwriter}" /></td>
            <td><fmt:formatDate pattern="yyyy/MM/dd" value="${board.bregDate}" /><br>
            </td>
            <td><fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="${board.bmodDate}" /><br>
            </td>
            <td>${board.bviewsCnt}</td>
        </tr>
    </c:if>
</c:forEach>
...(생략)...
```

## ☞ 방법2로 구현한 경우: 빨간색 코드 추가

```
...(생략)...
<c:forEach items="${boardList}" var="board"><%-- 컨트롤러에서 보낸 목록객체 이름: boardList --%>
    <c:if test="${board.bdelFlag == 1}">
        <tr style="background-color:Moccasin; text-align:center">
            <td><c:out value="${board.bno}" /></td>
            <td colspan="5"><em>작성자에 의하여 삭제된 게시글입니다.</em></td>
        </tr>
    </c:if>
    <c:if test="${board.bdelFlag == 0}">
        <tr>
            <td><c:out value="${board.bno}" /></td>
            <td style="text-align:left;">
                <a class='moveDetail' target="_self" href='<c:out value="${board.bno}" />'>
                    <c:out value="${board.btitle}" />
                </a>
            </td>
            <td><c:out value="${board.bwriter}" /></td>
            <td><fmt:formatDate pattern="yyyy/MM/dd" value="${board.bregDate}" /><br>
            </td>
            <td><fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="${board.bmodDate}" /><br>
            </td>
            <td>${board.bviewsCnt}</td>
        </tr>
    </c:if>
</c:forEach>

    </tbody>
</table><!-- /.table-responsive -->
<form id="frmSendValue">
</form>

...(생략)...
<%-- //게시물 행이나 제목 클릭 이벤트 처리: 게시물 상세 화면 이동//////////////////////////// --%>
```

```

<script>
var frmSendValue = $("#frmSendValue");

$(".moveDetail").on("click", function(e) {
    e.preventDefault(); // <a> 태그 기능 방지
    frmSendValue.append("<input type='hidden' name='bno' value=''" + $(this).attr("href") + "'/>");

    frmSendValue.attr("action", "${contextPath}/myboard/detail");
    frmSendValue.attr("method", "get");

    frmSendValue.submit();
});
</script>

<%@ include file="../myinclude/myfooter.jsp" %>

```

[테스트 - 브라우저에서 확인]

- 텀캣 서버를 기동
- 웹 브라우저에서 `http://localhost:8080/mypro00/myboard/list` URL로 접속
- [새글 등록] 버튼 클릭 → 서버처리(게시물 등록 화면 생성) → 게시물 등록 화면 표시
- 취소 버튼 클릭 → 서버처리(게시글 목록 화면 생성) → 게시물 목록 화면 표시
- 게시물 행 하나를 클릭 → 서버처리(게시물 상세 화면 생성) → 게시물 상세 화면 표시
- 목록 버튼 클릭 → 서버처리(게시물 목록 화면 생성) → 게시물 목록 화면 표시

The image contains two side-by-side screenshots of a web application interface.

**Left Screenshot (Board - List):**

- Title: My Admin Board
- Page: Board - List
- Section: 게시글 목록
- Buttons: 새글 등록 (New Article)
- Table Headers: 번호 (Number), 제목 (Title), 작성자 (Writer), 작성일 (Created Date), 수정일 (Modified Date), 조회수 (View Count)
- Data Rows:
  - 17: 화면 등록 테스트 제목 | test | 2021/06/20 Sun Jun 20 14:32:21 KST 2021 | 2021/06/20 14:32:21 | 10
  - 16: 게시물 등록 - 컨트롤러 테스트 제목 | test | 2021/06/19 Sat Jun 19 13:37:18 KST 2021 | 2021/06/19 13:37:18 | 2

**Right Screenshot (Board - Register):**

- Title: My Admin Board
- Page: Board - Register
- Section: 게시글 등록
- Form Fields:
  - 제목 (Title):
  - 내용 (Content):
  - 작성자 (Writer):
- Buttons: 등록 (Register) and 취소 (Cancel)

게시글 목록					
번호	제목	작성자	작성일	수정일	조회수
17	화면 등록 테스트 제목	test	2021/06/20 Sun Jun 20 14:32:21 KST 2021	2021/06/20 14:32:21 2021-06-20 14:32:21.0	10
16	게시물 등록 - 컨트롤러 테스트 제목	test	2021/06/19 Sat Jun 19 13:37:18 KST 2021	2021/06/19 13:37:18 2021-06-19 13:37:18.0	2

## Board - Detail

게시글 상세 - 조회수: 11

글번호	17
글제목	화면 등록 테스트 제목
글내용	화면 등록 테스트 내용
작성자	test

## Board - Detail

게시글 상세 - 조회수: 11

글번호	17
글제목	화면 등록 테스트 제목
글내용	화면 등록 테스트 내용
작성자	test
최종수정일 [등록일시: 2021/06/20 14:32:21]	2021/06/20 14:32:21
<button>수정</button>	<button>목록</button>

## Board - List

게시글 목록

번호	제목	작성자	작성일	수정일	조회수
17	화면 등록 테스트 제목	test	2021/06/20 Sun Jun 20 14:32:21 KST 2021	2021/06/20 14:32:21 2021-06-20 14:32:21.0	11
16	게시물 등록 - 컨트롤러 테스트 제목	test	2021/06/19 Sat Jun 19 13:37:18 KST 2021	2021/06/19 13:37:18 2021-06-19 13:37:18.0	2

## 7-9. 게시물 등록/삭제 작업 후, 모달을 통한 작업 처리 결과 표시

- ☞ 게시물 등록/삭제 작업 후, 게시물 목록 페이지로 이동 시, 각 작업의 처리에 대한 메시지를 모달을 통해 사용자에게 표시합니다.
- ☞ 모달 소스는 BootStrap Template 폴더의 pages 폴더에 notification.html 파일에 있는 것을 이용하면, 쉽게 구현할 수 있습니다.

→ 다음의 코드를 list.jsp 파일의 table 종료 태그(</table>) 밑에 추가합니다.

...(생략)...

```
</table><!-- /.table-responsive -->
<%-- Modal 모달 시작--%>
<div class="modal fade" id="myModal" tabindex="-1" role="dialog"
    aria-labelledby="myModalLabel" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-hidden="true">&times;</button>
                <h4 class="modal-title" id="myModalLabel">처리결과</h4>
            </div>
            <div class="modal-body">처리가 완료되었습니다.</div> ← <div> 영역에 처리 메시지가 표시됩니다.
            <div class="modal-footer">
                <button type="button" class="btn btn-default" data-dismiss="modal">확인</button>
            </div>
        </div><%-- END .modal-content --%>
    </div><%-- END .modal-dialog --%>
</div><%-- END .modal --%>

<form id="frmSendValue"><%-- 전달할 hidden 유형의 input 요소들이 추가되어 값들이 전달될 비어있는 form --%>
</form>
```

...(생략)...

→ 다음의 jQuery 코드를 list.jsp 파일의 <%@ include file="../myinclude/myfooter.jsp" %> JSP 지시자 위에 추가합니다.

```
...(생략)...
<script>
var result = '<c:out value="${result}" />'; //컨트롤러가 전달한 result 값을 변수에 저장

function checkModal(result) {
    if (result === '' || history.state) {
        return;
    } else if(result === 'successRemove'){
        var myMsg = "글이 삭제되었습니다";
    } else if (parseInt(result) > 0) {
        var myMsg = "게시글 " + parseInt(result) + " 번이 등록되었습니다.";
    }

    $(".modal-body").html(myMsg);
    $("#myModal").modal("show");
    myMsg='';
}

checkModal(result);
</script>

<%@ include file="../myinclude/myfooter.jsp" %>
```

[테스트 - 브라우저에서 확인]

→ 톰캣 서버를 기동

→ 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/list> URL로 접속

→ [새글 등록] 버튼 클릭 → 서버처리(게시물 등록 화면 생성) → 게시물 등록 화면 표시

→ 내용 기입 후, 등록 버튼 클릭 → 서버처리(게시물 등록, 게시글 목록 화면 생성) → 모달 창 표시, 확인 클릭 → 목록 화면 표시

The figure consists of four screenshots of a web application titled "My Admin Board".

- Board - List:** Shows a table of posts. One post (id 17) has been selected.
- Board - Register:** A modal window for creating a new post. It contains fields for Title (회원 새글등록 테스트2), Content (회원 새글 등록 테스트2 내용), and Author (test). Buttons for "등록" (Register) and "취소" (Cancel) are present.
- 처리결과:** A confirmation modal window stating "게시글 18 번이 등록되었습니다." (Post number 18 was registered successfully) with a "확인" (Confirm) button.
- Board - List:** Shows the updated list of posts. Post 18 is now listed with the title "회원 새글등록 테스트2" and post 17 is still listed.

번호	제목	작성자	작성일	수정일	조회수
17	화면 등록 테스트 제목	test	2021/06/20 Sun Jun 20 14:32:21 KST 2021	2021/06/20 14:32:21 2021-06-20 14:32:21.0	11
16	게시물 등록 -컨트롤러 테스트 제목	test	2021/06/19 Sat Jun 19 13:37:18 KST 2021	2021/06/19 13:37:18 2021-06-19 13:37:18.0	2
18	회원 새글등록 테스트2	test	2021/06/20 Sun Jun 20 19:47:19 KST 2021	2021/06/20 19:47:19 2021-06-20 19:47:19.0	0

→ 게시물 목록 중 게시물 행 하나를 클릭 → 서버처리(게시물 상세 화면 생성) → 게시물 상세 화면 표시 → 수정 버튼 클릭

→ 게시물 수정 페이지 표시 → 삭제 버튼 클릭 → 서버처리(게시물 삭제, 게시물 목록 화면 생성) → 모달 창 표시, 확인 클릭

→ 게시물 목록 화면 표시

The screenshot displays five windows illustrating the deletion process:

- Board - Detail**: Shows a post with ID 18, title "화면 새글등록 테스트2", content "화면 새글 등록 테스트2 내용", author "test", and creation date "2021/06/20 19:47:19". Buttons for "수정" (Modify), "삭제" (Delete), and "취소" (Cancel).
- Board - Modify**: Shows the same post after modification, with the title changed to "화면 새글등록 테스트2 - 수정" and content updated to "화면 새글 등록 테스트2 내용 - 수정". Buttons for "수정" (Modify), "삭제" (Delete), and "취소" (Cancel).
- Board - Detail**: Shows the post after modification, with the title and content reflecting the changes. The "삭제" (Delete) button is highlighted.
- 처리결과**: A modal window titled "처리결과" with the message "글이 삭제되었습니다." (The article has been deleted). A "확인" (Confirm) button is present.
- Board - List**: The main list of posts. Post 18 is highlighted in yellow with the message "작성자에 의해 삭제된 게시글입니다." (This post was deleted by the author). Other posts include 17 ("화면 등록 테스트 제목") and 16 ("게시물 등록 - 컨트롤러 테스트 제목"). A "새글 등록" (New Article Registration) button is visible at the top right.

## 7-10. 게시물 수정 후, 웹 브라우저 경고창을 통한 작업 처리 결과 표시

☞ 게시물 수정 작업 후, 게시물 상세 페이지로 이동 시, 작업의 처리에 대한 메시지를 브라우저의 경고창을 통해 사용자에게 표시합니다.

- src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지를 코드 작성부에 오픈
- 다음의 <script> 부터 </script> 까지의 JavaScript 및 jQuery 코드를, 제일 끝에 있는  
`<%@ include file="../myinclude/myfooter.jsp" %>` JSP 지시자 위에 추가합니다.

...(생략)...

```
<script>
var result = '<c:out value="${result}" />';

function checkModifyOperation(result) {
    if (result === '') || history.state) {
        return;
    } else if (result === 'successModify'){
        var myMsg = "글이 수정되었습니다";
    }

    alert(myMsg);
    myMsg='';
}

$(document).ready(function(){
    checkModifyOperation(result);
});
</script>
```

```
<%@ include file="../myinclude/myfooter.jsp" %>
```

→ 브라우저에서 확인 테스트는 다음의 순서로 스스로 해보시기 바랍니다.

- 터미널 서버를 기동
- 웹 브라우저에서 `http://localhost:8080/mypro00/myboard/list` URL로 접속
- 게시물 목록 중 하나의 글 제목 클릭 → 게시물 상세 화면으로 이동
- 수정 버튼 클릭 → 게시물 수정-삭제 화면으로 이동
- 제목과 내용을 적절히 수정 후, 수정 버튼 클릭
- "글이 수정되었습니다" 메시지가 표시된 알림창이 표시됨
- 알림창의 확인 클릭 → 수정 후의 게시물 상세 화면이 표시됨.

## 7-11. 게시물 등록/삭제 후 이동된 목록페이지에서 뒤로가기 방지 구현(모달 및 jQuery 이용)

☞ 게시물 등록 후, 표시된 목록페이지에서 웹 브라우저의 뒤로가기를 실행하면, 다시 값이 입력된 상태의 게시물 등록 페이지가 다시 표시됩니다. 이것은 브라우저가 이전에 게시물 등록 화면을 저장했다가 다시 표시해주는 기능을 이용한 것입니다(서버로의 요청이 없음). 그런데 이 상태에서 다시 등록 버튼을 클릭하면, 새글 등록이 다시 서버로 요청되어 수행됩니다. 이 과정을 반복되면, 서버에 동일한 데이터의 등록 요청이 계속 반복되며 데이터베이스에 동일한 글제목과 글내용이 저장되는 바람직하지 못한 상황이 발생됩니다.

이를 방지하기 위하여 게시물 등록 후에 호출된 게시물 목록페이지의 경우, 뒤로가기 버튼을 동작하지 못하게 방지할 필요가 있습니다.

→ 다음의 jQuery 코드를 list.jsp 파일의 `<%@ include file="../myinclude/myfooter.jsp" %>` JSP 지시자 위에 추가합니다.

...(생략)...

```
<script>
```

...(생략)...

`checkModal(result); //모달 동작 후, 아래의 history.pushState() 동작과 popstate 이벤트 리스너에 의해 뒤로가기 방지됨.`

```
</script>
```

```
<script>
```

`//popstate 이벤트를 처리하는 자바스크립트 리스너 추가, popstate는 간단히 브라우저의 뒤로가기 버튼 클릭 이벤트 이름입니다.`

`window.addEventListener('popstate', function(event) {`

`history.pushState(null, null, location.href); //뒤로가기 Block.`

`})`

`//페이지 로딩 시에, 실행되어 현재 목록페이지의 URL을 강제로 최근 URL로서 히스토리 객체에 다시 추가`

`history.pushState(null, null, location.href);`

```
</script>
```

```
<%@ include file="../myinclude/myfooter.jsp" %>
```

→ 화면에서의 테스트는 스스로 웹브라우저에서 다음의 절차대로 직접 수행합니다.

웹 브라우저 실행 → 게시물 목록페이지 → 게시물 등록 페이지 → 내용 작성 후, 등록 버튼 클릭

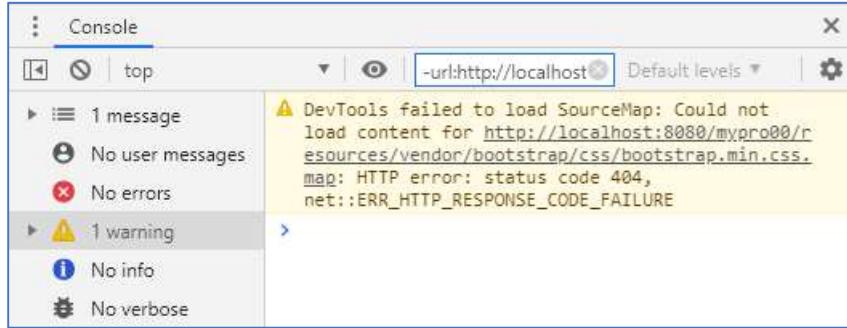
→ 모달 처리 결과 메시지 표시, 확인 클릭 → 게시물 목록페이지 표시 → 뒤로가기 버튼을 여러 번 클릭 → 등록 페이지로 이동 되면 않됨

게시물 목록 페이지, 목록 중 제목 하나를 클릭 → 게시물 상세 페이지 → 수정 버튼 클릭 → 게시물 수정 페이지 표시

→ 게시물 수정 페이지에서 삭제 버튼 클릭 → 모달 처리 결과 메시지 표시, 확인 클릭 → 게시물 목록페이지 표시

→ 뒤로가기 버튼을 여러 번 클릭 → 등록 페이지로 이동 되면 않됨

- ◆ 웹 브라우저의 개발자 도구 콘솔에 `bootstrap.min.css.map` 을 Load 할 수 없다는 아래와 같은 경고가 표시되는 경우,  
다음처럼 해결합니다.



- 사용 중인 부트스트랩 버전을 확인하기 위하여 `src/main/webapp/resources/vendor/bootstrap/css` 폴더에 있는 `bootstrap.min.css`를 코드작성 뷰에 오픈

```
/*
 * Bootstrap v3.3.7 (http://getbootstrap.com) ← BootStrap 3.3 버전의 최신 버전이 3.3.7임
 * Copyright 2011-2016 Twitter, Inc.
 * Licensed under MIT (https://github.com/twbs/bootstrap/blob/master/LICENSE)
 ...생략...
```

- 현재 구성된 Bootstrap의 버전이 v3.3.7임을 확인함

- 웹 브라우저에서 <http://getbootstrap.com> 사이트에 접속
  - 화면 중앙에 있는 Download 버튼 아래의 All releases를 클릭

- V3.x의 3.3 클릭 → Download Bootstrap 클릭 → v3.3.7 버전을 다운로드 받을 수 있는 페이지가 표시됨

- Download Bootstrap 클릭 → 3.3.7 버전이 다운로드 됨

- 다운로드 된 `bootstrap-3.3.7-dist.zip` 파일의 압축을 해제

- 압축해제 된 폴더 안에 있는 `css` 폴더로 이동 → `bootstrap.min.css.map` 파일이 존재

- `bootstrap.css.map`, `bootstrap.min.css.map` 파일을 복사

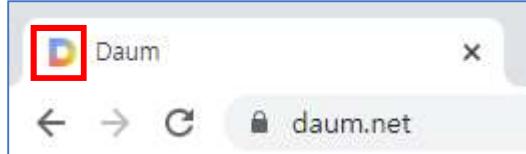
- 이클립스의 Project Explorer 뷰에서 `src/main/webapp/resources/vendor/bootstrap/css` 폴더에 붙여넣기

[조치 완료]

## ◆ 웹 브라우저의 개발자 도구 콘솔에 아래와 같은 favicon.ico 오류가 표시되는 경우의 해결법

① GET <http://localhost:8080/favicon.ico> 404 favicon.ico:1

☞ favicon은 웹 페이지 접속 시, 웹 브라우저의 탭 왼쪽에 표시되는 아이콘 이미지입니다(아래그림 빨간 박스).



☞ 서버로부터 전달된 HTML 내용을 웹 브라우저(특히, Chrome/FireFox)가 표시하면서, 탭에 표시할 favicon.ico를 전달받지 못한 경우, favicon.ico를 서버에 다시 요청 → 서버에서 브라우저로 전송할 favicon 이미지가 없는 경우, 웹 브라우저로 404 오류를 전송하게 되고, 따라서 다음 URL 요청 등이 처리되지 않은 상황이 발생됩니다.

☞ favicon에 의한 404 오류를 발생시키지 않으려면, favicon 이미지가 전송되도록 서버에 구성합니다.

문서에서는 무료 이미지를 다운로드 받아 사용합니다.

(1) 무료 favicon 이미지나 favicon.ico를 다운로드

→ <https://www.freecodecamp.org/freecodecamp/favicons> 접속

→ 원하는 favicon 이미지 다운로드(문서에서는 Dog 다운로드) → 압축해제

→ favicon.ico 파일이 압축해제 됨(사이즈가 다른 여러 개의 이미지가 있을 수도 있습니다.)

(2) 톰캣 서버에 favicon 사용 구성: 두 가지 방법이 있습니다.

(방법1) favicon.ico (반드시 이름이 favicon.ico어야 함) 파일을 복사 → mypro00 프로젝트 마우스 우버튼 클릭 → Paste(붙여넣기)

(방법2) favicon으로 사용하려는 이미지 파일이 png 같은 다른 형태의 이미지인 경우, 다음의 절차대로 구성합니다.

→ mypro00 프로젝트의 src/main/webapp/resources 폴더에 img 폴더 생성 → img 폴더에 다운로드 된 favicon 이미지 붙여 넣음  
문서 실습에서는 png 형식의 myfavicon.png 와 ico 형식의 favicon.ico (권장)를 붙여넣음

→ src/main/webapp/WEB-INF/views/myinclude/myheader.jsp 파일을 코드작성 뷰에 오픈

→ <head>와 </head> 사이에 다음의 코드를 작성

```
<!-- favicon.ico 404 Error 해결 -->
<!-- favicon 을 사용하지 않도록 설정-->
<!-- <link rel="shortcut icon" href="data:image/x-icon;," type="image/x-icon"> -->
<!-- 다른 형식의 이미지 파일을 favicon.ico 대신 설정(다른 경로의 이미지를 이용할 수도 있음) -->
<!-- <link rel="shortcut icon" href="${contextPath}/resources/img/myfavicon.png" type="image/png"> -->
<link rel="shortcut icon" href="${contextPath}/resources/img/favicon.ico" type="image/x-icon">

</head>
```

☞ type 속성에는 href 속성에 설정되는 favicon 이미지의 유형(.ico: image/x-icon, .png: image/png 등)을 설정합니다.

☞ favicon 구성을 사용하지 않는 경우, 주석을 참고합니다. 문서에서는 방법2의 favicon.ico를 이용하여 구성했습니다.

[조치 완료]

## [08] 게시물 목록 페이지: 페이징 구현

- 다음의 작업들이 진행됩니다.
  - 페이징 데이터 준비(SQL\*Developer에서 작업)
  - 페이징 데이터 저장 객체(DTO) 생성(MyBoardPagingDTO 클래스)
  - MyBoardMapper.xml 파일과 MyBoardMapper 인터페이스 수정
  - MyBoardService와 MyBoardController 수정
  - 게시물 목록화면 페이징 처리 구현
  - 화면 이동간 페이징 정보 유지 구현

☞ 포털 사이트에서 검색 결과나 게시판의 목록을 표시할 때, 표시할 양이 많을 경우, 여러 화면으로 걸쳐서 표시합니다. 따라서, 사용자가 화면을 이동하면서 표시된 내용을 확인하도록 구현하는 방법을 "페이징"이라고 합니다. 게시판이나 검색사이트의 하단에 표시된 아래 그림과 같은 일련의 숫자 링크를 의미합니다.



☞ 페이징 기능을 구현할 때는 다음의 두 가지 부분에 대한 처리를 고려해야 합니다.

- 서버에서 화면에 표시되는 데이터를 가져오기 위한 페이징 처리
- 화면에서의 페이징 처리

☞ 위의 2가지 처리 부분에 대한 고려사항 때문에, 페이징 시 필요한 데이터도 2 종류로 나눕니다

(1) 다음의 값들은 위의 두 가지 부분 구현 시에 모두 사용됩니다(문서에서는 페이징 기본값이라고 부르겠습니다).

- 페이징 번호
- 페이징당 표시되는 항목(행)의 개수

(2) 다음의 값들은 화면에서의 페이징 처리 시에만 사용됩니다.

- 한 화면에 표시되는 시작 페이징 번호와 끝 페이징 번호
- 한 화면에 표시되는 페이징 번호의 개수
- 이전/다음 표시 여부 지정 값
- 마지막 페이지에 표시되는 마지막 페이징 번호

☞ 마지막 페이징 번호를 구하기 위해선 데이터베이스로부터 게시물의 총 수를 구해와야 합니다.

☞ 현장에서는 2가지 방법으로 페이징 기능을 구현합니다.

- 페이징 기본값만 브라우저에게 전달하여, 브라우저의 JavaScript를 이용하여 추가적인 값을 결정해서, 화면에서 페이징 전체 구현
- 페이징 기본값과 화면에서 페이징 구현 시에 필요한 모든 값을 서버에서 브라우저로 전송해서, 화면에서 이를 이용하여 표시 구현

☞ 문서에서는 두 번째 구현방법인 페이징 기본값과 화면에서 페이징 구현 시에 필요한 모든 값을 서버에서 브라우저로 전송하는 방법으로 페이징 기능을 구현합니다.

- ☞ 목록 화면에, 사용자가 선택한 특정 페이지에서 표시될 행들을 가져오는 요청 SELECT문에, 위의 페이징 처리 값(페이지번호와 행수)들을 이용하여 다음처럼 작성합니다.

```

SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag -- 바깥쪽 SELECT문
FROM ( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* -- 안쪽 SELECT문
       FROM book_ex.tbl_myboard a
      WHERE bdelFlag = 0 AND ROWNUM <= (페이지번호 * 표시행수)
    )
WHERE rn >= ((페이지번호 * 표시행수) - (표시행수 - 1))

```

- ☞ /\*+ INDEX\_DESC (a pk\_myboard) \*/ 는, bno에 정의된 인덱스(pk\_myboard)를 이용하여 bno 컬럼값이 큰 값부터 작은 값 순서로 행을 찾아오도록 오라클 엔진에게 지시하는 오라클 SQL-힌트입니다. ORDER BY 절이 없어도 반환된 레코드가 bno 컬럼을 기준으로 내림차순으로 정렬됩니다. ROWNUM은, 오라클 데이터베이스에서 제공하는 1부터 시작되는 정수값으로, 행을 찾은 순서를 의미합니다.

- ☞ (페이지번호 \* 표시행수)는, 특정페이지에 표시되는 행들 중, 찾은 행 번호가 "제일 큰 번호"를 의미합니다. 또한 ((페이지번호 \* 표시행수) - (표시행수 - 1))는, 특정페이지에 표시되는 행들 중, 행 번호가 "제일 작은 번호"를 의미합니다. 예를 들면, **찾은 행 순서로**, 1페이지에는 1 ~ 10, 2페이지에는 11 ~ 20, 3페이지에는 21 ~ 30인 행들이 표시되어야 합니다.

- ☞ 따라서, 위의 SELECT문을 실행시키면, 안쪽 SELECT문 처리에 의해, 특정페이지에 표시되는, "찾은 행 번호가 제일 큰 것"보다 작은 행들만 바깥 쿼리로 반환됩니다(2페이지면, 찾은 행 순서로 1~ 20).

바깥쪽 SELECT문은, 안쪽 SELECT문으로부터 전달된 행들 중, 특정페이지에 표시되는 행들 중  
제일 작은 찾은 행번호보다 큰 행들((2페이지면, 찾은 행 순서로 11보다 큰 행)을 행 개수(10개)만큼 골라냅니다.

요약하면, 위의 SELECT문이 처리되면, bno 컬럼을 기준으로 내림차순으로 행을 찾아서 특정 페이지에 표시할 행 수만큼만 반환됩니다.

## 8-1. 페이징 기능 테스트 데이터 준비(SQL\*Developer에서 작업)

- ☞ 페이징 기능 구현 시, 작동유무를 확인하려면, 많은 수의 데이터가 데이터베이스의 book\_ex.tbl\_myboard 테이블에 존재해야 합니다.

→ SQL\*Developer를 이용하여 book\_ex 계정으로 xe 오라클 데이터베이스 서비스에 접속 후, 다음을 실행합니다.

```

DROP SEQUENCE book_ex.seq_myboard; -- 기존 시퀀스 삭제

CREATE SEQUENCE book_ex.seq_myboard; -- 동일한 시퀀스 생성

TRUNCATE TABLE book_ex.tbl_myboard ; -- 테이블 잘라내기(기존 행 모두 사라짐)
--DELETE FROM book_ex.tbl_myboard ;
--COMMIT ;

INSERT INTO book_ex.tbl_myboard
VALUES (book_ex.seq_myboard.nextval,
        '테스트제목'||book_ex.seq_myboard.currval ,
        '테스트내용'||book_ex.seq_myboard.currval,
        'test'||book_ex.seq_myboard.currval,
        DEFAULT, DEFAULT, DEFAULT, DEFAULT); --7번 수행

```

```
COMMIT ;
```

```
-- 스크립트 출력창에 (10만 이상) 개 행 이 삽입되었다는 메시지가 표시될 때까지 아래 INSERT 문반복 실행  
INSERT INTO book_ex.tbl_myboard (bno, btitle, bcontent, bwriter)  
SELECT book_ex.seq_myboard.nextval, btitle, bcontent, bwriter  
FROM book_ex.tbl_myboard ;
```

COMMIT ; -- 실제 행수는 (10만 이상 \* 2)의 행이 테이블에 저장되어 있게 됩니다. 문서에서는 약45만개의 행이 저장되었습니다.

```
UPDATE book_ex.tbl_myboard  
SET bdelFlag = 1  
WHERE bno LIKE '%8';
```

```
UPDATE book_ex.tbl_myboard  
SET bdelFlag = 1  
WHERE bno LIKE '%9';
```

COMMIT ; --bno 끝자리가 8 또는 9로 끝나는 행은 삭제 요청된 행(약 4만5천개 행)

```
SELECT count(*) FROM book_ex.tbl_myboard; -- 현재 행 개수 확인(20만개 정도의 행수가 표시되면 충분합니다)
```

→ SELECT문으로 TBL\_MYBOARD 테이블에 조회하여 데이터를 확인한 모습

The screenshot shows the Oracle SQL Developer interface with a query window titled 'BOOK.sql' containing the following SQL code:

```
1 SELECT bno, btitle, bcontent, bwriter, bregDate, bmodDate, bviewsCnt, bdelFlag  
2 FROM book_ex.tbl_myboard ORDER BY bno DESC;
```

The results grid displays 10 rows of data from the 'tbl\_myboard' table, ordered by bno in descending order. The columns are: BNO, BTITLE, BCONTENT, BWRITER, BREGDATE, BMODDATE, BVIEWSCNT, and BDELFLAG. The data includes test entries with bno values ranging from 458752 to 458743.

BNO	BTITLE	BCONTENT	BWRITER	BREGDATE	BMODDATE	BVIEWSCNT	BDELFLAG
1 458752	테스트제목7	테스트내용7	test7	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	0
2 458751	테스트제목6	테스트내용6	test6	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	0
3 458750	테스트제목5	테스트내용5	test5	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	0
4 458749	테스트제목4	테스트내용4	test4	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	1
5 458748	테스트제목3	테스트내용3	test3	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	1
6 458747	테스트제목2	테스트내용2	test2	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	0
7 458746	테스트제목1	테스트내용1	test1	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	0
8 458745	테스트제목7	테스트내용7	test7	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	0
9 458744	테스트제목6	테스트내용6	test6	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	0
10 458743	테스트제목5	테스트내용5	test5	2021/02/26 23:49:51	2021/02/26 23:49:51.000000000	0	0

## 8-2. 페이징 기능과 관련된 클래스들이 저장될 패키지 생성.

→ 페이징 기능은 비즈니스 처리와 관계없는 기능(화면에서의 표시관련 기능)이므로, `src/main/java` 폴더에 있는 `com.spring5213.mypro00.common` 패키지에, 서브 패키지로 `paging` 패키지를 생성한 후, 필요한 모듈을 제작합니다.

## 8-3. 게시물 페이징 값들을 저장할 DTO 클래스 생성

→ 생성된 `com.spring5213.mypro00.common.paging` 패키지에 `MyBoardPagingDTO` 클래스를 생성하고, 다음의 코드를 작성합니다.

```
package com.spring5213.mypro00.common.paging;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Setter
@Getter
@ToString
@EqualsAndHashCode
public class MyBoardPagingDTO {

    private int pageNum;           //현재 페이지 번호
    private int rowAmountPerPage;  //페이지당 출력할 레코드 개수

    //생성자를 통해 표시할 페이지번호와 페이지당 출력할 레코드 개수를 컨트롤러로 전달
    //생성자1: list.jsp가 처음 호출 시에, 페이지번호와 행수를 각각 1과 10으로 전달
    public MyBoardPagingDTO() {

        this.pageNum = 1 ;
        this.rowAmountPerPage = 10 ;

    }

    //생성자2: 목록 화면에서 페이징번호 클릭 시, 페이지번호와 행수를 각각 사용자가 선택한 페이징번호와 10으로 전달
    public MyBoardPagingDTO(int pageNum) {
        if(pageNum <= 0) {

            this.pageNum = 1 ;

        } else {

            this.pageNum = pageNum;
        }

        this.rowAmountPerPage = 10 ;
    }

    //생성자3: 목록 화면에서 사용자가 표시할 행수를 선택하고 페이징 번호 클릭 시,
    //페이지번호와 행수를 각각 사용자가 선택한 페이징번호와 표시행수로 전달
    public MyBoardPagingDTO(int pageNum, int rowAmountPerPage) {
        if(pageNum <= 0) {

            this.pageNum = 1 ;
```

```

} else {

    this.pageNum = pageNum;
}

if(rowAmountPerPage <= 0) {

    this.rowAmountPerPage = 10 ;

} else {

    this.rowAmountPerPage = rowAmountPerPage;
}

}
}
}

```

#### 8-4. 게시판 페이지 처리를 위한 Mybatis XML-매퍼 파일과 매퍼 인터페이스 수정

☞ 페이지 데이터를 이용하여 목록정보를 반환하는 SELECT문과 호출 메서드를 각각 XML-매퍼 파일과 매퍼 인터페이스에 추가합니다.

```

→ src/main/resources/com/spring5212/mypro00/mapper/MyBoardMapper.xml을 코드작성 뷰에 오픈
→ 기존 게시물 목록 조회 SELECT문을 주석처리 후, 다음의 빨간색 SELECT문 추가

<!-- 게시물 조회 - 목록 --><!--
<select id="selectMyBoardList" resultType="com.spring5213.mypro00.domain.MyBoardVO">
    SELECT * FROM book_ex.tbl_myboard
    WHERE bno > 0 AND bDelFlag = 0
    ORDER BY bno DESC
</select> -->

<!-- 게시물 조회 - 목록2: 페이지 고려: 삭제 요청된 행 제외 --><!--
<select id="selectMyBoardList" resultType="com.spring5213.mypro00.domain.MyBoardVO">
<![CDATA[
    SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag
    FROM ( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.*
            FROM book_ex.tbl_myboard a
            WHERE bno > 0 AND bdelFlag = 0 AND ROWNUM <= ( #{pageNum} * #{rowAmountPerPage} )
        )
    WHERE rn >= ( ( #{pageNum} * #{rowAmountPerPage} ) - ( #{rowAmountPerPage} - 1 ) )
]]>
</select>-->

<!-- 게시물 조회 - 목록2: 페이지 고려: 삭제 요청 된 행 포함 -->
<select id="selectMyBoardList" resultType="com.spring5213.mypro00.domain.MyBoardVO">
<![CDATA[
    SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag
    FROM ( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.*
            FROM book_ex.tbl_myboard a
            WHERE ROWNUM <= ( #{pageNum} * #{rowAmountPerPage} )
        )
    WHERE rn >= ( ( #{pageNum} * #{rowAmountPerPage} ) - ( #{rowAmountPerPage} - 1 ) )
]]>
</select>

```

☞ 위에 2개의 페이징을 고려한 SELECT문 코드가 있습니다. 실습에서는 삭제 요청된 행도 "사용자 요청으로 삭제됨"으로 화면에서 표시되도록 구현했으므로, 빨간색 SELECT문을 사용합니다. 주석처리(녹색) 된 다른 SELECT문은 `bdelFlag = 0` 조건이 추가되어 있으므로 삭제요청이 되지 않은 게시물 정보만 제공합니다.

→ src/main/java/com.spring5213.mypro00.mapper.MyBoardMapper.java 파일을 코드작성 뷰에 오픈하고, 기존 게시물 목록 조회 메서드를 주석처리 후, 페이징 데이터를 매개변수로 가지는 다음의 게시물 목록 조회 메서드를 추가합니다.

```
import com.spring5213.mypro00.common.paging.MyBoardPagingDTO;

...(생략)...

//게시물 조회 - 목록
//public List<MyBoardVO> selectMyBoardList();

//게시물 조회 - 목록(페이징 고려)
public List<MyBoardVO> selectMyBoardList(MyBoardPagingDTO myBoardPagingDTO);
```

☞ 위의 메서드를 추가하면, MyBoardPagingDTO를 임포트해야 합니다(Ctrl + Shift + O 키로 자동 완성됨).

☞ 메서드 추가 후에, 프로젝트의 MyBoardServiceImpl에 오류가 표시됩니다. 메서드 명세가 바뀌었기 때문입니다.

#### [테스트]

→ src/test/java/com.spring5213.mypro00.mapper.MyBoardMapperTests.java 파일을 코드작성 뷰에 오픈하고, MyBatis 매퍼에 새로 추가된 SELECT문을 테스트 하기 위한 다음의 메서드 추가(다른 테스트 메서드는 모두 주석처리).

```
import com.spring5213.mypro00.common.paging.MyBoardPagingDTO;

//게시물 목록 조회 테스트2 - 페이징 고려
@Test
public void testSelectBoardListWithPaging() {

    MyBoardPagingDTO myBoardPagingDTO = new MyBoardPagingDTO(); //기본 생성자 이용(1, 10)
    myBoardMapper.selectMyBoardList(myBoardPagingDTO).forEach(myBoard -> System.out.println(myBoard));

    myBoardPagingDTO = new MyBoardPagingDTO(2, 10);
    myBoardMapper.selectMyBoardList(myBoardPagingDTO).forEach(myBoard -> System.out.println(myBoard));
}
```

☞ 메소드 추가 시, 필요한 클래스에 대한 import 문도 포함시켰습니다. 메서드 추가 후, Ctrl + Shift + O 키를 동시에 누르면 필요한 클래스가 자동으로 import 되어 코드가 완성되므로, import는 직접 작성하지 않습니다.

→ 테스트 수행

→ 코드 작성 뷰에 표시된 MyBoardMapperTests.java 코드에서 마우스 우측버튼 클릭 → Run As → 2 JUnit Test 클릭

→ 테스트 실행 완료 시 JUnit 테스트 뷰에 녹색 진행 바가 표시됨

→ 녹색이 아닌 자주색 바가 표시되면, 테스트 실패 → 원인을 찾아 해결한 후, 다시 JUnit 테스트 수행

→ 콘솔에 표시된 로그 확인

[콘솔 로그 내용 일부(테스트 관련 빈 구성 로그와 HikariDataSource 빈 관련 로그는 생략)]

```

INFO : jdbc.sqlonly - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= ( 1 * 10 ) ) WHERE rn >= ( ( 1 * 10 ) - ( 10 - 1 ) )

INFO : jdbc.sqltiming - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= ( 1 * 10 ) ) WHERE rn >= ( ( 1 * 10 ) - ( 10 - 1 ) )
{executed in 112 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|458752 |테스트제목7 |테스트내용7 |test7 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458751 |테스트제목6 |테스트내용6 |test6 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458750 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458749 |테스트제목4 |테스트내용4 |test4 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |1
|458748 |테스트제목3 |테스트내용3 |test3 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |1
|458747 |테스트제목2 |테스트내용2 |test2 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458746 |테스트제목1 |테스트내용1 |test1 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458745 |테스트제목7 |테스트내용7 |test7 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458744 |테스트제목6 |테스트내용6 |test6 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458743 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
MyBoardVO(bno=458752, btitle=테스트제목7, bcontent=테스트내용7, bwriter=test7, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458751, btitle=테스트제목6, bcontent=테스트내용6, bwriter=test6, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458750, btitle=테스트제목5, bcontent=테스트내용5, bwriter=test5, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458749, btitle=테스트제목4, bcontent=테스트내용4, bwriter=test4, bviewsCnt=0, breplyCnt=0, bdelFlag=1, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458748, btitle=테스트제목3, bcontent=테스트내용3, bwriter=test3, bviewsCnt=0, breplyCnt=0, bdelFlag=1, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458747, btitle=테스트제목2, bcontent=테스트내용2, bwriter=test2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458746, btitle=테스트제목1, bcontent=테스트내용1, bwriter=test1, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458745, btitle=테스트제목7, bcontent=테스트내용7, bwriter=test7, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458744, btitle=테스트제목6, bcontent=테스트내용6, bwriter=test6, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458743, btitle=테스트제목5, bcontent=테스트내용5, bwriter=test5, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
INFO : jdbc.sqlonly - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= ( 2 * 10 ) ) WHERE rn >= ( ( 2 * 10 ) - ( 10 - 1 ) )

INFO : jdbc.sqltiming - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= ( 2 * 10 ) ) WHERE rn >= ( ( 2 * 10 ) - ( 10 - 1 ) )
{executed in 1 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|458742 |테스트제목4 |테스트내용4 |test4 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458741 |테스트제목3 |테스트내용3 |test3 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458740 |테스트제목2 |테스트내용2 |test2 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458739 |테스트제목1 |테스트내용1 |test1 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |1
|458738 |테스트제목7 |테스트내용7 |test7 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |1
|458737 |테스트제목6 |테스트내용6 |test6 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458736 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458735 |테스트제목4 |테스트내용4 |test4 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458734 |테스트제목3 |테스트내용3 |test3 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458733 |테스트제목2 |테스트내용2 |test2 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
MyBoardVO(bno=458742, btitle=테스트제목4, bcontent=테스트내용4, bwriter=test4, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458741, btitle=테스트제목3, bcontent=테스트내용3, bwriter=test3, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458740, btitle=테스트제목2, bcontent=테스트내용2, bwriter=test2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458739, btitle=테스트제목1, bcontent=테스트내용1, bwriter=test1, bviewsCnt=0, breplyCnt=0, bdelFlag=1, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458738, btitle=테스트제목7, bcontent=테스트내용7, bwriter=test7, bviewsCnt=0, breplyCnt=0, bdelFlag=1, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458737, btitle=테스트제목6, bcontent=테스트내용6, bwriter=test6, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458736, btitle=테스트제목5, bcontent=테스트내용5, bwriter=test5, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458735, btitle=테스트제목4, bcontent=테스트내용4, bwriter=test4, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458734, btitle=테스트제목3, bcontent=테스트내용3, bwriter=test3, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
MyBoardVO(bno=458733, btitle=테스트제목2, bcontent=테스트내용2, bwriter=test2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0)
```

## 8-5. MyBoardService 수정

☞ 게시물 목록 조회 메소드에 MyBoardPagingDTO를 매개변수로 받아서 처리하도록 수정합니다.

→ src/main/java/com.spring5213.mypro00.service.MyBoardService.java 파일을 코드 작성 뷰에 오픈,

기존 getBoardList() 메서드를 다음처럼 수정

```
import com.spring5213.mypro00.common.paging.MyBoardPagingDTO;

//게시물 목록 조회 서비스 - 페이지 고려
public List<MyBoardVO> getBoardList(MyBoardPagingDTO myBoardPagingDTO);
```

→ src/main/java/com.spring5213.mypro00.service.MyBoardServiceImpl.java 파일을 코드 작성 뷰에 오픈,

기존 getBoardList() 메서드를 다음처럼 수정

```
import com.spring5213.mypro00.common.paging.MyBoardPagingDTO;

//게시물 목록 조회 서비스 - 페이지 고려
@Override
public List<MyBoardVO> getBoardList(MyBoardPagingDTO myBoardPagingDTO){
    log.info("MyBoardService.getBoardList() 실행");
    return myBoardMapper.selectMyBoardList(myBoardPagingDTO);
}
```

→ [테스트] src/test/java/com.spring5213.mypro00.service.MyBoardServiceTests.java 파일을 코드작성 뷰에 오픈하고,

게시물 목록 테스트 메서드 수정(다른 테스트 메서드는 모두 주석처리).

```
import com.spring5213.mypro00.common.paging.MyBoardPagingDTO;

//게시물 목록 조회 서비스 테스트
@Test
public void testGetBoardList() {
    //페이지 고려 안함
    //myBoardService.getBoardList().forEach(myBoard -> log.info(myBoard)); ←주석처리

    //페이지 고려
    myBoardService.getBoardList(new MyBoardPagingDTO(2, 10)).forEach(myBoard -> log.info(myBoard)); ← 추가
}
```

☞ 메소드 추가 시, 필요한 클래스에 대한 import 문도 포함시켰습니다. 메서드 추가 후, Ctrl + Shift + O 키를 동시에 누르면 필요한 클래스가 자동으로 import 되어 코드가 완성되므로, import는 직접 작성하지 않습니다.

→ 테스트 수행 → 코드 작성 뷰에 표시된 MyBoardMapperTests.java 코드에서 마우스 우측버튼 클릭 → Run As → 2 JUnit Test 클릭

→ 콘솔에 표시된 로그 확인

[콘솔 로그 내용 일부(테스트 관련 빈 구성 로그와 HikariDataSource 빈 관련 로그는 생략)]

```
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoardList() 실행
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - 서비스에 전달된 페이지처리 데이터 객체: MyBoardPagingDTO(pageNum=2, rowAmountPerPage=10)
INFO : jdbc.sqlonly - SELECT bno, btitle, bcontent, bwritter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= ( 2 * 10 ) ) WHERE rn >= ( ( 2 * 10 ) - ( 10 - 1 ) )

INFO : jdbc.sqltiming - SELECT bno, btitle, bcontent, bwritter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
```

```

<= ( 2 * 10 ) ) WHERE rn >= ( ( 2 * 10 ) - ( 10 - 1 ) )
{executed in 85 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|
| bno | btitle | bcontent | bwriter | bregdate | bmoddate | bviewsCnt | breplyCnt | bdelFlag |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 458742 | 테스트제목4 | 테스트내용4 | test4 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 0 |
| 458741 | 테스트제목3 | 테스트내용3 | test3 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 0 |
| 458740 | 테스트제목2 | 테스트내용2 | test2 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 0 |
| 458739 | 테스트제목1 | 테스트내용1 | test1 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 1 |
| 458738 | 테스트제목7 | 테스트내용7 | test7 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 1 |
| 458737 | 테스트제목6 | 테스트내용6 | test6 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 0 |
| 458736 | 테스트제목5 | 테스트내용5 | test5 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 0 |
| 458735 | 테스트제목4 | 테스트내용4 | test4 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 0 |
| 458734 | 테스트제목3 | 테스트내용3 | test3 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 0 |
| 458733 | 테스트제목2 | 테스트내용2 | test2 | 2021-02-26 23:49:51.0 | 2021-02-26 23:49:51.0 | 0 | 0 | 0 |
|-----|-----|-----|-----|-----|-----|-----|-----|

```

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458742, btitle=테스트제목4, bcontent=테스트내용4, bwriter=test4, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458741, btitle=테스트제목3, bcontent=테스트내용3, bwriter=test3, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458740, btitle=테스트제목2, bcontent=테스트내용2, bwriter=test2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458739, btitle=테스트제목1, bcontent=테스트내용1, bwriter=test1, bviewsCnt=0, breplyCnt=0, bdelFlag=1, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458738, btitle=테스트제목7, bcontent=테스트내용7, bwriter=test7, bviewsCnt=0, breplyCnt=0, bdelFlag=1, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458737, btitle=테스트제목6, bcontent=테스트내용6, bwriter=test6, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458736, btitle=테스트제목5, bcontent=테스트내용5, bwriter=test5, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458735, btitle=테스트제목4, bcontent=테스트내용4, bwriter=test4, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458734, btitle=테스트제목3, bcontent=테스트내용3, bwriter=test3, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

INFO : com.spring5213.mypro00.service.MyBoardServiceTests - MyBoardVO(bno=458733, btitle=테스트제목2, bcontent=테스트내용2, bwriter=test2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0)

## 8-6. MyBoardController 설정

☞ 게시물 목록 조회 메소드에 MyBoardPagingDTO를 매개변수로 받아서 처리하도록 수정합니다.

→ src/main/java/com.spring5213.mypro00.controller.MyBoardController.java 파일을 코드 작성 뷰에 오픈,

기존 showBoardList() 메서드를 다음처럼 수정

```

import com.spring5213.mypro00.common.paging.MyBoardPagingDTO;

//게시물 목록 조회 - 페이지 고려
@GetMapping("/list")
public void showBoardList(MyBoardPagingDTO myBoardPagingDTO, Model model) {
    log.info("컨트롤러 - 게시물 목록 조회 시작.....");
    log.info("컨트롤러에 전달된 사용자의 페이지처리 데이터: " + myBoardPagingDTO);

    model.addAttribute("boardList", myBoardService.getBoardList(myBoardPagingDTO));
    log.info("컨트롤러 - 게시물 목록 조회 완료.....");
}

```

[테스트]

→ src/test/java/com.spring5213.mypro00.service.MyBoardControllerTests.java 파일을 코드작성 뷰에 오픈하고,

게시물 목록 테스트 메서드 수정(다른 테스트 메서드는 모두 주석처리).

```

//게시물 목록 조회 테스트
@Test

```

```

public void testShowBoardList() throws Exception {
    log.info(mockMvc.perform(MockMvcRequestBuilders.get("/myboard/list")
        .param("pageNum", "2") //페이지 테스트 시 추가
        .param("rowAmount", "10") //페이지 테스트 시 추가
    )
    .andReturn()
    .getModelAndView()
    .getModelMap());
}

```

- 테스트 수행 → 코드 작성 뷰에 표시된 MyBoardMapperTests.java 코드에서 마우스 우측버튼 클릭 → Run As → 2 JUnit Test 클릭  
→ 테스트 수행 후, 콘솔에 표시된 로그 확인

[콘솔 로그 내용 일부(테스트 관련 빈 구성 로그와 HikariDataSource 빈 관련 로그는 생략)]

```

INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 목록 조회 시작.....
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러에 전달된 사용자의 페이지링처리 데이터: MyBoardPagingDTO(pageNum=2,
rowAmountPerPage=10)
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoardList() 실행
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - 서비스에 전달된 페이지링처리 데이터 객체: MyBoardPagingDTO(pageNum=2, rowAmountPerPage=10)
INFO : jdbc.sqlonly - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= ( 2 * 10 ) ) WHERE rn >= ( ( 2 * 10 ) - ( 10 - 1 ) )

INFO : jdbc.sqltiming - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= ( 2 * 10 ) ) WHERE rn >= ( ( 2 * 10 ) - ( 10 - 1 ) )
{executed in 95 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|458742|테스트제목4|테스트내용4|test4|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|0
|458741|테스트제목3|테스트내용3|test3|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|0
|458740|테스트제목2|테스트내용2|test2|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|0
|458739|테스트제목1|테스트내용1|test1|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|1
|458738|테스트제목7|테스트내용7|test7|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|1
|458737|테스트제목6|테스트내용6|test6|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|0
|458736|테스트제목5|테스트내용5|test5|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|0
|458735|테스트제목4|테스트내용4|test4|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|0
|458734|테스트제목3|테스트내용3|test3|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|0
|458733|테스트제목2|테스트내용2|test2|2021-02-26 23:49:51.0|2021-02-26 23:49:51.0|0|0|0
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 목록 조회 완료.....
INFO : com.spring5213.mypro00.controller.MyBoardControllerTests - {myBoardPagingDTO=MyBoardPagingDTO(pageNum=2, rowAmountPerPage=10),
org.springframework.validation.BindingResult.myBoardPagingDTO=org.springframework.validation.BeanPropertyBindingResult: 0 errors,
boardList=[MyBoardVO(bno=458742, btitle=테스트제목4, bcontent=테스트내용4, bwriter=test4, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26
23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0), MyBoardVO(bno=458741, btitle=테스트제목3, bcontent=테스트내용3, bwriter=test3, bviewsCnt=0,
breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0), MyBoardVO(bno=458740, btitle=테스트제목2,
bcontent=테스트내용2, bwriter=test2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0),
MyBoardVO(bno=458739, btitle=테스트제목1, bcontent=테스트내용1, bwriter=test1, bviewsCnt=0, breplyCnt=0, bdelFlag=1, bregDate=Fri Feb 26 23:49:51
KST 2021, bmodDate=2021-02-26 23:49:51.0), MyBoardVO(bno=458738, btitle=테스트제목7, bcontent=테스트내용7, bwriter=test7, bviewsCnt=0,
breplyCnt=0, bdelFlag=1, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0), MyBoardVO(bno=458737, btitle=테스트제목6,
bcontent=테스트내용6, bwriter=test6, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0),
MyBoardVO(bno=458736, btitle=테스트제목5, bcontent=테스트내용5, bwriter=test5, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021,
bmodDate=2021-02-26 23:49:51.0), MyBoardVO(bno=458735, btitle=테스트제목4, bcontent=테스트내용4, bwriter=test4, bviewsCnt=0, breplyCnt=0,
bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0), MyBoardVO(bno=458734, btitle=테스트제목3,
bcontent=테스트내용3, bwriter=test3, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021, bmodDate=2021-02-26 23:49:51.0),
MyBoardVO(bno=458733, btitle=테스트제목2, bcontent=테스트내용2, bwriter=test2, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=Fri Feb 26 23:49:51 KST 2021,
bmodDate=2021-02-26 23:49:51.0}]
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown completed.

```

- 웹 브라우저에서의 테스트 수행

- 템켓 서버 기동 → 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/list?pageNum=2&rowAmount=10> URL로 접속

번호	제목	작성자	작성일	수정일	조회수
458742	테스트제목5	test5	2021/06/21 Mon Jun 21 08:28:13 KST 2021	2021/06/21 08:28:13 2021-06-21 08:28:13.0	0
458741	테스트제목4	test4	2021/06/21 Mon Jun 21 08:28:13 KST 2021	2021/06/21 08:28:13 2021-06-21 08:28:13.0	0
458740	테스트제목3	test3	2021/06/21 Mon Jun 21 08:28:13 KST 2021	2021/06/21 08:28:13 2021-06-21 08:28:13.0	0
458739	작성자에 의하여 삭제된 게시글입니다.				
458738	작성자에 의하여 삭제된 게시글입니다.				
458737	테스트제목7	test7	2021/06/21 Mon Jun 21 08:28:13 KST 2021	2021/06/21 08:28:13 2021-06-21 08:28:13.0	0
458736	테스트제목6	test6	2021/06/21 Mon Jun 21 08:28:13 KST 2021	2021/06/21 08:28:13 2021-06-21 08:28:13.0	0
458735	테스트제목5	test5	2021/06/21 Mon Jun 21 08:28:13 KST 2021	2021/06/21 08:28:13 2021-06-21 08:28:13.0	0
458734	테스트제목4	test4	2021/06/21 Mon Jun 21 08:28:13 KST 2021	2021/06/21 08:28:13 2021-06-21 08:28:13.0	0
458733	테스트제목3	test3	2021/06/21 Mon Jun 21 08:28:13 KST 2021	2021/06/21 08:28:13 2021-06-21 08:28:13.0	0

→ URL에서 pageNum을 바꿔가며 표시된 화면을 확인해 봅니다.

## 8-7. 화면에서의 페이징에 의한 게시물 목록 표시 기능 구현

☞ 단원 서두에 다음과 같은 화면에서 페이징 처리(페이징 번호 표시등)를 위해 필요한 데이터들의 의미를 설명했습니다.

- 한 화면에 표시되는 시작 페이징 번호와 끝 페이징 번호
  - 한 화면에 표시되는 페이징 번호의 개수: 지정해야 함
  - 이전/다음 표시 여부 지정 값
  - 마지막 페이지에 표시되는 마지막 페이징 번호
- ☞ 마지막 페이징 번호를 구하기 위해선 데이터베이스로부터 게시물의 총 수를 알아야 합니다.

☞ 구현 시에, 다음과 같은 추가적인 고려사항이 있습니다.

1 페이지부터 10 페이지까지는, 사용자가 원하는 데이터가 표시되는 화면의 페이징 번호를 클릭하더라도 화면 하단에 표시된 페이징 번호는 동일하게 1 ~ 10까지 표시됩니다.

동일한 방식으로 사용자가 선택한 페이지 번호가 11 페이지부터 20 페이지 사이의 번호일 때,

화면에 표시되는 페이징 번호들은 11 ~ 20 으로 동일하게 표시됩니다.

☞ 문서에서는, 페이징 이동 간에 동일하게 표시되는 페이징 번호들을 그룹으로 간주합니다.

☞ 또 다른 고려사항은, 예를 들어 설명합니다.

전체 게시물 총수는 250개, 한 페이지당 10개 행과 10개의 페이지 번호가 표시된다고 가정합니다.

이런 경우, 실제 전체 페이지 번호는 1 ~ 25 ( $250/10$ ) 까지입니다.

페이지 번호들은 1 ~ 10 그룹, 11 ~ 20 그룹, 그리고 마지막 그룹은 21 ~ 30이 아니라, 실제로 21 ~ 25까지가 표시되는 페이지 번호들이 됩니다.

간단히 요약하면, 맨마지막 페이지 그룹의 마지막 페이지 번호는, 실제 총 행수를 기준으로 구해진 마지막 페이지가 됩니다.

☞ 그리고, 클릭으로 화면에 표시된 목록정보의 페이지번호는, 항상 일정한 범위의 페이지 번호들의 그룹 중 하나의 페이지 숫자입니다.

- 위의 내용을 기준으로, 한 화면에 표시되는 시작 페이지 번호와 끝 페이지 번호는 다음의 방법으로 구해집니다.

페이지 데이터	구하는 방법
끝 페이지번호	(int) ( Math.ceil( 선택된 페이지 번호 / ( (double) 페이지번호 개수 ) ) * 페이지 번호 개수 )
시작 페이지번호	끝 페이지 번호 - ( 페이지번호 개수 - 1 )
실제 마지막 페이지 번호	(int) ( Math.ceil( 전체 게시물 총수 / ( (double) 페이지당 표시되는 행수 ) ) )

☞ Math.ceil(실수) 메서드는, 실수보다 큰 가장 작은 정수를 double 데이터유형으로 반환합니다.

따라서, 위와 같이 괄호를 적절히 사용하고, 적절한 Casting을 시켜야 원하는 값을 얻을 수 있습니다.

- 위의 공식으로 구하면, 선택된 페이지 번호가 22일 경우, 끝 페이지 번호가 30으로 구해집니다.

따라서, 제일 마지막 페이지 번호는 실제 마지막 페이지 번호로 처리되어야 합니다.

- 이전 표시 여부 지정 값은, 시작 페이지 번호가 1 보다 크면, 표시되도록 값을 설정하고,

다음 표시 여부 지정 값은, 끝 페이지번호가 실제 마지막 페이지번호 보다 작은 경우에 표시되도록 값을 설정합니다.

☞ 앞에서 설명한 페이지 값들을 화면으로 전달하기 위하여 다음의 내용을 구현합니다.

- 게시물 전체 수를 가져오는 매퍼 메서드 및 관련 SQL 문 추가

- 페이지 값이 저장되는 클래스 생성

- 값이 저장된 클래스의 객체를 게시물 목록 정보와 같이 Model 객체에 저장하여 컨트롤러를 통해 웹 브라우저에 전송

### (1) SQL 매퍼 파일과 매퍼 인터페이스 수정

☞ SQL 매퍼 파일에 게시물 총수를 구하는 SELECT문을 추가하고, 매퍼 인터페이스에 관련 매서드를 추가합니다.

→ src/main/resources/com/spring5212/mypro00/mapper 폴더의 MyBoardMapper.xml을 코드 작성 뷰에 오픈

→ 게시물의 총 수를 반환하는 다음의 SELECT문 추가

```
<!-- 게시물 총 개수 조회(페이지): 삭제 요청된 행 제외 -->
<!-- 아래 문장 사용 시, bdelFlag, bno를 키로 가지는 인덱스 추가 요망(성능) --><!--
<select id="selectRowAmountTotal" resultType="Long">
<![CDATA[
    SELECT count(*) FROM book_ex.tbl_myboard
    WHERE bno > 0 AND bdelFlag = 0
]]>
</select>-->

<!-- 게시물 총 개수 조회(페이지): 삭제 요청된 행 포함 -->
<select id="selectRowAmountTotal" resultType="Long">
<![CDATA[
    SELECT count(*) FROM book_ex.tbl_myboard
]]>
</select>
```

☞ 위에 2개의 게시물 총 개수를 조회하는 SELECT문 코드가 있습니다. 실습에서는 삭제 요청된 행도 포함해서 총 수를 구하는 빨간색 SELECT문을 사용합니다. 주석처리된 다른 SELECT문은 `bdelFlag = 0` 조건이 추가되어 있으므로 삭제요청이 되지 않은 게시물의 총 수를 조회합니다.

→ `src/main/java/com.spring5213.mypro00.mapper.MyBoardMapper.java` 인터페이스를 코드 작성 뷰에 오픈

→ SQL 매퍼에 추가된 게시물 총 수를 조회하는 SELECT문을 호출하는 메서드 추가

```
//게시물 총 개수 조회(페이징)  
public Long selectRowAmountTotal(MyBoardPagingDTO myBoardPagingDTO);
```

☞ 위에서 `MyBoardPagingDTO`를 매개변수로 전달받지 않아도 되지만, 이 후의 검색구현에서는 필요하므로 미리 구현해 놓습니다.

## (2) `MyBoardServiceImpl` 구현 객체 클래스와 `MyBoardService` 인터페이스 수정

☞ 비즈니스 계층의 서비스 인터페이스와 구현클래스에 게시물 총 수를 처리하는 메서드를 추가합니다.

→ `src/main/java/com.spring5213.mypro00.service.MyBoardService.java` 인터페이스를 코드 작성뷰에 오픈

→ `getRowAmountTotal()` 메서드 추가

```
//게시물 총 개수 조회 서비스 - 페이징 시 필요  
public Long getRowAmountTotal(MyBoardPagingDTO myBoardPagingDTO);
```

→ `src/main/java/com.spring5213.mypro00.service.MyBoardServiceImpl.java` 구현 클래스를 코드 작성뷰에 오픈

→ `getRowAmountTotal()` 재정의 메서드 추가

```
//게시물 총 개수 조회 서비스 - 페이징 시 필요  
@Override  
public Long getRowAmountTotal(MyBoardPagingDTO myBoardPagingDTO) {  
    log.info("MyBoardService.getRowAmountTotal()에 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);  
    return myBoardMapper.selectRowAmountTotal(myBoardPagingDTO);  
}
```

### (3) 페이지 데이터 저장 클래스 생성: MyBoardCreatingPagingDTO 클래스

☞ MyBoardCreatingPagingDTO 클래스에 페이지 값들이 저장되어 웹 브라우저에 전달됩니다.

→ src/main/java/com.spring5213.mypro00.common.paging 패키지에 MyBoardPagingCreatorDTO 클래스를 생성하고 아래의 내용을 작성합니다.

```
package com.spring5213.mypro00.common.paging;

import lombok.Getter;
import lombok.ToString;

@Getter
@ToString
public class MyBoardPagingCreatorDTO {

    private MyBoardPagingDTO myBoardPagingDTO; //페이지번호와 행 개수 저장 객체
    private int startPagingNum; //화면에 표시되는 시작 페이지 번호
    private int endPagingNum; //화면에 표시되는 끝 페이지 번호
    private boolean prev; //이전 버튼 표시 여부 결정 변수(true: 표시됨, false: 표시 안됨)
    private boolean next; //다음 버튼 표시 여부 결정 변수(true: 표시됨, false: 표시 안됨)
    private long rowAmountTotal; //전체 행 개수
    private int pagingNumCnt; //화면 하단에 표시할 기본 페이지 번호 개수(10)
    private int lastPageNum; //행 총수를 기준으로 한 실제 마지막 페이지 번호

    public MyBoardPagingCreatorDTO(long rowAmountTotal, MyBoardPagingDTO myBoardPagingDTO ) {
        this.myBoardPagingDTO = myBoardPagingDTO ;
        this.rowAmountTotal = rowAmountTotal;
        this.pagingNumCnt = 10;

        //계산된 끝-시작 페이지 번호:
        this.endPagingNum =
            (int)( Math.ceil(myBoardPagingDTO.getPageNum() / (this.pagingNumCnt * 1.0) ) ) * this.pagingNumCnt ;

        this.startPagingNum = this.endPagingNum - (this.pagingNumCnt -1);

        //행 총수를 기준으로 한 실제 마지막 페이지 번호 저장
        this.lastPageNum = (int)( Math.ceil( (rowAmountTotal * 1.0) / myBoardPagingDTO.getRowAmountPerPage() ) );

        //계산된 끝 페이지 번호가 실제 마지막 페이지 번호보다 크면, endPagingNum 값을 lastPageNum 값으로 대체
        if (lastPageNum < this.endPagingNum) {
            this.endPagingNum = lastPageNum ;
        }

        this.prev = this.startPagingNum > 1 ;
        this.next = this.endPagingNum < lastPageNum ;

        System.out.println("전달된 페이지 기본데이터-myBoardPagingDTO: " + myBoardPagingDTO.toString());
        System.out.println("끝 페이지번호: " + this.endPagingNum);
        System.out.println("시작 페이지번호: " + this.startPagingNum);
        System.out.println("이전버튼 표시 여부: " + this.prev);
        System.out.println("다음버튼 표시 여부: " + this.next);
        System.out.println("마지막 페이지 번호: " + this.lastPageNum);
    }
}
```

#### (4) MyBoardController 클래스 수정

☞ 게시물 목록 조회 메서드에 페이징 데이터를 사용자의 브라우저로 전달할 수 있도록 구현합니다.

→ src/main/java/com.spring5213.mypro00.controller.MyBoardController.java 클래스를 코드 작성부에 오픈

→ showBoardList() 메서드를 수정(빨간색 코드 추가)합니다.

```
//게시물 목록 조회 2 - 페이징 고려
@GetMapping("/list")
public void showBoardList(MyBoardPagingDTO myBoardPagingDTO, Model model) {
    log.info("컨트롤러 - 게시물 목록 조회 시작.....");
    log.info("컨트롤러에 전달된 사용자의 페이징처리 데이터: " + myBoardPagingDTO);

    model.addAttribute("boardList", myBoardService.getBoardList(myBoardPagingDTO));

    Long rowAmountTotal = myBoardService.getRowAmountTotal(myBoardPagingDTO);
    log.info("컨트롤러에 전달된 게시물 총 개수: " + rowAmountTotal);

    MyBoardPagingCreatorDTO myBoardPagingCreatorDTO =
        new MyBoardPagingCreatorDTO (rowAmountTotal, myBoardPagingDTO);
    log.info("컨트롤러에서 생성된 MyBoardPagingCreatorDTO 객체 정보: " + myBoardPagingCreatorDTO.toString());

    model.addAttribute("pagingCreator", myBoardPagingCreatorDTO );
    log.info("컨트롤러 - 게시물 목록(페이징고려) 조회 완료.....");
}
```

#### [테스트 - 웹 브라우저]

→ 템켓 서버 기동, → 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/list?pageNum=1&rowAmount=10> URL로 접속

#### [콘솔 로그 내용]

```
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 목록 조회 시작.....
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러에 전달된 사용자의 페이징처리 데이터: MyBoardPagingDTO(pageNum=1,
rowAmountPerPage=10)
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoardList() 실행
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - 서비스에 전달된 페이징처리 데이터 객체: MyBoardPagingDTO(pageNum=1, rowAmountPerPage=10)
INFO : jdbc.sqlonly - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= ( 1 * 10 ) ) WHERE rn >= ( ( 1 * 10 ) - ( 10 - 1 ) )

INFO : jdbc.sqltiming - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= ( 1 * 10 ) ) WHERE rn >= ( ( 1 * 10 ) - ( 10 - 1 ) )
```

```

{executed in 210 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewsCnt |breplyCnt |bdelflag |
|-----|-----|-----|-----|-----|-----|-----|-----|
|458752 |테스트제목7 |테스트내용7 |test7 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458751 |테스트제목6 |테스트내용6 |test6 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458750 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458749 |테스트제목4 |테스트내용4 |test4 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |1
|458748 |테스트제목3 |테스트내용3 |test3 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |1
|458747 |테스트제목2 |테스트내용2 |test2 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458746 |테스트제목1 |테스트내용1 |test1 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458745 |테스트제목7 |테스트내용7 |test7 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458744 |테스트제목6 |테스트내용6 |test6 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|458743 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0
|-----|-----|-----|-----|-----|-----|-----|-----|

```

INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getRowAmountTotal()에 전달된 MyBoardPagingDTO:  
MyBoardPagingDTO(pageNum=1, rowAmountPerPage=10)

INFO : jdbc.sqlonly - SELECT count(\*) FROM book\_ex.tbl\_myboard

INFO : jdbc.sqltiming - SELECT count(\*) FROM book\_ex.tbl\_myboard  
{executed in 17 msec}

INFO : jdbc.resultsettable -

count(*)
458752

INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러에 전달된 게시물 총 개수: 458752  
전달된 페이징 기본데이터-myBoardPagingDTO: MyBoardPagingDTO(pageNum=1, rowAmountPerPage=10)  
끝 페이징번호: 10  
시작 페이징번호: 1  
이전버튼 표시 여부: false  
다음버튼 표시 여부: true  
마지막 페이지 번호: 45876

INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러에서 생성된 MyBoardPagingCreatorDTO 객체 정보:  
MyBoardPagingCreatorDTO(myBoardPagingDTO=MyBoardPagingDTO(pageNum=1, rowAmountPerPage=10), startPagingNum=1, endPagingNum=10, prev=false, next=true, rowAmountTotal=458752, pagingNumCnt=10, lastPageNum=45876)

INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 목록 조회 완료.....

→ 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/list?pageNum=45876&rowAmount=10> URL(맨 마지막 페이지)로 접속

번호	제목	작성자	작성일	수정일	조회수
2	테스트제목2	test2	2021/06/21 Mon Jun 21 08:22:57 KST 2021	2021/06/21 08:22:58 2021-06-21 08:22:58.0	0
1	테스트제목1	test1	2021/06/21 Mon Jun 21 08:22:54 KST 2021	2021/06/21 08:22:55 2021-06-21 08:22:55.0	0

## 8-8. 화면에서의 페이징 숫자 표시 구현

☞ 사용자의 웹 브라우저에 표시되는 화면은 JSP-페이지에 의해서 생성된 HTML 내용입니다. 따라서, 게시물 목록 HTML을 생성하는 list.jsp 파일에 화면에 표시되는 페이징 번호들을 구현합니다. 실습에서는 Bootstrap 3.3.2 버전의 한글 매뉴얼을 제공하는 사이트에 접속하여, 필요한 소스를 구해서 작업합니다.

→ 웹 브라우저에서 <http://bootstrapk.com/components/#pagination> 의 URL로 접속(한글 Bootstrap 3 매뉴얼 사이트)

→ src/main/webapp/WEB-INF/views/myboard/list.jsp 파일을 코드 뷰에 오픈

→ 한글 부트스트랩 3 매뉴얼 사이트의 설명을 참조하면서 아래의 빨간색 코드를 list.jsp 파일의 </table> 태그 밑에 작성합니다.

```
...(생략)...
</table><!-- /.table-responsive -->
<!-- Pagination 시작 -->
<div class='pull-right'>
    <ul class="pagination pagination-sm">
        <!-- 페이징 버튼 클릭 시, jQuery로 페이지 번호를 전달하도록 a 태그에 전달된 pagingCreator 객체의 필드 지정 -->
        <c:if test="${pagingCreator.prev}">
            <li class="paginate_button previous">
                <a href="1">&laquo;</a><!-- 맨 앞으로 페이지로 이동 -->
            </li>
        </c:if>
        <c:if test="${pagingCreator.prev}">
            <li class="paginate_button previous">
                <a href="${pagingCreator.startPagingNum - 1}">이전</a><!-- 이전 페이징 그룹 끝 페이지로 이동 -->
            </li>
        </c:if>
        <!-- 페이징 그룹의 페이징 숫자(10개 표시) -->
        <c:forEach var="pageNum" begin="${pagingCreator.startPagingNum}" end="${pagingCreator.endPagingNum}">
            <!-- 선택된 숫자의 경우, Bootstrap의 active 클래스 이름 추가 -->
            <li class="paginate_button">
                <li class="paginate_button ${pagingCreator.myBoardPagingDTO.pageNum == pageNum ? "active":"" }">
                    <a href="${pageNum}">${pageNum}</a>
                </li>
            </c:forEach>
            <c:if test="${pagingCreator.next}">
                <li class="paginate_button next">
                    <a href="${pagingCreator.endPagingNum +1}">다음</a><!-- 다음 페이징 그룹의 첫 페이지로 이동 -->
                </li>
            </c:if>
            <c:if test="${pagingCreator.next}">
                <li class="paginate_button next">
                    <a href="${pagingCreator.lastPageNum}">&raquo;</a><%-- 맨 마지막으로 페이지로 이동 --%>
                </li>
            </c:if>
        </ul>
    </div><!-- Pagination 끝 -->

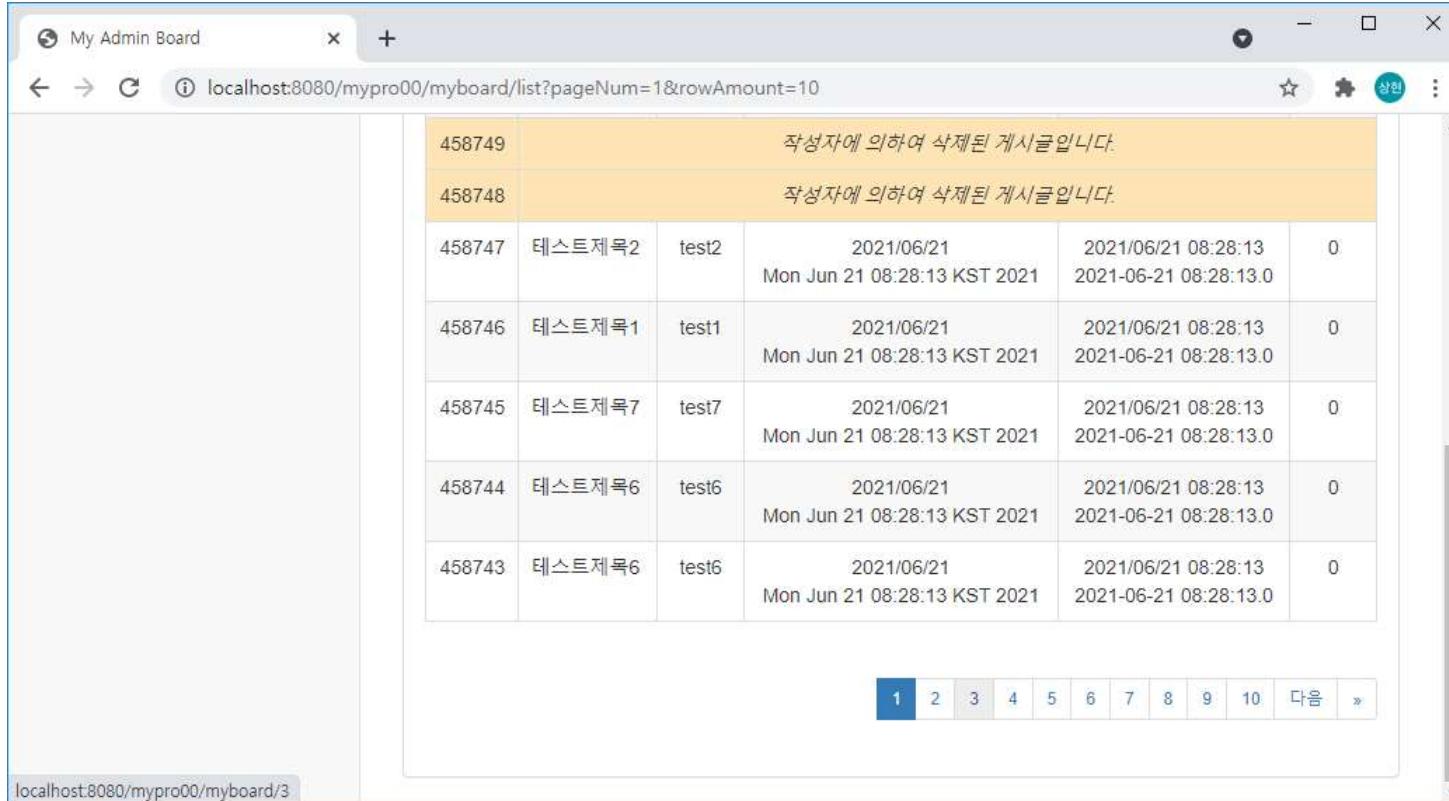
<%-- Modal 모달 시작--%>
...(생략)...
```

## [테스트] 웹 브라우저에서의 테스트 수행

→ 톰캣 서버 기동

→ 웹 브라우저에서 `http://localhost:8080/mypro00/myboard/list?pageNum=1&rowAmount=10` URL로 접속

→ 웹 브라우저에서 화면에 표시된 페이지 번호에 마우스를 위치시키면, 브라우저 왼쪽 하단에 이동되는 URL이 표시됨



## 8-9. 페이지 번호를 이용한 게시물 목록 화면 이동 구현

☞ 페이지 번호 버튼 클릭 시, 이동하려는 페이지 번호를 전달받아, 화면 이동이 수행되도록 `list.jsp` 파일에 jQuery 이벤트 동작 함수를 정의합니다. 이 때, 페이지 번호와 행수를 앞의 실습으로 생성한 `form`에 추가한 후, 이를 전송해서 서버에 요청되도록 구현합니다.

→ `src/main/webapp/WEB-INF/views/myboard/list.jsp` 파일을 코드 작성 뷰에 오픈

→ `list.jsp` 파일에 정의한 `id`가 `frmSendValue`인 `form`에 다음 빨간색 코드 추가

```
<%-- 전달할 hidden 유형의 input 요소들이 추가되어 값들이 전달될 비어있는 form --%>
<%-- 목록 화면에서 페이지 화면 이동 시, 페이지 데이터 전달을 위해 사용됨 --%>
<form id="frmSendValue">
    <input type='hidden' name='pageNum' value='${pagingCreator.myBoardPagingDTO.pageNum}'>
    <input type='hidden' name='rowAmountPerPage' value='${pagingCreator.myBoardPagingDTO.rowAmountPerPage}'>
    <input type='hidden' name='lastPageNum' value='${pagingCreator.lastPageNum}'>
</form>
```

→ 다음의 빨간색 jQuery 코드를 제일 밑의 `<%@ include file="../myinclude/myfooter.jsp" %>` 위에 추가

```
<script>
<%-- //페이지 버튼 클릭 이벤트 처리: 폼에 저장된 페이지번호를 클릭한 페이지번호로 변경한 후, 전송//--%>
//페이지 화면이동
$(".paginate_button a").on("click", function(e) {
```

```

e.preventDefault(); //a 태그의 클릭 시 동작 막음

//폼에 저장된 현재 화면의 페이지번호를 클릭한 페이징 버튼의 페이지번호로 변경
frmSendValue.find("input[name='pageNum']").val($(this).attr("href"));
frmSendValue.attr("action", "${contextPath}/myboard/list");
frmSendValue.attr("method", "get");
frmSendValue.submit();
});

</script>

<%@ include file="../myinclude/myfooter.jsp" %>

```

→ 게시물 등록날짜와 수정날짜에 2개의 값이 표시되는 것을 하나의 값이 표시되도록 수정

```

<c:forEach items="${boardList}" var="board"><!-- 컨트롤러에서 보낸 목록객체 이름: boardList --%>

...(생략)...
<td><fmt:formatDate pattern="yyyy/MM/dd" value="${board.bregDate}" /><br>
    <%-- ${board.bregDate} --%> ←주석처리 또는 삭제
</td>
<td><fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="${board.bmodDate}" /><br>
    <%-- ${board.bmodDate} --%> ←주석처리 또는 삭제
</td>
<td>${board.bviewsCnt}</td>
</tr>
</c:if>
</c:forEach>

</tbody>
</table><!-- /.table-responsive -->
...(생략)...

```

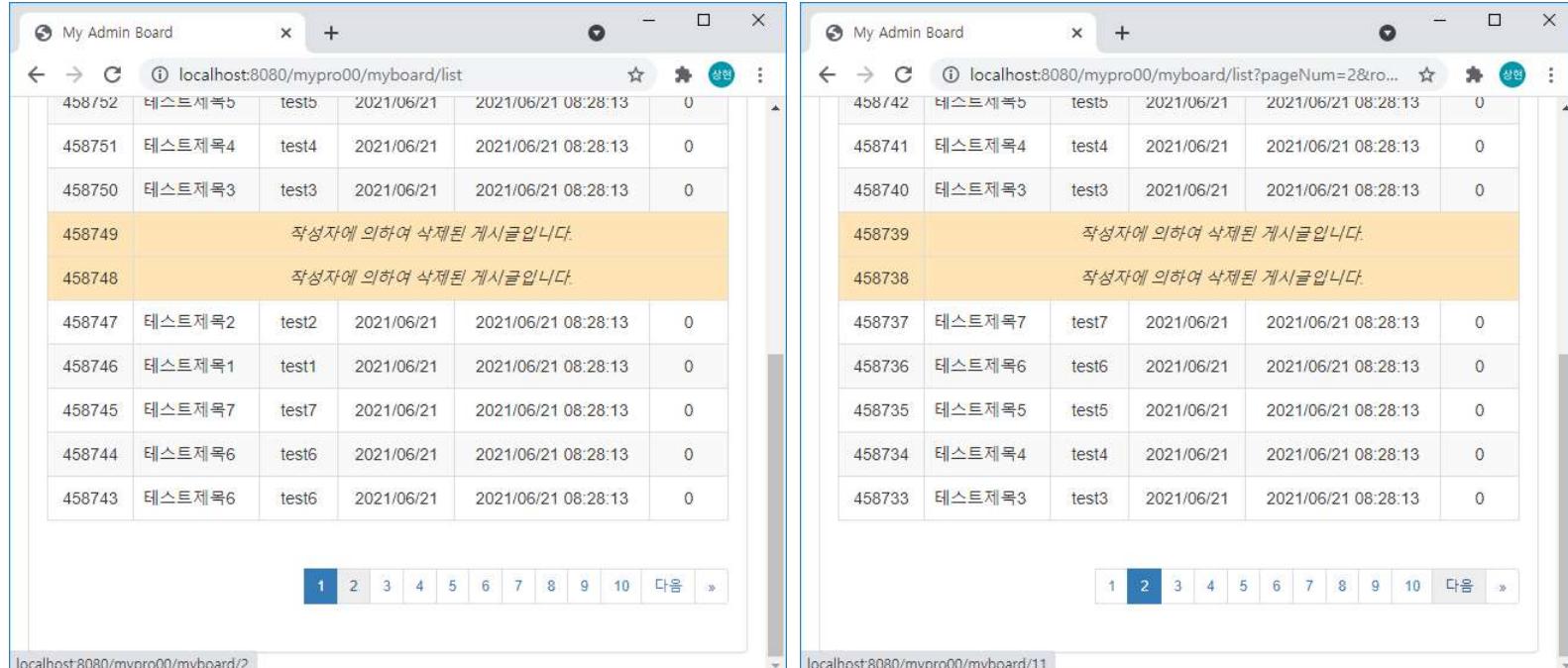
→ [테스트] 웹 브라우저에서의 테스트 수행

☞ 각 페이징 버튼을 클릭하면서, 페이지 이동, 게시물 내용 표시가 잘 수행되는지 테스트합니다.

→ 텐켓 서버 기동

→ 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/list> URL로 접속(웹 브라우저의 크기를 줄여놓고 테스트 함)

→ 2 클릭 → 다음 클릭



→ 맨 마지막 페이지로 이동 버튼[>>] 클릭 → 이전 버튼 클릭

The left screenshot shows a list of posts on page 11. Several posts are highlighted in yellow, indicating they have been deleted. The right screenshot shows the same list on page 10, where the deleted posts are no longer present.

→ 맨 앞 페이지 이동 버튼 [<<] 클릭

The left screenshot shows a list of posts on page 1. Several posts are highlighted in yellow, indicating they have been deleted. The right screenshot shows the same list on page 2, where the deleted posts are no longer present.

## 8-10. 화면 이동 간 페이징 처리 고려

☞ 현재 구현 상태까지는 목록화면에서 등록, 상세 화면 이동 후, 다시 목록 화면으로 이동 시에 무조건 1 페이지로 이동합니다.  
등록화면 → 목록화면 이동 시에는 1페이지로 이동하여 등록된 게시물을 확인할 수 있으므로 문제가 없습니다.

☞ 하지만 다음처럼 목록화면 → 다른 화면 → 목록화면으로 이동 시에는, 페이징 값들을 유지시켜, 처음의 목록 페이지가 표시되도록 해야 합니다.

- 게시물 목록화면 → 게시물 상세 화면 이동 → 게시물 목록화면 이동 시
- 게시물 목록화면 → 게시물 상세 화면 → 게시물 수정-삭제 화면, 삭제 처리 후 → 게시물 목록화면 이동 시
- 게시물 목록화면 → 게시물 상세 화면 → 게시물 수정-삭제 화면, 수정 처리 후 → 게시물 상세화면 → 목록화면 이동 시

☞ 화면이동 시에, 페이지 값들이 유지되도록 구현하는 방법으로, a 태그의 href 속성에 값을 추가하여 전달할 수도 있고, form에 hidden 유형의 값으로 추가하여 전달할 수도 있습니다. 문서에서는 form을 이용하여 페이지 값들이 화면이동 시에 전달되어 유지되도록 처리합니다.

(1) 게시물 목록화면 → 게시물 상세 화면 → 게시물 목록화면 이동 시 페이지 데이터 처리

☞ 게시물 목록화면에서 frmSendValue 아이디의 form을 통해, 페이지 데이터가 게시물 상세 화면 요청을 처리하는 컨트롤러의 메서드에 전달되도록, list.jsp 파일에 구현하는 것은, 페이지 번호 간 이동 구현 시에 이미 완료되었습니다.

☞ 따라서, 상세 화면 이동을 처리하는 컨트롤러의 메서드가 페이지 값을 받아서 상세 화면으로 전달할 수 있도록 수정합니다. 그리고, 게시물 상세 화면을 생성하는 JSP 페이지(detail.jsp)가 컨트롤러로부터 전달받은 페이지 데이터를 유지할 수 있도록 구현합니다.

→ src/main/java/com.spring5213.mypro00.controller/MyBoardController.java 클래스를 코드 작성 뷰에 오픈

→ showBoardDetail() 메서드 수정(빨간색 코드 추가)

```
// 특정 게시물 조회 페이지 호출: 목록페이지에서 호출됨
@GetMapping("/detail")
public void showBoardDetail(@RequestParam("bno") Long bno, Model model,
                           @ModelAttribute("myBoardPagingDTO") MyBoardPagingDTO myBoardPagingDTO ) {
    log.info("컨트롤러 - 게시물 조회 페이지 호출: " + bno);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    model.addAttribute("board", myBoardService.getBoard(bno));
    log.info("컨트롤러 - 화면으로 전달할 model: " + model);
}
```

☞ @ModelAttribute("myBoardPagingDTO") MyBoardPagingDTO myBoardPagingDTO 매개변수에 의해, 폼으로부터 메서드에 전달된 페이지 데이터는 myBoardPagingDTO 객체에 저장되고, Model 객체에 자동으로 myBoardPagingDTO 이름의 속성으로 추가되어 게시물 상세 화면을 생성하는 JSP 페이지(detail.jsp)로 전달됩니다.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지를 코드 작성 뷰에 오픈

→ frmSendValue 아이디의 form을 추가하고, 버튼 클릭 이벤트 처리 jQuery 코드를 수정 및 빨간색 코드를 추가합니다

```
<form id="frmSendValue"><!-- 폼을 추가(list.jsp와 동일한 아이디의 form) -->
    <input type='hidden' name='bno' id="bno" value='<c:out value="${board.bno}" />'>
    <input type='hidden' name='pageNum' value='${myBoardPagingDTO.pageNum}'>
    <input type='hidden' name='rowAmountPerPage' value='${myBoardPagingDTO.rowAmountPerPage}'>
</form>

<script>
var frmSendValue = $("#frmSendValue");

// 게시물 수정 페이지로 이동: 폼의 값을 전송해서 이동하는 형태로 변경
$("#BtnMoveModify").on("click", function(){
    //location.href='${contextPath}/myboard/modify?bno=<c:out value="${board.bno}" />'; ← 주석처리
    frmSendValue.attr("action", "${contextPath}/myboard/modify");
    frmSendValue.attr("method", "get");
    frmSendValue.submit();
})
```

```
//게시물 목록 페이지로 이동: 품의 값을 전송해서 이동하는 형태로 변경
$("#BtnMoveList").on("click", function(){
    //location.href="${contextPath}/myboard/list"; ←주석처리
    frmSendValue.find("#bno").remove(); // 목록화면 이동 시, bno 값 삭제
    frmSendValue.attr("action", "${contextPath}/myboard/list");
    frmSendValue.attr("method", "get");
    frmSendValue.submit();
})
</script>

<%@ include file="../myinclude/myfooter.jsp" %>
```

-- 목록화면 → 상세화면 → 목록화면 이동 처리 완료됨

(2) 게시물 상세화면 → 게시물 수정(삭제) 화면 이동 시 페이징 데이터 처리

☞ 게시물 상세화면에서의 구현(detail.jsp)은 앞 섹션에서 완료되었습니다.

☞ 따라서, 게시물 수정-삭제 화면으로 이동을 처리하는 컨트롤러의 메서드가 페이징 값을 받아서 수정-삭제 화면으로 전달할 수 있도록 수정합니다. 그리고, 게시물 수정-삭제 화면을 생성하는 JSP 페이지(modify.jsp)가 컨트롤러로부터 전달받은 페이징 데이터를 유지할 수 있도록 구현합니다. 단, modify.jsp 파일에는 이미 frmModify 아이디의 form이 있으므로 이를 이용해서 구현합니다.

→ src/main/java/com.spring5213.mypro00.controller/MyBoardController.java 클래스를 코드 작성 뷰에 오픈

→ showBoardModify() 메서드 수정(빨간색 코드 추가)

```
//특정 게시물 수정 페이지 호출: 조회 페이지에서 호출됨
//@ModelAttribute() 어노테이션을 사용하지 않고 구현
@GetMapping("/modify")
public void showBoardModify(@RequestParam("bno") Long bno, Model model,
                           MyBoardPagingDTO myBoardPagingDTO) {
    log.info("컨트롤러 - 게시물 수정 페이지 호출: " + bno);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    model.addAttribute("myBoardPagingDTO", myBoardPagingDTO); ← model 객체에 myBoardPagingDTO 속성으로 추가
    model.addAttribute("board", myBoardService.getBoardDetailModify(bno));
    log.info("컨트롤러 - 화면으로 전달할 model: " + model);
}
```

☞ 앞의 showBoardDetail() 메서드와 달리 showBoardModify() 메서드는 @ModelAttribute() 어노테이션을 사용하지 않고 구현합니다. 동일한 기능을 구현하는 다양한 방법을 학습하고자 합니다. 메서드로 전달된 페이징 데이터를 myBoardPagingDTO 매개변수에 저장되고, 이를 model 객체에 myBoardPagingDTO 이름의 속성으로 직접 추가하여 modify.jsp 페이지로 전달합니다.

→ src/main/webapp/WEB-INF/views/myboard/modify.jsp 페이지를 코드 작성 뷰에 오픈

→ 다음처럼, 버튼 밑에 빨간색 코드(hidden 유형의 input, 서버로부터 전달된 페이징 데이터가 저장됨)를 추가합니다.

... (생략) ...

```
<button type="button" class="btn btn-default" id="btnModify" data-oper="modify">수정</button>
<button type="button" class="btn btn-danger" id="btnRemove" data-oper="remove">삭제</button>
<button type="button" class="btn btn-info" id="btnList" data-oper="list">취소</button>
<%-- 추가 --%>
<input type='hidden' name='pageNum' value='${myBoardPagingDTO.pageNum}'>
<input type='hidden' name='rowAmountPerPage' value='${myBoardPagingDTO.rowAmountPerPage}'>
</form>
</div><!-- /.panel-body -->
```

```

</div><!-- /.panel -->
</div><!-- /.col-lg-12 -->
</div><!-- /.row -->
</div><!-- #page-wrapper -->
<script>

...(생략)...

```

-- 상세화면 → 수정-삭제 화면 이동 처리 완료됨

(3) 게시물 수정-삭제 화면 → 게시물 수정(서버 처리) → 게시물 상세 화면 이동 시 페이징 데이터 처리

☞ 게시물 수정-삭제 화면(modify.jsp)과 게시물 상세 화면에서의 구현(detail.jsp)은 앞 섹션에서 완료되었습니다.

☞ 따라서, 컨트롤러의 게시물 수정 메서드가, "페이징 값을 전달받고, 게시물 수정 처리 후, 게시물 상세 화면으로 전달"할 수 있도록 수정합니다.

☞ 그런데 수정 후, 게시물 상세 페이지 호출을 어떤 방식으로 처리할지를 결정해야 합니다.

이 때, 서버에서의 게시물 수정 요청은 **POST 방식으로 처리되며**, 처리 후의 게시물 상세 화면 이동은 URL 매핑을 통하여 JSP 페이지를 호출할 수 없고(URL은 /myboard/modify, 필요한 JSP는 myboard/detail.jsp 다름), 또한 "myboard/detail" 문자열을 반환해서 JSP 페이지를 호출하면, RedirectAttribute를 이용한 일회성 값을 전달 기능을 사용할 수 없습니다.

☞ 게시물 상세 페이지는 **GET 방식으로 호출되도록 구현했으므로**, 리다이렉트 방법을 이용하는 것이 지금 구현된 상태에서는 최선입니다. 그런데, Redirect 방식으로는 Model이나 @ModelAttribute를 사용해서 JSP 페이지로 페이징 데이터를 전달할 수 없으므로, **RedirectAttributes를 이용하여 값을 전달하도록 구현합니다.** 이 때, 값은 **MyBoardPagingDTO 같은 여러 개의 값을 가지는 객체로 전달될 수 없고, 하나의 값만 가진 객체유형으로 값을 전달해야 합니다.**(시도하면 아래의 오류가 발생됩니다)

```

WARN : org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver - Resolved
[org.springframework.beans.ConversionNotSupportedException: Failed to convert value of type
'com.spring5213.mypro00.common.paging.MyBoardPagingDTO' to required type 'java.lang.String'; nested exception is java.lang.IllegalStateException:
Cannot convert value of type 'com.spring5213.mypro00.common.paging.MyBoardPagingDTO' to required type 'java.lang.String': no matching editors or
conversion strategy found]

```

☞ RedirectAttribute 인터페이스는 스프링에서 제공하며, 컨트롤러에서 리다렉트 방식으로 다른 JSP 페이지로 이동 시에 전달할 값을 속성으로 추가하여 GET 방식으로 편하게 전달시킬 수 있습니다.

☞ 게시물 수정 후, 브라우저에 표시되는 게시물 상세 화면은, 게시물 조회수 증가를 방지하기 위하여, showBoardDetailMod() 메소드가 처리하도록 구현했으므로, showBoardDetailMod() 메소드가 페이징 데이터를 전달받아 detail.jsp 페이지로 전달되도록 구현합니다.

→ src/main/java/com.spring5213.mypro00.controller/MyBoardController.java 클래스를 코드 뷰에 오픈

→ showBoardModify() 메서드와 showBoardDetailMod() 메서드 수정(빨간색 코드 추가)

```

//게시물 수정 후 -> 조회페이지 호출 시(/detailmod)
//게시물 데이터를 화면까지 전달하기 위하여 Model을 사용
//RedirectAttributes를 사용하지 않음.

@GetMapping("/detailmod")
public String showBoardDetailMod(@RequestParam("bno") Long bno,
                                 Model model,           //JSP로 전달할 데이터 객체들을 저장
                                 MyBoardPagingDTO myBoardPagingDTO) { //전달된 페이징 값을 저장
    log.info("컨트롤러 - 게시물 수정 페이지 호출: " + bno);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    //수정 후 DB에 저장된 데이터를 다시 가져와서 model에 속성으로 바인딩
    model.addAttribute("board", myBoardService.getBoardDetailMod(bno));
}

```

```

model.addAttribute("myBoardPagingDTO", myBoardPagingDTO);
log.info("컨트롤러 - 화면으로 전달할 model: " + model);

return "myboard/detail";
}

//특정 게시물 수정 처리
//RedirectAttributes를 사용하는 방식
//model을 사용하지 않음.
@PostMapping("/modify")
public String modifyBoard( MyBoardVO myBoard,
                           RedirectAttributes redirectAttr,      //전달할 페이지 값을 저장
                           MyBoardPagingDTO myBoardPagingDTO){ //전달된 페이지 값을 저장
log.info("컨트롤러 - 게시물 수정 전달된 myBoard 값: " + myBoard);
log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

if (myBoardService.modifyBoard(myBoard)) {
    redirectAttr.addFlashAttribute("result", "successModify");
}
//RedirectAttribute 유형의 객체에 전달할 페이지 데이터를 속성으로 바인딩
redirectAttr.addFlashAttribute("bno", myBoard.getBno());
redirectAttr.addFlashAttribute("pageNum", myBoardPagingDTO.getPageNum());
redirectAttr.addFlashAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());

log.info("화면으로 전달될 redirectAttr: " + redirectAttr);

return "redirect:/myboard/detailmod" ;
}

```

-- 수정 화면 → (서버 처리) 게시물 수정 → 상세 화면 이동 처리 완료됨

(4) 게시물 수정-삭제 화면 → 게시물 삭제(서버 처리) → 게시물 목록 화면 이동 시 페이지 데이터 처리

☞ 게시물 수정-삭제 화면(modify.jsp)과 게시물 목록 화면에서의 구현(detail.jsp)은 앞 섹션에서 완료되었습니다.

☞ 따라서, 컨트롤러의 게시물 삭제 메서드가, "페이지 값을 전달받고, 게시물 삭제 처리 후, 게시물 목록 화면으로 전달" 할 수 있도록 수정합니다.

☞ 앞의 게시물 수정 시의 구현 방법과 동일하게 **RedirectAttributes**를 이용하여 페이지 값을 전달하도록 구현합니다.

→ src/main/java/com.spring5213.mypro00.controller 패키지의 **MyBoardController.java** 를 코드 뷰에 오픈  
 → 삭제처리와 관련된 메서드들을 동일한 방식으로 수정합니다(실제 사용되는 메서드는 bdelFlag를 1로 수정하는 메서드임).  
 빨간색 코드 추가만 하면 됩니다.

```

//특정 게시물 삭제 요청
@PostMapping("/delete")
public String setBoardDeleted(@RequestParam("bno") Long bno,
                           RedirectAttributes redirectAttr,      //전달할 페이지 값을 저장하는 객체
                           MyBoardPagingDTO myBoardPagingDTO ){ //전달된 페이지 값을 받음
log.info("컨트롤러 - 게시물 삭제(bdelFlag값변경 글번호): " + bno);
log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

if (myBoardService.setBoardDeleted(bno)) {
    redirectAttr.addFlashAttribute("result", "successRemove");
}

```

```

    redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
    redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
    log.info("화면으로 전달될 redirectAttr: " + redirectAttr);

    return "redirect:/myboard/list";
}

//특정 게시물 삭제 - 실제 삭제됨
@PostMapping("/remove")
public String removeBoard(@RequestParam("bno") Long bno,
                           RedirectAttributes redirectAttr,
                           MyBoardPagingDTO myBoardPagingDTO) {
    log.info("컨트롤러 - 게시물 삭제: 삭제되는 글번호: " + bno);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    if (myBoardService.removeBoard(bno)) {
        redirectAttr.addFlashAttribute("result", "successRemove");
    }

    redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
    redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
    log.info("화면으로 전달될 redirectAttr: " + redirectAttr);

    return "redirect:/myboard/list";
}

//삭제된 총 행수를 model 대신 RedirectAttributes로 전달, model 필요없음
//게시물 삭제 - 삭제 요청된 모든 게시물 삭제
@PostMapping("/removeAll")
public String removeAllDeletedBoard(RedirectAttributes redirectAttr,           //model 삭제,
                                     MyBoardPagingDTO myBoardPagingDTO) { //전달된 페이지 값들을 받음 추가
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    redirectAttr.addFlashAttribute("result", "successRemoveAll");
    //RedirectAttribute 유형의 객체에 전달할 페이지 데이터를 속성으로 바인딩
    redirectAttr.addAttribute("removedRowCnt", myBoardService.removeAllDeletedBoard());
    redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
    redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
    log.info("화면으로 전달될 redirectAttr: " + redirectAttr);

    return "redirect:/myboard/list";
}

```

-- 수정 화면 → (서버 처리) 게시물 삭제 → 목록 화면 이동 처리 완료됨

(5) 게시물 수정-삭제 화면 → 취소 버튼 클릭 → (목록 생성 서버 처리) → 게시물 목록 화면 이동 시 페이지 데이터 처리

☞ 수정/삭제 화면에서 취소버튼을 클릭하여 목록 화면 이동 시에는, form에 입력된 값들은 없애고, 페이지 데이터만 보내도록  
취소 버튼 클릭 이벤트 처리 jQuery 코드만 수정

→ src/main/webapp/WEB-INF/views/myboard/modify.jsp 페이지를 코드 작성 뷰에 오픈하고

취소 버튼 클릭 이벤트 처리 jQuery 코드 수정(빨간색 코드)

```

<script>

//form의 수정/삭제/목록보기 버튼 클릭 이벤트 처리
var frmModify = $("#frmModify");

```

```

$( 'button' ).on("click", function(e){

    //e.preventDefault(); //버튼 유형이 submit가 아니므로 설정할 필요 없음

    var operation = $(this).data("oper"); //각 버튼의 data-xxx 속성에 설정된 값을 저장
    //alert("operation: "+ operation);

    if(operation == "modify"){ //게시물 수정 요청
        frmModify.attr("action", "${contextPath}/myboard/modify");

    } else if(operation == "remove"){ //게시물 삭제 요청
        frmModify.attr("action", "${contextPath}/myboard/delete"); //bdelFlag만 업데이트
        //frmModify.attr("action", "${contextPath}/myboard/remove"); //실제 행 삭제

    } else if(operation == "list"){ //게시물 목록 화면 요청
        //기존 페이징 데이터 input 요소 복사
        var pageNumInput = $("input[name='pageNum']").clone(); //추가
        var rowAmountInput = $("input[name='rowAmountPerPage']").clone(); //추가

        frmModify.empty();

        frmModify.attr("action", "${contextPath}/myboard/list").attr("method","get");
        //복사된 input 요소를 다시 form에 추가
        frmModify.append(pageNumInput); //추가
        frmModify.append(rowAmountInput); //추가
    }

    frmModify.submit() ; //요청 전송
});

</script>

```

<%@ include file="../myinclude/myfooter.jsp" %>

-- 수정화면, 취소 클릭 → 목록화면 이동 처리 완료됨

#### [테스트]

→ Tomcat 서버를 기동한 후, 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/list> URL로 접속

→ 목록 → 상세 → 수정 → 목록 화면간 이동 시에 웹 브라우저의 URL 입력 부분의 주소에 페이지번호, 표시행수가 유지되는지 확인합니다.

단원 [08] 끝

## [09] 게시물 목록 페이지: 게시물 검색 구현

- 다음의 작업들이 진행됩니다.
  - 대표적인 게시물 검색 기준 확인 및 검색 기능 구현 시 고려사항
  - 매퍼 XML 파일에 검색을 위한 SQL 작성: MyBatis의 동적 태그 이용
  - 검색기능 구현
  - 화면 이동간 검색-페이지 유지 구현

### ☞ 대표적인 게시물 검색 기준

- 검색범위: 제목, 내용, 작성자 중 하나 이상을 검색범위로 선택하며, 처리 시에는 데이터베이스의 컬럼이 검색범위가 됩니다.
- 검색어: 선택된 검색범위에서 값들에 포함된 문자열을 지정하며, 처리 시, 데이터베이스 컬럼에 저장된 값에 포함된 문자열이 됩니다.

☞ 검색은 데이터베이스에서 검색어가 포함된 검색범위의 컬럼 들을 찾으면, 해당 컬럼값이 포함된 행을 목록화면에 전송하여 표시하는 것으로 처리됩니다. 화면에서 사용자에 의해 지정된 검색어(입력)와 검색범위(선택)는, 데이터베이스에 요청되는 SELECT문에서 사용되며, 데이터베이스에서 처리된 후 표시할 결과가 많을 경우 페이지으로 처리되어야 합니다.

따라서, 검색어와 검색범위도, 기본 페이지 값 저장 객체를 통해 서버에 전달되도록 구현됩니다.

☞ 주의할 것은 검색범위가 여러 개의 조합으로 구성될 수 있으므로 서로 다른 WHERE-조건절을 가지는 다수의 SQL문장이 필요하게 됩니다. 이런 상황은, **하나의 SQL문에 Mybatis의 동적 태그들을 이용하여**, 서로 다른 문장이 동일한 서버의 메서드에 전달되도록 구현할 수 있습니다.

☞ 표시할 목록정보를 데이터베이스로부터 가져오는 것은, 지금까지 게시물 목록 표시를 위해 사용되었던 SELECT문에 검색처리가 가능하도록, Mybatis의 동적 태그들을 이용하여, 조건절을 추가해서 구현합니다.

☞ 검색처리를 위해 필요한 검색범위와 검색어를 처리할 필드를 페이지 데이터 객체(MyBoardPagingDTO)에 추가합니다.

☞ 매퍼 인터페이스, 비즈니스 계층(서비스), 제어 계층(컨트롤러)에, 페이지 관련 처리를 모두 구현했기 때문에 수정할 것이 없습니다.

### ☞ 구현할 내용은 다음과 같습니다.

- MyBoardPagingDTO 클래스에 검색 기능을 위해 필요한 검색어, 검색범위 처리 필드 추가
- MyBoardPagingDTO 클래스에 사용자가 선택한 검색범위를 처리할 메서드 추가
  - ☞ 처리 로직은 뒤에서 설명합니다.
- 매퍼-XML 파일의 게시물 목록 조회 SELECT문장 수정
- 게시물 목록을 처리하여 표시하는 list.jsp 페이지에 사용자가 검색범위와 검색어를 입력하고 이것이 서버로 전달되도록 수정
- 화면 이동간 검색된 결과 목록의 페이지 유지 구현
  - ☞ 사용자 브라우저에서의 화면이동(예를 들면, 목록 화면 → 상세 화면 → 수정-삭제 화면 → 목록화면 이동) 시에도, 검색된 결과를 기반한 처음의 목록 화면이 표시될 수 있도록 페이지 기능이 유지되어야 합니다.

☞ 구현 시, 이해를 돋기 위하여, JSP 페이지에서의 구현을 먼저 수행합니다.

그런 후, MyBoardPagingDTO 클래스를 수정하고, Mybatis 매퍼를 구현합니다.

## 9-1. 게시물 목록 표시 JSP 페이지(list.jsp) 수정

☞ 게시물 목록 화면에서 사용자가 검색 기능을 사용할 수 있도록, 필요한 HTML 요소를 추가합니다.

☞ 추가된 요소들과 관련된 이벤트 처리 jQuery 실행문을 추가합니다.

- 표시 행수 선택 이벤트
- 검색 버튼 클릭 이벤트
- 검색 초기화 버튼 클릭 이벤트
- 검색 범위 선택 및 검색어 입력 이벤트

(1) 브라우저에서 사용자가 검색 기능을 사용하기 위해 필요한 HTML 요소 추가

☞ 다음의 HTML 요소를 목록 표시 table 요소 위에 추가합니다.

- 검색범위 선택을 위한 select 요소
- 검색어 입력을 위한 input 요소
- 전송을 위한 버튼
- 표시되는 게시물 개수를 선택할 수 있는 select 요소
- 재 검색을 할 수 있도록 기존 검색범위와 검색어를 초기화 하는 버튼

☞ 요소 추가 시, 페이징 데이터를 전달하기 위해 사용했던 frmSendValue 아이디의 form을 이용합니다.

→ src/main/webapp/WEB-INF/views/myboard/list.jsp 페이지를 코드 작성 뷰에 오픈, 다음의 지시대로 코드 작성

→ <script> 위에 작성된 <form id="frmSendValue"> 부터~ </form> 까지 내용을 잘라내기

→ <table> 시작 태그 위에 붙여넣기

→ 옮겨진 form에 다음의 HTML 요소를 추가(아래 빨간색 코드)

- 표시되는 게시물 개수를 선택할 수 있는 select 요소
- 검색범위 선택을 위한 select 요소
- 검색어 입력을 위한 input 요소
- 전송을 위한 버튼
- 재 검색을 할 수 있도록 기존 검색범위와 검색어를 초기화 하는 버튼

◦ 표시 행수, 선택된 검색범위, 입력된 검색어가 페이징 이동간에도 유지되도록, 컨트롤러부터 받아온 값을 value 속성에 설정.

...(생략)...

```
<div class="panel-body">

<%-- 전달할 hidden 유형의 input 요소들이 추가되어 값들이 전달될 비어있는 form --%>
<%-- 목록 화면에서 페이징 화면 이동 시, 페이징 데이터 전달을 위해 사용됨 --%>
<%-- 표시 행수/검색유형/검색어 입력 form 시작 --%>
<form class="form-inline" id="frmSendValue" action="${contextPath}/myboard/list" method="get">
    <div class="form-group">
        <label class="sr-only">frmSendValues</label>
        <select class="form-control" id="selectAmount" name="rowAmountPerPage"><!-- 표시 게시물 수 선택 -->
            <option value="10" <c:out value="${pagingCreator.myBoardPagingDTO.rowAmountPerPage eq '10' ? 'selected' : ''}" />>10개</option>
            <option value="20" <c:out value="${pagingCreator.myBoardPagingDTO.rowAmountPerPage eq '20' ? 'selected' : ''}" />>20개</option>
            <option value="50" <c:out value="${pagingCreator.myBoardPagingDTO.rowAmountPerPage eq '50' ? 'selected' : ''}" />>50개</option>
            <option value="100" <c:out value="${pagingCreator.myBoardPagingDTO.rowAmountPerPage eq '100' ? 'selected' : ''}" />>100개</option>
```

```

</select>

<select class="form-control" id="selectScope" name="scope"><!-- 검색 범위 선택 -->
    <option value="" <c:out value="${pagingCreator.myBoardPagingDTO.scope == null
        ? 'selected':'''}" /> 검색범위</option>
    <option value="T" <c:out value="${pagingCreator.myBoardPagingDTO.scope eq 'T'
        ? 'selected' : ''}" /> 제목</option>
    <option value="C" <c:out value="${pagingCreator.myBoardPagingDTO.scope eq 'C'
        ? 'selected' : ''}" /> 내용</option>
    <option value="W" <c:out value="${pagingCreator.myBoardPagingDTO.scope eq 'W'
        ? 'selected' : ''}" /> 작성자</option>
    <option value="TC" <c:out value="${pagingCreator.myBoardPagingDTO.scope eq 'TC'
        ? 'selected' : ''}" /> 제목 + 내용</option>
    <option value="TW" <c:out value="${pagingCreator.myBoardPagingDTO.scope eq 'TW'
        ? 'selected' : ''}" /> 제목 + 작성자</option>
    <option value="TCW" <c:out value="${pagingCreator.myBoardPagingDTO.scope eq 'TCW'
        ? 'selected' : ''}" /> 제목 + 내용 + 작성자</option>
</select>

<div class="input-group"><!-- 검색어 입력 -->
    <input class="form-control" id="inputKeyword" name="keyword" type="text" placeholder="검색어를 입력하세요"
        value='<c:out value="${pagingCreator.myBoardPagingDTO.keyword}" />' />
    <span class="input-group-btn"><!-- 전송버튼 -->
        <button class="btn btn-info" type="button" id="btnSearchGo">
            검색 &nbsp;<i class="fa fa-search"></i>
        </button>
    </span>
</div>

<div class="input-group"><!-- 검색 초기화 버튼 -->
    <button id="btnReset" class="btn btn-warning" type="reset">검색초기화</button>
</div><%-- /.form-group --%>

<input type='hidden' name='pageNum' value='${pagingCreator.myBoardPagingDTO.pageNum}'>
<input type='hidden' name='rowAmountPerPage' value='${pagingCreator.myBoardPagingDTO.rowAmountPerPage}'>
<input type='hidden' name='lastPageNum' value='${pagingCreator.lastPageNum}'>
</form><%-- END 검색범위 및 검색어 입력 폼 --%>
<br>

<table class="table table-striped table-bordered table-hover"
    style="width:100%; text-align:center;" >
    <thead>
        ...
        ...
    <thead>
        ...
        ...
    <tbody>
        ...
        ...
    <tbody>
</table>

```

- ☞ 표시 행수, 선택된 검색범위, 입력된 검색어가 페이지 이동간에도 유지되도록, 컨트롤러부터 받아온 값이 value 속성에 설정됩니다.
- ☞ 이 때, 선택된 표시 행수와 검색범위는 서버로부터 전달된 값에 해당되는 옵션이 선택되도록 3항 연산자 조건식을 이용하였습니다.
- ☞ 검색범위의 특정 옵션을 선택하면, "T", "C", "W", "TC", "TW", "TCW" 중 하나의 문자열이 서버로 전달됩니다.  
예를 들어, "TCW" 문자열이 서버로 전달되면, 이 문자열 값을 ["T", "C", "W"] 배열로 변환시킨 후, 변환된 배열을 MyBatis가 사용하여, SELECT문의 조건식을 완성합니다.
- ☞ 서버에서 전달된 검색범위의 문자열을 문자열 배열로 변환하는 메서드는, **MyBoardPagingDTO** 클래스에 구현합니다(다음 섹션).

→ 다음의 추가한 요소에 대한 이벤트 처리 jQuery 코드(<script>부터 </script>까지)를

<%@ include file="../myinclude/myfooter.jsp" %> 위에 추가합니다.

...(생략)...

```
<%-- 검색 관련 요소의 이벤트 처리 --%>
<script>
//표시행수 변경 이벤트 처리
$("#selectAmount").on("change", function(){
    frmSendValue.find("input[name='pageNum']").val(1);
    frmSendValue.attr("action", "${contextPath}/myboard/list");
    frmSendValue.attr("method", "get");
    frmSendValue.submit();
});

//검색버튼 클릭 이벤트 처리
$("#btnSearchGo").on("click", function(e) {
    if (!$("#selectScope").find("option:selected").val()) {
        alert("검색범위를 선택하세요");
        return false;
    }

    if (!frmSendValue.find("input[name='keyword']").val()) {
        alert("검색어를 입력하세요");
        return false;
    }

    frmSendValue.find("input[name='pageNum']").val("1");
    //e.preventDefault();

    frmSendValue.submit();
});

//검색초기화 버튼 이벤트처리
//버튼 초기화 시, 1페이지에 목록 정보 다시 표시
$("#btnReset").on("click", function(){
    $("#selectAmount").val(10);
    $("#selectScope").val("");
    $("#inputKeyword").val("");
    $("#hiddenPageNum").val(1);
    $("#hiddenLastPageNum").val("");

    frmSendValue.submit();
});

</script>

<%@ include file="../myinclude/myfooter.jsp" %>
```

☞ 추가한 jQuery 이벤트 처리 실행문들은 뒤에서 필요한 것을 모두 구성 후에 테스트 합니다.

## 9-2. MyBoardPagingDTO 클래스 수정

☞ 검색범위와 검색어 필드를 포함시키고, 화면에서 전달된 검색범위 값을 처리할 메서드를 추가합니다.

→ src/main/java/com.spring5213.mypro00.common.paging.MyBoardPagingDTO.java 를 코드 작성 뷰에 오픈

→ 다음의 빨간색 필드와 메서드를 추가.

```
package com.spring5213.mypro00.common.paging;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Setter
@Getter
@ToString
@EqualsAndHashCode
public class MyBoardPagingDTO {

    private int pageNum;           //현재 페이지 번호
    private int rowAmountPerPage; //페이지당 출력할 레코드 개수
    private String scope;          //검색범위(scope - T:btitle, C:bcontent, W:bwriter)
    private String keyword;        //검색어

    //검색범위 값 처리를 위한 메소드: 화면에서 선택된 TCW 값을 ["T", "C", "W"] 배열로 반환
    //이 메서드는 Mybatis 엔진에 의해서 자동으로 호출되어 사용됨(SQL 매퍼 파일의 SELECT문에서 설명함)
    public String[] getScopeArray() {
        return scope == null ? new String[] {} : scope.split("");
    }

    //생성자를 통해 표시할 페이지번호와 페이지당 출력할 레코드 개수를 컨트롤러로 전달
    //생성자1: list.jsp가 처음 호출 시에, 페이지번호와 행수를 각각 1과 10으로 전달
    public MyBoardPagingDTO() {
        this.pageNum = 1 ;
        this.rowAmountPerPage = 10 ;
    }

    //생성자2: 목록 화면에서 페이지번호 클릭 시, 페이지번호와 행수를 각각 사용자가 선택한 페이지번호와 10으로 전달
    public MyBoardPagingDTO(int pageNum) {
        if(pageNum <= 0) {
            this.pageNum = 1 ;
        } else {
            this.pageNum = pageNum;
        }
        this.rowAmountPerPage = 10 ;
    }

    //생성자3: 목록 화면에서 사용자가 표시할 행수를 선택하고 페이지 번호 클릭 시,
    //페이지번호와 행수를 각각 사용자가 선택한 페이지번호와 표시행수로 전달
    public MyBoardPagingDTO(int pageNum, int rowAmountPerPage) {
        if(pageNum <= 0) {
            this.pageNum = 1 ;
        } else {
            this.pageNum = pageNum;
        }

        if(rowAmountPerPage <= 0) {
            this.rowAmountPerPage = 10 ;
        } else {
            this.rowAmountPerPage = rowAmountPerPage;
        }
    }
}
```

### 9-3. SQL 매퍼 파일의 SQL 문 수정

- 검색범위와 검색어에 따라 SELECT 문의 조건절이 동적으로 완성되어 처리될 수 있도록, Mybatis 동적태그를 활용합니다.
- 이 때, (1) 목록을 조회하는 SELECT문과, (2) 게시물 총 개수를 조회하는 SELECT 문을 수정해야 하며, 수정 후에, 매퍼 테스트를 수행합니다.

→ src/main/resources/com/spring5212/mypro00/mapper/MyBoardMapper.xml 파일을 코드 작성 뷰에 오픈

→ 게시물 목록을 조회하는 SELECT 문을 다음처럼 수정합니다(빨간색 코드 추가).

```
<!-- 게시물 목록 조회(페이징, 검색): 삭제 요청된 행 포함 -->
<select id="selectMyBoardList" resultType="com.spring5213.mypro00.domain.MyBoardVO">
<![CDATA[
    SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag
    FROM ( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.*
            FROM book_ex.tbl_myboard a
            WHERE
        ]]> ← 여기서부터 추가 시작
    <trim prefix="(" suffix=")" AND" prefixOverrides="OR">
        <foreach item='scope' collection="scopeArray"> ←① colle아래 설명 참고
            <trim prefix="OR">
                <choose>
                    <when test="scope == 'T'.toString()">btitle LIKE '%'||#{keyword}||'%'
                    <when test="scope == 'C'.toString()">bcontent LIKE '%'||#{keyword}||'%'
                    <when test="scope == 'W'.toString()">bwriter LIKE '%'||#{keyword}||'%'
                </choose>
            </trim>
        </foreach>
    </trim>
    <![CDATA[ ← 여기까지 추가 끝
        ROWNUM <= #{pageNum} * #{rowAmountPerPage}
    )
    WHERE rn >= ( ( #{pageNum} * #{rowAmountPerPage} ) - ( #{rowAmountPerPage} - 1 ) )
]]>
</select>
```

#### ◆ ① 설명

- foreach 요소의 item 속성에 지정된 변수에는 'T', 'C', 'W' 중 하나의 값이 할당되어, 아래의 <choose> 하위요소인 <when> 요소의 test 속성에 명시된 조건식에서 사용됩니다.

- item 요소의 값은 collection 속성에 명시된 scopeArray 변수의 배열로부터 값이 전달됩니다.

- scopeArray 변수에 배열이 할당되는 과정은 다음과 같습니다.

사용자가 브라우저의 게시물 목록 화면에서 검색범위(예, 제목+내용+작성자)를 선택하면, 그에 해당하는 "TCW" 값이 폼을 통해 톰캣 서버로 전달 → MyBoardController의 showBoardList() 메서드의 MyBoardPagingDTO 객체에 저장

→ MyBoardPagingDTO 객체가 MyBoardServiceImpl의 getBoardList() 메서드에게 전달

→ MyBoardPagingDTO 객체가 MyBoardMapper를 통해 SQL 매퍼 파일의 selectMyBoardList 아이디의 SELECT 문장(위의 문장)에 전달

→ Mybatis 엔진에 의해 문장이 처리되면서, collection 속성에 설정된 scopeArray 변수에 대한 getScopeArray() getter 메서드를 전달된 MyBoardPagingDTO 객체로부터 호출

→ MyBoardPagingDTO.getScope() 메서드가 scope 필드에 저장된 사용자가 선택한 검색범위값("TCW")를 처리하여 ['T', 'C', 'W'] 배열을 반환

→ Mybatis 엔진이 반환된 ['T', 'C', 'W'] 배열을 collection 속성에 설정된 scopeArray 변수에 할당

→ foreach 요소가 처리됨, 즉 배열로부터 문자값을 하나씩 item 속성의 변수에 할당한 후, 아래의 when 요소의 test 속성의 조건평가

- 따라서, 앞에서 MyBoardPagingDTO 클래스에 정의한 getScopeArray() 메서드는, 실질적으로 Mybatis 엔진이 foreach 요소의

collection 속성의 변수에 값을 할당하기 위하여 사용됩니다.

따라서, MyBoardPagingDTO 클래스의 getScopeArray() 메서드 이름은, collection 속성에 지정된 변수의 getter 형식으로 지정되어야 합니다.

→ 게시물 총 개수를 조회하는 기준 SELECT 문에 다음처럼 <where>부터 </where>까지를 추가합니다.

```
<!-- 게시물 조회 - 총 게시물 개수(페이징, 검색고려): 삭제 요청된 행 포함 -->
<select id="selectRowAmountTotal" resultType="Long">
<![CDATA[
    SELECT count(*) FROM book_ex.tbl_myboard
]]
<where> ← 여기서부터 추가 시작
    <trim prefix="(" suffix=")" prefixOverrides="OR">
        <foreach item='scope' collection="scopeArray">
            <trim prefix="OR">
                <choose>
                    <when test="scope == 'T'.toString()">btitle LIKE '%'||#{keyword}|| '%'</when>
                    <when test="scope == 'C'.toString()">bcontent LIKE '%'||#{keyword}|| '%'</when>
                    <when test="scope == 'W'.toString()">bwriter LIKE '%'||#{keyword}|| '%'</when>
                </choose>
            </trim>
        </foreach>
    </trim>
</where> ← 여기까지 추가 끝
</select>
```

◆ 다음은 삭제 요청된 행을 포함하지 않을 경우의 SQL 매퍼 XML에 해당하는 코드입니다.

```
<!-- 게시물 목록 조회(페이징, 검색): 삭제 요청된 행 제외 --><!--
<select id="selectMyBoardList" resultType="com.spring5213.mypro00.domain.MyBoardVO">
<![CDATA[
    SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag
    FROM ( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.*
            FROM book_ex.tbl_myboard a
            WHERE
]]
<trim prefix="(" suffix=") AND" prefixOverrides="OR">
    <foreach item='scope' collection="scopeArray">
        <trim prefix="OR">
            <choose>
                <when test="scope == 'T'.toString()">btitle LIKE '%'||#{keyword}|| '%'</when>
                <when test="scope == 'C'.toString()">bcontent LIKE '%'||#{keyword}|| '%'</when>
                <when test="scope == 'W'.toString()">bwriter LIKE '%'||#{keyword}|| '%'</when>
            </choose>
        </trim>
    </foreach>
</trim>
<![CDATA[
        bno > 0 AND bdelFlag = 0 AND ROWNUM <= ( #{pageNum} * #{rowAmountPerPage} )
    )
    WHERE rn >= ( ( #{pageNum} * #{rowAmountPerPage} ) - ( #{rowAmountPerPage} - 1 ) )
]]
</select> -->
```

```
<!-- 게시물 총 개수 조회(검색 페이징): 삭제 요청된 행 제외 --><!--
```

```
<select id="selectRowAmountTotal" resultType="Long">
<![CDATA[
    SELECT count(*) FROM book_ex.tbl_myboard
    WHERE
]]
<trim prefix="(" suffix=") AND" prefixOverrides="OR">
    <foreach item='scope' collection="scopeArray">
```

```

<trim prefix="OR">
    <choose>
        <when test="scope == 'T'.toString()">btitle LIKE '%'||#{keyword}||'%'</when>
        <when test="scope == 'C'.toString()">bcontent LIKE '%'||#{keyword}||'%'</when>
        <when test="scope == 'W'.toString()">bwriter LIKE '%'||#{keyword}||'%'</when>
    </choose>
</trim>
</foreach>
</trim>
<![CDATA[
    bno > 0 AND bdelFlag = 0
]]>
</select> -->

```

### [매퍼 구성 테스트]

- ☞ 문서의 콘솔 로그는 테스트 관련 빈 구성 로그와 HikariDataSource 빈 관련 로그는 생략합니다.
- ☞ 로그에 표시된 SQL 문과 총 행수, 레코드를 잘 확인합니다. 테스트 메서드의 검색어와 검색범위의 주석을 바꿔서 해제하면서 테스트해보세요.

→ src/test/java/com.spring5213.mypro00.mapper/MyBoardMapperTests.java 클래스를 코드 작성 뷰에 오픈

→ 기존 테스트 메서드를 모두 주석처리 후, 다음의 메서드 추가

```

@Test
public void testSearchBoardWithPaging() {
    MyBoardPagingDTO myBoardPagingDTO = new MyBoardPagingDTO(); //기본 생성자 이용(1, 10)
    //myBoardPagingDTO.setKeyword("5");
    myBoardPagingDTO.setKeyword("용5");
    //myBoardPagingDTO.setKeyword("5");

    //myBoardPagingDTO.setScope("T");
    //myBoardPagingDTO.setScope("C");
    //myBoardPagingDTO.setScope("W");
    //myBoardPagingDTO.setScope("TC");
    myBoardPagingDTO.setScope("TCW");
    //myBoardPagingDTO.setScope("TW");
    //myBoardPagingDTO.setScope("CW");
    log.info("행 총 개수: " + myBoardMapper.selectRowAmountTotal(myBoardPagingDTO));

    List<MyBoardVO> list = myBoardMapper.selectMyBoardList(myBoardPagingDTO);
    list.forEach(board -> log.info(board));
}

```

→ 테스트 수행

- 코드 작성 뷰에 표시된 MyBoardControllerTests.java 코드에서 마우스 우측버튼 클릭 → Run As → 2 JUnit Test 클릭
- 콘솔에 표시된 로그 확인

[콘솔 로그 내용 일부: 다음은 검색어와 검색범위가 없을 때와 검색어와 검색범위 있을 때(빨간색)를 각각 테스트 한 로그입니다]

```

//검색어와 검색범위 없을 때
INFO : jdbc.sqlonly - SELECT count(*) FROM book_ex.tbl_myboard

INFO : jdbc.sqltiming - SELECT count(*) FROM book_ex.tbl_myboard
{executed in 85 msec}
INFO : jdbc.resultsettable -
|-----|
|count(*)|
|-----|

```

```

|458751 |
|-----|
INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - 행 총 개수: 458751
INFO : jdbc.sqlonly - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= 1 * 10 ) WHERE rn >= ( ( 1 * 10 ) - ( 10 - 1 ) )

INFO : jdbc.sqltiming - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE ROWNUM
<= 1 * 10 ) WHERE rn >= ( ( 1 * 10 ) - ( 10 - 1 ) )
{executed in 23 msec}
INFO : jdbc.resultsettable -
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
458752 |테스트제목7 |테스트내용7 |test7 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |1 |0 |0 |
458751 |테스트제목6 |테스트내용6 |test6 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458750 |테스트제목5:수정 |테스트제목5:수정 |test5 |2021-02-26 23:49:51.0 |2021-02-27 17:29:34.0 |7 |0 |1 |
458749 |테스트제목4 |테스트내용4 |test4 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |1 |
458748 |테스트제목3 |테스트내용3 |test3 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |1 |
458747 |테스트제목2 |테스트내용2 |test2 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458746 |테스트제목1 |테스트내용1 |test1 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458745 |테스트제목7 |테스트내용7 |test7 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458744 |테스트제목6 |테스트내용6 |test6 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458743 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

```

...(생략)...

//검색어와 검색범위 있을 때

```

INFO : jdbc.sqlonly - SELECT count(*) FROM book_ex.tbl_myboard WHERE ( btitle LIKE '%'||'용5'|| '%' OR bcontent LIKE
'%'||'용5'|| '%' OR bwriter LIKE '%'||'용5'|| '%' )

```

```

INFO : jdbc.sqltiming - SELECT count(*) FROM book_ex.tbl_myboard WHERE ( btitle LIKE '%'||'용5'|| '%' OR bcontent LIKE
'%'||'용5'|| '%' OR bwriter LIKE '%'||'용5'|| '%' )
{executed in 209 msec}

```

```

INFO : jdbc.resultsettable -
-----|-----|
|count(*) |
-----|-----|
|65535 |
-----|-----|

```

INFO : com.spring5213.mypro00.mapper.MyBoardMapperTests - 행 총 개수: 65535

```

INFO : jdbc.sqlonly - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE (
btitle LIKE '%'||'용5'|| '%' OR bcontent LIKE '%'||'용5'|| '%' OR bwriter LIKE '%'||'용5'|| '%' )
AND ROWNUM <= 1 * 10 ) WHERE rn >= ( ( 1 * 10 ) - ( 10 - 1 ) )

```

```

INFO : jdbc.sqltiming - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE (
btitle LIKE '%'||'용5'|| '%' OR bcontent LIKE '%'||'용5'|| '%' OR bwriter LIKE '%'||'용5'|| '%' )
AND ROWNUM <= 1 * 10 ) WHERE rn >= ( ( 1 * 10 ) - ( 10 - 1 ) )
{executed in 26 msec}

```

```

INFO : jdbc.resultsettable -
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |btitle |bcontent |bwriter |bregdate |bmoddate |bviewscnt |breplycnt |bdelflag |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
458743 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458736 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458731 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458724 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458717 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458714 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458707 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458700 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458693 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
458683 |테스트제목5 |테스트내용5 |test5 |2021-02-26 23:49:51.0 |2021-02-26 23:49:51.0 |0 |0 |0 |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

```

...(생략)...

## 9-4. 검색 테스트 데이터 준비 및 웹 브라우저에서의 검색 결과에 대한 페이징 확인

- ☞ 검색을 위한 데이터를 생성하기 위하여 일부 행들을 업데이트 합니다.
- ☞ 검색어와 검색 범위를 지정하여 검색 후, 페이지 번호를 클릭하여 검색 결과 내에서의 페이징 이동이 수행되는지, 각 버튼의 동작이 정상적으로 처리되는지 확인합니다.

### (1) 데이터 준비

→ SQL\*Developer를 이용하여 book\_ex 계정으로 데이터베이스에 접속한 후, 다음의 UPDATE문을 실행합니다.

```
-- 검색 테스트 데이터 준비
```

```
UPDATE book_ex.tbl_myboard
SET btitle='새로 수정'||btitle
WHERE bno LIKE '22__6';    --2 2 _ _ _ 6 : 1000개 행 수정됨, 문서에서의 총 행수가 약46만개라서 1000개입니다.
```

```
UPDATE book_ex.tbl_myboard
SET bcontent='새로 수정'||bcontent
WHERE bno LIKE '22__4';    --2 2 _ _ _ 4 : 1000개 행 수정됨
```

```
UPDATE book_ex.tbl_myboard
SET bwriter='user새로'
WHERE bno LIKE '22__2';    --2 4 _ _ _ 2 : 1000개 행 수정됨
```

```
COMMIT;
```

### (2) 웹 브라우저 테스트: 검색 결과의 페이징 이동 테스트

- ☞ 웹 브라우저에서 다음의 지시대로 스스로 테스트 합니다.

→ 톰캣 서버 기동

→ 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/list> URL로 접속

→ 표시 행 수를 50, 검색 범위는 "제목+내용+작성자", 검색어는 "새로"를 지정한 후, 검색 버튼 클릭

→ 콘솔창에 총 행수가 약 3000개 가까이 표시되는 확인

→ 페이지 번호를 클릭하면서 화면의 표시행수, 검색어, 검색범위의 표시 값이 유지되는지 확인

## 9-5. 목록, 상세, 수정-삭제 화면 이동간 검색-페이징 유지

☞ JSP 페이지에는 검색 페이징 데이터를 받아 유지되는 form에 검색어와 검색범위를 추가하고, 컨트롤러의 수정-삭제를 처리하는 메서드에 해당 값들이 상세 또는 목록으로 전달되도록 RedirectAttributes 객체에 검색-페이징 값들을 속성으로 추가합니다.

→ src/main/webapp/WEB-INF/views/myboard 폴더에 있는 다음의 JSP 페이지들을 각각 코드 뷰에 오픈 후, 각각 수정(빨간색 코드)

→ detail.jsp 파일: frmSendValue 아이디의 폼에 검색어와 검색범위 hidden 유형의 input 추가

```
<form id="frmSendValue"><!-- 검색 페이징 데이터 전달 -->
    <input type='hidden' name='bno' id="bno" value=''>
    <input type='hidden' name='pageNum' value='${myBoardPagingDTO.pageNum}'>
    <input type='hidden' name='rowAmountPerPage' value='${myBoardPagingDTO.rowAmountPerPage}'>
    <input type='hidden' name='scope' value='${myBoardPagingDTO.scope}'>
    <input type='hidden' name='keyword' value='${myBoardPagingDTO.keyword}'>
</form>
```

→ modify.jsp 파일: 기존에 검색어와 검색범위 hidden 유형의 input 추가, 취소(목록이동) 버튼 클릭 처리 jQuery 코드 수정

```
<%-- 검색 페이징 데이터가 저장되는 hidden 유형의 input --%>
    <input type='hidden' name='pageNum' value='${myBoardPagingDTO.pageNum}'>
    <input type='hidden' name='rowAmountPerPage' value='${myBoardPagingDTO.rowAmountPerPage}'>
    <input type='hidden' name='scope' value='${myBoardPagingDTO.scope}'> ←추가
    <input type='hidden' name='keyword' value='${myBoardPagingDTO.keyword}'> ←추가
</form>
```

...(생략)...

```
<script>
//form의 수정/삭제/목록보기 버튼 클릭 이벤트 처리
var frmModify = $("#frmModify");

$('button').on("click", function(e){

    var operation = $(this).data("oper"); //각 버튼의 data-xxx 속성에 설정된 값을 저장

    if(operation == "modify"){ //게시물 수정 요청
        frmModify.attr("action", "${contextPath}/myboard/modify");

    } else if(operation == "remove"){ //게시물 삭제 요청
        frmModify.attr("action", "${contextPath}/myboard/delete"); //bdelFlag만 업데이트
        //frmModify.attr("action", "${contextPath}/myboard/remove"); //실제 행 삭제 시 사용

    } else if(operation == "list"){ //게시물 목록 화면 요청
        //기존 페이징 데이터 input 요소 복사
        var pageNumInput = $("input[name='pageNum']").clone();
        var rowAmountInput = $("input[name='rowAmountPerPage']").clone();
        var scopeInput = $("input[name='scope']").clone(); ←추가
        var keywordInput = $("input[name='keyword']").clone(); ←추가

        frmModify.empty();

        frmModify.attr("action", "${contextPath}/myboard/list").attr("method", "get");
        //복사된 input 요소를 다시 form에 추가
        frmModify.append(pageNumInput);
        frmModify.append(rowAmountInput);
        frmModify.append(scopeInput); ←추가
        frmModify.append(keywordInput); ←추가
    }
})
```

```

    }

    frmModify.submit() ; //요청 전송

});
</script>

```

→ src/main/java/com.spring5213.mypro00.controller/**MyBoardController.java** 를 코드 작성 뷰에 오픈

→ 수정 처리 메서드 수정: 빨간색 코드 추가

```

//특정 게시물 수정 처리
//RedirectAttributes를 사용하는 방식
//model을 사용하지 않음.
@PostMapping("/modify")
public String modifyBoard(MyBoardVO myBoard,
                         RedirectAttributes redirectAttr,           //전달할 페이지 값들을 저장
                         MyBoardPagingDTO myBoardPagingDTO){      //전달된 페이지 값들을 저장
    log.info("컨트롤러 - 게시물 수정 전달된 myBoard 값: " + myBoard);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    if (myBoardService.modifyBoard(myBoard)) {
        redirectAttr.addFlashAttribute("result", "successModify");
    }
    //RedirectAttribute 유형의 객체에 전달할 페이지 데이터를 속성으로 바인딩
    redirectAttr.addAttribute("bno", myBoard.getBno());
    redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
    redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
    redirectAttr.addAttribute("scope", myBoardPagingDTO.getScope());      ←추가
    redirectAttr.addAttribute("keyword", myBoardPagingDTO.getKeyword());   ←추가
    log.info("화면으로 전달될 redirectAttr: " + redirectAttr);

    return "redirect:/myboard/detailmod" ;
}

```

→ 삭제 처리 메서드(3개) 수정: 빨간색 코드 추가

```

//특정 게시물 삭제 요청
@PostMapping("/delete")
public String setBoardDeleted(@RequestParam("bno") Long bno,
                             RedirectAttributes redirectAttr,           //전달할 페이지 값들을 저장
                             MyBoardPagingDTO myBoardPagingDTO){      //전달된 페이지 값들을 저장
    log.info("컨트롤러 - 게시물 삭제(bdelFlag값변경 글번호): " + bno);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    if (myBoardService.setBoardDeleted(bno)) {
        redirectAttr.addFlashAttribute("result", "successRemove");
    }
    //RedirectAttribute 유형의 객체에 전달할 페이지 데이터를 속성으로 바인딩
    redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
    redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
    redirectAttr.addAttribute("scope", myBoardPagingDTO.getScope());      ← 추가
    redirectAttr.addAttribute("keyword", myBoardPagingDTO.getKeyword());   ← 추가
    log.info("화면으로 전달될 redirectAttr: " + redirectAttr);

    return "redirect:/myboard/list";
}

//특정 게시물 실제 삭제
@PostMapping("/remove")
public String removeBoard(@RequestParam("bno") Long bno,
                         RedirectAttributes redirectAttr,           //전달할 페이지 값들을 저장

```

```

        MyBoardPagingDTO myBoardPagingDTO ) { //전달된 페이지 값들을 저장
log.info("컨트롤러 - 게시물 삭제: 삭제되는 글번호: " + bno);
log.info("컨트롤러 - 전달된 MyBoardPagingDTO: "+ myBoardPagingDTO);

if (myBoardService.removeBoard(bno)) {
    redirectAttr.addFlashAttribute("result", "successRemove");
}
//RedirectAttribute 유형의 객체에 전달할 페이지 데이터를 속성으로 바인딩
redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
redirectAttr.addAttribute("scope", myBoardPagingDTO.getScope()); ← 추가
redirectAttr.addAttribute("keyword", myBoardPagingDTO.getKeyword()); ← 추가
log.info("화면으로 전달될 redirectAttr: "+ redirectAttr);

return "redirect:/myboard/list";
}

//삭제된 총 행수를 model 대신 RedirectAttributes로 전달, model 필요없음
//게시물 삭제 - 삭제 요청된 모든 게시물 삭제
@PostMapping("/removeAll")
public String removeAllDeletedBoard(RedirectAttributes redirectAttr, //전달할 페이지 값들과 삭제된 행 수를 저장
                                     MyBoardPagingDTO myBoardPagingDTO ) { //전달된 페이지 값들을 저장
log.info("컨트롤러 - 전달된 MyBoardPagingDTO: "+ myBoardPagingDTO);

redirectAttr.addFlashAttribute("result", "successRemoveAll");
//RedirectAttribute 유형의 객체에 전달할 페이지 데이터를 속성으로 바인딩
redirectAttr.addAttribute("removedRowCnt", myBoardService.removeAllDeletedBoard());
redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
redirectAttr.addAttribute("scope", myBoardPagingDTO.getScope()); ← 추가
redirectAttr.addAttribute("keyword", myBoardPagingDTO.getKeyword()); ← 추가
log.info("화면으로 전달될 redirectAttr: "+ redirectAttr);

return "redirect:/myboard/list";
}

```

## [테스트]

☞ 웹 브라우저에서 지금까지의 구성 내용을 아래의 절차대로 스스로 수행하여 잘 작동되는지와 콘솔 로그를 확인합니다.

→ 톰캣 서버 기동

→ 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/list> URL로 접속

→ 표시 행 수를 20, 검색 범위는 "제목", 검색어는 "새로"를 지정한 후, 검색 버튼 클릭

→ 콘솔창에 총 행수가 1000개가 표시되는 확인

→ 상세, 수정 화면 이동 간 검색-페이지 값들이 화면에서 유지되는지 확인

→ 콘솔창에 표시된 로그에서 myBoardPagingDTO의 검색-페이지 필드 값이 유지되는지도 같이 확인

## [참고] UriComponentsBuilder를 이용하는 URI-링크 생성

☞ 테스트 시에 브라우저 주소창을 보면(아래 그림), URL에 페이지 데이터들이 파라미터로 추가되어 표시됩니다.

이 때, 한글 값(예, 그림의 새로)은 %코드값으로 변환된 값으로 처리됩니다(URL을 복사해서 메모장에 붙여보면 %코드값을 확인할 수 있습니다.).

The screenshot shows a web application interface titled "My Admin Board". On the left, there's a sidebar with navigation links: Dashboard, Charts, Tables, Forms, UI Elements, Multi-Level Dropdown, and Sample Pages. The main content area is titled "Board - List" and has a sub-section titled "게시글 목록". At the top of this section, there are search and filter controls: "10개" dropdown, "제목 + 내용 + 작성자" dropdown, "새로" button, "검색" button, and "검색초기화" button. Below these are four rows of data in a table:

번호	제목	작성자	작성일	수정일	조회수
229696	새로 수정테스트제목6	test6	2021/06/21	2021/06/21 08:28:13	0
229694	테스트제목4	test4	2021/06/21	2021/06/21 08:28:13	0
229692	테스트제목2	user새로	2021/06/21	2021/06/21 08:28:13	0
229686	새로 수정테스트제목3	test3	2021/06/21	2021/06/21 08:28:13	0

☞ 실습에서는 컨트롤러 클래스의 메서드에서 리다이렉트 방식의 JSP 페이지를 호출할 때,

RedirectAttributes 클래스의 addAttribute() 메서드를 이용하여 검색 페이지 데이터가 전달되도록 구성했습니다.

따라서, 전달된 값이 URL 뒤에 GET 방식으로 전달되었기 때문에 URL이 그림처럼 표시됩니다.

☞ org.springframework.web.util.UriComponentsBuilder 스프링 API 클래스는 여러 개의 파라미터들을 연결해서

URL 형태로 만들어주는 기능을 제공합니다.

특히, 많은 개수의 파라미터들을 GET 방식으로 URL에 노출시켜서 전달되도록 코드를 작성할 때 유용합니다.

또한 JSP 페이지에서는 JavaScript를 사용할 수 없는 상황에서 UriComponentsBuilder로 생성된 URI 링크를 처리해야 할 때 유용하게 사용되기도 합니다.

→ src/main/java/com.spring5213.mypro00.common.paging.MyBoardPagingDTO.java 클래스를 코드 작성부에 오픈

→ 검색-페이지 데이터를 전달하는 MyBoardPagingDTO 클래스에, UriComponentsBuilder 클래스를 이용하여,

전달할 값을 파라미터로 처리하여 URI에 GET 방식으로 추가해주는 다음의 addPagingParamsToURI() 메서드를 추가합니다.

```
//import org.springframework.web.util.UriComponentsBuilder;
```

```
public String addPagingParamsToURI () {
    UriComponentsBuilder uriBuilder =
        UriComponentsBuilder.fromPath("")
            .queryParam("pageNum", this.pageNum)
            .queryParam("rowAmountPerPage", this.rowAmountPerPage)
            .queryParam("scope", this.scope)
            .queryParam("keyword", this.keyword) ;
    String uriString = uriBuilder.toUriString();
    System.out.println("생성된 파라미터 추가 URI String: " + uriString);
    return uriString;
}
```

- 생성된 메서드를 이용하여 전달할 파라미터들이 URI에 추가될 수 있도록 MyBoardController의 setBoardDeleted() 메서드를 수정합니다.

```
//특정 게시물 삭제 요청
@PostMapping("/delete")
public String setBoardDeleted(@RequestParam("bno") Long bno,
                               RedirectAttributes redirectAttr, //전달할 페이징 값들을 저장
                               MyBoardPagingDTO myBoardPagingDTO){ //전달된 페이징 값들을 저장
    log.info("컨트롤러 - 게시물 삭제(bdelFlag값변경 글번호): " + bno);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    if (myBoardService.setBoardDeleted(bno)) {
        redirectAttr.addFlashAttribute("result", "successRemove");
    }
    //RedirectAttribute 유형의 객체에 전달할 페이징 데이터를 속성으로 바인딩
    //redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum()); ←주석처리
    //redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage()); ←주석처리
    //redirectAttr.addAttribute("scope", myBoardPagingDTO.getScope()); ←주석처리
    //redirectAttr.addAttribute("keyword", myBoardPagingDTO.getKeyword()); ←주석처리
    //log.info("화면으로 전달될 redirectAttr: " + redirectAttr); ←주석처리

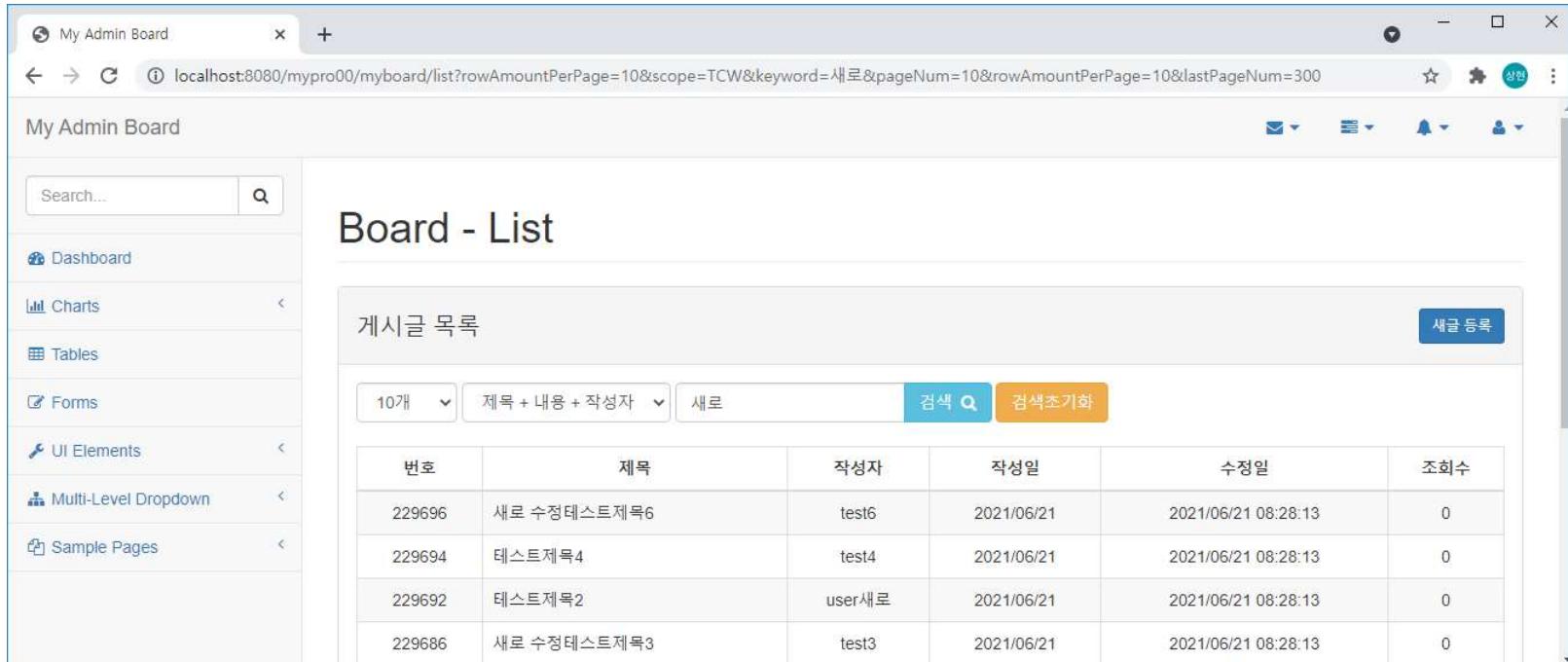
    //return "redirect:/myboard/list";
    return "redirect:/myboard/list" + myBoardPagingDTO.addPagingParamsToURI(); ←수정
}
```

[웹 브라우저 테스트: UriComponentsBuilder를 사용한 메서드 작동]

- 템켓 서버기동

→ 웹브라우저에에서 <http://localhost:8080/mypro00/myboard/list> URL로 접속 → 게시물 목록 페이지

→ 검색범위: 제목 + 내용 + 작성자 선택, 검색어: 새로 입력, 검색 버튼 클릭 → 표시된 목록 화면에서 10 페이지로 이동



→ 첫 번째 게시물의 행 클릭 → 게시물 상세 페이지 이동

The screenshot shows a web application window titled 'My Admin Board'. The URL is [localhost:8080/mypro00/myboard/detail?rowAmountPerPage=10&scope=TCW&keyword=새로&pageNum=10&rowAmountPerPage=10&lastPageNum=300&bno...](http://localhost:8080/mypro00/myboard/detail?rowAmountPerPage=10&scope=TCW&keyword=새로&pageNum=10&rowAmountPerPage=10&lastPageNum=300&bno...). The main content area is titled 'Board - Detail' and displays a single post with the following details:

- 게시글 상세 - 조회수: 1
- 글번호: 229696
- 글제목: 새로 수정테스트제목6
- 글내용: 테스트내용6

→ 수정 버튼 클릭 → 게시물 수정-삭제 페이지 이동

The screenshot shows the same web application window, but the main content area is titled 'Board - Modify'. The URL is [localhost:8080/mypro00/myboard/modify?bno=229696&pageNum=10&rowAmountPerPage=10&scope=TCW&keyword=새로&lastPageNum=300&rowAmountPerPage=10&bno=229696](http://localhost:8080/mypro00/myboard/modify?bno=229696&pageNum=10&rowAmountPerPage=10&scope=TCW&keyword=새로&lastPageNum=300&rowAmountPerPage=10&bno=229696). The form fields are identical to the previous screen:

- 게시글 수정-삭제
- 글번호: 229696
- 글제목: 새로 수정테스트제목6
- 글내용: 테스트내용6

→ 삭제 버튼 클릭 → 게시물 목록페이지가 표시되는 확인

The screenshot shows the 'Board - Modify' page again, but now it displays a message indicating the post has been deleted:

게시글 수정-삭제 완료

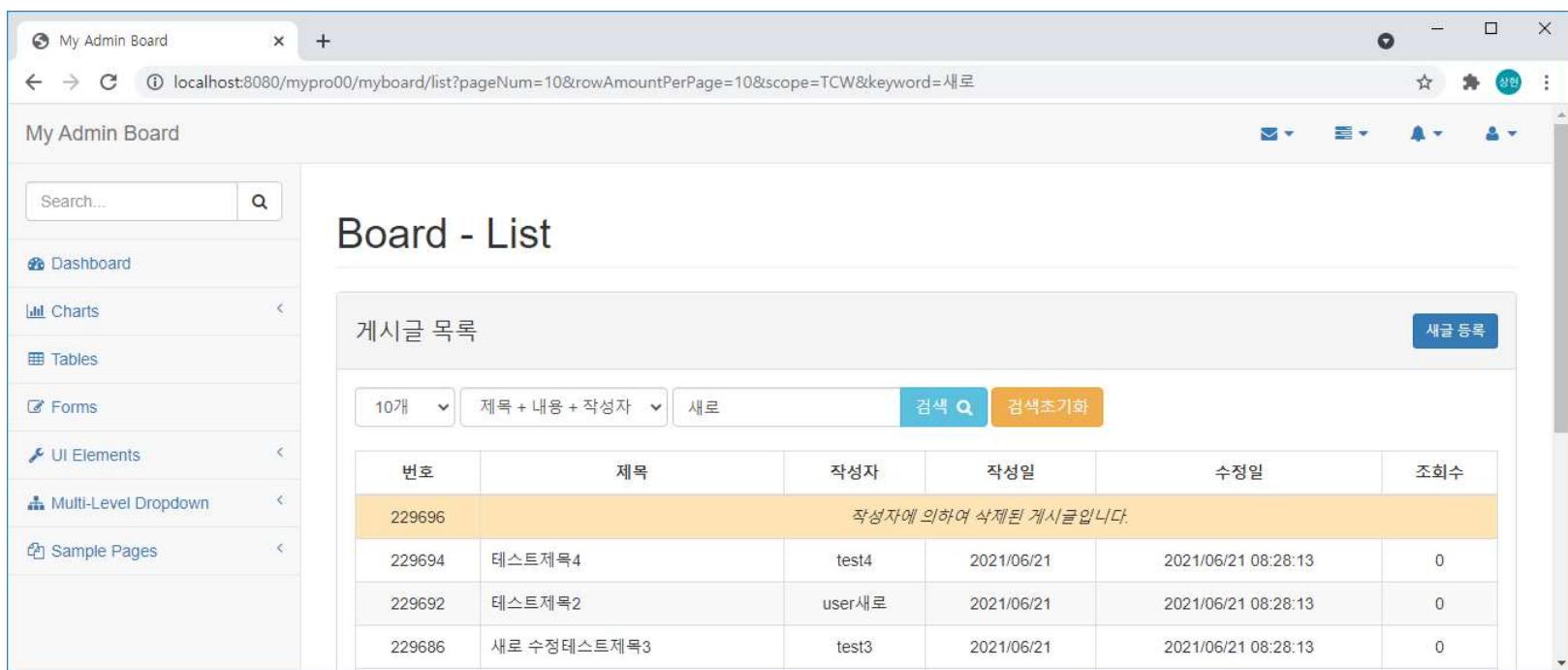
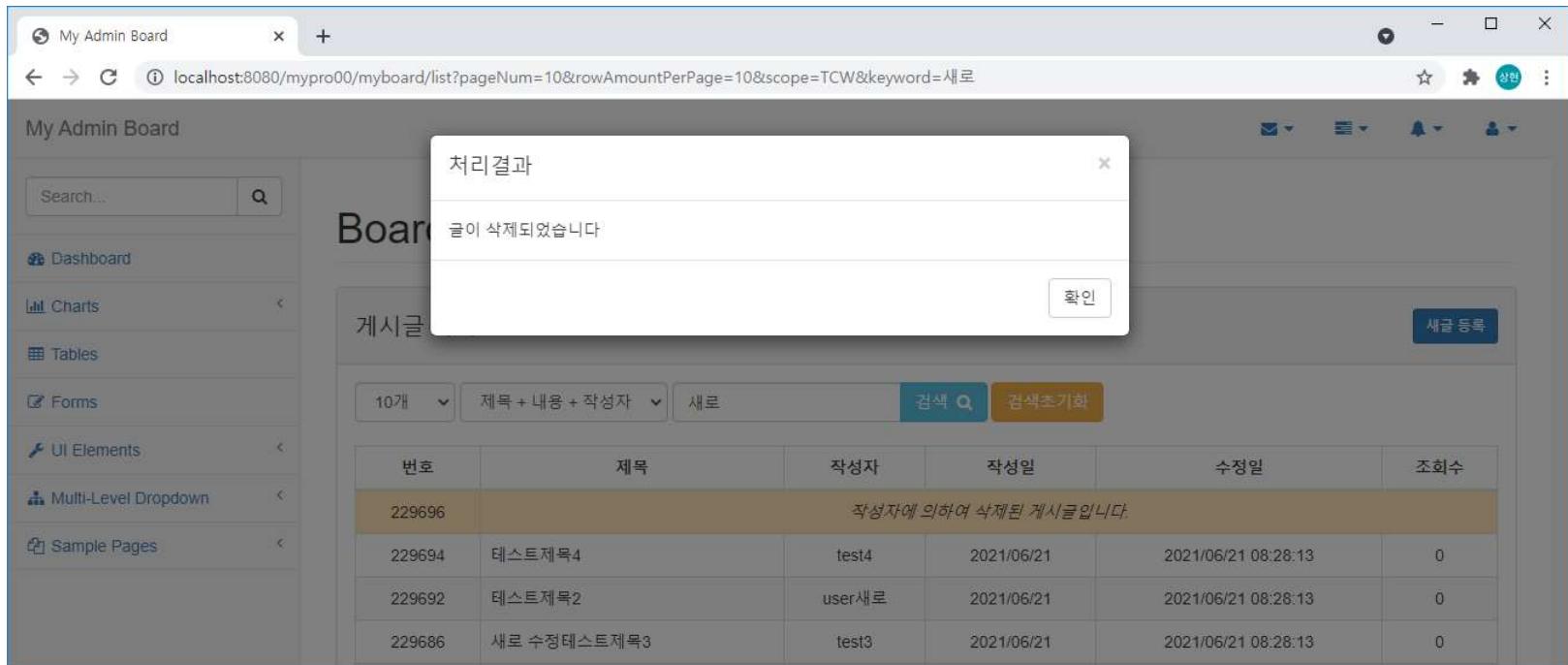
작성자: test6

최종수정일 [등록일시: 2021/06/21 08:28:13]

2021/06/21 08:28:13

수정 삭제 취소

The '삭제' button is highlighted in red.



☞ 웹 브라우저의 URL 표시란에 검색 페이지ing 데이터가 정상적으로 유지되는 것을 확인할 수 있습니다.

#### [addPagingParamsToURI() 메서드 실행 시 콘솔에 표시된 내용]

```
... (생략)...
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 삭제(bdelFlag값변경 글번호): 229696
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 전달된 MyBoardPagingDTO: MyBoardPagingDTO(pageNum=10, rowAmountPerPage=10, scope=TCW, keyword=새로)
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.setBoardDeleted()에 전달된 bno: 229696
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET bdelFlag = 1 WHERE bno = 229696

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET bdelFlag = 1 WHERE bno = 229696
{executed in 1 msec}
생성된 파라미터 추가 URI String: ?pageNum=10&rowAmountPerPage=10&scope=TCW&keyword=%EC%83%88%EB%A1%9C
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 목록 조회 시작.....
INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러에 전달된 사용자의 페이지ing 처리 데이터: MyBoardPagingDTO(pageNum=10, rowAmountPerPage=10, scope=TCW, keyword=새로)
INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.getBoardList() 실행
INFO : jdbc.sqlonly - SELECT bno, btitle, bcontent, bwriter, bregdate, bmodDate, bviewsCnt, breplyCnt, bdelFlag FROM
( SELECT /*+ INDEX_DESC (a pk_myboard) */ ROWNUM rn, a.* FROM book_ex.tbl_myboard a WHERE (
btitle LIKE '%'||'새로'|| '%' OR bcontent LIKE '%'||'새로'|| '%' OR bwriter LIKE '%'||'새로'|| '%'
) AND ROWNUM <= 10 * 10 ) WHERE rn >= ( ( 10 * 10 ) - ( 10 - 1 ) )
... (생략)...
```

## [10] 게시물 댓글 처리 - REST-API와 jQuery의 Ajax 이용

- 다음의 작업들이 진행됩니다.
  - REST-API 개요 및 Spring의 REST-API 간단 소개
  - 댓글 처리를 위한 데이터베이스 구성
  - 댓글 처리를 위한 도메인 클래스 생성
  - 댓글 처리를 위한 매퍼 XML 파일 및 매퍼 인터페이스 생성 및 테스트
  - 댓글 목록에 대한 페이징 처리
  - 비즈니스 계층(서비스)과 제어계층(컨트롤러) 구현 및 테스트
  - 화면 구성
    - JavaScript 클로저(Closure) 개요
    - JavaScript ajax를 이용한 데이터 이동 구현
    - 댓글 화면 구성
    - 테스트

### 10-1. REST-API 개요 및 Spring의 REST-API 간단 소개

- ☞ 과거에는 서버가 JSP를 통해 모든 데이터가 담겨진 HTML을 생성한 후, 이를 브라우저로 전달하여, 브라우저는 이를 화면에서 보여주는 역할을 했습니다. 현재의 모바일 시대에서는, 스마트폰에서 실행되는 "앱"이라는 애플리케이션이, 서버의 데이터를 받아서, 자신만의 방식으로 처리한 후, 앱 화면에 표시합니다. 따라서, 스마트 폰의 앱은, 완성된 HTML이 아니라, 앱 자신이 처리해서 표시할 수 있는 순수한 데이터만을 서버에 요구합니다.
- ☞ 이처럼 서버의 역할은 점점 더 순수하게 데이터에 대한 처리를 목적으로 하는 형태로 바뀌고 있습니다. 또한, 브라우저와 앱은 서버에서 전달하는 데이터를 이용해서, 앱/브라우저 스스로가 별도의 방식을 통해서 이를 사용하여 표시하는 형태로 전환하고 있습니다.
- ☞ 이러한 변화 속에서 웹의 URI(Uniform Resource Identifier) 의미도 조금 다르게 변화되고 있습니다.

과거에는 하나의 서비스 내에서 웹페이지를 이동하더라도 브라우저에 표시된 주소는 변화하지 않도록 웹페이지를 구현하는 방식을 선호했습니다. 반면에 최근 웹 페이지들은 대부분 페이지를 이동하면 브라우저 내의 주소 역시 같이 이동하는 방식을 사용합니다.
- ☞ REST는 'Representational State Transfer'의 약어로, 하나의 URI는 하나의 고유한 리소스를 대표하도록 설계된다는 개념에, 전송방식을 결합해서, 원하는 작업을 지정하는 표현 방법입니다.

예를 들어 '/myboard/123'은, 게시물 중에서 123번이라는 고유한 의미를 가지도록 설계하고, 이에 대한 처리는 GET, POST 방식과 같이 추가적인 정보를 통해서 결정합니다.
- ☞ REST에서는 GET과 POST 외에도 PUT, DELETE 등 다양한 HTTP 전송방식을 사용하여 데이터를 전달합니다.

전송방식	기능
POST	생성(CREATE)
GET	조회(READ)
PUT 또는 PATCH	수정(UPDATE)
DELETE	삭제(DELETE)

☞ 기본-키 컬럼(PK)을 이용해서 한 행의 조회/수정/삭제 처리가 가능하기 때문에, REST 방식으로 동작하는 URL을 설계할 때는 기본-키 컬럼을 기준으로 작성하는 것이 편리합니다. 단 댓글의 목록표시는 한 행을 처리하는 것이 아니므로 기본-키 컬럼을 사용할 수 없기 때문에 파라미터로 필요한 게시물의 번호와 페이지 번호 정보들을 URL에서 표현하는 방식을 사용하게 됩니다.

☞ 회원이라는 자원을 대상으로, URI에 전송방식을 결합하여 REST 방식으로 표현하면 다음과 같이 됩니다.

서버의 작업	HTTP 전송방식	서버의 작업을 REST 방식으로의 표현한 URI	Spring 의 HTTP 전송방식 지원 어노테이션
회원 등록	POST	/members/news	@PostMapping
회원 조회	GET	/members/{id}	@GetMapping
회원 수정	PUT	/member/{id} + body{json 데이터 등}	@PutMapping
회원 삭제	DELETE	/member/{id}	@DeleteMapping

☞ 따라서 REST 방식은 다음과 같이 구성된다고 정리할 수 있습니다.



☞ 스프링은, @RequestMapping처럼, REST 방식의 데이터 전송 처리를 위한 여러 종류의 Annotation과 기능을 제공합니다.

다음은 대표적인 스프링의 REST와 관련된 Annotation입니다.

어노테이션	기능
@ResponseBody	<ul style="list-style-type: none"> <li>- 메서드 선언부에 명시하며, 해당 메서드가 데이터만 응답에 실어서 요청 브라우저에게 전송</li> <li>- 일반 컨트롤러에서도 사용 가능</li> <li>- Spring 3부터 지원</li> </ul>
@RestController	<ul style="list-style-type: none"> <li>- 클래스 선언부에 명시되며, 해당 클래스가 REST 방식으로 처리되는 메소드만 있는 컨트롤러임을 지정</li> <li>- 메서드들은 데이터를 요청 웹 브라우저에게 전송(JSP로 전송하지 않음)</li> <li>- 메서드의 리턴 타입으로, 사용자가 정의한 클래스 타입을 사용할 수 있고, 이는 Spring에 의해 자동으로 JSON이나 XML로 바꿔어 전달됨</li> <li>- Spring 4부터 지원</li> </ul>
@PathVariable	URL 경로에 있는 값을 파라미터로 추출할 때 사용
@CrossOrigin	Ajax의 크로스 도메인 문제를 해결해주는 어노테이션
@RequestBody	주로 사용자가 보낸 JSON 데이터를 컨트롤러에서 원하는 타입의 자바객체로 바인딩 처리

☞ 다음의 표는 @RestController 어노테이션이 설정된 클래스 내에 정의된 메서드들의 반환타입을 정리한 것입니다.

메서드의 반환타입이 요청 웹 브라우저로 전송될 때는, @XxxxxxMapping 어노테이션의 produce 속성에 명시된 MIME-유형으로 변환되어 전송됩니다.

반환타입	메서드 반환 자바타입	메서드에 지정하는 방법
단순 문자열	String	@GetMapping( value='/test/getText', produce="text/plain; charset=utf-8")
사용자 객체	MyReplyVO	@GetMapping( value='/reply/1' ) 또는 @GetMapping( value='/reply/1', produces={ "application/xml; charset=UTF-8", "application/json; charset=UTF-8" } )

☞ 반환타입 정리 표 계속

반환타입	메서드 반환 자바타입	메서드에 지정하는 방법
리스트 객체	List<MyReplyVO>	@GetMapping( value='/reply/list' ) 또는 @GetMapping( value='/reply/list', produces={"application/xml; charset=UTF-8", "application/json; charset=UTF-8" } )
맵 객체	Map< <b>String</b> , MyReplyVO>  → XML로 반환되는 경우, 키가 태그 이름으로 사용되므로, 문자열로 지정해야 함	@GetMapping( value='/reply/replyMap' ) 또는 @GetMapping( value='/reply/replyMap', produces={"application/xml; charset=UTF-8", "application/json; charset=UTF-8" } )
ResponseEntity 타입	ResponseEntity<List<MyReplyVO>>  ☞ 데이터와 함께 "HTTP 헤더의 상태"를 요청한 브라우저에 전달하기 위해 사용	@GetMapping( value = "/pages/{bno}/{page}", produces = { "application/xml; charset=UTF-8", "application/json; charset=UTF-8" } )  - 메서드의 return 문 예시 return ResponseEntity.status(HttpStatus.OK) .body(eplyService.getList(bno));  return new ResponseEntity<>(replyService.getList(bno), HttpStatus.OK);

☞ 다음은 @RestController 어노테이션이 설정된 클래스 내에 정의된 메서드에서 사용되는 매개변수 유형에 대한 사용 방식의 예입니다.

(사용 예 - 1: @PathVariable 어노테이션)

```
@GetMapping( value = "/myreplies/pages/{bno}/{page}",  
produces = { "application/json; charset=UTF-8", "application/xml; charset=UTF-8" })  
public ResponseEntity<MyReplyPageDTO> showReplyList(@PathVariable("bno") Long bno, @PathVariable("page") int page) {  
...}
```

☞ @PathVariable은, 일반 컨트롤러에서도 사용이 가능하며, URL 경로의 일부를 값으로 가져와서, 파라미터로 정의할 때 사용됩니다.

단, 값을 파라미터로 정의할 때 파라미터의 유형은, int, double과 같은 기본 자료형은 사용할 수 없습니다.

위에서 int → Integer로 해주어야 함. 그러나 현실은 int도 됩니다.

(사용 예 - 2: @RequestBody 어노테이션)

```
@PostMapping(value = "/new",  
consumes = {"application/json; charset=UTF-8"},  
produces = { "text/plain; charset=UTF-8" })  
public ResponseEntity<String> create(@RequestBody MyReplyVO myReplyVO) { ... }
```

☞ @RequestBody는, 전달된 브라우저 요청(Request)의 내용(body)을 이용해서 해당 파라미터의 타입으로 변환을 요구합니다.

☞ 내부적으로 HttpMessageConverter 타입의 객체들을 이용해서 다양한 포맷의 입력 데이터를 서버내의 클래스 유형으로 변환할 수 있습니다. 대부분의 경우, 웹 브라우저에서 전송된 JSON 데이터를 원하는 타입의 객체로 변환해야 하는 경우에 사용됩니다.

☞ @RequestBody 어노테이션은, GET 방식의 메서드에서는 사용될 수 없습니다.

## 10-2. 댓글처리를 위한 데이터베이스 구성

- ☞ 특정 게시물의 댓글은, 대부분의 경우, 게시물 상세화면에서 게시물 상세 정보 하단에 목록형태로 표시됩니다.
- ☞ 데이터베이스에 이러한 댓글의 조회를 빠르게 처리되도록 테이블을 다음처럼 구성하는 것이 좋습니다.
  - 기본키가 사용하는 인덱스의 키 컬럼을 (게시물번호, 댓글번호) 순으로 지정하여 인덱스를 생성합니다.
  - 기본키를 게시물번호, 댓글번호 순으로 지정하고, 미리 생성된 인덱스를 사용하도록 지정합니다.
- ☞ 데이터베이스의 테이블은, 간단히 요약해서, "업무 분석 → 필요 데이터 도출 → 관계를 기반한 데이터 그룹핑 → 테이블 설계" 순의 과정을 거쳐, Entiry-Relationship Diagram(ERD)을 작성하고, 이로부터 "테이블 정의서"라는 문서를 작성하고, 작성된 테이블 정의서를 기반하여 특정 RDBMS 제품에 대한 테이블 생성 구문을 작성합니다.

→ SQL\*Developer를 이용하여 book\_ex 계정으로 오라클 데이터베이스에 접속하여 댓글 데이터를 저장할 테이블과 관련 객체들을 생성

--댓글답글 저장 테이블 생성

```
CREATE TABLE book_ex.tbl_myreply (
    bno NUMBER(10,0) NOT NULL,
    rno NUMBER(10,0),
    rcontent VARCHAR2(4000) NOT NULL,
    rwriter VARCHAR2(50) NOT NULL,
    rRegDate DATE DEFAULT sysdate,
    rModDate Timestamp(0) DEFAULT systimestamp(0),
    prno NUMBER(10,0) --댓글(부모키)의 답글 처리를 위한 부모키
) TABLESPACE users;
```

--기본키 제약조건이 사용할 인덱스 생성

```
CREATE INDEX book_ex.idx_myreply_bno_rno ON book_ex.tbl_myReply(bno, rno);
```

--기본키 제약조건

```
ALTER TABLE book_ex.tbl_myReply
ADD CONSTRAINT pk_myreply PRIMARY KEY (bno, rno)
USING INDEX book_ex.idx_myreply_bno_rno;
```

--외래키 제약조건 추가(bno --> myboard.bno 참조)

--게시물 삭제 시 관련 댓글 모두 삭제(ON DELETE CASCADE)

```
ALTER TABLE book_ex.tbl_myreply
ADD CONSTRAINT fk_bno_reply_board FOREIGN KEY (bno)
    REFERENCES book_ex.tbl_myBoard(bno) ON DELETE CASCADE;
```

--부모 --자식

--외래키 제약조건 추가(prno --> myreply.rno 참조, 댓글의 답글 관계)

--참조하는 기본키가 bno, rno로 구성되었으므로, 동일하게 bno, prno로 지정해야 함.

--부모 댓글 삭제 시 자식 댓글 모두 삭제(ON DELETE CASCADE)

```
ALTER TABLE book_ex.tbl_myreply
ADD CONSTRAINT fk_bno_prno_myreply FOREIGN KEY (bno, prno)
    REFERENCES book_ex.tbl_myreply(bno, rno) ON DELETE CASCADE;
```

--입력데이터 초기화: 테스트 시, 댓글 입력 시에

--ORA-02291: integrity constraint (BOOK\_EX.FK\_BNO\_PRNO\_MYREPLY) violated - parent key not found 오류가 발생되는 경우, 다음의 조치를 수행합니다.

```
--DELETE FROM book_ex.tbl_myreply ; -- tbl_myreply 테이블의 모든 데이터 삭제
```

```
--COMMIT ; -- 트랜잭션 커밋
```

```
--DROP SEQUENCE book_ex.seq_myreply ; -- 시퀀스 삭제
```

-- 아래의 시퀀스 생성부터 테스트 데이터 재입력을 수행한 후, 테스트를 다시 진행합니다.

--시퀀스 생성

```
CREATE SEQUENCE book_ex.seq_myreply START WITH 1;
```

--테스트 데이터 입력

--1) 부모글 확인: 결과의 첫번째 행의 bno 값 확인

```
SELECT bno, btitle, bcontent, bwriter, bregDate, bmodDate, bviewsCnt, bdelFlag
FROM book_ex.tbl_myboard ORDER BY bno DESC; --229377
```

--2) 아래의 INSERT 문을 5 번 실행

```
INSERT INTO book_ex.tbl_myReply(bno, rno, rcontent, rwriter, rRegDate, rModDate, prno)
VALUES (229377, book_ex.seq_myreply.NEXTVAL, '229377 의 댓글' || book_ex.seq_myreply.CURRVAL,
'user' || book_ex.seq_myreply.CURRVAL, DEFAULT, DEFAULT, DEFAULT);
```

```
COMMIT ;
```

--3) 댓글의 댓글 데이터 생성(5 개의 INSERT 문을 1 번씩 실행): 글번호를 자신의 결과 값으로 변경 후 수행.

```
INSERT INTO book_ex.tbl_myReply(bno, rno, rcontent, rwriter, rRegDate, rModDate, prno)
VALUES (229377, book_ex.seq_myreply.NEXTVAL, '229377 의 댓글 1 의 댓글 6', 'test02', DEFAULT, DEFAULT, 1);
```

```
INSERT INTO book_ex.tbl_myReply(bno, rno, rcontent, rwriter, rRegDate, rModDate, prno)
VALUES (229377, book_ex.seq_myreply.NEXTVAL, '229377 의 댓글 1 의 댓글 7', 'test03', DEFAULT, DEFAULT, 1);
```

```
INSERT INTO book_ex.tbl_myReply(bno, rno, rcontent, rwriter, rRegDate, rModDate, prno)
VALUES (229377, book_ex.seq_myreply.NEXTVAL, '229377 의 댓글 1 의 댓글 8', 'test04', DEFAULT, DEFAULT, 1);
```

```
INSERT INTO book_ex.tbl_myReply(bno, rno, rcontent, rwriter, rRegDate, rModDate, prno)
VALUES (229377, book_ex.seq_myreply.NEXTVAL, '229377 의 댓글 6 의 댓글 9', 'test04', DEFAULT, DEFAULT, 6);
```

```
INSERT INTO book_ex.tbl_myReply(bno, rno, rcontent, rwriter, rRegDate, rModDate, prno)
VALUES (229377, book_ex.seq_myreply.NEXTVAL, '229377 의 댓글 6 의 댓글 10', 'test02', DEFAULT, DEFAULT, 6);
```

```
INSERT INTO book_ex.tbl_myReply(bno, rno, rcontent, rwriter, rRegDate, rModDate, prno)
VALUES (229377, book_ex.seq_myreply.NEXTVAL, '229377 의 댓글 11', 'test02', DEFAULT, DEFAULT, 2);
```

```
INSERT INTO book_ex.tbl_myReply(bno, rno, rcontent, rwriter, rRegDate, rModDate, prno)
VALUES (229377, book_ex.seq_myreply.NEXTVAL, '229377 의 댓글 11 의 댓글 12', 'test03', DEFAULT, DEFAULT, 11);
```

```
COMMIT ;
```

--입력된 데이터 확인

```
SELECT * FROM book_ex.tbl_myreply ORDER BY bno, rno ;
```

[SELECT 문 결과]

SQL   인출된 모든 행: 12(0초)						
BNO	RNO	RCONTENT	RWRITER	RREGDATE	RMODDATE	PRNO
1 458753	12	458753의 댓글11의 댓글12	test03	21/06/22 21/06/22 17:47:41.000000000		11
2 458753	11	458753의 댓글11	test02	21/06/22 21/06/22 17:47:39.000000000		2
3 458753	10	458753의 댓글6의 댓글10	test02	21/06/22 21/06/22 17:47:38.000000000		6
4 458753	9	458753의 댓글6의 댓글9	test04	21/06/22 21/06/22 17:47:36.000000000		6
5 458753	8	458753의 댓글1의 댓글8	test04	21/06/22 21/06/22 17:47:34.000000000		1
6 458753	7	458753의 댓글1의 댓글7	test03	21/06/22 21/06/22 17:47:33.000000000		1
7 458753	6	458753의 댓글1의 댓글6	test02	21/06/22 21/06/22 17:47:30.000000000		1
8 458753	5	458753의 댓글5	user5	21/06/22 21/06/22 17:47:16.000000000	(null)	
9 458753	4	458753의 댓글4	user4	21/06/22 21/06/22 17:47:15.000000000	(null)	
10 458753	3	458753의 댓글3	user3	21/06/22 21/06/22 17:47:14.000000000	(null)	
11 458753	2	458753의 댓글2	user2	21/06/22 21/06/22 17:47:13.000000000	(null)	
12 458753	1	458753의 댓글1	user1	21/06/22 21/06/22 17:47:12.000000000	(null)	

### 10-3. JSON/XML 처리를 위한 라이브러리 설치(이미 설치되어 있음)

- 브라우저와 서버 사이에서 XML이나 JSON 형식의 데이터를 주고 받을 때, 서버 상에 이를 처리할 수 있는 라이브러리를 추가
  - 자바 인스턴스 객체를 JSON 객체나 JSON 유형 문자열 또는 XML로 변환하기 위한 라이브러리

→ mypro00 프로젝트의 pom.xml 파일을 코드 작성 뷰에 오픈 → 다음의 라이브러리 설치 정보 추가

```
<!-- 새로추가: jackson-databind (자바 객체 -> JSON/XML 변환 라이브러리)-->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20210307</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.6</version>
</dependency>
```

### 10-4. 댓글처리를 위한 MyReplyVO 도메인 클래스 생성

→ src/main/java/com.spring5213.mypro00.domain 패키지에 MyReplyVO 클래스를 생성하고, 다음의 코드를 추가

```
package com.spring5213.mypro00.domain;

import java.sql.Timestamp;
import lombok.Data;

@Data
public class MyReplyVO {

    private long bno;
    private long rno;
    private String rcontent;
    private String rwriter;
    private Timestamp rregDate;
    private Timestamp rmodDate;
    private long prno;
    private int lvl; //계층쿼리의 level 값을 저장할 필드
}
```

## 10-5. 댓글처리를 위한 매퍼 인터페이스와 SQL 매퍼 XML 파일 생성 및 매퍼 테스트

☞ 기본적인 CRUD(CREATE/READ/UPDATE/DELETE)를 처리할 수 있는 메서드들을 추가합니다.

☞ 댓글의 경우, 댓글의 답글이 있을 수 있으므로, 오라클 계층-쿼리를 이용하여 작성합니다.

### (1) MyReplyMapper 인터페이스 생성

→ src/main/java/com.pring5212.mypro00.mapper 패키지에 MyReplyMapper 인터페이스를 생성하고, 다음 내용을 작성

```
package com.spring5213.mypro00.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Param;

import com.spring5213.mypro00.domain.MyReplyVO;

public interface MyReplyMapper {

    //특정 게시물에 대한 댓글 목록 조회
    public List<MyReplyVO> selectMyReplyList(@Param("bno") Long bno);

    //특정 게시물에 대한 댓글 등록
    public long insertMyReplyForBoard(MyReplyVO myReplyVO);

    //댓글에 대한 답글 등록
    public long insertMyReplyForReply(MyReplyVO myReplyVO);

    //특정 게시물에 대한 특정 댓글/답글 조회
    public MyReplyVO selectMyReply(@Param("bno") Long bno, @Param("rno") Long rno);

    //특정 게시물에 대한 특정 댓글/답글 수정
    public int updateMyReply(MyReplyVO myReply);

    //특정 게시물에 대한 특정 댓글/답글 삭제
    public int deleteMyReply(@Param("bno") Long bno, @Param("rno") Long rno);

    //특정 게시물에 대한 모든 댓글 삭제(관리자) -- myreply.bno 컬럼의 F.K에 ON DELETE CASCADE 를 사용하는 경우, 필요 없음
    //public int deleteAllReply(@Param("bno") Long bno);
}
```

☞ 위에서 @Param 어노테이션은 MyBatis에서 제공되는 것으로, 2개 이상의 데이터를 매개변수로서 SQL 매퍼 XML에 저장된 SQL 문장으로 전달합니다. 댓글 목록의 경우, 필요 없지만 이 후에 페이징 구현을 고려해서 미리 작성해 놓습니다.

☞ MyBatis에서 두 개 이상이 데이터를 파라미터로 전달하기 위하여, 2개 이상의 값을 저장할 수 있는 Map 객체나 별도의 자바 객체를 이용하여 전달하거나 @Param 어노테이션을 이용하는 방법이 있으며, 이 중 @Param 어노테이션을 이용하는 방법이 제일 간단합니다.

☞ @Param 어노테이션으로 전달된 값은 SQL 매퍼 XML 파일의 SQL문장에서 #{매개변수이름} 형식으로 사용됩니다.

## (2) MyReplyMapper.xml SQL 매퍼 파일 생성

→ src/main/resources/com/pring5212/mypro00/mapper 폴더에 MyReplyMapper.xml 파일을 생성하고, 다음의 내용을 작성

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.spring5213.mypro00.mapper.MyReplyMapper">

    <!-- 특정 게시물에 대한 댓글 목록 조회 -->
    <select id="selectMyReplyList" resultType="com.spring5213.mypro00.domain.MyReplyVO">
        <![CDATA[
            SELECT bno, rno, rcontent, rwriter, rregDate, rmodDate, prno, LEVEL AS lvl
            FROM (
                SELECT /*+ INDEX_ASC (a IDX_MYREPLY_BNO_RNO) */ *
                FROM book_ex.tbl_myreply
                WHERE bno = #{bno} ) a
            START WITH prno IS NULL
            CONNECT BY PRIOR rno = prno
        ]]>
    </select>

    <!-- 특정 게시물에 대한 댓글 등록-->
    <insert id="insertMyReplyForBoard">
        <selectKey keyProperty="rno" order="BEFORE" resultType="long">
            SELECT book_ex.seq_myreply.NEXTVAL FROM dual
        </selectKey>
        INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno)
        VALUES (#{bno}, #{rno}, #{rcontent}, #{rwriter}, DEFAULT, DEFAULT, DEFAULT)
    </insert>

    <!-- 댓글에 대한 답글 등록-->
    <insert id="insertMyReplyForReply">
        <selectKey keyProperty="rno" order="BEFORE" resultType="long">
            SELECT book_ex.seq_myreply.NEXTVAL FROM dual
        </selectKey>
        INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno)
        VALUES (#{bno}, #{rno}, #{rcontent}, #{rwriter}, DEFAULT, DEFAULT, #{prno})
    </insert>

    <!-- 특정 게시물에 대한 특정 댓글/답글 조회-->
    <select id="selectMyReply" resultType="com.spring5213.mypro00.domain.MyReplyVO">
        <![CDATA[
            SELECT * FROM book_ex.tbl_myreply WHERE bno= #{bno} AND rno = #{rno}
        ]]>
    </select>

    <!-- 특정 게시물에 대한 특정 댓글/답글 수정 -->
    <update id="updateMyReply">
        <![CDATA[
            UPDATE book_ex.tbl_myreply
            SET rcontent = #{rcontent},
            rmodDate = DEFAULT
            WHERE bno= #{bno} AND rno = #{rno}
        ]]>
    </update>

    <!-- 특정 게시물에 대한 특정 댓글/답글 삭제 -->
    <delete id="deleteMyReply">
        <![CDATA[
            DELETE FROM book_ex.tbl_myreply WHERE bno= #{bno} AND rno = #{rno}
        ]]>
    </delete>

</mapper>
```

### (3) MyReplyMapper.xml 사용 구성

☞ 구성된 MyBoardMapper 인터페이스와 MyBoardMapper.xml 파일을 mybatis-spring 라이브러리를 통해 Mybatis에서 사용될 수 있도록 구성합니다.

→ src/main/webapp/WEB-INF/spring/mybatis-context.xml 파일을 오픈하고, 다음의 코드(빨간색 코드)를 추가합니다.

```
...(생략)...

<!-- 4. mybatis-spring 연동 (HikariDataSource 빈 주입 받음) -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mapperLocations">
        <!-- 매퍼파일들의 위치는 src/main/resources 폴더에 com/spring5213/mypro00/mapper 폴더구조가 있고
            매퍼파일이 위치해야 함-->
        <list>
            <value>classpath:com/spring5213/mypro00/mapper/MyBoardMapper.xml</value>
            <value>classpath:com/spring5213/mypro00/mapper/MyReplyMapper.xml</value> ←추가
        </list>
    </property>
</bean>

<!-- 5. mybatis-spring 연동 (SqlSessionFactoryBean 빈 주입 받음) -->
<!-- SqlSessionFactory 보다 쓰래드에 안전 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>

<!-- 매퍼인터페이스 자동 빈 생성 -->
<!-- 설정된 패키지에 DAO클래스를 대신하는 매퍼-인터페이스가 있습니다. -->
<mybatis-spring:scan base-package="com.spring5213.mypro00.mapper"/>

</beans>
```

### [매퍼 테스트]

→ SQL\*Developer에서 book\_ex 계정으로 접속한 후, 다음의 SELECT 문으로 TBL\_MYBOARD 테이블에 조회하여 가장 최신 게시물의 bno 값을 확인합니다. 또한 TBL\_MYREPLY 테이블에 대한 SELECT문을 테스트 마다 실행하여 데이터베이스에서의 변경 사항을 확인합니다.

-- 가장 최근 게시물의 게시물 번호(bno) 확인 SELECT 문

```
SELECT MAX(bno) FROM book_ex.tbl_myboard ; -- 확인된 bno 값을 테스트 메서드에서 bno 값으로 사용합니다.
```

-- 댓글 테이블 데이터 확인 SELECT문

```
SELECT * FROM book_ex.tbl_myreply ORDER BY bno DESC, rno ASC ;
```

## (1) MyReplyMapper 인터페이스와 SQL 매퍼 파일 테스트를 위한 테스트 클래스 준비

→ src/test/java/com.spring5213.mypro00.mapper 패키지에 MyReplyMapperTests 클래스 생성하고

다음의 내용 작성

```
package com.spring5213.mypro00.mapper;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/mybatis-context.xml")
@Log4j
public class MyReplyMapperTests {

    @Setter(onMethod_ = @Autowired)
    private MyReplyMapper myReplyMapper;

}
```

☞ 위의 클래스에 테스트가 완료된 메서드는 주석처리 후, 새로운 메서드를 하나씩 추가하면서 하나의 메서드 씩 테스트를 수행합니다.

☞ 테스트 방법은, 변경내용 저장 후, 마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트 수행 → 콘솔에 표시된 로그 확인 순으로 진행합니다(문서에서는, 콘솔 로그 중 테스트 환경 구성 로그와 HikariDataSource 로그는 제외 시켰습니다).

☞ 이 후에는 테스트 메서드와 테스크 결과 로그를 테스트 별로 명시합니다.

## (2) 매퍼 주입 확인 테스트

→ 테스트 메서드 추가

```
@Setter(onMethod_ = @Autowired)
private MyReplyMapper myReplyMapper;

//매퍼 인스턴스 생성 테스트
@Test
public void testMapper() {
    log.info(myReplyMapper);
}
```

[테스트 후, 콘솔 표시 로그 일부]

```
INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - org.apache.ibatis.binding.MapperProxy@7cbc3762
```

### (3) 특정 게시물에 대한 댓글 목록 조회 테스트

→ 테스트 메서드 추가

```
import java.util.List;
import com.spring5213.mypro00.domain.MyReplyVO;

//특정 게시물에 대한 댓글 목록 조회 테스트
@Test
public void testSelectMyReplyList() {

    Long targetBno = 458753L; //SQL Developer에서 확인된 가장 최신 게시물의 bno를 지정
    List<MyReplyVO> myReplies = myReplyMapper.selectMyReplyList(targetBno);
    myReplies.forEach(myReply -> System.out.println(myReply));
}
```

[테스트 후, 콘솔 표시 로그 일부]

```
INFO : jdbc.sqlonly - SELECT bno, rno, rcontent, rwriter, rregDate, rmodDate, prno, LEVEL AS lvl FROM ( SELECT /*+
INDEX_ASC (a IDX_MYREPLY_BNO_RNO) */ * FROM book_ex.tbl_myreply WHERE bno = 458753 ) a START
WITH prno IS NULL CONNECT BY PRIOR rno = prno

INFO : jdbc.sqltiming - SELECT bno, rno, rcontent, rwriter, rregDate, rmodDate, prno, LEVEL AS lvl FROM ( SELECT /*+
INDEX_ASC (a IDX_MYREPLY_BNO_RNO) */ * FROM book_ex.tbl_myreply WHERE bno = 458753 ) a START
WITH prno IS NULL CONNECT BY PRIOR rno = prno
{executed in 158 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |rno |rcontent |rwriter |rregdate |rmoddate |prno |lvl |
|-----|-----|-----|-----|-----|-----|-----|-----|
|458753 |1 |458753의 댓글1 |user1 |2021-06-22 17:47:12.0 |2021-06-22 17:47:12.0 |[null] |1 |
|458753 |6 |458753의 댓글1의 댓글6 |test02 |2021-06-22 17:47:30.0 |2021-06-22 17:47:30.0 |1 |2 |
|458753 |9 |458753의 댓글6의 댓글9 |test04 |2021-06-22 17:47:35.0 |2021-06-22 17:47:36.0 |6 |3 |
|458753 |10 |458753의 댓글6의 댓글10 |test02 |2021-06-22 17:47:37.0 |2021-06-22 17:47:38.0 |6 |3 |
|458753 |7 |458753의 댓글1의 댓글7 |test03 |2021-06-22 17:47:32.0 |2021-06-22 17:47:33.0 |1 |2 |
|458753 |8 |458753의 댓글1의 댓글8 |test04 |2021-06-22 17:47:34.0 |2021-06-22 17:47:34.0 |1 |2 |
|458753 |2 |458753의 댓글2 |user2 |2021-06-22 17:47:13.0 |2021-06-22 17:47:13.0 |[null] |1 |
|458753 |11 |458753의 댓글11 |test02 |2021-06-22 17:47:39.0 |2021-06-22 17:47:39.0 |2 |2 |
|458753 |12 |458753의 댓글11의 댓글12 |test03 |2021-06-22 17:47:41.0 |2021-06-22 17:47:41.0 |11 |3 |
|458753 |3 |458753의 댓글3 |user3 |2021-06-22 17:47:14.0 |2021-06-22 17:47:14.0 |[null] |1 |
|458753 |4 |458753의 댓글4 |user4 |2021-06-22 17:47:14.0 |2021-06-22 17:47:15.0 |[null] |1 |
|458753 |5 |458753의 댓글5 |user5 |2021-06-22 17:47:15.0 |2021-06-22 17:47:16.0 |[null] |1 |
|-----|-----|-----|-----|-----|-----|-----|-----|
MyReplyVO(bno=458753, rno=1, rcontent=458753의 댓글1, rwriter=user1, rregDate=2021-06-22 17:47:12.0, rmodDate=2021-06-22 17:47:12.0, prno=0, lvl=1)
MyReplyVO(bno=458753, rno=6, rcontent=458753의 댓글1의 댓글6, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0,
prno=1, lvl=2)
MyReplyVO(bno=458753, rno=9, rcontent=458753의 댓글6의 댓글9, rwriter=test04, rregDate=2021-06-22 17:47:35.0, rmodDate=2021-06-22 17:47:36.0,
prno=6, lvl=3)
MyReplyVO(bno=458753, rno=10, rcontent=458753의 댓글6의 댓글10, rwriter=test02, rregDate=2021-06-22 17:47:37.0, rmodDate=2021-06-22 17:47:38.0,
prno=6, lvl=3)
MyReplyVO(bno=458753, rno=7, rcontent=458753의 댓글1의 댓글7, rwriter=test03, rregDate=2021-06-22 17:47:32.0, rmodDate=2021-06-22 17:47:33.0,
prno=1, lvl=2)
MyReplyVO(bno=458753, rno=8, rcontent=458753의 댓글1의 댓글8, rwriter=test04, rregDate=2021-06-22 17:47:34.0, rmodDate=2021-06-22 17:47:34.0,
prno=1, lvl=2)
MyReplyVO(bno=458753, rno=2, rcontent=458753의 댓글2, rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 17:47:13.0, prno=0, lvl=1)
MyReplyVO(bno=458753, rno=11, rcontent=458753의 댓글11, rwriter=test02, rregDate=2021-06-22 17:47:39.0, rmodDate=2021-06-22 17:47:39.0, prno=2,
lvl=2)
MyReplyVO(bno=458753, rno=12, rcontent=458753의 댓글11의 댓글12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0,
prno=11, lvl=3)
MyReplyVO(bno=458753, rno=3, rcontent=458753의 댓글3, rwriter=user3, rregDate=2021-06-22 17:47:14.0, rmodDate=2021-06-22 17:47:14.0, prno=0, lvl=1)
MyReplyVO(bno=458753, rno=4, rcontent=458753의 댓글4, rwriter=user4, rregDate=2021-06-22 17:47:14.0, rmodDate=2021-06-22 17:47:15.0, prno=0, lvl=1)
MyReplyVO(bno=458753, rno=5, rcontent=458753의 댓글5, rwriter=user5, rregDate=2021-06-22 17:47:15.0, rmodDate=2021-06-22 17:47:16.0, prno=0, lvl=1)
```

#### (4) 특정 게시물에 대한 댓글 등록 테스트

→ 테스트 메서드 추가

```
//특정 게시물에 대한 댓글 등록 테스트
@Test
public void testInsertMyReplyForBoard() {
    MyReplyVO myReply = new MyReplyVO();

    myReply.setBno(458753L); //SQL Developer에서 확인된 가장 최신 게시물의 bno를 지정
    myReply.setRcontent("매퍼 댓글 입력 테스트 ");
    myReply.setRwriter("replyer");
    log.info("매퍼 - 추가 전 댓글 객체: " + myReply);

    myReplyMapper.insertMyReplyForBoard(myReply);
    log.info("매퍼 - 추가 후 댓글 객체: " + myReply);
    myReplyMapper.selectMyReply(458753L, myReply.getRno());
}
```

[테스트 후, 콘솔 표시 로그 일부]

```
INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - 매퍼 - 추가 전 댓글 객체: MyReplyVO(bno=458753, rno=0, rcontent=매퍼 댓글 입력 테스트 , rwriter=replyer, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : jdbc.sqlonly - SELECT book_ex.seq_myreply.NEXTVAL FROM dual

INFO : jdbc.sqltiming - SELECT book_ex.seq_myreply.NEXTVAL FROM dual
{executed in 98 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|13     |
|-----|

INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 13, '매퍼 댓글 입력 테스트 ', 'replyer', DEFAULT, DEFAULT, DEFAULT)

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 13, '매퍼 댓글 입력 테스트 ', 'replyer', DEFAULT, DEFAULT, DEFAULT)
{executed in 12 msec}
INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - 매퍼 - 추가 후 댓글 객체: MyReplyVO(bno=458753, rno=13, rcontent=매퍼 댓글 입력 테스트 , rwriter=replyer, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 13

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 13
{executed in 17 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|
|bno   |rno   |rcontent      |rwriter   |rregdate       |rmoddate       |prno   |
|-----|-----|-----|-----|-----|-----|-----|
|458753|13    |매퍼 댓글 입력 테스트 |replyer   |2021-06-22 20:33:20.0|2021-06-22 20:33:21.0|[null] |
|-----|-----|-----|-----|-----|-----|-----|
```

#### (5) 댓글에 대한 답글 등록 테스트

→ 테스트 메서드 추가

```
//댓글의 답글 등록 테스트
@Test
public void testInsertMyReplyForReply() {

    MyReplyVO myReply = new MyReplyVO();
    myReply.setBno(458753L); //SQL Developer에서 확인된 가장 최신 게시물의 bno를 지정
    myReply.setRcontent("매퍼 댓글의 답글 입력 테스트 ");
    myReply.setRwriter("test03");
    myReply.setPrno(1L);
    log.info("매퍼 - 추가 전 답글 객체: " + myReply);

    myReplyMapper.insertMyReplyForReply(myReply);
    log.info("매퍼 - 추가 후 답글 객체: " + myReply);

    myReplyMapper.selectMyReply(458753L, myReply.getRno());
}
```

### [테스트 후, 콘솔 표시 로그 일부]

```

INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - 매퍼 - 추가 전 댓글 객체: MyReplyVO(bno=458753, rno=0, rcontent=매퍼 댓글의 답글 입력 테스트 , rwriter=test03, rregDate=null, rmodDate=null, prno=1, lvl=0)
INFO : jdbc.sqlonly - SELECT book_ex.seq_myreply.NEXTVAL FROM dual

INFO : jdbc.sqltiming - SELECT book_ex.seq_myreply.NEXTVAL FROM dual
{executed in 85 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|14    |
|-----|

INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES (458753, 14, '매퍼 댓글의 답글 입력 테스트 ', 'test03', DEFAULT, DEFAULT, 1)

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES (458753, 14, '매퍼 댓글의 답글 입력 테스트 ', 'test03', DEFAULT, DEFAULT, 1)
{executed in 9 msec}
INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - 매퍼 - 추가 후 댓글 객체: MyReplyVO(bno=458753, rno=14, rcontent=매퍼 댓글의 답글 입력 테스트 , rwriter=test03, rregDate=null, rmodDate=null, prno=1, lvl=0)
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 14

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 14
{executed in 14 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|
|bno  |rno |rcontent      |rwriter |rregdate       |rmoddate      |prno |
|-----|-----|-----|-----|-----|-----|-----|
|458753|14  |매퍼 댓글의 답글 입력 테스트 |test03 |2021-06-22 20:35:06.0 |2021-06-22 20:35:06.0 |1   |
|-----|-----|-----|-----|-----|-----|-----|

```

### (6) 특정 댓글/답글 조회 테스트

#### → 테스트 메서드 추가

```

//특정 댓글/답글 조회 테스트
@Test
public void testSelectMyReply() {

    Long targetBno = 458753L; //SQL Developer에서 확인된 가장 최신 게시물의 bno를 지정.
    Long targetRno = 2L;

    //458753L
    MyReplyVO myReply = myReplyMapper.selectMyReply(targetBno, targetRno);
    log.info("특정 댓글/답글 조회 테스트: " + myReply);
}

```

### [테스트 후, 콘솔 표시 로그 일부]

```

INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 2

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 2
{executed in 116 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|
|bno  |rno |rcontent      |rwriter |rregdate       |rmoddate      |prno |
|-----|-----|-----|-----|-----|-----|-----|
|458753|2   |458753의 댓글2 |user2   |2021-06-22 17:47:13.0 |2021-06-22 17:47:13.0 |[null] |
|-----|-----|-----|-----|-----|-----|-----|

INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - 특정 댓글/답글 조회 테스트: MyReplyVO(bno=458753, rno=2, rcontent=458753의 댓글2, rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 17:47:13.0, prno=0, lvl=0)

```

## (7) 특정 댓글/답글 수정 테스트

→ 테스트 메서드 추가

```
//특정 댓글 수정 테스트
@Test
public void testUpdateMyReply() {

    Long targetBno = 458753L; //SQL Developer에서 확인된 가장 최신 게시물의 bno를 지정.
    Long targetRno = 2L;

    MyReplyVO myReply = myReplyMapper.selectMyReply(targetBno, targetRno);
    myReply.setRcontent("매퍼 - 수정 테스트..");

    int count = myReplyMapper.updateMyReply(myReply);
    log.info("UPDATE COUNT: " + count);
    log.info(myReplyMapper.selectMyReply(targetBno, targetRno));
}
```

[테스트 후, 콘솔 표시 로그 일부]

```
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 2
INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 2
{executed in 115 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|
|bno |rno |rcontent |rwriter |rregdate |rmoddate |prno |
|-----|-----|-----|-----|-----|-----|-----|
|458753 |2 |458753의 댓글2 |user2 |2021-06-22 17:47:13.0 |2021-06-22 17:47:13.0 |[null] |
|-----|-----|-----|-----|-----|-----|-----|
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myreply SET rcontent = '매퍼 - 수정 테스트..', rmodDate = DEFAULT WHERE bno= 458753
AND rno = 2
INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myreply SET rcontent = '매퍼 - 수정 테스트..', rmodDate = DEFAULT WHERE bno= 458753
AND rno = 2
{executed in 3 msec}
INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - UPDATE COUNT: 1
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 2
INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 2
{executed in 0 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|
|bno |rno |rcontent |rwriter |rregdate |rmoddate |prno |
|-----|-----|-----|-----|-----|-----|-----|
|458753 |2 |매퍼 - 수정 테스트.. |user2 |2021-06-22 17:47:13.0 |2021-06-22 20:40:25.0 |[null] |
|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - MyReplyVO(bno=458753, rno=2, rcontent=매퍼 - 수정 테스트.., rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0, lvl=0)
```

## (8) 특정 댓글/답글 삭제 테스트

→ 테스트 메서드 추가

```
//특정 게시물에 대한 특정 댓글 삭제 테스트
@Test
public void testDeleteMyReply() {

    Long targetBno = 458753L; //SQL Developer에서 확인된 가장 최신 게시물의 bno를 지정.
    Long targetRno = 4L;
    int count = myReplyMapper.deleteMyReply(targetBno, targetRno);
    log.info("DELETE COUNT: " + count);
    log.info(myReplyMapper.selectMyReply(targetBno, targetRno));
}
```

[테스트 후, 콘솔 표시 로그 일부]

```
INFO : jdbc.sqlonly - DELETE FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 4
INFO : jdbc.sqltiming - DELETE FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 4
{executed in 17 msec}
```

```

INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - DELETE COUNT: 1
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 4

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 4
{executed in 114 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|
|bno |rno |rcontent |rwriter |rregdate |rmoddate |prno |
|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----| ←삭제되었으므로 조회내용이 없습니다.

INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests -

```

→ 테스트 후, SQL\*Developer에서 book\_ex.tbl\_myreply 테이블에 조회하여 데이터 확인.

```
SQL> SELECT * FROM book_ex.tbl_myreply ORDER BY bno, rno;
```

BN0	RNO RCONTENT	RWRITER	RREGDATE	RMODDATE	PRNO
458753	1 458753 의 댓글 1	user1	21/06/22 21:06/22 17:47:12.000000000		
458753	2 매퍼 - 수정 테스트..	user2	21/06/22 21:06/22 20:40:25.000000000		
458753	3 458753 의 댓글 3	user3	21/06/22 21:06/22 17:47:14.000000000		
458753	5 458753 의 댓글 5	user5	21/06/22 21:06/22 17:47:16.000000000		
458753	6 458753 의 댓글 1 의 댓글 6	test02	21/06/22 21:06/22 17:47:30.000000000		1
458753	7 458753 의 댓글 1 의 댓글 7	test03	21/06/22 21:06/22 17:47:33.000000000		1
458753	8 458753 의 댓글 1 의 댓글 8	test04	21/06/22 21:06/22 17:47:34.000000000		1
458753	9 458753 의 댓글 6 의 댓글 9	test04	21/06/22 21:06/22 17:47:36.000000000		6
458753	10 458753 의 댓글 6 의 댓글 10	test02	21/06/22 21:06/22 17:47:38.000000000		6
458753	11 458753 의 댓글 11	test02	21/06/22 21:06/22 17:47:39.000000000		2
458753	12 458753 의 댓글 11 의 댓글 12	test03	21/06/22 21:06/22 17:47:41.000000000		11
458753	13 매퍼 댓글 입력 테스트	replayer	21/06/22 21:06/22 20:33:21.000000000		
458753	14 매퍼 댓글의 답글 입력 테스트	test03	21/06/22 21:06/22 20:35:06.000000000		1

13 개의 행이 선택됨

## 10-6. 댓글 목록에 대한 페이징 처리

☞ 게시물의 댓글이 많은 경우에도, 페이징 처리를 고려해야 합니다.

☞ 게시물과 다르게 댓글 목록 표시/댓글 추가/댓글 수정-삭제는 모두 게시물 상세 화면(detail.jsp)에서 Ajax를 통해 데이터만 받아와서 처리되므로 페이지 이동이 발생되지 않습니다.  
따라서, 댓글 목록을 표시하기 위한 페이징만 고려하면 됩니다.

### (1) MyReplyPagingDTO.java 클래스 생성

→ 프로젝트의 src/main/java/com.spring5213.mypro00.common.paging 패키지에 MyReplyPagingDTO.java 클래스 생성

다음의 코드 작성: bno 값을 포함시킵니다.

```
package com.spring5213.mypro00.common.paging;
```

```

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Setter
@Getter
@ToString
@EqualsAndHashCode
public class MyReplyPagingDTO {

    private long bno; //게시물 번호

```

```

private Integer pageNum;           //현재 페이지 번호
private int rowAmountPerPage; //페이지당 출력할 레코드 개수

//생성자 - 댓글의 페이징번호 클릭 시,
//페이지번호와 행수를 각각 사용자가 선택한 페이징번호와 10 으로 전달

public MyReplyPagingDTO(long bno, Integer pageNum) {
    this.bno = bno;

    if(pageNum == null) {

        this.pageNum = 1 ;

    } else {

        this.pageNum = pageNum;

    }

    this.rowAmountPerPage = 10 ;
}

}

```

## (2) MyReplyMapper 인터페이스 파일 수정

```

→ src/main/java/com.spring5212.mypro00.mapper.MyReplyMapper 인터페이스 파일 수정
→ selectMyReplyList() 메소드를 페이징 처리가 가능하게 수정하고, 특정 게시물의 댓글 총 개수를 확인하는 메서드 추가
//특정 게시물에 대한 댓글 목록 조회(페이징)
public List<MyReplyVO> selectMyReplyList(@Param("myReplyPagingDTO") MyReplyPagingDTO myReplyPagingDTO);

//특정 게시물의 댓글 총 개수확인
public int selectReplyTotalByBno(@Param("myReplyPagingDTO") MyReplyPagingDTO myReplyPagingDTO);

```

## (3) MyReplyMapper.xml SQL 매퍼 파일 수정

```

→ src/main/resources/com/spring5212/mypro00/mapper 폴더에 있는 MyReplyMapper.xml 파일 수정
→ 기존 댓글 목록 조회 SELECT 문을 페이징 처리가 가능하게 수정하고, 특정 게시물의 댓글 총 개수를 조회하는 SELECT문을 추가
<!-- 댓글 목록 - 계층 쿼리 (페이징) -->
<select id="selectMyReplyList" resultType="com.spring5213.mypro00.domain.MyReplyVO">
<![CDATA[
    SELECT bno, rno, rcontent, rwriter, rregDate, rmodDate, prno, lvl
    FROM ( SELECT ROWNUM rn, b.*
            FROM ( SELECT LEVEL lvl, a.*
                    FROM ( SELECT /*+ INDEX_ASC (a IDX_MYREPLY_BNO_RNO) */ *
                            FROM book_ex.tbl_myreply
                            WHERE bno = #{myReplyPagingDTO.bno}
                        ) a
                    START WITH prno IS NULL
                    CONNECT BY PRIOR rno = prno
                ) b
        )
    WHERE rn BETWEEN #{myReplyPagingDTO.pageNum} * #{myReplyPagingDTO.rowAmountPerPage}
        - (#{myReplyPagingDTO.rowAmountPerPage} - 1)
        AND #{myReplyPagingDTO.pageNum} * #{myReplyPagingDTO.rowAmountPerPage}
]]>
</select>

```

```

<!-- 특정 게시물의 댓글 총 개수 확인 -->
<select id="selectReplyTotalByBno" resultType="int">
<![CDATA[
    SELECT /*+ INDEX (TBL_MYREPLY IDX_MYREPLY_BNO_RNO) */ count(*)
    FROM book_ex.tbl_myreply
    WHERE bno = #{myReplyPagingDTO.bno}
]]>
</select>

```

### [테스트-댓글의 검색 페이징 쿼리]

→ src/test/java/com.spring5213.mypro00.mapper.MyReplyMapperTest.java 클래스를 오픈

→ 기존 테스트 메서드 모두를 주석처리 후, 다음의 메서드 추가 후 테스트 수행

```

// 특정 게시물에 대한 댓글 목록 조회(페이징) 테스트
@Test
public void testSelectMyReplyListPaging() {
    long targetBno = 458753L; //SQL Developer에서 확인된 가장 최신 게시물의 bno를 지정

    MyReplyPagingDTO myReplyPagingDTO = new MyReplyPagingDTO(targetBno, 1);

    long totalReplyCnt = myReplyMapper.selectReplyTotalByBno(myReplyPagingDTO);
    log.info("댓글 총 개수: " + totalReplyCnt);

    List<MyReplyVO> myReplies = myReplyMapper.selectMyReplyList(myReplyPagingDTO);
    myReplies.forEach(myReply -> System.out.println(myReply));
}

```

### [아이클립스 콘솔 로그 일부]

```

INFO : jdbc.sqlonly - SELECT /*+ INDEX (TBL_MYREPLY IDX_MYREPLY_BNO_RNO) */ count(*) FROM book_ex.tbl_myreply WHERE
bno = 458753

INFO : jdbc.sqltiming - SELECT /*+ INDEX (TBL_MYREPLY IDX_MYREPLY_BNO_RNO) */ count(*) FROM book_ex.tbl_myreply WHERE
bno = 458753
{executed in 113 msec}
INFO : jdbc.resultsettable -
|-----|
|count(*)|
|-----|
|13   |
|-----|

INFO : com.spring5213.mypro00.mapper.MyReplyMapperTests - 댓글 총 개수: 13
INFO : jdbc.sqlonly - SELECT bno, rno, rcontent, rwriter, rregDate, rmodDate, prno, lvl FROM ( SELECT ROWNUM rn,
b.* FROM ( SELECT LEVEL lvl, a.* FROM ( SELECT /*+ INDEX_ASC (a IDX_MYREPLY_BNO_RNO) */ * FROM
book_ex.tbl_myreply WHERE bno = 458753 ) a START WITH prno IS NULL CONNECT BY PRIOR rno = prno
) b ) WHERE rn BETWEEN 1 * 10 - (10 - 1) AND 1 * 10
{executed in 32 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |rno |rcontent      |rwriter       |rregdate      |rmoddate      |prno  |lvl  |
|-----|-----|-----|-----|-----|-----|-----|-----|
|458753|1  |458753 의 댓글 1    |user1        |2021-06-22 17:47:12.0|2021-06-22 17:47:12.0|[null]|1   |
|458753|6  |458753 의 댓글 1 의 댓글 6 |test02       |2021-06-22 17:47:30.0|2021-06-22 17:47:30.0|1     |2   |
|458753|9  |458753 의 댓글 6 의 댓글 9 |test04       |2021-06-22 17:47:35.0|2021-06-22 17:47:36.0|6     |3   |
|458753|10 |458753 의 댓글 6 의 댓글 10 |test02       |2021-06-22 17:47:37.0|2021-06-22 17:47:38.0|6     |3   |
|458753|7  |458753 의 댓글 1 의 댓글 7    |test03       |2021-06-22 17:47:32.0|2021-06-22 17:47:33.0|1     |2   |
|458753|8  |458753 의 댓글 1 의 댓글 8    |test04       |2021-06-22 17:47:34.0|2021-06-22 17:47:34.0|1     |2   |
|458753|14 |매퍼 댓글의 답글 입력 테스트 |test03       |2021-06-22 20:35:06.0|2021-06-22 20:35:06.0|1     |2   |
|458753|2  |매퍼 - 수정 테스트..       |user2        |2021-06-22 17:47:13.0|2021-06-22 20:40:25.0|[null]|1   |
|458753|11 |458753 의 댓글 11          |test02       |2021-06-22 17:47:39.0|2021-06-22 17:47:39.0|2     |2   |
|458753|12 |458753 의 댓글 11 의 댓글 12 |test03       |2021-06-22 17:47:41.0|2021-06-22 17:47:41.0|11    |3   |
|-----|-----|-----|-----|-----|-----|-----|-----|
```

MyReplyVO(bno=458753, rno=1, rcontent=458753 의 댓글 1, rwriter=user1, rregDate=2021-06-22 17:47:12.0, rmodDate=2021-06-22 17:47:12.0, prno=0, lvl=1)  
MyReplyVO(bno=458753, rno=6, rcontent=458753 의 댓글 1 의 댓글 6, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2)  
MyReplyVO(bno=458753, rno=9, rcontent=458753 의 댓글 6 의 댓글 9, rwriter=test04, rregDate=2021-06-22 17:47:35.0, rmodDate=2021-06-22 17:47:36.0, prno=6, lvl=3)

```

MyReplyVO(bno=458753, rno=10, rcontent=458753 의 댓글 6 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:37.0, rmodDate=2021-06-22 17:47:38.0,
prno=6, lvl=3)
MyReplyVO(bno=458753, rno=7, rcontent=458753 의 댓글 1 의 댓글 7, rwriter=test03, rregDate=2021-06-22 17:47:32.0, rmodDate=2021-06-22 17:47:33.0,
prno=1, lvl=2)
MyReplyVO(bno=458753, rno=8, rcontent=458753 의 댓글 1 의 댓글 8, rwriter=test04, rregDate=2021-06-22 17:47:34.0, rmodDate=2021-06-22 17:47:34.0,
prno=1, lvl=2)
MyReplyVO(bno=458753, rno=14, rcontent=매퍼 댓글의 답글 입력 테스트 , rwriter=test03, rregDate=2021-06-22 20:35:06.0, rmodDate=2021-06-22 20:35:06.0,
prno=1, lvl=2)
MyReplyVO(bno=458753, rno=2, rcontent=매퍼 - 수정 테스트.., rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0,
lvl=1)
MyReplyVO(bno=458753, rno=11, rcontent=458753 의 댓글 11, rwriter=test02, rregDate=2021-06-22 17:47:39.0, rmodDate=2021-06-22 17:47:39.0, prno=2,
lvl=2)
MyReplyVO(bno=458753, rno=12, rcontent=458753 의 댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0,
prno=11, lvl=3)

```

#### (4) MyReplyPagingCreatorDTO.java 클래스 생성

→ src/main/java/com.spring5213.mypro00.common.paging 패키지에 MyReplyPagingCreatorDTO.java 클래스 생성하고

다음의 코드 작성: 페이지 관련 필요한 데이터와 댓글 목록을 모두 전달

```

package com.spring5213.mypro00.common.paging;

import java.util.List;

import com.spring5213.mypro00.domain.MyReplyVO;

import lombok.Getter;
import lombok.ToString;

public class MyReplyPagingCreatorDTO {

    private MyReplyPagingDTO myReplyPaging ; //페이지번호와 행 개수 저장 객체
    private int startPagingNum; //화면에 표시되는 시작 페이지 번호
    private int endPagingNum; //화면에 표시되는 끝 페이지 번호
    private boolean prev; //이전 버튼 표시 여부 결정 변수(true: 표시됨, false: 표시 안됨)
    private boolean next; //다음 버튼 표시 여부 결정 변수(true: 표시됨, false: 표시 안됨)
    private long replyTotalByBno; //전체 행 개수
    private int pagingNumCnt; //화면 하단에 표시할 기본 페이지 번호(10)
    private int lastPageNum ; //마지막 페이지 번호

    private List<MyReplyVO> replyList;

    public MyReplyPagingCreatorDTO(long replyTotalByBno,
                                   MyReplyPagingDTO myReplyPaging,
                                   List<MyReplyVO> replyList) {
        this.myReplyPaging = myReplyPaging ;
        this.replyTotalByBno = replyTotalByBno;
        this.replyList = replyList ;
        this.pagingNumCnt = 10;

        //계산된 끝-시작 페이지 번호:
        this.endPagingNum =
            (int)( Math.ceil(myReplyPaging.getPageNum() / (this.pagingNumCnt * 1.0) ) ) * this.pagingNumCnt ;

        this.startPagingNum = this.endPagingNum - (this.pagingNumCnt -1);

        //행 총수를 기준으로 한 실제 마지막 페이지 번호
        this.lastPageNum = (int)( Math.ceil( (replyTotalByBno * 1.0) / myReplyPaging.getRowAmountPerPage() ) );

        //계산된 끝 페이지 번호가 실제 마지막 페이지 번호보다 크면, endPagingNum 값을 lastPageNum 으로 대체
        if (lastPageNum < this.endPagingNum) {
            this.endPagingNum = lastPageNum ;
        }

        this.prev = this.startPagingNum > 1 ;
        this.next = this.endPagingNum < lastPageNum ;
    }
}

```

```

        System.out.println("댓글-전달된 페이징 기본데이터-MyReplyPagingDTO: " + myReplyPaging.toString());
        System.out.println("댓글-끝 페이징번호: " + this.endPagingNum);
        System.out.println("댓글-시작 페이징번호: " + this.startPagingNum);
        System.out.println("댓글-이전버튼 표시 여부: " + this.prev);
        System.out.println("댓글-다음버튼 표시 여부: " + this.next);
        System.out.println("전달된 댓글 목록 데이터: " + this.replyList);
    }

}

```

## 10-7. 댓글 처리를 위한 비즈니스 계층(서비스) 구현

→ src/main/java/com.spring5213.mypro00.service 패키지에 MyReplyService 인터페이스를 생성하고 다음의 코드를 작성

```

package com.spring5213.mypro00.service;

import com.spring5213.mypro00.common.paging.MyReplyPagingCreatorDTO;
import com.spring5213.mypro00.common.paging.MyReplyPagingDTO;
import com.spring5213.mypro00.domain.MyReplyVO;

public interface MyReplyService {

    //특정 게시물에 대한 댓글 목록 조회(페이징1)
    //public List<MyReplyVO> getReplyListByBno(MyReplyPagingDTO myReplyPaging);

    //특정 게시물에 대한 댓글 목록 조회(페이징2-수정)
    public MyReplyPagingCreatorDTO getReplyListByBno(MyReplyPagingDTO myReplyPaging);

    //특정 게시물의 댓글 총 개수확인
    public long getReplyTotalByBno(MyReplyPagingDTO myReplyPaging);

    //특정 게시물에 대한 댓글 등록: rno 반환
    public long registerReplyForBoard(MyReplyVO myReply);

    //댓글의 답글 등록: rno 반환
    public long registerReplyForReply(MyReplyVO myReply);

    //특정 게시물에 대한 특정 댓글/답글 조회
    public MyReplyVO getReply(long bno, long rno);

    //특정 게시물에 대한 특정 댓글/답글 수정
    public int modifyReply(MyReplyVO myReply);

    //특정 게시물에 대한 특정 댓글/답글 삭제
    public int removeReply(long bno, long rno);

}

```

→ src/main/java 폴더의 com.spring5213.mypro00.service 패키지에 있는 MyReplyServiceImpl 구현객체 클래스 생성하고 MyReplyMapper 인터페이스의 메서드를 이용하여 비즈니스 로직 처리에 필요한 데이터를 처리할 수 있도록 다음의 코드를 작성

```

package com.spring5213.mypro00.service;

import org.springframework.stereotype.Service;

```

```

import com.spring5213.mypro00.common.paging.MyReplyPagingCreatorDTO;
import com.spring5213.mypro00.common.paging.MyReplyPagingDTO;
import com.spring5213.mypro00.domain.MyReplyVO;
import com.spring5213.mypro00.mapper.MyReplyMapper;

import lombok.extern.log4j.Log4j;

@Service
@Log4j
//{@AllArgsConstructor
public class MyReplyServiceImpl implements MyReplyService {

    //자동주입 방법1: 생성자 자동 주입 방식으로 주입 시
    private MyReplyMapper myReplyMapper;

    public MyReplyServiceImpl(MyReplyMapper myReplyMapper) {
        this.myReplyMapper = myReplyMapper;
    }
}

/*
//자동주입 방법2: @Autowired를 이용한 자동 주입(setter 자동 주입과 동일)
//    @Autowired
//    private MyReplyMapper myReplyMapper;
//
//자동주입 방법3: setter를 이용한 자동 주입//lombok 어노테이션 이용
//    @Setter(onMethod_ = @Autowired)
//    private MyReplyMapper myReplyMapper;
//
//    private MyReplyMapper myReplyMapper;
//    @Autowired
//    public void setMapper(MyReplyMapper myReplyMapper) {
//        this.myReplyMapper = myReplyMapper;
//    }
*/
//특정 게시물에 대한 댓글 목록 조회(페이징-전체댓글 수 고려)
@Override
public MyReplyPagingCreatorDTO getReplyListByBno(MyReplyPagingDTO myReplyPaging) {
    log.info("댓글-서비스-게시물에 대한 댓글 목록 조회 - 전달된 MyReplyPagingDTO: " + myReplyPaging);
    MyReplyPagingCreatorDTO myReplyPagingCreator =
        new MyReplyPagingCreatorDTO(
            myReplyMapper.selectReplyTotalByBno(myReplyPaging),
            myReplyPaging,
            myReplyMapper.selectMyReplyList(myReplyPaging));
    log.info("댓글-서비스-게시물에 대한 댓글 목록 조회 - 생성된 myReplyPagingCreatorDTO: " + myReplyPagingCreator);
    log.info("댓글-서비스-게시물에 대한 댓글 목록 조회 - myReplyPagingCreator가 컨트롤러로 전달됨");

    return myReplyPagingCreator;
}

//특정 게시물의 댓글 총 개수확인
@Override
public long getReplyTotalByBno(MyReplyPagingDTO myReplyPaging) {
    return myReplyMapper.selectReplyTotalByBno(myReplyPaging);
}

//특정 게시물에 대한 댓글 등록: rno 반환
@Override
public long registerReplyForBoard(MyReplyVO myReply) {
    log.info("댓글-서비스-게시물에 대한 댓글 등록 시 처음 전달된 myReplyVO: " + myReply);

    //게시물에 대한 댓글 등록 처리
    myReplyMapper.insertMyReplyForBoard(myReply);
    log.info("댓글-서비스 - 게시물에 대한 댓글 등록: 등록된 myReply: " + myReply);
    log.info("댓글-서비스 - 게시물에 대한 댓글 등록: 등록된 rno(byGetter): " + myReply.getRno());
    log.info("댓글-서비스 - 게시물에 대한 댓글 등록: 등록된 rno가 컨트롤러로 전달됨 " );

    return myReply.getRno();
}

```

```

//댓글의 답글 등록: rno 반환
@Override
public long registerReplyForReply(MyReplyVO myReply) {
    log.info("댓글-서비스-게시물에 대한 댓글 등록 시 myReplyVO: " + myReply);

    //댓글에 대한 답글 등록 DB 저장
    myReplyMapper.insertMyReplyForReply(myReply);

    log.info("댓글-서비스 - 댓글에 대한 답글 등록: 등록된 myReply: " + myReply);
    log.info("댓글-서비스 - 댓글에 대한 답글 등록: 등록된 rno(ByGetter): " + myReply.getRno());
    log.info("댓글-서비스 - 댓글에 대한 답글 등록: 등록된 rno가 컨트롤러로 전달됨");

    return myReply.getRno();
}

//특정 댓글 조회
@Override
public MyReplyVO getReply(long bno, long rno) {
    log.info("댓글-서비스-댓글-조회 시 전달된 bno: " + bno);
    log.info("댓글-서비스-댓글-조회 시 전달된 rno: " + rno);

    MyReplyVO myReply = myReplyMapper.selectMyReply(bno, rno);
    log.info("댓글-서비스-댓글-조회: 매퍼로부터 전달된 myReply: " + myReply);
    log.info("댓글-서비스-댓글-조회: myReply가 컨트롤러로 전달됨");

    return myReply;
}

//특정 댓글 수정: 수정된 행수인 1 반환
@Override
public int modifyReply(MyReplyVO myReply) {
    log.info("댓글-서비스-댓글-수정시 전달된 myReplyVO: " + myReply);

    Integer modCnt = myReplyMapper.updateMyReply(myReply);
    log.info("댓글-서비스-수정된 댓글 개수: " + modCnt);

    return modCnt;
}

//특정 댓글 삭제: 삭제된 행수인 1 반환
@Override
public int removeReply(long bno, long rno) {
    log.info("댓글-서비스-댓글-삭제시 전달된 bno: " + bno);
    log.info("댓글-서비스-댓글-삭제시 전달된 rno: " + rno);

    Integer delCnt = myReplyMapper.deleteMyReply(bno, rno);
    log.info("댓글-서비스-삭제된 댓글 개수: " + delCnt);

    return delCnt;
}
}

```

## [MyReplyService 테스트]

☞ 테스트 전에 SQL\*Developer에서 댓글이 있는 게시물 번호 확인해서 코드 수정 후 테스트를 수행합니다.(문서에서는, 458753 입니다)

## (1) MyReplyServiceTests 클래스 생성

→ src/test/java/com.spring5213.mypro00.service 패키지에 MyReplyServiceTests 클래스 생성하고

다음의 내용 작성

```
package com.spring5213.mypro00.service;

import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@WebAppConfiguration //테스트 시, DispatcherServlet의 servlet-context.xml 설정 구성파일(들)을 사용하기 위한 어노테이션
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({ "file:src/main/webapp/WEB-INF/spring/mybatis-context.xml",
                      "file:src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml"})
@Log4j
public class MyReplyServiceTests {
    //JUnit 테스트에서는 생성자 주입 방식은 오류를 발생하므로, Setter를 이용한 주입방식을 사용해야 합니다.
    @Setter(onMethod_ = { @Autowired })
    private MyReplyService myReplyService;

}
```

☞ 위의 클래스에 테스트가 완료된 메서드는 주석처리 후, 새로운 메서드를 하나씩 추가하면서 테스트를 수행합니다.

☞ 테스트 방법은, 변경내용 저장 후, 마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트 수행

→ 콘솔에 표시된 로그 확인 순으로 진행합니다. 문서에서는 표시된 로그 중 테스트 환경 구성 로그와 HikariDataSource 로그는 제외했습니다. 문서에는 테스트 메서드와 처리로그만 명시합니다.

## (2) 서비스 객체 주입 확인 테스트

→ 테스트 메서드 추가

```
//JUnit 테스트에서는 생성자 주입 방식은 오류를 발생하므로, Setter를 이용한 주입방식을 사용해야 합니다.
```

```
@Setter(onMethod_ = { @Autowired })
```

```
private MyReplyService myReplyService; ←주석 처리하면 됩니다.
```

```
//MyReplyService 빈 생성 유무 확인 테스트 ←테스트 후, 메서드만 주석 처리
```

```
@Test
```

```
public void testMyReplyServiceExist() {
```

```
    log.info(myReplyService);
```

```
    assertNotNull(myReplyService);
```

[테스트 후, 콘솔 표시 로그 일부]

```
INFO : com.spring5213.mypro00.service.MyReplyServiceTests - com.spring5213.mypro00.service.MyReplyServiceImpl@779dfe55
```

### (3) 게시물에 대한 댓글 목록 조회 테스트

☞ 댓글이 있는 게시물의 bno: 458753

→ 테스트 메서드 추가

```
// 특정 게시물의 댓글 목록 조회 테스트
@Test
public void testGetReplyList() {
    MyReplyPagingCreatorDTO myReplyPagingCreator =
        myReplyService.getReplyListByBno(new MyReplyPagingDTO(458753, 1));
    log.info("댓글-서비스-게시물에 대한 댓글 목록 조회 테스트 - 반환된 MyReplyPagingCreatorDTO: "
        + myReplyPagingCreator );
}
```

[테스트 후, 콘솔 표시 로그 일부]

```
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 목록 조회 - 전달된 MyReplyPagingDTO:
MyReplyPagingDTO(bno=458753, pageNum=1, rowAmountPerPage=10)
INFO : jdbc.sqlonly - SELECT /*+ INDEX (TBL_MYREPLY IDX_MYREPLY_BNO_RNO) */ count(*) FROM book_ex.tbl_myreply WHERE
bno = 458753

INFO : jdbc.sqltiming - SELECT /*+ INDEX (TBL_MYREPLY IDX_MYREPLY_BNO_RNO) */ count(*) FROM book_ex.tbl_myreply WHERE
bno = 458753
{executed in 104 msec}
INFO : jdbc.resultsettable -
|-----|
|count(*) |
|-----|
|13   |
|-----|

INFO : jdbc.sqlonly - SELECT bno, rno, rcontent, rwriter, rregDate, rmodDate, prno, lvl FROM ( SELECT ROWNUM rn,
b.* FROM ( SELECT LEVEL lvl, a.* FROM ( SELECT /*+ INDEX_ASC (a IDX_MYREPLY_BNO_RNO) */ * FROM
book_ex.tbl_myreply WHERE bno = 458753 ) a START WITH prno IS NULL CONNECT BY PRIOR rno = prno
) b ) WHERE rn BETWEEN 1 * 10 - (10 - 1) AND 1 * 10

INFO : jdbc.sqltiming - SELECT bno, rno, rcontent, rwriter, rregDate, rmodDate, prno, lvl FROM ( SELECT ROWNUM rn,
b.* FROM ( SELECT LEVEL lvl, a.* FROM ( SELECT /*+ INDEX_ASC (a IDX_MYREPLY_BNO_RNO) */ * FROM
book_ex.tbl_myreply WHERE bno = 458753 ) a START WITH prno IS NULL CONNECT BY PRIOR rno = prno
) b ) WHERE rn BETWEEN 1 * 10 - (10 - 1) AND 1 * 10
{executed in 10 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|
|bno |rno |rcontent      |rwriter |rregdate       |rmoddate      |prno   |lvl  |
|-----|-----|-----|-----|-----|-----|-----|-----|
|458753|1  |458753 의 댓글 1  |user1   |2021-06-22 17:47:12.0|2021-06-22 17:47:12.0|[null] |1   |
|458753|6  |458753 의 댓글 1 의 댓글 6 |test02  |2021-06-22 17:47:30.0|2021-06-22 17:47:30.0|1     |2   |
|458753|9  |458753 의 댓글 6 의 댓글 9 |test04  |2021-06-22 17:47:35.0|2021-06-22 17:47:36.0|6     |3   |
|458753|10 |458753 의 댓글 6 의 댓글 10 |test02  |2021-06-22 17:47:37.0|2021-06-22 17:47:38.0|6     |3   |
|458753|7  |458753 의 댓글 1 의 댓글 7 |test03  |2021-06-22 17:47:32.0|2021-06-22 17:47:33.0|1     |2   |
|458753|8  |458753 의 댓글 1 의 댓글 8 |test04  |2021-06-22 17:47:34.0|2021-06-22 17:47:34.0|1     |2   |
|458753|14 |매퍼 댓글의 답글 입력 테스트 |test03  |2021-06-22 20:35:06.0|2021-06-22 20:35:06.0|1     |2   |
|458753|2  |매퍼 - 수정 테스트..   |user2   |2021-06-22 17:47:13.0|2021-06-22 20:40:25.0|[null] |1   |
|458753|11 |458753 의 댓글 11    |test02  |2021-06-22 17:47:39.0|2021-06-22 17:47:39.0|2     |2   |
|458753|12 |458753 의 댓글 11 의 댓글 12 |test03  |2021-06-22 17:47:41.0|2021-06-22 17:47:41.0|11    |3   |
|-----|-----|-----|-----|-----|-----|-----|-----|
```

댓글-전달된 페이지 기본데이터-MyReplyPagingDTO: MyReplyPagingDTO(bno=458753, pageNum=1, rowAmountPerPage=10)  
댓글-끝 페이지번호: 2  
댓글-시작 페이지번호: 1  
댓글-이전버튼 표시 여부: false  
댓글-다음버튼 표시 여부: false  
전달된 댓글 목록 데이터: [MyReplyVO(bno=458753, rno=1, rcontent=458753 의 댓글 1, rwriter=user1, rregDate=2021-06-22 17:47:12.0, rmodDate=2021-06-22 17:47:12.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=6, rcontent=458753 의 댓글 1 의 댓글 6, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=9, rcontent=458753 의 댓글 6 의 댓글 9, rwriter=test04, rregDate=2021-06-22 17:47:35.0, rmodDate=2021-06-22 17:47:36.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=10, rcontent=458753 의 댓글 6 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:37.0, rmodDate=2021-06-22 17:47:38.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=7, rcontent=458753 의 댓글 1 의 댓글 7, rwriter=test03, rregDate=2021-06-22 17:47:32.0, rmodDate=2021-06-22 17:47:33.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=8, rcontent=458753 의 댓글 1 의 댓글 8, rwriter=test04, rregDate=2021-06-22 17:47:34.0, rmodDate=2021-06-22 17:47:34.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=14, rcontent=매퍼 댓글의 답글 입력 테스트 , rwriter=test03, rregDate=2021-06-22 20:35:06.0, rmodDate=2021-06-22 20:35:06.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=2, rcontent=매퍼 - 수정 테스트.., rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=11, rcontent=458753 의 댓글 11, rwriter=test02, rregDate=2021-06-22 17:47:39.0, rmodDate=2021-06-22 17:47:39.0, prno=2, lvl=2), MyReplyVO(bno=458753, rno=12, rcontent=458753 의 댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=3)]  
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 목록 조회 - 생성된 myReplyPagingCreatorDTO:  
MyReplyPagingCreatorDTO(myReplyPaging=MyReplyPagingDTO(bno=458753, pageNum=1, rowAmountPerPage=10), startPagingNum=1, endPagingNum=2, prev=false, next=false, replyTotalByBno=13, pagingNumCnt=10, lastPageNum=2, replyList=[MyReplyVO(bno=458753, rno=1, rcontent=458753 의 댓글 1, rwriter=user1, rregDate=2021-06-22 17:47:12.0, rmodDate=2021-06-22 17:47:12.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=6, rcontent=458753 의 댓글 1 의 댓글 6, rwriter=test02, rregDate=2021-06-22 17:47:12.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=9, rcontent=458753 의 댓글 6 의 댓글 9, rwriter=test04, rregDate=2021-06-22 17:47:12.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=10, rcontent=458753 의 댓글 6 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:12.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=7, rcontent=458753 의 댓글 1 의 댓글 7, rwriter=test03, rregDate=2021-06-22 17:47:12.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=8, rcontent=458753 의 댓글 1 의 댓글 8, rwriter=test04, rregDate=2021-06-22 17:47:12.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=14, rcontent=매퍼 댓글의 답글 입력 테스트 , rwriter=test03, rregDate=2021-06-22 20:35:06.0, rmodDate=2021-06-22 20:35:06.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=2, rcontent=매퍼 - 수정 테스트.., rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=11, rcontent=458753 의 댓글 11, rwriter=test02, rregDate=2021-06-22 17:47:39.0, rmodDate=2021-06-22 17:47:39.0, prno=2, lvl=2), MyReplyVO(bno=458753, rno=12, rcontent=458753 의 댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=3)], prevPageNum=1, nextPageNum=2, prevPageLink=false, nextPageLink=true]

```

17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=11, rcontent=458753 의 댓글 11, rwriter=test02, rregDate=2021-06-22 17:47:39.0, rmodDate=2021-06-22 17:47:39.0, prno=2, lvl=2), MyReplyVO(bno=458753, rno=12, rcontent=458753 의 댓글 11의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=3)])
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 목록 조회 - myReplyPagingCreator 가 컨트롤러로 전달됨
INFO : com.spring5213.mypro00.service.MyReplyServiceTests - 댓글-서비스-게시물에 대한 댓글 목록 조회 테스트- 반환된 MyReplyPagingCreatorDTO:
MyReplyPagingCreatorDTO(myReplyPaging=MyReplyPagingDTO(bno=458753, pageNum=1, rowAmountPerPage=10), startPagingNum=1, endPagingNum=2, prev=false, next=false, replyTotalByBno=13, pagingNumCnt=10, lastPageNum=2, replyList=[MyReplyVO(bno=458753, rno=1, rcontent=458753 의 댓글 1, rwriter=user1, rregDate=2021-06-22 17:47:12.0, rmodDate=2021-06-22 17:47:12.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=6, rcontent=458753 의 댓글 1의 댓글 6, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=9, rcontent=458753 의 댓글 6의 댓글 9, rwriter=test04, rregDate=2021-06-22 17:47:35.0, rmodDate=2021-06-22 17:47:36.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=10, rcontent=458753 의 댓글 6의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:37.0, rmodDate=2021-06-22 17:47:38.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=7, rcontent=458753 의 댓글 1의 댓글 7, rwriter=test03, rregDate=2021-06-22 17:47:32.0, rmodDate=2021-06-22 17:47:33.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=8, rcontent=458753 의 댓글 1의 댓글 8, rwriter=test04, rregDate=2021-06-22 17:47:34.0, rmodDate=2021-06-22 17:47:34.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=14, rcontent=매퍼 댓글의 답글 입력 테스트 , rwriter=test03, rregDate=2021-06-22 20:35:06.0, rmodDate=2021-06-22 20:35:06.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=2, rcontent=매퍼 - 수정 테스트.., rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=11, rcontent=458753 의 댓글 11, rwriter=test02, rregDate=2021-06-22 17:47:39.0, rmodDate=2021-06-22 17:47:39.0, prno=2, lvl=2), MyReplyVO(bno=458753, rno=12, rcontent=458753 의 댓글 11의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=3)])

```

#### (4) 게시물에 대한 댓글 등록 테스트

☞ 댓글이 있는 게시물의 bno: **458753**

→ 테스트 메서드 추가

```

//게시물에 대한 댓글 등록(rno 반환) 테스트
@Test
public void testRegisterReplyForBoard() {
    MyReplyVO myReply = new MyReplyVO();
    myReply.setBno(458753L);
    myReply.setRcontent("서비스 게시물에 대한 댓글 등록 테스트");
    myReply.setRwriter("test");

    Long registeredRno = myReplyService.registerReplyForBoard(myReply);
    log.info("서비스 게시물에 대한 댓글 등록 테스트-등록된 댓글번호: " + registeredRno);
}

```

#### [테스트 후, 콘솔 표시 로그 일부]

```

INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 등록 시 처음 전달된 myReplyVO: MyReplyVO(bno=458753, rno=0, rcontent=서비스 게시물에 대한 댓글 등록 테스트, rwriter=test, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : jdbc.sqlonly - SELECT book_ex.seq_myreply.NEXTVAL FROM dual

INFO : jdbc.sqltiming - SELECT book_ex.seq_myreply.NEXTVAL FROM dual
{executed in 95 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|16     | ← 답글 등록 테스트에서, 답글이 등록되는 댓글의 번호로 사용됩니다.
|-----|

INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 16, '서비스 게시물에 대한 댓글 등록 테스트', 'test', DEFAULT, DEFAULT, DEFAULT)

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 16, '서비스 게시물에 대한 댓글 등록 테스트', 'test', DEFAULT, DEFAULT, DEFAULT)
{executed in 11 msec}

INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 게시물에 대한 댓글 등록: 등록된 myReply: MyReplyVO(bno=458753, rno=16, rcontent=서비스 게시물에 대한 댓글 등록 테스트, rwriter=test, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 게시물에 대한 댓글 등록: 등록된 rno(byGetter): 16
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 게시물에 대한 댓글 등록: 등록된 rno 가 컨트롤러로 전달됨
INFO : com.spring5213.mypro00.service.MyReplyServiceTests - 서비스 게시물에 대한 댓글 등록 테스트-등록된 댓글번호: 16

```

## (5) 댓글에 대한 답글 등록 테스트

☞ 댓글이 있는 게시물의 bno: 458753

☞ 답글이 등록되는 댓글의 rno는 앞의 테스트에서 등록한 댓글번호를 이용합니다(문서에서는 16L).

→ 테스트 메서드 추가

```
//게시물의 댓글에 대한 답글 등록(rno 반환) 테스트
@Test
public void testRegisterReplyForReply() {
    MyReplyVO myReply = new MyReplyVO();
    myReply.setBno(458753L);
    myReply.setRcontent("서비스 테스트 - 게시물의 댓글에 대한 답글 등록 테스트");
    myReply.setRwriter("test");
    myReply.setPrno(16L); //앞에서 등록한 댓글번호

    Long registeredRno = myReplyService.registerReplyForReply(myReply);
    log.info("서비스 게시물의 댓글에 대한 답글 등록 테스트-등록된 댓글번호: " + registeredRno);
}
```

[테스트 후, 콘솔 표시 로그 일부]

```
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글에 대한 답글 등록 시 myReplyVO: MyReplyVO(bno=458753, rno=0, rcontent=서비스 테스트 - 게시물의 댓글에 대한 댓글 등록 테스트, rwriter=test, rregDate=null, rmodDate=null, prno=16, lvl=0)
INFO : jdbc.sqlonly - SELECT book_ex.seq_myreply.NEXTVAL FROM dual

INFO : jdbc.sqltiming - SELECT book_ex.seq_myreply.NEXTVAL FROM dual
{executed in 97 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|17     |
|-----|

INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 17, '서비스 테스트 - 게시물의 댓글에 대한 댓글 등록 테스트', 'test', DEFAULT, DEFAULT, 16)

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 17, '서비스 테스트 - 게시물의 댓글에 대한 댓글 등록 테스트', 'test', DEFAULT, DEFAULT, 16)
{executed in 11 msec}
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 댓글에 대한 답글 등록: 등록된 myReply: MyReplyVO(bno=458753, rno=17, rcontent=서비스 테스트 - 게시물의 댓글에 대한 답글 등록 테스트, rwriter=test, rregDate=null, rmodDate=null, prno=16, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 댓글에 대한 답글 등록: 등록된 rno(ByGetter): 17
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 댓글에 대한 답글 등록: 등록된 rno 가 컨트롤러로 전달됨
INFO : com.spring5213.mypro00.service.MyReplyServiceTests - 서비스 게시물의 댓글에 대한 답글 등록 테스트-등록된 댓글번호: 17
```

## (6) 특정 게시물에 대한 특정 댓글/답글 조회 테스트

☞ 댓글이 있는 게시물의 bno: 458753

☞ 댓글의 답글 등록 테스트에서 등록된 답글의 글번호를 이용합니다(문서에서는 17L).

→ 테스트 메서드 추가

```
//특정 게시물에 대한 특정 댓글 조회
@Test
public void testGetReply() {
    log.info(myReplyService.getReply(458753L, 17L));
}
```

[테스트 후, 콘솔 표시 로그 일부]

```

INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회 시 전달된 bno: 458753
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회 시 전달된 rno: 17
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 17

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 17
{executed in 117 msec}
INFO : jdbc.resultsettable -
+-----+-----+-----+-----+-----+
|bno |rno |rcontent |rwriter |rregdate |rmoddate |prno |
+-----+-----+-----+-----+-----+
|458753 |17 |서비스 테스트 - 게시물의 댓글에 대한 답글 등록 테스트 |test |2021-06-23 09:55:04.0 |2021-06-23 09:55:04.0 |16 |
+-----+-----+-----+-----+-----+

INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회: 매퍼로부터 전달된 myReply: MyReplyVO(bno=458753, rno=17, rcontent=서비스 테스트 - 게시물의 댓글에 대한 댓글 등록 테스트, rwriter=test, rregDate=2021-06-23 09:55:04.0, rmodDate=2021-06-23 09:55:04.0, prno=16, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회: myReply 가 컨트롤러로 전달됨
INFO : com.spring5213.mypro00.service.MyReplyServiceTests - MyReplyVO(bno=458753, rno=17, rcontent=서비스 테스트 - 게시물의 댓글에 대한 댓글 등록 테스트, rwriter=test, rregDate=2021-06-23 09:55:04.0, rmodDate=2021-06-23 09:55:04.0, prno=16, lvl=0)

```

## (7) 특정 게시물에 대한 특정 댓글 수정 테스트

☞ 댓글이 있는 게시물의 bno: **458753**

☞ 댓글/답글 조회 테스트에서 조회된 댓글의 번호를 이용합니다(문서에서는 **17L**).

→ 테스트 메서드 추가

```

//게시물에 대한 특정 댓글 수정 테스트
@Test
public void testModifyReply() {

    MyReplyVO myReply = myReplyService.getReply(458753L, 17L);
    if (myReply == null) {
        return;
    }

    myReply.setRcontent("서비스-댓글 수정테스트입니다.수정");
    log.info("서비스-댓글 수정테스트 시 반환된 값(수정된 댓글 수): " + myReplyService.modifyReply(myReply));
    log.info("서비스-댓글 수정테스트: 수정 후 조회(수정된 myReplyVO): " + myReplyService.getReply(458753L, 17L));
}

```

## [테스트 후, 콘솔 표시 로그 일부]

```

INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회 시 전달된 bno: 458753
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회 시 전달된 rno: 17
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 17

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 17
{executed in 111 msec}
INFO : jdbc.resultsettable -
+-----+-----+-----+-----+-----+
|bno |rno |rcontent |rwriter |rregdate |rmoddate |prno |
+-----+-----+-----+-----+-----+
|458753 |17 |서비스 테스트 - 게시물의 댓글에 대한 답글 등록 테스트 |test |2021-06-23 09:55:04.0 |2021-06-23 09:55:04.0 |16 |
+-----+-----+-----+-----+-----+

INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회: 매퍼로부터 전달된 myReply: MyReplyVO(bno=458753, rno=17, rcontent=서비스 테스트 - 게시물의 댓글에 대한 답글 등록 테스트, rwriter=test, rregDate=2021-06-23 09:55:04.0, rmodDate=2021-06-23 09:55:04.0, prno=16, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회: myReply 가 컨트롤러로 전달됨
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-수정시 전달된 myReplyVO: MyReplyVO(bno=458753, rno=17, rcontent=서비스-댓글 수정테스트입니다.수정, rwriter=test, rregDate=2021-06-23 09:55:04.0, rmodDate=2021-06-23 09:55:04.0, prno=16, lvl=0)
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myreply SET rcontent = '서비스-댓글 수정테스트입니다.수정', rmodDate = DEFAULT WHERE bno= 458753 AND rno = 17

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myreply SET rcontent = '서비스-댓글 수정테스트입니다.수정', rmodDate = DEFAULT WHERE bno= 458753 AND rno = 17
{executed in 2 msec}
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-수정된 댓글 개수: 1
INFO : com.spring5213.mypro00.service.MyReplyServiceTests - 서비스-댓글 수정테스트 시 반환된 값(수정된 댓글 수): 1
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회 시 전달된 bno: 458753
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회 시 전달된 rno: 17
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 17

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 17
{executed in 1 msec}
INFO : jdbc.resultsettable -

```

bno	rno	rcontent	rwriter	rregdate	rmddate	prno
458753	17	서비스-댓글 수정테스트입니다.수정	test	2021-06-23 09:55:04.0	2021-06-23 10:04:52.0	16

```

INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회: 매퍼로부터 전달된 myReply: MyReplyVO(bno=458753, rno=17, rcontent=서비스-댓글 수정테스트입니다.수정, rwriter=test, rregDate=2021-06-23 09:55:04.0, rmodDate=2021-06-23 10:04:52.0, prno=16, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회: myReply 가 컨트롤러로 전달됨
INFO : com.spring5213.mypro00.service.MyReplyServiceTests - 서비스-댓글 수정테스트: 수정 후 조회(수정된 myReplyVO): MyReplyVO(bno=458753, rno=17, rcontent=서비스-댓글 수정테스트입니다.수정, rwriter=test, rregDate=2021-06-23 09:55:04.0, rmodDate=2021-06-23 10:04:52.0, prno=16, lvl=0)

```

## (8) 특정 게시물에 대한 특정 댓글 삭제 테스트

- ☞ 댓글이 있는 게시물의 bno: **458753**
- ☞ 댓글/답글 수정 테스트에서 수정된 댓글의 번호를 이용합니다(문서에서는 **17L**).

→ 테스트 메서드 추가

```
//게시물에 대한 특정 댓글 삭제
@Test
public void testRemoveReply() {
    log.info("서비스: 특정 게시물 삭제 테스트: " + myReplyService.removeReply(458753L, 17L));
}
```

### [테스트 후, 콘솔 표시 로그 일부]

```

INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-삭제시 전달된 bno: 458753
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-삭제시 전달된 rno: 17
INFO : jdbc.sqlonly - DELETE FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 17

INFO : jdbc.sqltiming - DELETE FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 17
{executed in 17 msec}
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-삭제된 댓글 개수: 1
INFO : com.spring5213.mypro00.service.MyReplyServiceTests - 서비스: 특정 게시물 삭제 테스트: 1

```

## 10-7. 댓글 처리를 위한 제어 계층(컨트롤러) 구현

- ☞ REST 방식(Spring에서 제공하는 REST API)으로 처리할 때 주의해야 하는 점은, 브라우저나 외부에서 서버에 전달 및 요청하는 데이터의 포맷과 서버에서 보내주는 데이터의 포맷을 명확히 설계해야 합니다. 예를 들어, 댓글 등록의 경우, 브라우저에서는 JSON 타입으로 된 댓글 데이터를 전송하고, 서버에서는 댓글의 처리 결과가 정상적으로 되었는지 문자열로 결과를 알려 주도록 합니다.
- ☞ 다음은 아래에서 작성되는 각 메서드를 요약한 내용입니다.
- registerReplyForBoard()와 registerReplyForReply()는 다음의 내용으로 구성됩니다.
    - @PostMapping에 의해 POST 방식으로만 동작됩니다.
    - consumes 속성("application/json")에 의해 서버는 사용자로부터 JSON 형식의 데이터만 받아서 사용하도록 지정됩니다.
    - produces 속성({MediaType.TEXT\_PLAIN\_VALUE})에 설정된 텍스트 형식의 데이터(ResponseEntity<String>)를 생성해서 사용자에게 전송하게 됩니다.
    - @RequestBody ReplyVO replyVO를 파라미터로 사용하여, JSON 데이터를 ReplyVO 타입으로 변환하도록 지정됩니다.
    - ReplyServiceImpl의 postNewReplyArticle()를 호출하여 댓글을 등록시킨 후 반환된 숫자를 확인하여 브라우저에 '200 OK' 혹은 '500 Internal Server Error'를 반환합니다.
  - showReplyList()는 다음의 내용으로 구성됩니다.
    - @GetMapping에 의해 GET 방식으로만 동작됩니다.
    - produces 속성 ({"application/json"})에 설정된 JSON형식의 데이터(ResponseEntity<List<ReplyVO>>)를 생성해서 사용자에게 전송하게 됩니다.
    - 파라미터로 @PathVariable()을 이용하여 URL에 사용된 {이름}을 매개변수로 가져옵니다.
    - MyReplyPagingDTO(page, 10) 객체를 생성하여, ReplyServiceImpl의 getReplyListByBnoWithPaging()에게 bno 값과 같이 파라미터로 전달됩니다.
    - ReplyServiceImpl의 getReplyListByBnoWithPaging()로부터 반환된 특정 게시물의 댓글목록(List<ReplyVO>)과 '200 OK' 상태를 사용자에게 반환합니다.
  - 그 외의 댓글의 수정/삭제/조회 처리 메서드는 위와 유사한 방식으로 JSON 형식이나 문자열을 반환하도록 설계합니다.
    - 댓글 수정은 'PUT' 방식이나 'PATCH' 방식으로 처리되며, 실제 수정되는 데이터는 JSON 포맷으로 서버에 전달되도록 @RequestBody를 이용해서 처리됩니다.
- ☞ 이 때, @RequestBody로 처리되는 데이터는 일반 파라미터나 @PathVariable 파라미터를 처리할 수 없기 때문에 직접 처리해주어야 합니다.

- ☞ 컨트롤러의 각 메서드에는 다음처럼 사용자의 URL 요청이 매핑되도록 구현합니다

동작(기능)	방식	사용자 요청URL
게시물에 대한 댓글 목록 조회	GET	/replies/pages/{bno}/{page}
게시물에 대한 댓글 등록(rno 반환)	POST	/replies/{bno}/new
게시물에 대한 댓글의 답글 등록(rno 반환)	POST	/replies/{bno}/{prno}/new
게시물에 대한 특정 댓글 조회	GET	/replies/{bno}/{rno}
게시물에 대한 특정 댓글 수정	POST 또는 PATCH	/replies/{bno}/{rno}
게시물에 대한 특정 댓글 삭제	DELETE	/replies/{bno}/{rno}

→ src/main/java/com.spring5213.mypro00.controller 패키지에 MyReplyController 클래스를 생성하고 다음의 코드를 작성

```
package com.spring5213.mypro00.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.spring5213.mypro00.common.paging.MyReplyPagingCreatorDTO;
import com.spring5213.mypro00.common.paging.MyReplyPagingDTO;
import com.spring5213.mypro00.domain.MyReplyVO;
import com.spring5213.mypro00.service.MyReplyService;

import lombok.extern.log4j.Log4j;

@RequestMapping(value= {"replies"})
@RestController //컨트롤러 클래스 내에 메소드들은 JSP 파일을 호출할 수 없습니다.(이유: 데이터만 전달해 주므로)
@Log4j
//@AllArgsConstructor
public class MyReplyController {

    //생성자 자동 주입
    private MyReplyService myReplyService;

    public MyReplyController(MyReplyService myReplyService) {
        this.myReplyService = myReplyService;
    }

    /*
    //매핑 URL 목록
    //게시물에 대한 댓글 목록 조회:          GET      /replies/pages/{bno}/{page}
    //게시물에 대한 댓글 등록(rno 반환):    POST     /replies/{bno}/new
    //게시물에 대한 댓글의 답글 등록(rno 반환): POST     /replies/{bno}/{prno}/new
    //게시물에 대한 특정 댓글 조회:          GET      /replies/{bno}/{rno}
    //게시물에 대한 특정 댓글 수정:          POST:PATCH /replies/{bno}/{rno}
    //게시물에 대한 특정 댓글 삭제:          DELETE   /replies/{bno}/{rno}
    */

    //게시물에 대한 댓글 목록 조회(페이지 - 전체댓글 수 고려)
    @GetMapping(value = "/pages/{bno}/{page}",
                //메소드가 웹브라우저로 전달할 데이터의 MIME-유형을 지정
                produces = { "application/json; charset=UTF-8", "application/xml; charset=UTF-8" })
    public ResponseEntity<MyReplyPagingCreatorDTO> showReplyList(@PathVariable("bno") Long bno,
                                                                @PathVariable("page") Integer pageNum) {
        log.info("댓글-컨트롤러 - 댓글목록 표시-URI 추출 pageNum: " + pageNum);
        log.info("댓글-컨트롤러 - 댓글목록 표시-URI 추출 bno: " + bno);

        MyReplyPagingDTO myReplyPaging = new MyReplyPagingDTO(bno, pageNum);
        log.info("댓글-컨트롤러 - 댓글목록 표시 - 생성된 MyReplyPagingDTO: " + myReplyPaging);

        MyReplyPagingCreatorDTO replyPagingCreator = myReplyService.getReplyListByBno(myReplyPaging);

        ResponseEntity<MyReplyPagingCreatorDTO> responseEntity =
            new ResponseEntity<>(replyPagingCreator, HttpStatus.OK);

        log.info("댓글-컨트롤러 - 댓글목록 표시 - 브라우저로 전달되는 ResponseEntity<>: " + responseEntity);

        return responseEntity;
    }
    //테스트: 톰캣 기동 후
    //브라우저에서 http://localhost:8080/replies/pages/4849721/1      --> XML 형식으로 표시됨
}
```

```
//브라우저에서 http://localhost:8080/replies/pages/4849721/1.json <--JSON 형식으로 표시됨
```

```
//게시물에 대한 댓글 등록: rno 반환
@PostMapping(value = "/{bno}/new",
            consumes = {"application/json; charset=UTF-8"}, //웹브라우저로부터 메소드가 전달받는 MIME 유형을 지정
            produces = { "text/plain; charset=UTF-8" }) //{@ MediaType.TEXT_PLAIN_VALUE }
public ResponseEntity<String> registerReplyForBoard(@PathVariable("bno") Long bno,
                                                       @RequestBody MyReplyVO myReply) {
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-URI 추출 bno: " + bno);
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-myReply.getBno: " + myReply.getBno());
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-서비스로 전달되는 MyReplyVO: " + myReply);

    Long registerdRno = myReplyService.registerReplyForBoard(myReply);
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-반환된 rno(registerdRno): " + registerdRno);
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-myReply.getRno: " + myReply.getRno());

    //삼항연산자로 반환값 결정
    return registerdRno != null ? new ResponseEntity<>("게시물의 댓글 등록 성공", HttpStatus.OK)
        : new ResponseEntity<>("게시물의 댓글 등록 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}
```

```
//게시물에 대한 댓글의 답글 등록: rno 반환
@PostMapping(value = "/{bno}/{prno}/new",
            consumes = {"application/json; charset=UTF-8"},
            produces = { "text/plain; charset=UTF-8" })
public ResponseEntity<String> registerReplyForReply(@PathVariable("bno") Long bno,
                                                       @PathVariable("prno") Long prno,
                                                       @RequestBody MyReplyVO myReply) {
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-URI 추출 bno: " + bno);
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-myReply.getBno: " + myReply.getBno());
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-URI 추출 prno: " + prno);
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-myReply.getPrno: " + myReply.getPrno());
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-전달된 MyReplyVO: " + myReply);

    Long registerdRno = myReplyService.registerReplyForReply(myReply);
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-반환된 rno(registerdRno): " + registerdRno);
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-myReply.getRno: " + myReply.getRno());

    return registerdRno != null
        ? new ResponseEntity<>("댓글에 대한 답글 등록 성공", HttpStatus.OK)
        : new ResponseEntity<>("댓글에 대한 답글 등록 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}
```

```
//게시물에 대한 특정 댓글/답글 조회
@GetMapping( value = "/{bno}/{rno}",
            produces = { "application/json; charset=UTF-8", "application/xml; charset=UTF-8" })
public ResponseEntity<MyReplyVO> showReply(@PathVariable("bno") Long bno,
                                             @PathVariable("rno") Long rno) {
    log.info("댓글-컨트롤러 - 댓글 조회- URI 추출 bno: " + bno);
    log.info("댓글-컨트롤러 - 댓글 조회- URI 추출 rno: " + rno);

    MyReplyVO myReply = myReplyService.getReply(bno, rno);
    log.info("댓글-컨트롤러 댓글 조회 - 브라우저로 전달되는 MyReplyVO: " + myReply);

    return new ResponseEntity<>(myReply, HttpStatus.OK);
}
```

```
//HTTP PUT : 리소스의 전체를 업데이트(누락된 값 --> null로 처리)
// PATCH : 리소스의 일부를 업데이트
//데이터베이스에서 누락된 값을 DEFAULT 등으로 처리해서 항상 전체를 업데이트 시키므로, PUT/PATCH 차이가 없음.
```

```
//게시물에 대한 특정 댓글/답글 수정
@RequestMapping(value = "/{bno}/{rno}",
                method = { RequestMethod.PUT, RequestMethod.PATCH },
                consumes = "application/json; charset=UTF-8",
                produces = "text/plain; charset=UTF-8" )
```

```

public ResponseEntity<String> modifyReply(@PathVariable("bno") Long bno,
                                         @PathVariable("rno") Long rno,
                                         @RequestBody MyReplyVO myReply) {
    log.info("댓글-컨트롤러 - 댓글 조회-URI 추출 bno: " + bno);
    log.info("댓글-컨트롤러 - 댓글 조회-URI 추출 rno: " + rno);
    log.info("댓글-컨트롤러 - 게시물에 대한 댓글 수정-전달된 MyReplyVO: " + myReply);

    myReply.setBno(bno);
    myReply.setRno(rno);

    int modCnt = myReplyService.modifyReply(myReply);
    log.info("댓글-컨트롤러 댓글 수정 - 수정된 댓글 수: " + modCnt);

    return modCnt == 1
        ? new ResponseEntity<>("댓글 수정 성공", HttpStatus.OK)
        : new ResponseEntity<>("댓글 수정 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}

//게시물에 대한 특정 댓글/답글 삭제
@DeleteMapping(value = "/{bno}/{rno}", produces = { "text/plain; charset=UTF-8" })
public ResponseEntity<String> removeReply(@PathVariable("bno") Long bno,
                                         @PathVariable("rno") Long rno) {
    log.info("댓글-컨트롤러 - 댓글 삭제-URI 추출 bno: " + bno);
    log.info("댓글-컨트롤러 - 댓글 삭제-URI 추출 rno: " + rno);

    int delCnt = myReplyService.removeReply(bno, rno);
    log.info("댓글-컨트롤러 - 댓글 삭제-삭제된 댓글 수: " + delCnt);

    return delCnt == 1
        ? new ResponseEntity<>("댓글 삭제 성공", HttpStatus.OK)
        : new ResponseEntity<>("댓글 삭제 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}
}

```

## [MyReplyController 테스트]

☞ 테스트 전에 SQL\*Developer에서 댓글이 있는 게시물 번호를 확인해서 코드 수정 후 테스트를 수행합니다.(문서에서는, 458753 입니다)

### (1) MyReplyControllerTests 클래스 생성

→ src/test/java/com.spring5213.mypro00.controller 패키지에 MyReplyControllerTests 클래스 생성하고

다음의 내용 작성

```

package com.spring5213.mypro00.controller;

//녹색 import 문은 테스트 시에 추가되는 import 문입니다.
//아래의 2 import 문은 코드에서, 오류를 마우스 우클릭 --> 메시지에서 static 임포트를 직접 수행해야 추가됩니다.
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
//import org.slf4j.Logger;
//import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;

```

```

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;
import com.google.gson.Gson;
import com.spring5213.mypro00.domain.MyReplyVO;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

//테스트 메소드를 실행하는 어노테이션
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration //DispatcherServlet의 구성설정파일을 사용하여 컨트롤러를 테스트하기 위해 필요한 설정
@ContextConfiguration({ "file:src/main/webapp/WEB-INF/spring/mybatis-context.xml",
                      "file:src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml" })
@Log4j
public class MyReplyControllerTests {
    //Log4j 롬복 어노테이션을 사용하지 않는 경우, 다음처럼 로그 객체를 직접 호출해야 합니다.
    //private static final Logger logger = LoggerFactory.getLogger(MyReplyControllerTests.class);

    //테스트 환경 구성
    //컨트롤러 테스트를 위해서는 WebApplicationContext를 객체(웹 컨텍스트 객체)로 주입 받아야 합니다.
    @Setter(onMethod_ = { @Autowired })
    private WebApplicationContext ctx;

    //테스트 객체를 담을 필드
    private MockMvc mockMvc;

    //테스트 전에 테스트 객체를 생성하여 테스트 환경 설정
    @Before
    public void setup() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(ctx).build();
    } //테스트 환경 구성-끝

}

```

☞ 위의 클래스에 **테스트가 완료된 메서드는 주석처리 후, 새로운 메서드를 하나씩 추가**하면서 테스트를 수행합니다.

☞ 테스트 방법은, **변경내용 저장** 후, **마우스 우버튼 클릭 → Run As → 2 JUnit Test 클릭 → 테스트 수행**

→ 콘솔에 표시된 로그 확인 순으로 진행합니다. 문서에서는 표시된 로그 중 테스트 환경 구성 로그와 HikariDataSource 로그는 제외했습니다. 문서에는 테스트 메서드와 처리로그만 명시합니다.

(2) 게시물에 대한 댓글 목록 조회 테스트(URL: GET /replies/pages/{bno}/{page})

☞ 댓글이 있는 게시물의 bno: **458753**

→ 테스트 메서드 추가

```

//게시물에 대한 댓글 목록 조회 테스트: GET /replies/pages/{bno}/{page}
@Test
public void testShowReplyList() throws Exception {

    int resultStatus =
        mockMvc.perform(get("/replies/pages/458753/1")
                      //서버가 보낸 것이 JSON 일 때만 처리(컨트롤러의 produces에 대응)
                      .accept("application/json; charset=UTF-8")
                      //서버로 보내는 데이터에 대한 인코딩을 UTF-8로 지정
                      .characterEncoding("utf-8"))

```

```
.andDo(print())// 콘솔 출력  
.andReturn()  
.getResponse()  
.getStatus();  
log.info("웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): " + resultStatus );  
}
```

## [테스트 후, 콘솔 표시 로그 일부]

INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글목록 표시-URI 추출 pageNum: 1  
 INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글목록 표시-URI 추출 bno: 458753  
 INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글목록 표시 - 생성된 MyReplyPagingDTO: MyReplyPagingDTO(bno=458753, pageNum=1, rowAmountPerPage=10)  
 INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 목록 조회 - 전달된 MyReplyPagingDTO: MyReplyPagingDTO(bno=458753, pageNum=1, rowAmountPerPage=10)  
 INFO : jdbc.sqlonly - SELECT /\*+ INDEX (TBL\_MYREPLY IDX\_MYREPLY\_BNO\_RNO) \*/ count(\*) FROM book\_ex.tbl\_myreply WHERE bno = 458753

INFO : jdbc.sqltiming - SELECT /\*+ INDEX (TBL\_MYREPLY IDX\_MYREPLY\_BNO\_RNO) \*/ count(\*) FROM book\_ex.tbl\_myreply WHERE bno = 458753  
 {executed in 105 msec}  
 INFO : jdbc.resultsettable -  

-----	count(*)	-----
-----	14	-----
-----	-----	-----

INFO : jdbc.sqlonly - SELECT bno, rno, rcontent, rwriter, rregDate, rmodDate, prno, lvl FROM ( SELECT ROWNUM rn, b.\* FROM ( SELECT LEVEL lvl, a.\* FROM ( SELECT /\*+ INDEX\_ASC (a IDX\_MYREPLY\_BNO\_RNO) \*/ \* FROM book\_ex.tbl\_myreply WHERE bno = 458753 ) a START WITH prno IS NULL CONNECT BY PRIOR rno = prno ) b ) WHERE rn BETWEEN 1 \* 10 - (10 - 1) AND 1 \* 10

INFO : jdbc.sqltiming - SELECT bno, rno, rcontent, rwriter, rregDate, rmodDate, prno, lvl FROM ( SELECT ROWNUM rn, b.\* FROM ( SELECT LEVEL lvl, a.\* FROM ( SELECT /\*+ INDEX\_ASC (a IDX\_MYREPLY\_BNO\_RNO) \*/ \* FROM book\_ex.tbl\_myreply WHERE bno = 458753 ) a START WITH prno IS NULL CONNECT BY PRIOR rno = prno ) b ) WHERE rn BETWEEN 1 \* 10 - (10 - 1) AND 1 \* 10  
 {executed in 15 msec}  
 INFO : jdbc.resultsettable -  

----- ----- ----- ----- ----- ----- ----- -----	----- ----- ----- ----- ----- ----- ----- -----
bno  rno  rcontent  rwriter  rregdate  rmoddate  prno  lvl	----- ----- ----- ----- ----- ----- ----- -----
----- ----- ----- ----- ----- ----- ----- -----	----- ----- ----- ----- ----- ----- ----- -----
458753  1  458753 의 댓글 1  user1  2021-06-22 17:47:12.0  2021-06-22 17:47:12.0  [null]  1	----- ----- ----- ----- ----- ----- ----- -----
458753  6  458753 의 댓글 1 의 댓글 6  test02  2021-06-22 17:47:30.0  2021-06-22 17:47:30.0  1  2	----- ----- ----- ----- ----- ----- ----- -----
458753  9  458753 의 댓글 6 의 댓글 9  test04  2021-06-22 17:47:35.0  2021-06-22 17:47:36.0  6  3	----- ----- ----- ----- ----- ----- ----- -----
458753  10  458753 의 댓글 6 의 댓글 10  test02  2021-06-22 17:47:37.0  2021-06-22 17:47:38.0  6  3	----- ----- ----- ----- ----- ----- ----- -----
458753  7  458753 의 댓글 1 의 댓글 7  test03  2021-06-22 17:47:32.0  2021-06-22 17:47:33.0  1  2	----- ----- ----- ----- ----- ----- ----- -----
458753  8  458753 의 댓글 1 의 댓글 8  test04  2021-06-22 17:47:34.0  2021-06-22 17:47:34.0  1  2	----- ----- ----- ----- ----- ----- ----- -----
458753  14  매파 댓글의 답글 입력 테스트  test03  2021-06-22 20:35:06.0  2021-06-22 20:35:06.0  1  2	----- ----- ----- ----- ----- ----- ----- -----
458753  2  매파 - 수정 테스트..  user2  2021-06-22 17:47:13.0  2021-06-22 20:40:25.0  [null]  1	----- ----- ----- ----- ----- ----- ----- -----
458753  11  458753 의 댓글 11  test02  2021-06-22 17:47:39.0  2021-06-22 17:47:39.0  2  2	----- ----- ----- ----- ----- ----- ----- -----
458753  12  458753 의 댓글 11 의 댓글 12  test03  2021-06-22 17:47:41.0  2021-06-22 17:47:41.0  11  3	----- ----- ----- ----- ----- ----- ----- -----
----- ----- ----- ----- ----- ----- ----- -----	----- ----- ----- ----- ----- ----- ----- -----

댓글-전달된 페이지 기본데이터-MyReplyPagingDTO: MyReplyPagingDTO(bno=458753, pageNum=1, rowAmountPerPage=10)  
 댓글-끝 페이지번호: 2  
 댓글-시작 페이지번호: 1  
 댓글-이전버튼 표시 여부: false  
 댓글-다음버튼 표시 여부: false  
 전달된 댓글 목록 데이터: [MyReplyVO(bno=458753, rno=1, rcontent=458753 의 댓글 1, rwriter=user1, rregDate=2021-06-22 17:47:12.0, rmodDate=2021-06-22 17:47:12.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=6, rcontent=458753 의 댓글 1 의 댓글 6, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=9, rcontent=458753 의 댓글 6 의 댓글 9, rwriter=test04, rregDate=2021-06-22 17:47:35.0, rmodDate=2021-06-22 17:47:36.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=10, rcontent=458753 의 댓글 6 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:37.0, rmodDate=2021-06-22 17:47:38.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=7, rcontent=458753 의 댓글 1 의 댓글 7, rwriter=test03, rregDate=2021-06-22 20:35:06.0, rmodDate=2021-06-22 20:35:06.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=8, rcontent=458753 의 댓글 1 의 댓글 8, rwriter=test04, rregDate=2021-06-22 17:47:34.0, rmodDate=2021-06-22 17:47:34.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=14, rcontent=458753 의 댓글 6 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=2, rcontent=458753 의 댓글 11, rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=11, rcontent=458753 의 댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=2), MyReplyVO(bno=458753, rno=12, rcontent=458753 의 댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=3)]]  
 INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 목록 조회 - 생성된 myReplyPagingCreatorDTO:  
 MyReplyPagingCreatorDTO(myReplyPaging=MyReplyPagingDTO(bno=458753, pageNum=1, rowAmountPerPage=10), startPagingNum=1, endPagingNum=2, prev=false, next=false, replyTotalByBno=14, pagingNumCnt=10, lastPageNum=2, replyList=[MyReplyVO(bno=458753, rno=1, rcontent=458753 의 댓글 1, rwriter=user1, rregDate=2021-06-22 17:47:12.0, rmodDate=2021-06-22 17:47:12.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=6, rcontent=458753 의 댓글 1 의 댓글 6, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=9, rcontent=458753 의 댓글 1 의 댓글 9, rwriter=test04, rregDate=2021-06-22 17:47:35.0, rmodDate=2021-06-22 17:47:36.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=10, rcontent=458753 의 댓글 1 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:37.0, rmodDate=2021-06-22 17:47:38.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=7, rcontent=458753 의 댓글 1 의 댓글 7, rwriter=test03, rregDate=2021-06-22 20:35:06.0, rmodDate=2021-06-22 20:35:06.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=8, rcontent=458753 의 댓글 1 의 댓글 8, rwriter=test04, rregDate=2021-06-22 17:47:34.0, rmodDate=2021-06-22 17:47:34.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=14, rcontent=458753 의 댓글 6 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=2, rcontent=458753 의 댓글 11, rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=11, rcontent=458753 의 댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=2), MyReplyVO(bno=458753, rno=12, rcontent=458753 의 댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=3])]  
 INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 목록 조회 - myReplyPagingCreator 가 컨트롤러로 전달됨  
 INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글목록 표시 - 브라우저로 전달되는 ResponseEntity<>: <200 OK OK, MyReplyPagingCreatorDTO(myReplyPaging=MyReplyPagingDTO(bno=458753, pageNum=1, rowAmountPerPage=10), startPagingNum=1, endPagingNum=2, prev=false, next=false, replyTotalByBno=14, pagingNumCnt=10, lastPageNum=2, replyList=[MyReplyVO(bno=458753, rno=1, rcontent=458753 의 댓글 1, rwriter=user1, rregDate=2021-06-22 17:47:12.0, rmodDate=2021-06-22 17:47:12.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=6, rcontent=458753 의 댓글 1 의 댓글 6, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=9, rcontent=458753 의 댓글 1 의 댓글 9, rwriter=test04, rregDate=2021-06-22 17:47:35.0, rmodDate=2021-06-22 17:47:36.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=10, rcontent=458753 의 댓글 1 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:37.0, rmodDate=2021-06-22 17:47:38.0, prno=6, lvl=3), MyReplyVO(bno=458753, rno=7, rcontent=458753 의 댓글 1 의 댓글 7, rwriter=test03, rregDate=2021-06-22 20:35:06.0, rmodDate=2021-06-22 20:35:06.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=8, rcontent=458753 의 댓글 1 의 댓글 8, rwriter=test04, rregDate=2021-06-22 17:47:34.0, rmodDate=2021-06-22 17:47:34.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=14, rcontent=458753 의 댓글 6 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=2, rcontent=458753 의 댓글 11, rwriter=user2, rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=11, rcontent=458753 의 댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=2), MyReplyVO(bno=458753, rno=12, rcontent=458753 의 댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=3])]

```

댓글 1 의 댓글 6, rwriter=test02, rregDate=2021-06-22 17:47:30.0, rmodDate=2021-06-22 17:47:30.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=9,
rcontent=458753 의 댓글 6 의 댓글 9, rwriter=test04, rregDate=2021-06-22 17:47:35.0, rmodDate=2021-06-22 17:47:36.0, prno=6, lvl=3),
MyReplyVO(bno=458753, rno=10, rcontent=458753 의 댓글 6 의 댓글 10, rwriter=test02, rregDate=2021-06-22 17:47:37.0, rmodDate=2021-06-22 17:47:38.0,
prno=6, lvl=3), MyReplyVO(bno=458753, rno=7, rcontent=458753 의 댓글 1 의 댓글 7, rwriter=test03, rregDate=2021-06-22 17:47:32.0, rmodDate=2021-06-22
17:47:33.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=8, rcontent=458753 의 댓글 1 의 댓글 8, rwriter=test04, rregDate=2021-06-22 17:47:34.0,
rmodDate=2021-06-22 17:47:34.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=14, rcontent=매퍼 댓글의 답글 입력 테스트 , rwriter=test03, rregDate=2021-
06-22 20:35:06.0, rmodDate=2021-06-22 20:35:06.0, prno=1, lvl=2), MyReplyVO(bno=458753, rno=2, rcontent=매퍼 - 수정 테스트.., rwriter=user2,
rregDate=2021-06-22 17:47:13.0, rmodDate=2021-06-22 20:40:25.0, prno=0, lvl=1), MyReplyVO(bno=458753, rno=11, rcontent=458753 의 댓글 11,
rwriter=test02, rregDate=2021-06-22 17:47:39.0, rmodDate=2021-06-22 17:47:39.0, prno=2, lvl=2), MyReplyVO(bno=458753, rno=12, rcontent=458753 의
댓글 11 의 댓글 12, rwriter=test03, rregDate=2021-06-22 17:47:41.0, rmodDate=2021-06-22 17:47:41.0, prno=11, lvl=3)]),[]>

MockHttpServletRequest:
  HTTP Method = GET
  Request URI = /replies/pages/458753/1
  Parameters = {}
  Headers = [Accept:"application/json; charset=UTF-8"]
  Body = null
Session Attrs = {}

Handler:
  Type = com.spring5213.mypro00.controller.MyReplyController
  Method = com.spring5213.mypro00.controller.MyReplyController#showReplyList(Long, Integer)

Async:
  Async started = false
  Async result = null

Resolved Exception:
  Type = null

ModelAndView:
  View name = null
  View = null
  Model = null

FlashMap:
  Attributes = null

MockHttpServletResponse:
  Status = 200
  Error message = null
  Headers = [Content-Type:"application/json;charset=UTF-8"]
  Content type = application/json;charset=UTF-8
  Body =
>{"myReplyPaging":{"bno":458753,"pageNum":1,"rowAmountPerPage":10}, "startPagingNum":1,"endPagingNum":2,"prev":false,"next":false,"replyTotalByBno":14,"pagingNumCnt":10,"lastPageNum":2,"replyList":[{"bno":458753,"rno":1,"rcontent":"458753 의 댓글 1","rwriter":"user1","rregDate":1624351632000,"rmodDate":1624351632000,"prno":0,"lvl":1},{"bno":458753,"rno":6,"rcontent":"458753 의 댓글 6","rwriter":"test02","rregDate":1624351650000,"rmodDate":1624351650000,"prno":1,"lvl":2}, {"bno":458753,"rno":9,"rcontent":"458753 의 댓글 9","rwriter":"test04","rregDate":1624351655000,"rmodDate":1624351656000,"prno":6,"lvl":3}, {"bno":458753,"rno":10,"rcontent":"458753 의 댓글 10","rwriter":"test02","rregDate":1624351657000,"rmodDate":1624351658000,"prno":6,"lvl":3}, {"bno":458753,"rno":7,"rcontent":"458753 의 댓글 1의 댓글 7","rwriter":"test03","rregDate":1624351652000,"rmodDate":1624351653000,"prno":1,"lvl":2}, {"bno":458753,"rno":8,"rcontent":"458753 의 댓글 1의 댓글 8","rwriter":"test04","rregDate":1624351654000,"rmodDate":1624351654000,"prno":1,"lvl":2}, {"bno":458753,"rno":14,"rcontent":"매퍼 댓글의 답글 입력 테스트 ","rwriter":"test03","rregDate":1624361706000,"rmodDate":1624361706000,"prno":1,"lvl":2}, {"bno":458753,"rno":2,"rcontent":"매퍼 - 수정 테스트..","rwriter":"user2","rregDate":1624351633000,"rmodDate":1624362025000,"prno":0,"lvl":1}, {"bno":458753,"rno":11,"rcontent":"458753 의 댓글 11","rwriter":"test02","rregDate":1624351659000,"rmodDate":1624351659000,"prno":2,"lvl":2}, {"bno":458753,"rno":12,"rcontent":"458753 의 댓글 11 의 댓글 12","rwriter":"test03","rregDate":1624351661000,"rmodDate":1624351661000,"prno":11,"lvl":3}]}
  Forwarded URL = null
  Redirected URL = null
  Cookies = []
INFO : com.spring5213.mypro00.controller.MyReplyControllerTests - 웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): 200

```

### (3) 게시물에 대한 댓글 등록 테스트(URL: POST /replies/{bno}/new)

☞ 댓글이 있는 게시물의 bno: **458753**

→ 테스트 메서드 추가

```

//게시물에 대한 댓글 등록(rno 반환):POST /replies/{bno}/new (브라우저에서 JSON으로 보내는 것처럼 테스트 코드 작성)
@Test
public void testRegisterReplyForBoard() throws Exception {

    //MyReplyVO 객체 생성
    MyReplyVO myReply = new MyReplyVO();

    //myReplyVO에 입력값 지정
    myReply.setBno(458753L);
    myReply.setRcontent("컨트롤러-게시물에 대한 댓글 등록 테스트: JSON입력테스트");
    myReply.setRwriter("testJson");

    //myReply를 JSON 스트링으로 변환

```

```

String myReplyJsonStr = new Gson().toJson(myReply);

//JsonString 객체 콘솔 출력
log.info("컨트롤러-게시물에 대한 댓글 등록 테스트: JSON-String 객체(myReplyJsonStr): "+myReplyJsonStr);

//ResponseEntity가 반환하는 상태값을 표시
int resultStatus
    = mockMvc.perform(post("/replies/458753/new")
                    //서버가 보낸 것이 String 일 때만 처리(컨트롤러의 @*Mapping의 produces 속성에 대응)
                    .accept("text/plain; charset=UTF-8")
                    //서버에게 보낼 것이 JSON 임을 header에 명시(컨트롤러의 @*Mapping의 consumes 속성에 대응)
                    .contentType("application/json; charset=UTF-8")
                    //보내는 데이터에 대한 인코딩 설정입니다.
                    .characterEncoding("utf-8")
                    //테스트에 처리되는 데이터
                    .content(myReplyJsonStr))
                    .andDo(print())//print()를 직접 static으로 import, 콘솔에 출력
                    .andReturn()
                    .getResponse()
                    .getStatus();
log.info("웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): " + resultStatus);
}

```

#### [테스트 후, 콘솔 표시 로그 일부]

```

//게시물에 대한 댓글 등록(rno 반환):POST /replies/{bno}/new, JSON 형식 입력하여 테스트

INFO : com.spring5213.mypro00.controller.MyReplyControllerTests - 컨트롤러-게시물에 대한 댓글 등록 테스트: JSON-String
객체(myReplyJsonStr):{"bno":458753,"rno":0,"rcontent":"컨트롤러-게시물에 대한 댓글 등록 테스트: JSON 입력테스트","rwriter":"testJson","prno":0,"lvl":0}
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러-게시물에 대한 댓글 등록-URI 추출 bno: 458753
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러-게시물에 대한 댓글 등록-myReply.getBno: 458753
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러-게시물에 대한 댓글 등록-서비스로 전달되는 MyReplyVO: MyReplyVO(bno=458753,
rno=0, rcontent=컨트롤러-게시물에 대한 댓글 등록 테스트: JSON 입력테스트, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 등록 시 처음 전달된 myReplyVO: MyReplyVO(bno=458753, rno=0,
rcontent=컨트롤러-게시물에 대한 댓글 등록 테스트: JSON 입력테스트, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : jdbc.sqlonly - SELECT book_ex.seq_myreply.NEXTVAL FROM dual

INFO : jdbc.sqltiming - SELECT book_ex.seq_myreply.NEXTVAL FROM dual
{executed in 110 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|19   |
|-----|

INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 19, '컨트롤러-게시물에 대한 댓글 등록 테스트: JSON 입력테스트', 'testJson', DEFAULT, DEFAULT, DEFAULT)

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 19, '컨트롤러-게시물에 대한 댓글 등록 테스트: JSON 입력테스트', 'testJson', DEFAULT, DEFAULT, DEFAULT)
{executed in 10 msec}
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 게시물에 대한 댓글 등록: 등록된 myReply: MyReplyVO(bno=458753, rno=19,
rcontent=컨트롤러-게시물에 대한 댓글 등록 테스트: JSON 입력테스트, rwriter=testJson, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 게시물에 대한 댓글 등록: 등록된 rno(byGetter): 19
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 게시물에 대한 댓글 등록: 등록된 rno 가 컨트롤러로 전달됨
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러-게시물에 대한 댓글 등록-반환된 rno(registerdRno): 19
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러-게시물에 대한 댓글 등록-myReply.getRno: 19

MockHttpServletRequest:
    HTTP Method = POST
    Request URI = /replies/458753/new
    Parameters = {}
        Headers = [Content-Type:"application/json; charset=UTF-8", Accept:"text/plain; charset=UTF-8", Content-Length:"151"]
        Body = {"bno":458753,"rno":0,"rcontent":"컨트롤러-게시물에 대한 댓글 등록 테스트: JSON 입력테스트","rwriter":"testJson","prno":0,"lvl":0}
    Session Attrs = {}

Handler:
    Type = com.spring5213.mypro00.controller.MyReplyController
    Method = com.spring5213.mypro00.controller.MyReplyController#registerReplyForBoard(Long, MyReplyVO)

Async:
    Async started = false
    Async result = null

Resolved Exception:
    Type = null

ModelAndView:
    View name = null
    View = null
    Model = null

FlashMap:

```

```

Attributes = null

MockHttpServletResponse:
    Status = 200
    Error message = null
    Headers = [Content-Type:"text/plain;charset=UTF-8", Content-Length:"33"]
    Content type = text/plain;charset=UTF-8
    Body = 게시물의 댓글 등록 성공
    Forwarded URL = null
    Redirected URL = null
    Cookies = []
INFO : com.spring5213.mypro00.controller.MyReplyControllerTests - 웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): 200

```

(4) 게시물에 대한 댓글의 답글 등록 테스트(URL: POST /replies/{bno}/{prno}/new)

☞ 댓글이 있는 게시물의 bno: **458753**

답글이 등록되는 댓글의 rno는 앞의 테스트에서 등록한 댓글번호를 이용합니다(문서에서는 **19L**).

→ 테스트 메서드 추가

```

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post

//게시물에 대한 댓글의 답글 등록(rno 반환): POST /replies/{bno}/{prno}/new
@Test
public void testRegisterReplyForReply() throws Exception {

    //replyVO 객체 생성
    MyReplyVO myReply = new MyReplyVO();

    //replyVO에 입력값 지정
    myReply.setBno(458753L);
    myReply.setRcontent("컨트롤러-댓글에 대한 답글 등록 테스트");
    myReply.setRwriter("test00");
    myReply.setPrno(19L); //부모댓글의 rno를 받아와야 함

    //myReply를 JSON 스트링으로 변환
    String myReplyJsonStr = new Gson().toJson(myReply);

    //JsonString 객체 콘솔 출력
    log.info("컨트롤러-게시물의 댓글에 대한 답글 등록 테스트: JSON-String 객체(myReplyJsonStr): " + myReplyJsonStr);

    //ResponseEntity가 반환하는 상태값을 표시
    int resultStatus
        = mockMvc.perform(post("/replies/458753/19/new")
                //서버가 보낸 것이 String 일 때만 처리(컨트롤러의 @*Mapping의 produces 속성에 대응)
                .accept("text/plain; charset=UTF-8")
                //서버에게 보낼 것이 JSON임을 header에 명시(컨트롤러의 @*Mapping의 consumes 속성에 대응)
                .contentType("application/json; charset=UTF-8")
                //보내는 데이터에 대한 인코딩 설정입니다.
                .characterEncoding("utf-8")
                //테스트에 처리되는 데이터
                .content(myReplyJsonStr) )
                .andDo(print())//print()를 직접 static으로 import, 콘솔에 출력
                .andReturn()
                .getResponse()
                .getStatus());
    log.info("웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): " + resultStatus);
}

```

[테스트 후, 콘솔 표시 로그 일부]

```

//게시물에 대한 댓글의 답글 등록: (rno 반환): POST /replies/{bno}/{prno}/new
INFO : com.spring5213.mypro00.controller.MyReplyControllerTests - 컨트롤러-게시물의 댓글에 대한 답글 등록 테스트: JSON-String
객체(myReplyJsonStr): {"bno":458753, "rno":0, "rcontent":"컨트롤러-댓글에 대한 답글 등록 테스트", "rwriter":"test00", "prno":19, "lvl":0}
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글에 대한 답글 등록-URI 추출 bno: 458753
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글에 대한 답글 등록-myReply.getBno: 458753

```

```

INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글에 대한 답글 등록-URI 추출 prno: 19
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글에 대한 답글 등록-myReply.getRno: 19
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글에 대한 답글 등록-전달된 MyReplyVO: MyReplyVO(bno=458753, rno=0,
rcontent=컨트롤러-댓글에 대한 댓글 등록 테스트, rwriter=test00, rregDate=null, rmodDate=null, prno=19, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 등록 시 myReplyVO: MyReplyVO(bno=458753, rno=0,
rcontent=컨트롤러-댓글에 대한 댓글 등록 테스트, rwriter=test00, rregDate=null, rmodDate=null, prno=19, lvl=0)
INFO : jdbc.sqlonly - SELECT book_ex.seq_myreply.NEXTVAL FROM dual

INFO : jdbc.sqltiming - SELECT book_ex.seq_myreply.NEXTVAL FROM dual
{executed in 90 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|20    |
|-----|

INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 20, '컨트롤러-댓글에 대한 댓글 등록 테스트', 'test00', DEFAULT, DEFAULT, 19)

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 20, '컨트롤러-댓글에 대한 댓글 등록 테스트', 'test00', DEFAULT, DEFAULT, 19)
{executed in 10 msec}
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 댓글에 대한 댓글 등록: 등록된 myReply: MyReplyVO(bno=458753, rno=20,
rcontent=컨트롤러-댓글에 대한 댓글 등록 테스트, rwriter=test00, rregDate=null, rmodDate=null, prno=19, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 댓글에 대한 댓글 등록: 등록된 rno(ByGetter): 20
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스 - 댓글에 대한 댓글 등록: 등록된 rno가 컨트롤러로 전달됨
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글에 대한 답글 등록-반환된 rno(registerdRno): 20
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글에 대한 답글 등록-myReply.getRno: 20

MockHttpServletRequest:
    HTTP Method = POST
    Request URI = /replies/458753/19/new
    Parameters = {}
        Headers = [Content-Type:"application/json; charset=UTF-8", Accept:"text/plain; charset=UTF-8", Content-Length:"126"]
        Body = {"bno":458753,"rno":0,"rcontent":"컨트롤러-댓글에 대한 댓글 등록 테스트","rwriter":"test00","prno":19,"lvl":0}
    Session Attrs = {}

Handler:
    Type = com.spring5213.mypro00.controller.MyReplyController
    Method = com.spring5213.mypro00.controller.MyReplyController#registerReplyForReply(Long, Long, MyReplyVO)

Async:
    Async started = false
    Async result = null

Resolved Exception:
    Type = null

ModelAndView:
    View name = null
    View = null
    Model = null

FlashMap:
    Attributes = null

MockHttpServletResponse:
    Status = 200
    Error message = null
    Headers = [Content-Type:"text/plain;charset=UTF-8", Content-Length:"37"]
    Content type = text/plain;charset=UTF-8
    Body = 댓글에 대한 답글 등록 성공
    Forwarded URL = null
    Redirected URL = null
    Cookies = []
INFO : com.spring5213.mypro00.controller.MyReplyControllerTests - 웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): 200

```

## (5) 게시물에 대한 특정 댓글/답글 조회 테스트(URL: GET /replies/{bno}/{rno})

☞ 댓글이 있는 게시물의 bno: **458753**

조회되는 댓글/답글의 rno는 앞의 테스트에서 등록한 댓글번호를 이용합니다(문서에서는 **19L**).

→ 테스트 메서드 추가

```

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get

//게시물에 대한 특정 댓글 조회 테스트: GET /replies/{bno}/{rno}
@Test
public void testShowReply() throws Exception{
    int resultStatus =
        mockMvc.perform(get("/replies/458753/19"))

```

```
//서버가 보낸 것이 JSON 일 때만 처리(컨트롤러의 produces에 대응)
    .accept("application/json; charset=UTF-8")
    //서버로 보내는 데이터에 대한 인코딩을 UTF-8로 지정
    .characterEncoding("utf-8"))
    .andDo(print())// 콘솔 출력
    .andReturn()
    .getResponse()
    .getStatus();
log.info("웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): " + resultStatus );
}
```

## [테스트 후, 콘솔 표시 로그 일부]

//게시물에 대한 특정 댓글 조회 테스트: GET /replies/{bno}/{rno}

```
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글 조회- URI추출 bno: 458753
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글 조회- URI추출 rno: 19
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회 시 전달된 bno: 458753
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회 시 전달된 rno: 19
INFO : jdbc.sqlonly - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 19

INFO : jdbc.sqltiming - SELECT * FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 19
{executed in 107 msec}
INFO : jdbc.resultsettable -
|-----+-----+-----+-----+-----+-----|
|bno |rno |rcontent |rwriter |rregdate |rmoddate |prno |
|-----+-----+-----+-----+-----+-----+
|458753 |19 |컨트롤러-게시물에 대한 댓글 등록 테스트: JSON입력테스트 |testJson |2021-06-23 17:05:56.0 |2021-06-23 17:05:57.0 |[null] |
|-----+-----+-----+-----+-----+-----+-----|
```

INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회: 맵퍼로부터 전달된 myReply: MyReplyVO(bno=458753, rno=19, rcontent=컨트롤러-게시물에 대한 댓글 등록 테스트: JSON입력테스트, rwriter=testJson, rregDate=2021-06-23 17:05:56.0, rmodDate=2021-06-23 17:05:57.0, prno=0, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-조회: myReply가 컨트롤러로 전달됨
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 댓글 조회 - 브라우저로 전달되는MyReplyVO: MyReplyVO(bno=458753, rno=19, rcontent=컨트롤러-게시물에 대한 댓글 등록 테스트: JSON입력테스트, rwriter=testJson, rregDate=2021-06-23 17:05:56.0, rmodDate=2021-06-23 17:05:57.0, prno=0, lvl=0)

MockHttpServletRequest:

```
HTTP Method = GET
Request URI = /replies/458753/19
Parameters = {}
Headers = [Accept:"application/json; charset=UTF-8"]
Body = null
Session Attrs = {}
```

Handler:

```
Type = com.spring5213.mypro00.controller.MyReplyController
Method = com.spring5213.mypro00.controller.MyReplyController#showReply(Long, Long)
```

Async:

```
Async started = false
Async result = null
```

Resolved Exception:

```
Type = null
```

ModelAndView:

```
View name = null
View = null
Model = null
```

FlashMap:

```
Attributes = null
```

MockHttpServletResponse:

```
Status = 200
Error message = null
Headers = [Content-Type:"application/json;charset=UTF-8"]
Content type = application/json;charset=UTF-8
Body = {"bno":458753,"rno":19,"rcontent":"컨트롤러-게시물에 대한 댓글 등록 테스트: JSON입력테스트","rwriter":"testJson","rregDate":1624435556000,"rmodDate":1624435557000,"prno":0,"lvl":0}
Forwarded URL = null
Redirected URL = null
Cookies = []
```

INFO : com.spring5213.mypro00.controller.MyReplyControllerTests - 웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): 200

(6) 게시물에 대한 특정 댓글/답글 수정 테스트(URL: `PUT /replies/{bno}/{rno}` 또는 `PATCH /replies/{bno}/{rno}`)

☞ 댓글이 있는 게시물의 `bno: 458753`

조회되는 댓글/답글의 `rno`는 앞의 테스트에서 등록한 댓글번호를 이용합니다(문서에서는 [19L](#)).

→ 테스트 메서드 추가

```
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.patch

//게시물에 대한 특정 댓글/답글 수정 테스트 : PUT /replies/{bno}/{rno}, PATCH /replies/{bno}/{rno}
@Test
public void testModifyReply() throws Exception{

    MyReplyVO myReply = new MyReplyVO();
    myReply.setRno(19L);
    myReply.setRcontent("댓글-컨트롤러-수정테스트 - 댓글 수정(put)");

    //myReply를 JSON 스트링으로 변환
    String myReplyJsonStr = new Gson().toJson(myReply);
    System.out.println("=====JSON-String 객체로 변환:" + myReplyJsonStr);

    int resultStatus =
        mockMvc.perform(put("/replies/458753/19")
            //mockMvc.perform(patch("/replies/458753/19")
            //서버가 보낸 것(컨트롤러의 @*Mapping의 produces 속성에 대응)
            .accept("text/plain; charset=UTF-8")
            //서버에게 보낼 것(컨트롤러의 @*Mapping의 consumes 속성에 대응)
            .contentType("application/json; charset=UTF-8")
            .content(myReplyJsonStr))
        .andDo(print())// 콘솔 출력
        .andReturn()
        .getResponse()
        .getStatus();
    log.info("웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): " + resultStatus);
}
```

[테스트 후, 콘솔 표시 로그 일부]

```
//게시물에 대한 특정 댓글 수정 테스트 : PUT /replies/{bno}/{rno}, PATCH /replies/{bno}/{rno}

=====JSON-String 객체로 변환:{"bno":0,"rno":19,"rcontent":"댓글-컨트롤러-수정테스트 - 댓글 수정(put)","prno":0,"lvl":0}
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글 조회-URI 주출 bno: 458753
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글 조회-URI 주출 rno: 19
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 게시물에 대한 댓글 수정-전달된 MyReplyVO: MyReplyVO(bno=0, rno=19, rcontent=댓글-컨트롤러-수정테스트 - 댓글 수정(put), rwriter=null, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-수정시 전달된 myReplyVO: MyReplyVO(bno=458753, rno=19, rcontent=댓글-컨트롤러-수정테스트 - 댓글 수정(put), rwriter=null, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myreply SET rcontent = '댓글-컨트롤러-수정테스트 - 댓글 수정(put)', rmodDate = DEFAULT WHERE bno= 458753 AND rno = 19

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myreply SET rcontent = '댓글-컨트롤러-수정테스트 - 댓글 수정(put)', rmodDate = DEFAULT WHERE bno= 458753 AND rno = 19
{executed in 16 msec}
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-수정된 댓글 개수: 1
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 댓글 수정 - 수정된 댓글 수: 1

MockHttpServletRequest:
    HTTP Method = PUT
    Request URI = /replies/458753/19
    Parameters = {}
        Headers = [Content-Type:"application/json; charset=UTF-8", Accept:"text/plain; charset=UTF-8", Content-Length:"105"]
        Body = {"bno":0,"rno":19,"rcontent":"댓글-컨트롤러-수정테스트 - 댓글 수정(put)","prno":0,"lvl":0}
    Session Attrs = {}

Handler:
    Type = com.spring5213.mypro00.controller.MyReplyController
    Method = com.spring5213.mypro00.controller.MyReplyController#modifyReply(Long, Long, MyReplyVO)

Async:
    Async started = false
    Async result = null

Resolved Exception:
    Type = null

ModelAndView:
```

```

View name = null
View = null
Model = null

FlashMap:
Attributes = null

MockHttpServletResponse:
Status = 200
Error message = null
Headers = [Content-Type:"text/plain;charset=UTF-8", Content-Length:"20"]
Content type = text/plain;charset=UTF-8
Body = 댓글 수정 성공
Forwarded URL = null
Redirected URL = null
Cookies = []
INFO : com.spring5213.mypro00.controller.MyReplyControllerTests - 웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): 200

```

## (7) 게시물에 대한 특정 댓글/답글 삭제 테스트(URL: DELETE: /replies/{bno}/{rno})

☞ 댓글이 있는 게시물의 bno: 458753

조회되는 댓글/답글의 rno는 앞의 테스트에서 등록한 댓글번호를 이용합니다(문서에서는 19L).

→ 테스트 메서드 추가

```

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete

//게시물에 대한 특정 댓글 삭제 테스트: DELETE: /replies/{bno}/{rno}
@Test
public void testRemoveReply() throws Exception{
    int resultStatus =
        mockMvc.perform(delete("/replies/458753/19").accept("text/plain; charset=UTF-8"))
            .andDo(print())
            .andReturn()
            .getResponse()
            .getStatus();
    log.info("웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): " + resultStatus );
}

```

## [테스트 후, 콘솔 표시 로그 일부]

```

//게시물에 대한 특정 댓글 삭제 테스트: DELETE: /replies/{bno}/{rno}

INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글 삭제-URI 추출 bno: 458753
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글 삭제-URI 추출 rno: 19
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-삭제시 전달된 bno: 458753
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-댓글-삭제시 전달된 rno: 19
INFO : jdbc.sqlonly - DELETE FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 19

INFO : jdbc.sqltiming - DELETE FROM book_ex.tbl_myreply WHERE bno= 458753 AND rno = 19
{executed in 18 msec}
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-삭제된 댓글 개수: 1
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러 - 댓글 삭제-삭제된 댓글 수: 1

MockHttpServletRequest:
HTTP Method = DELETE
Request URI = /replies/458753/19
Parameters = {}
Headers = [Accept:"text/plain; charset=UTF-8"]
Body = <no character encoding set>
Session Attrs = {}

Handler:
    Type = com.spring5213.mypro00.controller.MyReplyController
    Method = com.spring5213.mypro00.controller.MyReplyController#removeReply(Long, Long)

Async:
    Async started = false
    Async result = null

Resolved Exception:
    Type = null

ModelAndView:
    View name = null
    View = null
    Model = null

```

```

FlashMap:
    Attributes = null

MockHttpServletResponse:
    Status = 200
    Error message = null
    Headers = [Content-Type:"text/plain;charset=UTF-8", Content-Length:"20"]
    Content type = text/plain;charset=UTF-8
    Body = 댓글 삭제 성공
    Forwarded URL = null
    Redirected URL = null
    Cookies = []
INFO : com.spring5213.mypro00.controller.MyReplyControllerTests - 웹브라우저에 전달되는 ResponseEntity 객체의 처리 상태 코드(resultStatus): 200

```

☞ 다음은 테스트로 동작이 잘되는지 확인된 경우, log4j코드와 주석을 없애서 정리한 MyReplyController.java 코드입니다.

☞ 참고만 합니다. 실습에서는 사용하지 않습니다.

```

package com.spring5213.mypro00.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.spring5213.mypro00.common.paging.MyReplyPagingCreatorDTO;
import com.spring5213.mypro00.common.paging.MyReplyPagingDTO;
import com.spring5213.mypro00.domain.MyReplyVO;
import com.spring5213.mypro00.service.MyReplyService;

import lombok.extern.log4j.Log4j;

@RequestMapping(value= {"replies"})
@RestController
@Log4j
@AllArgsConstructor
public class MyReplyController {

    private MyReplyService myReplyService;

    //게시물에 대한 댓글 목록 조회(페이징-전체댓글 수 고려)
    @GetMapping(value = "/pages/{bno}/{page}",
            produces = { "application/json; charset=UTF-8", "application/xml; charset=UTF-8" })
    public ResponseEntity<MyReplyPagingCreatorDTO> showReplyList(@PathVariable("bno") Long bno,
                                                               @PathVariable("page") Integer pageNum) {

        MyReplyPagingDTO myReplyPaging = new MyReplyPagingDTO(bno, pageNum);

        MyReplyPagingCreatorDTO replyPagingCreator = myReplyService.getReplyListByBno(myReplyPaging);

        return new ResponseEntity<>(replyPagingCreator, HttpStatus.OK);
    }

    //게시물에 대한 댓글 등록: rno 반환
    @PostMapping(value = "/{bno}/new",
            consumes = {"application/json; charset=UTF-8"},
            produces = { "text/plain; charset=UTF-8" })
    public ResponseEntity<String> registerReplyForBoard(@PathVariable("bno") Long bno,
                                                       @RequestBody MyReplyVO myReply) {
        Long registerdRno = myReplyService.registerReplyForBoard(myReply);

        return registerdRno != null
                ? new ResponseEntity<>("게시물의 댓글 등록 성공", HttpStatus.OK)
                : new ResponseEntity<>("게시물의 댓글 등록 실패", HttpStatus.INTERNAL_SERVER_ERROR);
    }

    //게시물에 대한 댓글의 답글 등록: rno 반환
}

```

```

@PostMapping(value = "/{bno}/{prno}/new",
            consumes = {"application/json; charset=UTF-8"},
            produces = { "text/plain; charset=UTF-8" })
public ResponseEntity<String> registerReply(@PathVariable("bno") Long bno,
                                              @PathVariable("prno") Long prno,
                                              @RequestBody MyReplyVO myReply) {
    return myReplyService.registerReplyForReply(myReply) != null
        ? new ResponseEntity<>("댓글에 대한 답글 등록 성공", HttpStatus.OK)
        : new ResponseEntity<>("댓글에 대한 답글 등록 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}

//게시물에 대한 특정 댓글/답글 조회
@GetMapping( value = "/{bno}/{rno}",
              produces = { "application/json; charset=UTF-8", "application/xml; charset=UTF-8" })
public ResponseEntity<MyReplyVO> showReply(@PathVariable("bno") Long bno,
                                             @PathVariable("rno") Long rno) {
    return new ResponseEntity<>( myReplyService.getReply(bno, rno), HttpStatus.OK);
}

//게시물에 대한 특정 댓글/답글 수정
@RequestMapping(value = "/{bno}/{rno}",
                 method = { RequestMethod.PUT, RequestMethod.PATCH },
                 consumes = "application/json; charset=UTF-8",
                 produces = "text/plain; charset=UTF-8" )
public ResponseEntity<String> modifyReply(@PathVariable("bno") Long bno,
                                            @PathVariable("rno") Long rno,
                                            @RequestBody MyReplyVO myReply) {
    myReply.setBno(bno);
    myReply.setRno(rno);

    return myReplyService.modifyReply(myReply) == 1
        ? new ResponseEntity<>("댓글 수정 성공", HttpStatus.OK)
        : new ResponseEntity<>("댓글 수정 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}

//게시물에 대한 특정 댓글/답글 삭제
@DeleteMapping(value = "/{bno}/{rno}", produces = { "text/plain; charset=UTF-8" })
public ResponseEntity<String> removeReply(@PathVariable("bno") Long bno,
                                           @PathVariable("rno") Long rno) {
    return myReplyService.removeReply(bno, rno) == 1
        ? new ResponseEntity<>("댓글 삭제 성공", HttpStatus.OK)
        : new ResponseEntity<>("댓글 삭제 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}

```

## 10-8 웹 브라우저와 서버간 Ajax 를 이용한 데이터 이동 구현

- ☞ 클라이언트 웹 브라우저가 서버에 데이터 전송하고, 필요한 데이터를 전송받는 Ajax 코드는 별도의 JavaScript 파일(.js 파일)로 구성하여, JSP 페이지에서 HTML 요소에 대한 이벤트나 화면 동작을 처리하는 JavaScript/jQuery 코드와는 분리시켜 구현하는 것이 좋습니다.
- ☞ 조회, 수정, 삭제, 입력 같은 다양한 데이터 처리 방식에 따라 다수의 Ajax 함수들이 필요하고, 이들 Ajax 가 포함된 여러 함수들을 하나의 모듈로 구현하는 것이 좋습니다.
- ☞ JavaScript에서 가장 많이 사용하는 모듈 패턴은, 관련 있는 함수들을 하나의 묶음으로 구성하는 것을 의미하며, JavaScript의 클로저(Closer)가 대표적입니다.
- ☞ 문서의 실습에서도 여러가지 Ajax 함수들을 자바스크립트의 클로저 형태의 모듈로서 별도의 JavaScript 파일로 구현합니다.

### ◆ JavaScript 클로저(Closer) 개요

- ☞ 클라이언트 브라우저에서 동작하는 JavaScript 클로저(Closer)를 이해하려면 다음의 JavaScript 기본 지식이 필요합니다.
  - JavaScript 함수 생성 방법
  - JavaScript에서의 변수 중첩 범위와 중첩함수 거동
  - JavaScript 일급 객체와 일급함수(first-class object/first-class function)
  - **자유변수**
  - **자기 호출 함수(self-invoking function)** 또는 **즉시 호출 함수(immediately invoked function)**
- ☞ 자바스크립트 변수는 지역(local) 또는 전역(global)의 변수 유효 범위를 갖습니다.
  - 기본적으로 함수 내에서 정의한 변수는, 정의된 함수 내에서만 사용되는 지역 변수(Local Variable)입니다.  
**지역변수는 외부의 다른 함수에서는 사용될 수 없습니다.** 즉, 지역변수는 함수가 호출될 때 함수 안에서 생성되고, 함수가 종료될 때 지워집니다.
  - 함수 밖에서 정의된 변수는 전역 변수(Global Variable)입니다.  
전역 변수는, 브라우저의 윈도우(window) 객체에 속합니다. 따라서, 전역 변수는 페이지 내의 모든 스크립트에서 사용될 수 있습니다.  
**전역 변수는, 선언된 웹 페이지가 유지되는 동안에 유지됩니다.**
  - 같은 이름을 가지는 전역변수와 지역변수가 있을 때, 이 둘을 서로 다른 변수입니다.  
하나를 변경하더라도 다른 것에는 영향이 없습니다.
  - **함수 내에서 전역 변수와 같은 이름의 지역변수를 정의하면, 함수내에서는 지역변수만 사용할 수 있습니다.**
  - 함수 내에서 var 키워드 없이 변수를 선언하면, 이 변수는 항상 전역변수입니다(이런 방법의 전역변수 사용은 권장하지 않습니다).
- ☞ 중첩 함수(Nested Function)는 **함수내에서 또 함수를 정의하고 사용**하는 것을 말합니다.
  - 함수1의 내부에 함수2가 정의되고, 함수1과 함수2에 각각 변수1과 변수2가 정의된 경우,  
내부 함수(중첩함수의 안에 있는 함수)인 함수2의 기준으로, 변수2는 함수2의 지역변수가 되고, 함수1에 선언된 변수1은 함수1에는 지역변수이면서, 함수2의 입장에서는 변수2는 전역변수가 되므로 함수2의 입장에서는 두 변수를 모두 사용할 수 있습니다.  
반면에, 외부 함수인 함수1의 입장에서는 변수1이 지역변수가 되고, 변수2는 접근할 수 없습니다.
- ☞ 웹 브라우저에 표시된 HTML에 아래의 코드가 포함되어 있을 경우, 페이지가 로딩될 때, 아래의 코드가 실행되어, global\_name 변수가 선언되고 → printName() 함수가 생성되고, → printName() 함수가 실행됩니다.  
printName() 함수가 실행되면, outerVal 변수 선언 → showName() 함수 생성 → showName() 함수가 실행이 순차적으로 수행됩니다.

`showName()` 함수의 실행에 의해, `innerVar` 변수 선언 → "전역변수" 표시 → "**PRINTNAME의 지역변수**" 표시 → "**SHOWNAME의 지역변수**" 표시가 수행되고, `showName()` 함수는 종료됩니다. `showName()` 함수의 종료와 함께 `innerVar` 변수가 삭제됩니다. 이 작업이 끝나면 `printName()` 함수 실행이 끝나게 되고, `printName()` 함수 종료와 함께, `outerVar` 변수가 삭제됩니다.

`global_name` 변수는 페이지가 유지되는 동안 브라우저에 유지됩니다. 페이지가 바뀌면, `global_name` 변수가 삭제됩니다.

```
var global_name = "전역변수";
```

```
function printName() {  
  
    var outerVar = "PRINTNAME 의 지역변수";  
  
    function showName() {  
        var innerVar = "SHOWNAME 의 지역변수";  
        console.log(global_name);  
        console.log(outer_name);  
        console.log(inner_name);  
    }  
  
    showName();  
}  
  
printName(); // ← 함수 실행
```

☞ 다음의 특징을 가지는 것을 일급 객체 또는 일급 함수라고 합니다(위키피디아)

- 다른 함수나 메서드의 인자로 전달될 수 있어야 함
- 함수의 반환값으로 사용되어 값을 수정할 수 있어야 함
- 변수에 할당될 수 있어야 함

☞ JavaScript 함수는, 다른 함수의 파라미터로서 전달되거나 반환값으로 사용될 수 있고, 다른 함수의 값을 수정할 수 있으며, 변수에 할당될 수 있기 때문에 일급함수이며, JavaScript에서 함수는 객체이므로, JavaScript 함수는 일급객체입니다(아래 예제 참조).

```
var global_name = "전역변수";  
  
function makePrinter() {  
  
    var outer_name = "외부함수변수";  
  
    function printName() {  
        var inner_name = "내부함수변수";  
        console.log(global_name);  
        console.log(outer_name);  
        console.log(inner_name);  
    }  
  
    return printName; // ← 함수를 반환  
}  
  
var print = makePrinter(); // ← makePrinter() 함수 실행으로 반환된 printName() 함수가 print 변수에 저장됨  
// 이 때, print 변수에 저장된 것이 함수이므로 print도 함수입니다.  
print(); // ← print() 함수 실행
```

☞ 위에서 `var print = makePrinter();` 실행문으로, `makePrinter()`의 실행으로 반환된 `printName()` 함수가 클로저입니다.

☞ 클로저로써 가장 중요한 특징은, `outer_name`, `inner_name` 변수가 `makePrinter()` 함수가 종료되어 사라진 후에도, `print();` 함수의 실행 시에 표시(유지됨)됩니다.

즉, 클로저(`print` 변수에 저장된 `printName()` 함수)는, 자신을 포함하고 있는 외부 함수의 인자, 지역변수 등을 외부 함수가 종료된 후에도 사용할 수 있습니다.

이렇게 클로저(`printName()` 함수)에 의해 사용되는, 종료된 함수(`makePrinter()` 함수)의 변수를 **자유 변수(free variable)**라고

부릅니다. 클로저가 생성될 때, 클로저에 포함된 지역 변수들을 자유 변수로 만드는 것을 **캡쳐(capture)** 라고 합니다.

**이 자유변수(클로저 내의 변수)는 외부에서는 직접 접근할 수 없고, 항상 클로저를 통해서만 사용할 수 있습니다.**

객체 지향언어의 **private** 멤버 변수와 유사한 효과를 냅니다. 이처럼 클로저는 내부에 **자유변수를 가질 수 있습니다.**

☞ 다음은 변수에 익명함수가 반환되어 저장된 예제입니다. 즉, 아래 예제에서는 익명함수가 클로저입니다.

```
function makeGreeting(name) {  
    var greeting = "안녕! ";  
  
    return function() { //익명함수가 클로저  
        console.log(greeting + name); //greeting 이 자유변수.  
    };  
  
}  
  
var g1 = makeGreeting("홍길동");  
var g2 = makeGreeting("강감찬");  
  
g1();  
g2();  
  
결과)  
안녕! 홍길동  
안녕! 강감찬
```

makeGreeting(name) 함수에 name 매개변수를 지정하여 실행하면 반환된 익명 함수가 각각 g1, g2 변수에 할당됩니다.

이 때, **함수 내의 greeting 변수의 값과 매개변수의 값도 포함**되어 있습니다.

즉, 반환 된 익명함수를 통해, makeGreeting()의 인자와 내부의 지역변수가 모두 캡쳐 되어서 자유변수가 됨을 보여주고 있습니다.

인자로 주어지는 값에 따라 다른 기능을 하는 클로저가 필요하다면 이러한 형태로 작성을 하면 될 것입니다.

☞ 아래의 예제는 **익명함수를 반환하는 다른 예입니다.** ← 이것이 중요합니다.

하나의 페이지에서 클로저를 여러 개 생성할 필요 없이, **자기호출(self-invoking) 또는 즉시호출(immediately invoked) 함수 패턴**을 사용하여 하나의 클로저를 만들면 됩니다. **자기 호출 함수는 정의와 동시에 즉시 실행되고, 한 번만 실행됩니다. 또한 인자를 통해 함수내부로 값을 보낼 수도 있습니다.**

```
var print = ( function(name) { //← name 매개변수를 가지는 익명함수 1  
  
    var greeting = "익명함수 1 의-지역변수 ";  
  
    return function() { //← 익명함수 2, 익명함수 1 이 실행되면 익명함수 2 가 반환됨.  
        console.log(greeting +", " + name); //← 익명함수 2 는 익명함수 1 의 지역변수를 포함함.  
    };  
}  
  
( " 나는 name 매개변수입니다" ); //← name 에 할당될 초기값 지정 , //← () 가 자기 호출 함수임을 의미
```

print();

print() 실행 결과: 익명함수 1 의-지역변수, 나는 name 매개변수입니다 ← 초기값이 지정된 경우

```
var print = ( function(name) { //← name 매개변수를 가지는 익명함수 1  
  
    var greeting = "익명함수 1 의-지역변수 ";  
  
    return function() { //← 익명함수 2, 익명함수 1 이 실행되면 익명함수 2 가 반환됨.  
        console.log(greeting +", " + name); //← 익명함수 2 는 익명함수 1 의 지역변수를 포함함.  
    };  
}  
  
( ); //← name 에 초기값 지정 안함 ← () 가 자기 호출 함수임을 의미
```

print();

print() 실행 결과: 익명함수 1 의-지역변수 ← 초기값이 지정되지 않은 경우

☞ 아래에서 counter는 반환된 익명함수(클로저)가 할당된 변수입니다.

이 예제처럼 클로저를 만들면, 객체지향언어의 객체가 가지는 private 멤버 변수, private 메소드, public 메소드와 유사한 기능을 구현할 수 있습니다.

// 카운터 클로저를 생성합니다.

```
var counter =  
  
  ( function() {  
  
    var privateCounter = 0 ; // private 변수입니다.(외부 접근 불가)  
  
    function changeCounter(val) {// private 함수입니다.(외부 접근 불가)  
  
      privateCounter += val ;  
    }  
  
    return { // public 함수를 가지는 객체를 반환합니다.  
  
      inc: function() { // 증가 기능을 가지는 public 함수.  
        changeCounter(1);  
      },  
  
      dec: function() { // 감소 기능을 가지는 public 함수.  
        changeCounter(-1);  
      },  
  
      val: function() { // 현재값 조회 기능을 가지는 public 함수  
        return privateCounter;  
      }  
    }; // 반환되는 객체  
  } )(); //← counter 클로저는 즉시 실행함수임  
  
counter.inc(); ← privateCounter 가 1 이 됩니다.  
counter.inc(); ← privateCounter 가 2 이 됩니다.  
console.log("after increment : " + counter.val());  
counter.dec(); ← privateCounter 가 1 이 됩니다.  
console.log("after decrement : " + counter.val());
```

### (1) 게시물 상세 페이지 수정

☞ WEB-INF/views/myboard/detail.jsp 페이지를 다음처럼 수정: 한 페이지에 많은 내용이 표시되므로 기존 내용 정리 및 표시 위치 재조정

→ 게시물 표시 내용 정리(빨간색 코드 부분 추가 및 수정)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"  
pageEncoding="UTF-8"%>  
  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>  
  
<c:set var="contextPath" value="${pageContext.request.contextPath}" />  
  
<%@ include file="../../myinclude/myheader.jsp" %>  
  
<div id="page-wrapper">  
  <div class="row">
```

```

<div class="col-lg-12">
    <h3 class="page-header"
        style="white-space: nowrap;">Board - Detail: <c:out value="${board.bno}" />번 게시물</h3>
    </div>
<%-- 게시물 상세 표시 시작 --%>
<div class="row">
    <div class="col-lg-12">
        <div class="panel panel-default">
            <div class="panel-heading">
                <div class="row">
                    <div class="col-md-3" style="white-space: nowrap; height: 45px; padding-top:11px;">
                        <strong style="font-size:18px;">${board.bwriter}님 작성</strong>
                    </div>
                    <div class="col-md-3" style="white-space: nowrap; height: 45px; padding-top:16px;">
                        <span class="text-primary" style="font-size: smaller; height: 45px; padding-top: 19px;">
                            <span>
                                <span>등록일:&nbsp;</span>
                                <strong><fmt:formatDate pattern="yyyy-MM-dd HH:mm:ss" value="${board.bregDate}" /></strong>
                            <span>&nbsp;&nbsp;</span>
                            </span>
                            <span>조회수:&nbsp;<strong><c:out value="${board.bviewsCnt}" /></strong></span>
                        </span>
                    </div>
                    <div class="col-md-6" style="height: 45px; padding-top:6px;"><%-- vertical-align: middle; --%>
                        <div class="button-group pull-right">
                            <button type="button" id="btnToModify" data-oper="modify"
                                class="btn btn-primary"><span>수정</span></button>
                            <button type="button" id="btnToList" data-oper="list"
                                class="btn btn-info"><span>목록</span></button>
                        </div>
                    </div>
                </div>
            </div><%-- /.panel-heading --%>
</div><%-- /.panel-body --%>

<div class="panel-body form-horizontal">

    <div class="form-group">
        <label class="col-sm-2 control-label" style="white-space: nowrap;">글제목</label>
        <div class="col-sm-10">
            <input class="form-control" name="btitle" value='<c:out value="${board.btitle}" />'>
            <span>readonly="readonly"</span>
        </div>
    </div>

    <div class="form-group">
        <label class="col-sm-2 control-label" style="white-space: nowrap;">글내용</label>
        <%-- <textarea>와 </textarea>는 한 줄에 작성되어야 필요없는 공백이 포함되지 않음 --%>
        <div class="col-sm-10">
            <textarea class="form-control" rows="3" name="bcontent" style="resize: none;"><c:out value="${board.bcontent}" /></textarea>
        </div>
    </div>

    <div class="form-group">
        <label class="col-sm-2 control-label" style="white-space: nowrap;">최종수정일</label>
        <div class="col-sm-10">
            <input class="form-control" name="bmodDate"
                value='<fmt:formatDate pattern="yyyy/MM/dd HH:mm:ss" value="${board.bmodDate}" />'>
            <span>readonly="readonly"</span>
        </div>
    </div><%-- /.panel-body --%>
</div><%-- /.panel --%>
</div><%-- /.col-lg-12 --%>
</div><%-- /.row --%>

```

```

<form id="frmSendValue"><%-- 폼을 추가 --%>
    <input type='hidden' name='bno' id="bno" value=''>
    <input type='hidden' name='pageNum' value='${myBoardPagingDTO.pageNum}'>
    <input type='hidden' name='rowAmountPerPage' value='${myBoardPagingDTO.rowAmountPerPage}'>
    <input type='hidden' name='scope' value='${myBoardPagingDTO.scope}'>
    <input type='hidden' name='keyword' value='${myBoardPagingDTO.keyword}'>
</form>
</div><%-- /#page-wrapper --%>

<%-- 게시물 상세 표시 끝 --%>

<script>

var frmSendValue = $("#frmSendValue");

//게시물 수정 페이지로 이동: 폼의 값을 전송해서 이동하는 형태로 변경
$("#btnToModify").on("click", function(){
    frmSendValue.attr("action", "${contextPath}/myboard/modify");
    frmSendValue.attr("method", "get");
    frmSendValue.submit();
})

//게시물 목록 페이지로 이동: 폼의 값을 전송해서 이동하는 형태로 변경
$("#btnToList").on("click", function(){
    frmSendValue.find("#bno").remove(); // 목록화면 이동 시, bno 값 삭제
    frmSendValue.attr("action", "${contextPath}/myboard/list");
    frmSendValue.attr("method", "get");
    frmSendValue.submit();
})
</script>
<script>
var result = '<c:out value="${result}" />';

function checkModifyOperation(result) {
    if (result === '' || history.state) {
        return;
    } else if (result === 'successModify'){
        var myMsg = "글이 수정되었습니다";
    }

    alert(myMsg);
    myMsg= '';
}
checkModifyOperation(result);

</script>

<%@ include file="../myinclude/myfooter.jsp" %>

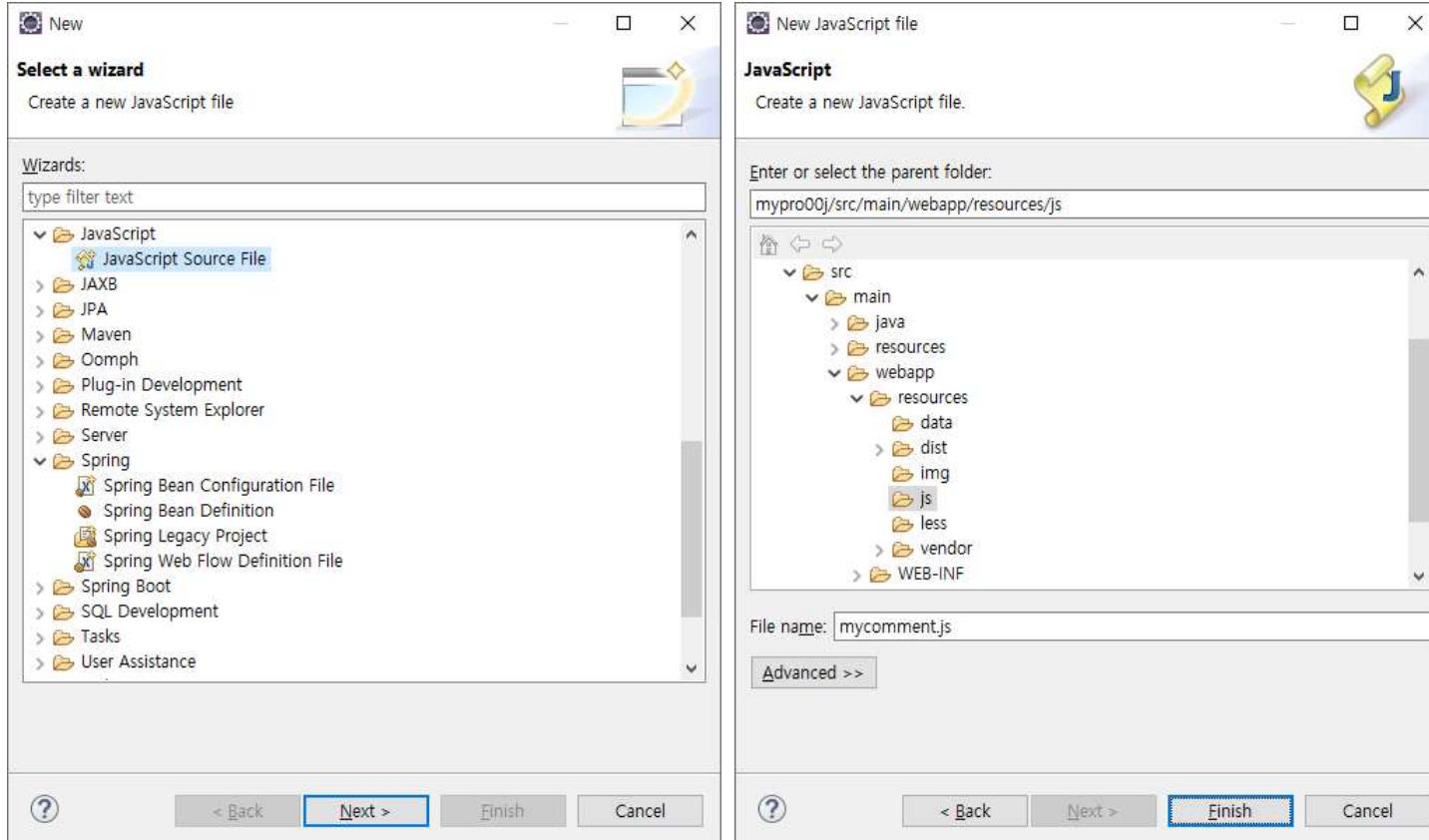
```

### (3) ajax-클로저 모듈이 정의될 JavaScript 파일 생성 및 동작 확인

→ src/main/webapp/resources/js 폴더에 mycomment.js 파일 생성

→ New → Other → JavaScript 폴더 → JavaScript Source File 항목 선택 → Next 클릭

→ 폴더 이름 확인 후, File name 입력란에 mycomment.js 입력, finish 클릭



→ 코드 작성 뷰에 표시된 mycomment.js 파일에 다음 코드 작성.

```
/*
 * mycomment.js: 댓글/답글 데이터 처리용 Ajax Closure Module */

alert("댓글처리 클로저 모듈 실행됨=====");
```

→ mycomment.js 파일은 게시물의 조회 페이지에서 처리되는 댓글/답글의 데이터 이동을 위해 사용되므로

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지의 기존 </script> 종료태그 밑에 다음의 코드 추가

```
</script>
<script type="text/javascript" src="${contextPath}/resources/js/mycomment.js"></script> ←추가
<script>

</script>
<%@ include file="../myinclude/myfooter.jsp" %>
```

[테스트]

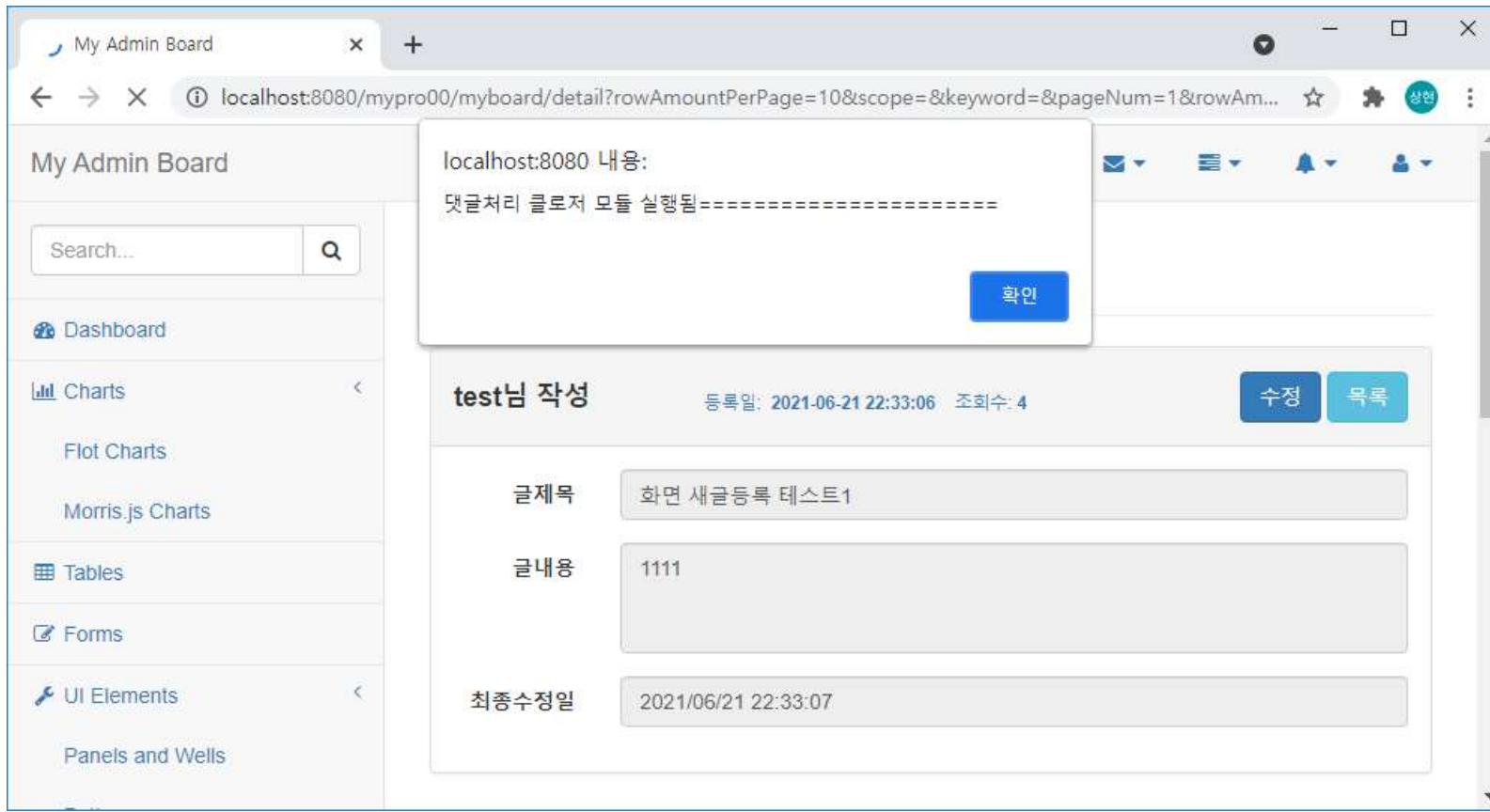
→ 톰캣 서버 기동(이 후의 실습에서 진행되는 테스트를 위해 톰캣 서버는 계속 작동 중인 상태를 유지합니다)

→ 웹브라우저에서 http://localhost:8080/mypro00/myboard/list' URL로 게시물 목록 페이지 접속

→ 표시된 목록에서 제일 처음에 표시된 게시물 클릭 → 게시물 상세 화면 이동

→ 경고창에 "댓글처리 클로저 모듈 실행됨===== 표시되고, 클릭한 게시물의 상세 화면이 표시됨 → 테스트 완료

→ 테스트 후, mycomment.js 파일의 alert() 실행문은 삭제 또는 주석처리 합니다.



(4) 댓글 목록 데이터 요청을 하는 ajax가 포함된 클로저 형태의 함수 추가

→ src/main/webapp/resources/js/mycomment.js 파일을 코드 작성 뷰에 오픈 후, 다음의 코드 작성

```
/**
 * myreply.js: 댓글/답글 데이터 처리용 Ajax Closure Module
 */

var myCommentCclsr = ( function() {

    //브라우저의 상세 페이지에서 Javascript에 의해 호출 시 전달받는 매개변수 설명
    //bnoAndPage: 서버로 전달할 값,
    //callback: 서버 처리 성공 시 브라우저에 의해 처리되는 함수
    //error: 서버 처리 실패 시 브라우저에 의해 처리되는 함수

    //댓글 목록(페이징) - ajax() 함수 사용
    function getCntList(pagingParams, callback, error) {

        var bno = pagingParams.bno;
        var page = pagingParams.page || 1;

        console.log("getCntList()가 전달받은 bno: " + bno);
        console.log("getCntList()가 전달받은 page: " + page);
        console.log("getCntList()함수의 ajax 처리 시작.....");

        //댓글 목록 조회 컨트롤러의 매팅 URL: GET /replies/pages/{bno}/{page}
        //$.ajax() 함수는, 자바스크립트 객체를 매개값으로 받아 처리
        $.ajax({
            type: 'get', //전송유형: get 방식
            url: "/mypro00/replies/pages/" + bno + "/" + page, //컨트롤러 메소드 호출 URL
            dataType: 'json', //서버로부터 받는 값들이 JSON 형식임(서버의 produces 속성에 대응)
            //json, xml, text, html, script 값 설정 가능
            //서버처리 성공 시 실행 함수, XHR (XML Http Request)
            //result 매개변수를 통해 서버로부터 전달받은 데이터가 callback 함수에 의해 처리됨
            success : function(replyPagingCreator, status, xhr){
                if(callback){ //callback 매개변수에 함수가 있으면 ,참
                    callback(replyPagingCreator);
                }
            },
            //서버처리 실패 시, 매개변수로 전달된 함수 실행, XHR (XML Http Request)
            error : function(xhr, status, er){

```

```

        if(error){ //error 매개변수에 함수가 있으면 ,참
            error(er);
        }
    }
};

//.ajax END
}//getCmtList END

return {
    getCmtList : getCmtList
};

})(); //즉시 실행 함수

```

☞ 위의 getCmtList() 함수는 \$.ajax() 대신 \$.getJSON() 함수를 이용하여 작성될 수 있습니다.

○ \$.getJSON() 함수는, 컨트롤러 메서드에 get 방식으로 요청하고, 결과를 JSON 객체를 받는 함수이며, 아래의 형식으로 사용됩니다.

[형식] \$.getJSON(url, [data,서버로 전달할 데이터], callback-function)

단, 서버에서 오류발생 시, 처리할 함수가 지정되지 못하므로, 오류 시 처리할 로직이 필요한 경우, fail() 메서드를 사용하여 아래처럼 별도로 지정해야 합니다.

○ fail() 메서드는 jQuery Ajax Chaining 메서드로서 ajax의 error 속성을 대체하여 사용됩니다. 문서에서는 실습하지 않습니다.

#### //댓글 목록(페이지) - getJSON() 함수 사용

```

function getCmtList(bnoAndPage, callback, error) {

    var bno = bnoAndPage.bno;
    var page = bnoAndPage.page || 1;

    //$.getJSON() 함수는, 컨트롤러 메서드에 get 방식으로 요청하고, 결과를 JSON 객체를 받음
    //#[형식] $.getJSON(url, [data,서버로 전달할 데이터], callback-function)
    //서버에서 오류발생 시, 처리할 함수가 지정되지 못하므로, fail() 함수로 별도로 지정해야 함

    $.getJSON( "/mypro00/replies/pages/" + bno + "/" + page ,
        function(replyPagingCreator) {
            if (callback) {
                callback(replyPagingCreator);
            }
        } //callback 처리함수 END
    ).fail(function(xhr, status, err) { //jQuery Ajax Chaining 메서드: fail 메서드 사용.ajax의 error를 대체)
        if (error) {
            error();
        }
    }
);
}//getCmtList END

```

[테스트: getCmtList(pagingParams, callback, error) 작동 테스트]

☞ mycomment.js의 모듈 함수는 detail.jsp 페이지에서 호출되어 사용되므로, 테스트 코드를 detail.jsp 파일에 작성합니다.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지를 코드 뷰에 오픈하고, 다음의 빨간색 코드를

기존 JavaScript 코드 밑에 작성

```

</script><%--게시물 상세 화면에서 사용되는 JavaScript 코드의 끝--%>
<script type="text/javascript" src="${contextPath}/resources/js/mycomment.js"></script>
<script>
<%--mycomment.js 모듈의 테스트 코드입니다. 아래 코드는 테스트 후에는 주석 처리합니다.////////////////////--%>

//게시물 번호 저장
var bnoValue = '<c:out value="${board.bno}" />';

```

```
//댓글 목록 데이터 요청/반기 Ajax 테스트
myCommentClsr.getCommentList()
    {bno:bnoValue, page:1}, //1번째 bnoAndPage의 인자값(JS객체)
    function(replyPagingCreator){ //2번째 callback 인자값: 성공 시 실행되는 함수(익명블록)
        for (var i = 0, len = replyPagingCreator.replyList.length || 0 ; i < len; i++){
            console.log(replyPagingCreator.replyList[i]);
        }
    }
);
</script>

<%@ include file="../myinclude/myfooter.jsp" %>
```

### [테스트]

- 다음의 순서로 테스트를 수행합니다.
- 웹 브라우저 실행, [F12] 키로 "개발자 도구" 표시 → Console 탭에 전달된 데이터가 표시되어야 함
- 웹 브라우저 개발자 도구 창에서 Network 탭 페이지 표시
- 게시물 목록 화면 이동 → 앞의 실습에서 댓글이 추가된 게시물 클릭(문서에서는 458753번 게시물)
- 게시물 상세 페이지 이동 시, 개발자 도구에 표시된 내용 중,
- Name 란의 1 파일 표시되어야 하며, 1을 클릭한 후, 표시된 내용에서 Preview 탭에 브라우저로 전달된 데이터가 표시되어야 함.

The screenshot shows the Network tab of the Chrome DevTools. A red box highlights the 'Network' tab at the top. Another red box highlights the 'Preview' tab under the Headers section for the selected request. The request details show a comment list for comment ID 1, with various fields like bno, pageNum, rowAmountPerPage, and replyList. The replyList contains multiple comment entries, each with fields like rcontent, rwriter, rregDate, and rmodDate. The 'Preview' tab displays the raw JSON data of the replyList.

☞ 테스트 코드의 `console.log()` 처리 결과는 개발자 도구의 Console 페이지에서 확인할 수 있습니다.

- ☞ <http://localhost:8080/mypro00/myboard/detail> URL 요청이 서버에 전달되어 처리 후, detail.jsp 페이지(JSP-서블릿)에 의해서 생성된 HTML 내용이 웹브라우저로 전달된 후, 웹 브라우저에서의 처리 과정은 다음과 같습니다.
- HTML 내용이 위에서부터 아래로 처리되어 웹브라우저에 표시되기 위한 처리가 진행됩니다.
- mycomment.js 라이브러리가 로드되어 구현된 코드가 실행됨
  - myCommentClsr.getCommentList() 함수가 메모리에 로드 완료
  - HTML 내의 테스트 코드가 처리됨

- myCommentClsr.getCommentList() 호출(이 때, 필요한 매개변수가 지정되어 호출됨)
- 메모리에 이미 실행 중인 myCommentClsr.getCommentList() 함수가 실행
- myCommentClsr.getCommentList() 함수 내의 ajax에 의해 서버에 데이터 요청
  - 서버에서 브라우저의 ajax로 댓글 목록 데이터 전달됨: 이 때 전달된 파일이 네트워크 탭에서 확인된 1 파일
  - myCommentClsr.getCommentList()의 2번째 매개변수인 callback 매개변수에 지정된 `function(replyPagingCreator){}` 함수에 댓글 목록 데이터 전달
  - `function(replyPagingCreator){}` 함수에 의한 처리가 수행
  - 개발자 도구의 Console 페이지에 표시됨.
- ☞ Network 페이지(Preview 탭)에서는 전달된 데이터를 확인할 수 있습니다.

#### (5) 댓글 등록/답글 등록을 위한 ajax가 포함된 클로저 형태의 함수 추가

→ src/main/webapp/resources/js/mycomment.js 파일에 다음의 코드를 기준 함수와 return문 사이에 추가합니다.

```
//댓글 등록
function registerCmt(comment, callback, error){
  var bno = comment.bno;
  console.log("registerCmt()에 전달된 bno: " + bno);
  console.log("registerCmt() 함수의 comment 등록 ajax 처리 시작.....");
  
  //댓글 등록 컨트롤러의 매팅 URL: GET /replies/{bno}/new
  //data: 브라우저가 서버에게 보내는 데이터
  //contentType: 서버에게 보내는 데이터의 MIME타입 설정, 컨트롤러의 consumes 속성에 대응
  $.ajax( { type: "post",
            url: "/mypro00/replies/" + bno + "/new",
            data: JSON.stringify(comment),
            contentType : "application/json; charset=utf-8",
            success : function(result, status, xhr){
              if(callback){
                callback(result);
              }
            },
            error : function(xhr, status, er){
              if(error){
                error(er);
              }
            }
          }
    );//.ajax END
};//registerCmt 함수 END

//답글 등록:
function registerReply(reply, callback, error){
  var bno = reply.bno;
  var prno = reply.prno;
  console.log("registerReply() 전달받은 bno: " + bno);
  console.log("registerReply() 전달받은 prno: " + prno);
  console.log("registerReply() 함수의 댓글에 대한 답글 등록 ajax 처리 시작.....");

  //답글 등록 컨트롤러의 매팅 URL: GET /replies/{bno}/{prno}/new
  //data: 브라우저가 서버에게 보내는 데이터
  //contentType: 서버에게 보내는 데이터의 MIME타입 설정, 컨트롤러의 consumes 속성에 대응
  $.ajax( { type: "post",
            url: "/mypro00/replies/" + bno + "/" + prno + "/new",
            data: JSON.stringify(reply),
            contentType : "application/json; charset=utf-8",
            dataType : 'text',
            success : function(result, status, xhr){
              if(callback){
                callback(result);
              }
            }
          }
    );//.ajax END
};//registerReply 함수 END
```

```

        }
    },
    error : function(xhr, status, er){
        if(error){
            error(er);
        }
    }
}
); // ajax END
}//registerReply 함수 END

return {
    getCmtList : getCmtList,           // 댓글 목록           ← 콤마(,) 추가
    registerCmt : registerCmt,         // 게시물의 댓글 등록 ← 추가
    registerReply : registerReply     // 댓글의 답글 등록 ← 추가
};

})(); // 즉시 실행 함수

```

☞ JSON.stringify()는, 객체를 JSON-String 객체로 변환하는 JavaScript 메서드입니다(웹 브라우저에 내장됨).

자세한 내용은 [https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global\\_Objects/JSON/stringify](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify) 페이지를 참고합니다.

[테스트: registerCmt(comment, callback, error), registerReply(reply, callback, error) 작동 테스트]

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지에 다음의 테스트 코드 작성

→ 기존 테스트 코드 주석처리!!! 후, 그 밑에 작성합니다

```

// 댓글등록 테스트
myCommentClsr.registerCmt(
    {bno:bnoValue, rcontent:"JS-클로저-댓글입력테스트입니다.", rwriter: "user7"} , // 1번째: comment 인자값
    function(result){ // 2번째: callback 인자값: 성공 시 실행되는 함수
        alert("myReplyClsr.registerCmt() 처리결과 " + result);
    }
);

// 답글등록 테스트
myCommentClsr.registerReply(
    {bno: bnoValue, prno: 1, rcontent: "JS-클로저-댓글의 답글입력테스트입니다.", rwriter: "user10"}, // 1번째: callback 인자값: 성공 시 실행되는 익명블록 형태의 콜백함수
    function(result){ // 2번째 callback 인자값: 성공 시 실행되는 익명블록 형태의 콜백함수
        alert("myReplyClsr.registerReply() 처리결과 " + result);
    }
);

```

[웹 브라우저 테스트]

→ 다음의 순서로 테스트를 수행합니다.

- 웹 브라우저 실행, [F12] 키로 "개발자 도구" 표시
- 웹 브라우저 개발자 도구 창에서 Network 탭 페이지 표시
- 게시물 목록 화면 이동 → 첫번째 게시물 클릭(문서에서는 458753번 게시물)
- 댓글 등록 알림창 표시, 확인 클릭
- 답글 등록 알림창 표시, 확인 클릭
- 게시물 상세 페이지 이동 시, 개발자 도구의 Network 페이지에 표시된 내용 중,
- Name란에 2개의 new 항목이 표시되어야 하며,
  - new를 각각 클릭, Preview 탭 내용 확인 → Headers 탭 페이지의 제일 밑에 서버로 전달한 데이터 확인
- SQL\*Developer에서 입력된 댓글과 답글 확인

DevTools - localhost:8080/mypro00/myboard/detail?rowAmountPerPage=10&scope=&keyword=&pageNum=1&rowAmo...

Elements Sources Network Console Performance Memory Application Security Lighthouse

Preserve log Disable cache No throttling

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other Has blocked cookies

Blocked Requests

10000 ms 20000 ms 30000 ms 40000 ms 50000 ms 60000 ms 70000 ms

Name Headers Preview Response Initiator Timing Cookies

**X-Requested-With: XMLHttpRequest**

Request Payload view source

```
{bno: "458753", rcontent: "JS-클로저-댓글입력테스트입니다.", rwriter: "user7"}
  bno: "458753"
  rcontent: "JS-클로저-댓글입력테스트입니다."
  rwriter: "user7"
```

Console

top Filter Default levels No Issues 2 hidden

7 messages
 5 user messages
 mycomm... 5
 No errors

registerCmt()에 전달된 bno: 458753
 registerCmt() 함수의 comment 등록 ajax 처리 시작.....
 registerReply() 전달받은 bno: 458753
 registerReply() 전달받은 prno: 1
 registerReply() 함수의 댓글에 대한 답글 등록 ajax 처리 시작.....

DevTools - localhost:8080/mypro00/myboard/detail?rowAmountPerPage=10&scope=&keyword=&pageNum=1&rowAmo...

Elements Sources Network Console Performance Memory Application Security Lighthouse

Preserve log Disable cache No throttling

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other Has blocked cookies

Blocked Requests

10000 ms 20000 ms 30000 ms 40000 ms 50000 ms 60000 ms 70000 ms

Name Headers Preview Response Initiator Timing Cookies

**X-Requested-With: XMLHttpRequest**

Request Payload view source

```
{bno: "458753", prno: 1, rcontent: "JS-클로저-댓글의 답글입력테스트입니다.", rwriter: "user10"}
  bno: "458753"
  prno: 1
  rcontent: "JS-클로저-댓글의 답글입력테스트입니다."
  rwriter: "user10"
```

Console

top Filter Default levels No Issues 2 hidden

7 messages
 5 user messages
 mycomm... 5
 No errors

registerCmt()에 전달된 bno: 458753
 registerCmt() 함수의 comment 등록 ajax 처리 시작.....
 registerReply() 전달받은 bno: 458753
 registerReply() 전달받은 prno: 1
 registerReply() 함수의 댓글에 대한 답글 등록 ajax 처리 시작.....

SQL 워크시트(W) 내역

CONN\_BOOK\_EX

워크시트 질의 작성기

```
1 SELECT * FROM book_ex.tbl_myreply
2 WHERE rcontent LIKE '%클로저%'
3 ORDER BY bno, rno;
```

스크립트 출력 × 질의 결과 ×

SQL | 인출된 모든 행: 2(0초)

BNO	RNO	RCONTENT	RWRITER	RREGDATE	RMODDATE	PRNO
1	458753	22 JS-클로저-댓글입력테스트입니다.	user7	21/06/24 21/06/24 09:40:10.000000000	(null)	
2	458753	23 JS-클로저-댓글의 답글입력테스트입니다.	user10	21/06/24 21/06/24 09:40:10.000000000	1	

→ 표시된 댓글/답글을 이 후에 삭제 테스트에서 사용합니다(rno 값을 잘 기억)

## (6) 댓글-답글 조회 및 댓글-답글 수정 처리를 위한 ajax가 포함된 클로저 형태의 함수 추가

→ src/main/webapp/resources/js/mycomment.js 파일에 다음의 코드를 기존 함수와 return문 사이에 추가합니다.

```
//jQuery Ajax의 get() 메소드 사용: url 주소로 GET 방식으로 요청, 객체로 받음
//get( url [, data] [, success(data, textStatus, jqXHR)] [, dataType] )
//댓글 조회
function getCommentReply(bnoAndRno, callback, error) {

    var bno = bnoAndRno.bno;
    var rno = bnoAndRno.rno;
    console.log("getCommentReply() 전달받은 bno: " + bno);
    console.log("getCommentReply() 전달받은 rno: " + rno);
    console.log("getCommentReply() 함수의 특정 댓글에 조회 ajax 처리 시작.....");

    //댓글 조회 컨트롤러의 매팅 URL: GET /replies/{bno}/{rno} ← .json 추가
    $.get( "/mypro00/replies/" + bno + "/" + rno + ".json", //json 형식으로 전송됨
        function(result) {
            if (callback) {
                callback(result);
            }
        }).fail(function(xhr, status, err) {
            if (error) {
                error();
            }
        });
    };//fail END
}//getCommentReply 함수 END

//댓글 수정: 수정된 특정 댓글을 서버로 전송
function modifyCommentReply(comment, callback, error) {

    var bno = comment.bno;
    var rno = comment.rno;
    console.log("modifyCommentReply() 전달받은 bno: " + bno + " 전달받은 rno: " + rno);
    console.log("modifyCommentReply() 함수의 특정 댓글 수정 ajax 처리 시작.....");

    //댓글 수정 컨트롤러의 매팅 URL: PUT 또는 PATCH /replies/{bno}/{rno}
    $.ajax( {type: "put",
        url : "/mypro00/replies/" + bno + "/" + rno,
        data: JSON.stringify(comment),
        contentType: "application/json; charset=utf-8",
        dataType : "text", // 서버로부터 전달받는 데이터 유형을 설정(xml, text, html, json 등)
        success: function(modifyResult, status, xhr) {
            if (callback) {
                callback(modifyResult);
            }
        },
        error: function(xhr, status, er) {
            if (error) {
                error(er);
            }
        }
    });
    };//ajax END
}//update END

return {
    getCommentList : getCommentList,           //댓글 목록
    registerComment : registerComment,         //댓글 등록
    registerReply : registerReply,             //답글 등록
    getCommentReply : getCommentReply,          //댓글-답글 조회
    modifyCommentReply : modifyCommentReply    //댓글-답글 수정
};

})(); //즉시 실행 함수
```

[테스트: `getCmtReply(bnoAndRno, callback, error)`, `modifyCmtReply(comment, callback, error)` 작동 테스트]

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지에 다음의 테스트 코드 작성

→ 기존 테스트 코드 주석처리!!! 후, 그 밑에 작성합니다

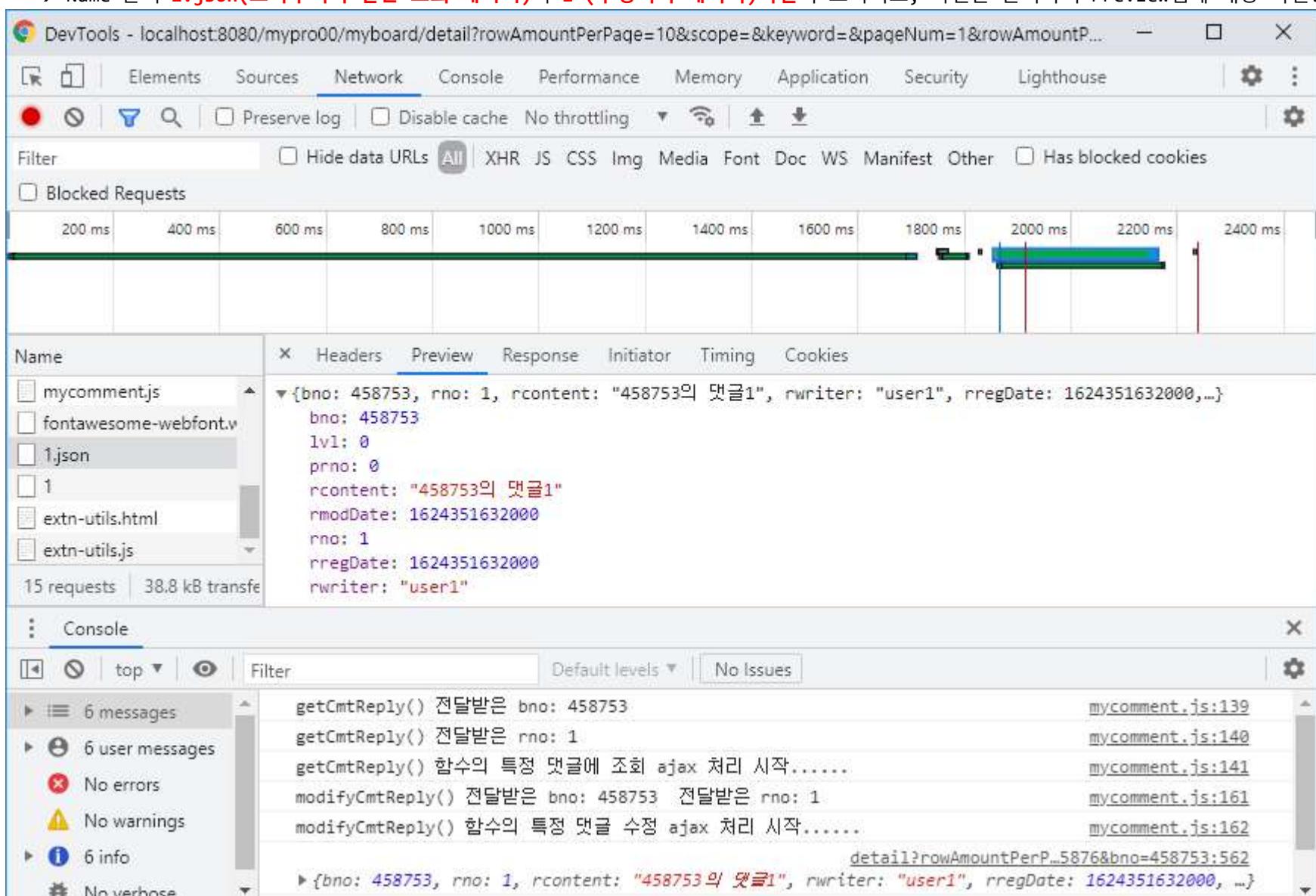
```
//댓글-답글 조회 테스트
myCommentClsr.getCmtReply(
    {bno: bnoValue, rno: 1 },
    function(data){
        console.log(data);
    }
);

//댓글-답글 수정 테스트
myCommentClsr.modifyCmtReply(
    {bno : bnoValue, rno : 1, rcontent: "JS클로저에 의한 댓글 수정====="},
    function(modifyResult) {
        alert(modifyResult + "- ajax 처리 완료");
    }
);
```

[웹 브라우저 테스트]

→ 다음의 순서로 테스트를 수행합니다.

- 웹 브라우저 실행, [F12] 키로 "개발자 도구" 표시
- 웹 브라우저 개발자 도구 창에서 Network 탭 페이지 표시
- 게시물 목록 화면 이동 → 목록의 첫번째 게시물 클릭(문서에서는 458753번 게시물)
- 게시물 상세 페이지 이동 시, 개발자 도구에 표시된 내용 중,
- Name 란의 1.json(브라우저가 받은 조회 데이터)과 1 (수정처리 메시지)파일이 표시되고, 파일을 클릭하여 Preview탭에 내용 확인.



## (7) 댓글-답글 삭제 처리를 위한 ajax가 포함된 클로저 형태의 함수 추가

→ src/main/webapp/resources/js/mycomment.js 파일에 다음의 코드를 기존 함수와 return문 사이에 추가합니다.

```
//댓글 삭제(실제 삭제)
function removeCmtReply(comment, callback, error) {

    var bno = comment.bno;
    var rno = comment.rno;
    var rwriter = comment.rwriter
    console.log("removeCmtReply() 전달받은 bno: " + bno);
    console.log("removeCmtReply() 전달받은 rno: " + rno);
    console.log("removeCmtReply() 전달받은 rwriter: " + rwriter);
    console.log("removeCmtReply() 함수의 댓글 삭제 ajax 처리 시작.....");

    //댓글 삭제 컨트롤러의 매팅 URL: DELETE /replies/{bno}/{rno}
    $.ajax( { type: "delete",
        url : "/mypro00/replies/" + bno + "/" + rno,
        data : JSON.stringify({bno: bno, rno: rno, rwriter: rwriter}),
        contentType : "application/json; charset=utf-8",
        success : function(removeResult, status, xhr) {
            if (callback) {
                callback(removeResult);
            }
        },
        error: function(xhr, status, er) {
            if (error) {
                error(er);
            }
        }
    }); //ajax END
} //removeCmtReply 함수 END

return {
    getCmtList : getCmtList,           //댓글 목록
    registerCmt : registerCmt,         //댓글 등록
    registerReply : registerReply,       //답글 등록
    getCmtReply : getCmtReply,          //댓글-답글 조회
    modifyCmtReply : modifyCmtReply,    //댓글-답글 수정
    removeCmtReply : removeCmtReply    //댓글-답글 삭제
};

})(); //즉시 실행 함수
```

[테스트: removeCmtReply(comment, callback, error) 작동 테스트]

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지에 다음의 테스트 코드 작성

→ 기존 테스트 코드 주석처리!!! 후, 그 밑에 작성합니다

```
//댓글-답글 삭제 테스트(실제 삭제 발생)
myCommentClsr.removeCmtReply(
    {bno : bnoValue, rno : 22, rwriter: "user10"}, //댓글-답글 입력 테스트 시, SQL*Developer에서 확인한 rno를 사용
    function(deleteResult) {
        console.log(deleteResult);
        if (deleteResult === "댓글/답글 삭제 성공") {
            alert(deleteResult + ": 댓글/답글이 삭제되었습니다(ByRemoveCmtReply).");
        }
    },
    function(err) {
        alert("오류로 인한 댓글/답글 삭제 작업 취소...");
    }
);
```

## [웹 브라우저 테스트]

→ 다음의 순서로 테스트를 수행합니다.

- 웹 브라우저 실행, [F12] 키로 "개발자 도구" 표시 → Console 탭에 전달된 데이터가 표시되어야 함
- 웹 브라우저 개발자 도구 창에서 Network 탭 페이지 표시
- 게시물 목록 화면 이동 → 목록의 첫번째 게시물 클릭(문서에서는 458753번 게시물)
- 게시물 상세 페이지 이동 시, 개발자 도구에 표시된 내용 중,
- Name 란의 22 파일 표시되어야 하며, 22을 클릭한 후, 표시된 내용에서 Header 탭 페이지 내용을 확인합니다.

The screenshot shows the Network tab in the DevTools. A request for '22' is selected. The Headers tab shows 'X-Requested-With: XMLHttpRequest'. The Request Payload tab shows the JSON data: {bno: "458753", rno: 22, rwriter: "user10"}.

The Console tab shows the following log entries:

- removeCmtReply() 전달받은 bno: 458753
- removeCmtReply() 전달받은 rno: 22
- removeCmtReply() 전달받은 rwriter: user10
- removeCmtReply() 함수의 댓글 삭제 ajax 처리 시작.....
- 댓글 삭제 성공

The log ends with a URL: detail?rowAmountPerP...5876&bno=458753:579

☞ mycomment.js 작성 및 테스트 후, detail.jsp 파일에 작성된 ajax-클로저 모듈 테스트를 위한 모든 실행문을 주석으로 처리합니다!

## 10-9 웹 브라우저에서의 댓글 표시를 위한 JSP 페이지 화면 구성

☞ 댓글은 게시물의 상세화면에서 표시되고 모든 처리가 이루어지므로, 게시물의 상세 정보를 표시하는 detail.jsp 파일에 화면에 관련된 모든 것을 구현합니다.

### (1) 댓글 입력 요소와 댓글 목록 표시 요소 추가

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지를 코드 작성 뷰에 오픈

→ 기존 게시물의 정보 표시 화면 요소 밑에, </div><!-- /#page-wrapper --> 요소 위에, 다음의 빨간색 코드 추가

```
...(생략)... ← 게시물 상세보기 표시 부분 코드
    <input type='hidden' name='keyword' value='${myBoardPagingDTO.keyword}'>
</form>
<%-- 게시물 상세 표시 끝 --%>

<%-- 댓글 화면 표시 시작, 붙여넣기 후, 아래의 style 태그 내용은 <div id="page-wrapper"> 태그 위에 옮겨놓을 것 --%>
<style>
.txtBoxCmt, .txtBoxComment {
    overflow: hidden; resize: vertical; min-height: 100px; color: black;
}
</style>

<div class="row">
    <div class="col-lg-12">
        <div class="panel panel-default" >
            <div class="panel-heading">
                <p style="margin-bottom: 0px; font-size: 16px;">
                    <strong style="padding-top: 2px;">
                        <span>댓글&ampnbsp<cc:out value="${board.breplyCnt}"/>개</span>
                        <span>&ampnbsp</span>
                        <button type="button" id="btnChgCmtReg" class="btn btn-info btn-sm">댓글 작성</button>
                        <button type="button" id="btnRegCmt" class="btn btn-warning btn-sm"
                                style="display:none">댓글 등록</button>
                        <button type="button" id="btnCancelRegCmt" class="btn btn-warning btn-sm"
                                style="display:none">취소</button>
                    </strong>
                </p>
            </div> <%-- /.panel-heading --%>

            <div class="panel-body">
<%-- 댓글 입력창 div 시작 --%>
                <div class="form-group" style="margin-bottom: 5px;">
                    <textarea class="form-control txtBoxCmt" name="rcontent"
                            placeholder="댓글작성을 원하시면,&#10; 댓글 작성 버튼을 클릭해주세요."
                            readonly="readonly"
                    ></textarea>
                </div>
                <hr style="margin-top: 10px; margin-bottom: 10px;">
<%-- 댓글 입력창 div 끝 --%>
                <ul class="chat">
                    <%-- 댓글 목록 표시 영역 - JavaScript로 내용이 생성되어 표시됩니다.--%>
                    </ul><%-- /.chat --%>
            </div><%-- /.panel-body --%>

            <div class="panel-footer" id="showCmtPagingNums">
                <%-- 댓글 목록의 페이지 번호 표시 영역 - JavaScript로 내용이 생성되어 표시됩니다.--%>
            </div>
        </div><%-- /.panel --%>
    </div><%-- .col-lg-12 --%>
</div><%-- .row : 댓글 화면 표시 끝 --%>
<%-- 댓글 페이지 데이터 저장 form --%>
<form id="frmCmtPagingValue">
    <input type='hidden' name='pageNum' value=' ' />
```

```

<input type='hidden' name='rowAmountPerPage' value='10' />
</form>

</div><%-- /#page-wrapper --%> ← page-wrapper 종료 div는 화면 표시 요소 제일 밑에 위치해야 함.

...(생략)...

```

- 댓글 목록의 화면 표시 확인을 위한 샘플 코드 추가
- 앞에서 작성한 코드에 있는 댓글 목록 표시 영역의 `<ul>`과 `</ul>` 태그 사이에 다음의 빨간색 코드(댓글 목록 표시 요소 샘플) 추가
- ☞ 댓글 목록 표시 요소 샘플 코드가 이 후에 댓글 표시 HTML 생성을 처리하는 jQuery 코드 작성 시, 템플릿처럼 사용됩니다.

```

<!-- 댓글 목록 표시 영역 시작 -->
<ul class="chat">
    <li class="left clearfix commentLi" data-bno="123456" data-rno="12">
        <div>
            <div>
                <span class="header info-rwriter">
                    <strong class="primary-font">user00</strong>
                    <span>&ampnbsp</span>
                    <small class="text-muted">2018-01-01 13:13</small>
                </span>
                <p>앞으로 사용할 댓글 표시 기본 템플릿입니다.</p>
            </div>
            <div class="btncComment" style="margin-bottom:10px">
                <button type="button" style="display:inline-block"
                        class="btn btn-primary btn-xs btnChgReg">답글 작성</button>
                <button type="button" style="display:none"
                        class="btn btn-warning btn-xs btnRegCmt">답글 등록</button>
            <hr class="txtBoxCmtHr" style="margin-top:10px; margin-bottom:10px">
            <textarea class="form-control txtBoxCmtMod" name="rcontent" style="margin-bottom:10px"
                      placeholder="답글작성을 원하시면,&#10;답글 작성 버튼을 클릭해주세요.">
            </textarea>
        </div>
    </div>
    </li>
</ul><%-- /.chat --%>

```

### [화면 확인]

- 톰캣 서버 기동 → 웹브라우저에서 게시물 목록 화면 접속 → 게시물 상세 화면 이동 → 화면구성 확인



## (2) 댓글 목록 표시 구현

- ☞ 현재 detail.jsp 에는 특정 게시물의 상세 정보만 담겨진 HTML이 생성되도록 구현되어 있습니다.  
소스코드에는 댓글목록이 표시되는 HTML 요소는 현재 ul 요소 외에는 없습니다.  
즉, 댓글 목록은 게시물 정보가 처음 표시될 때, detail.jsp 에 작성된 JavaScript와 jQuery 코드의 실행에 의해서  
댓글 목록이 표시될 HTML이 생성되어 ul 태그 사이에 표시되도록 프로그램적으로 구현됩니다.
- ☞ 앞에서 컨트롤러, 서비스, 매퍼, ajax 가 포함된 클로저 구현 시, 이미 댓글 목록에 대한 페이지 처리를 기준으로 구현했으므로,  
댓글/답글 목록 데이터는 페이지 데이터와 함께, replyPagingCreator 이름의 MyReplyPagingCreatorDTO 클래스 객체에 저장된 후,  
최종적으로는 JSON 객체로 변환되어 웹 브라우저에 전달됩니다. 따라서, jQuery에서 사용될 각 데이터들은 replyPagingCreator 객체의  
각 필드에 저장된 값을 가져오는 형태로 구현되어야 합니다.
- ☞ 다음의 요구사항이 반영되도록 댓글 목록 표시 코드를 구현합니다.
  - 게시물 상세 화면이 처음 브라우저에 표시 될 때, 게시물에 댓글이 있는 경우, 댓글 목록이 같이 표시되어야 함.  
댓글이 없는 경우, 댓글이 없다고 표시되어야 함.  
댓글이 많은 경우, 페이지로 처리되어 목록이 표시되어야 함
  - 각 댓글은 <ul> 요소의 하부요소인 <li> 요소에 하나의 댓글 처리되어 보여야 함.  
→ 이 때 각 댓글의 밑에는 **답글작성 버튼**이 표시되어야 함.
  - 댓글에 대하여 표시되는 답글은, 매퍼에서 오라클의 계층쿼리에 의해서 표시될 데이터가 전달되므로,  
레벨에 따라 앞에 빙칸이 추가되어, 답글 내용이 들여쓰는 효과로 표시되도록 구현
  - 이 후에 구현될 댓글/답글의 수정삭제를 위하여 필요한 bno 값과 rno 값을  
li 요소에 data-bno와 data-rno 속성으로 값을 미리 지정.  
  
☞ HTML5 data-xxx 속성에 대해서는 [https://developer.mozilla.org/ko/docs/Learn/HTML/Howto/데이터\\_속성\\_사용하기](https://developer.mozilla.org/ko/docs/Learn/HTML/Howto/데이터_속성_사용하기) 사이트를  
참고합니다.
  - 페이지 번호 표시 및 페이지 기능 구현
    - ajax로 전송된 날짜시간 값을 정수가 아닌 날짜시간 형태로 표시되도록 수정  
☞ 이 기능 관련 함수는 detail.jsp 파일에 구현할 수도 있지만,  
mycomment.js 파일의 클로저 함수를 사용해서 웹 브라우저로 전달된 날짜시간 값을 처리하는 것으로  
mycomment.js 파일의 클로저에 날짜 시간 표시 형식 함수를 구현하는 것이 좋습니다.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지를 코드 작성부에 오픈

→ 다음 코드가 작성된 위치로 이동 → bnoValue 변수 선언문 밑에 댓글 목록 표시 코드를 추가합니다.

```
<script type="text/javascript" src="${contextPath}/resources/js/mycomment.js"></script>
<script>
//게시물 번호 저장
var bnoValue = '<c:out value="${board.bno}" />'; ← 이 코드 밑에 댓글 목록 표시 코드 추가
```

→ 추가 코드- 1: 댓글 목록 표시 함수

```
var commentUL = $(".chat");
var frmCmtPagingValue = $("#frmCmtPagingValue");

//댓글 목록표시 함수 선언: ajax 클로저 모듈 함수를 호출함
function showCmtList(page){
    //댓글 목록 데이터를 요청하는 클로저 모듈 함수 호출
    myCommentClsr.getCmtList(
        {bno: bnoValue, page: page || 1 },
        function(replyPagingCreator) { //ajax에서 실행할 callback 함수
            console.log("서버로부터 전달된 pageNum(replyPagingCreator.myRelyPaging.pageNum): "
                + replyPagingCreator.myReplyPaging.pageNum);

            frmCmtPagingValue.find("input[name='pageNum']")
                .val(replyPagingCreator.myReplyPaging.pageNum);

            console.log("폼에 저장된 페이지번호 pageNum(): "
                + frmCmtPagingValue.find("input[name='pageNum']").val());
            var str="";
        }

        //댓글이 없으면(replyList가 null), if 문의 블록이 실행되고, 함수 실행이 중지됨(return), 따라서, for문 실행 않됨
        if( ! replyPagingCreator.replyList ){
            str += '<li class="left clearfix commentLi" '
                + ' style="text-align: center; background-color: lightCyan; '
                + ' height: 35px; margin-bottom: 0px; padding-bottom: 0px; '
                + ' padding-top: 6px; border: 1px dotted;">'
                + ' <strong>등록된 댓글이 없습니다.</strong></li>';
            commentUL.html(str);
            return;
        }

        //답글 표시 for문(<ul></ul>내에 아래의 li 요소들이 for문에 의해서 생성되어 표시됨)
        for (var i = 0, len = replyPagingCreator.replyList.length; i < len; i++) {
            str += '<li class="left clearfix commentLi" data-bno="' + bnoValue
                + ' data-rno="' + replyPagingCreator.replyList[i].rno + '">'
                + ' <div>';

            //댓글에 대한 답글 들여쓰기
            if(replyPagingCreator.replyList[i].lvl == 1){ //댓글 들여쓰기 안함
                str += ' <div>';
            } else if (replyPagingCreator.replyList[i].lvl == 2){ //답글 들여쓰기
                str += ' <div style="padding-left: 25px">';
            } else if (replyPagingCreator.replyList[i].lvl == 3){ //답글의 답글 들여쓰기
                str += ' <div style="padding-left: 50px">';
            } else if (replyPagingCreator.replyList[i].lvl == 4){ //답글의 답글 들여쓰기
                str += ' <div style="padding-left: 75px">';
            } else { //답글의 레벨이 5이상이면 동일한 들여쓰기
                str += ' <div style="padding-left: 100px">';
            }
            str += ' <span class="header info-rwriter">'
                + ' <strong class="primary-font">';
            if(replyPagingCreator.replyList[i].lvl > 1){ //답글인 경우, 앞에 아이콘 표시
                str += ' <i class="fa fa-reply fa-fw"></i>&ampnbsp';
            }
            str += ' replyPagingCreator.replyList[i].rwriter
                + ' </strong>'
                + ' <span>&ampnbsp</span>'
                + ' <small class="text-muted">'
                    + ' replyPagingCreator.replyList[i].rmodDate //수정날짜가 정수값으로 표시됨
                + ' </small>';

            if(replyPagingCreator.replyList[i].lvl > 1){ //답글인 경우, 답글을 표시
                str += ' <small>&ampnbsp 답글</small>';
            }
            str += ' </span>'
                + ' <p data-bno=' + replyPagingCreator.replyList[i].bno
                + ' data-rno=' + replyPagingCreator.replyList[i].rno
                + ' data-rwriter=' + replyPagingCreator.replyList[i].rwriter
```

```

+ '          >' + replyPagingCreator.replyList[i].rcontent + '</p>'
+ '      </div>';
str += '      <div class="bttnsReply" style="margin-bottom:10px">
+ '          <button type="button" style="display:inline-block" '
+ '              class="btn btn-primary btn-xs btnChgReplyReg">
+ '                  답글 작성</span></button>
+ '      </div>';
str += '  </div>
+ '</li>';
}//for-End
//UL 태그 내에 HTML 내용 표시
commentUL.html(str); //html() 사용 시, 새로운 내용으로 교체.
//commentUL.append(str); //append 사용 시, 기존 내용 밑에 새로운 내용 추가 (페이지 대신 사용).
}///.end callback 매개변수의 익명 함수
);///.end myCommentClsr.getCommentList
}///.end showCmtList

$(document).ready(function(){//페이지 로딩 시 함수 실행 ← 전체 JavaScript 내용 중 제일 마지막에 위치해야 함
showCmtList(1);
});

```

- ☞ 문서에서는 댓글/답글의 글 내용(replyPagingCreator.replyList[i].rcontent)은, <li> 요소 내에서 <p> 요소에 표시되도록 구현했습니다. 따라서, <p> 태그에도 data-rno, data-bno 속성을 지정하여 rno, bno 값을 가지도록 구현했습니다. 추가적으로, data-rwriter 속성으로 rwriter 값도 미리 지정했습니다(스프링 시큐리티 실습에서 사용).
- ☞ 제일 마지막에 \$(document).ready(function(){ showCmtList(1); }); 실행문에 의해, 서버로부터 전달된 게시물 상세 HTML 내용이 로딩될 때, 댓글/답글의 목록이 표시되도록 구현했습니다. 단, \$(document).ready(function(){ showCmtList(1); }); 실행문은 항상 제일 마지막에 위치하는 것이 적극 권장입니다. 따라서, 이 후로 작성되는 코드는 \$(document).ready(function(){ showCmtList(1); }); 실행문 위에 작성하시기 바랍니다.

- 추가 코드- 2: 페이지 번호 생성 함수 코드 추가 및 작성된 showCmtList() 함수에 페이지 번호 표시 코드 추가  
→ showCmtList() 함수 밑에 다음의 함수 추가

```

//댓글 목록에 표시할 페이지번호 생성 함수: replyPagingCreator로 부터 받아온 값을 이용
function showCmtPagingNums(pageNum, startPagingNum, endPagingNum, prev, next, lastPageNum ) {

    if(endPagingNum >= lastPageNum){
        endPagingNum = lastPageNum;
    }

    var str = "<div class='text-center'>
        + "    <ul class='pagination pagination-sm'>";
    //맨 앞으로
    if(prev){
        str += "        <li class='page-item'>
            + "            <a class='page-link' href='"+1+"'>
            + "                <span aria-hidden='true'>&laquo;</span>
            + "            </a>
            + "        </li>";
    }
    //이전 페이지
    if(next){
        str += "        <li class='page-item'>
            + "            <a class='page-link' href='"+(startPagingNum -1)+"'>이전</a>
            + "        </li>";
    }
}

```

```

//선택된 페이지 번호 Bootstrap의 색상표시
for(var i = startPagingNum ; i <= endPagingNum; i++){
    //active는 Bootstrap 클래스 이름
    var active = ( ( pageNum == i ) ? "active" : "" );

    str += "      <li class='page-item "+active+">"
    + "          <a class='page-link' href='"+i+"'>"+i+"</a>"
    + "      </li>";
}

if(next){
    str += "      <li class='page-item">
        + "          <a class='page-link' href='"+(endPagingNum + 1)+">다음</a>""
        + "      </li>";
}

//맨 마지막으로
if(next){
    str += "      <li class='page-item'">
        + "          <a class='page-link' href='"+(lastPageNum)+">&raquo;</a>""
        + "      </li>";
}

str += "  </ul>
+ "</div>";

$("#showCmtPagingNums").html(str); //#showCmtPagingNums div에 표시
}

```

→ showCmtList() 함수에 페이지 번호 표시 코드(빨간색 코드) 추가

```

//UL 태그 내에 HTML 내용 표시
commentUL.html(str); //html 사용 시, 새로운 내용으로 교체.
//commentUL.append(str); //append 사용 시, 기존 내용 밑에 새로운 내용 추가 (페이지 대신 사용).

//페이지번호 표시 함수 호출
showCmtPagingNums(replyPagingCreator.myReplyPaging.pageNum,
                  replyPagingCreator.startPagingNum,
                  replyPagingCreator.endPagingNum,
                  replyPagingCreator.prev,
                  replyPagingCreator.next,
                  replyPagingCreator.lastPageNum);

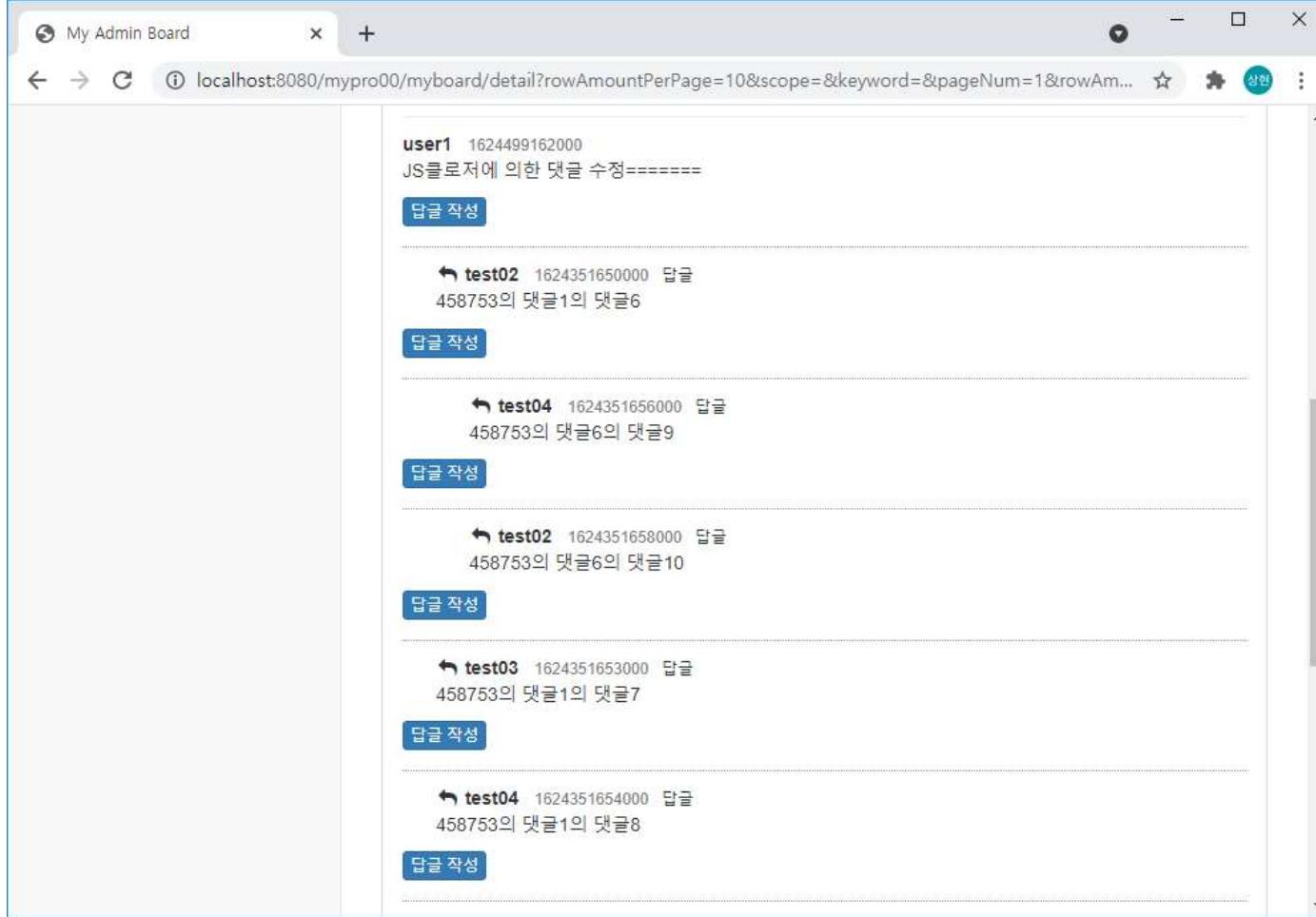
```

## [화면 확인]

톰캣 서버 기동 → 게시물 목록 → 댓글 많은 게시물 번호(458753, 게시물 목록에서 제일 첫 게시물)의 게시물 상세 페이지 이동

→ 아래 그림처럼, 페이징 처리에 의해서 10개의 댓글-답글이 표시됩니다.

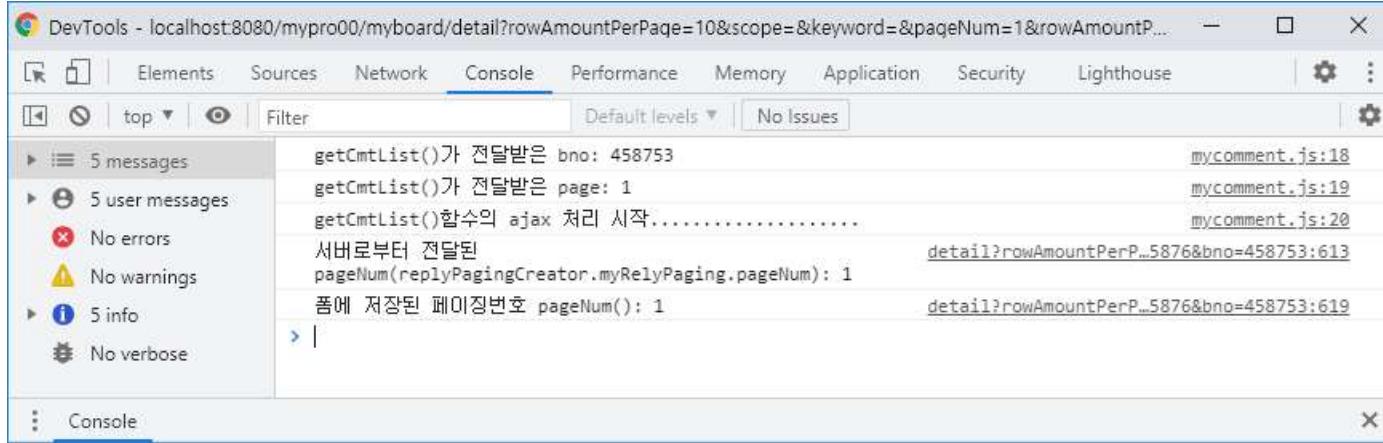
→ F12키로 개발자 도구 오픈 → Console에 오류표시가 없어야 합니다.



The screenshot shows a web browser window titled "My Admin Board". The URL is "localhost:8080/mypro00/myboard/detail?rowAmountPerPage=10&scope=&keyword=&pageNum=1&rowAm...". The page displays a forum post by "user1" with ID "1624499162000". The post content is "JS클로저에 의한 댓글 수정=====". Below the post, there are 10 replies from users "test02", "test04", and "test02" again. Each reply includes a "Reply" button. The replies are as follows:

- test02 1624351650000 댓글 458753의 댓글1의 댓글6
- test04 1624351656000 댓글 458753의 댓글6의 댓글9
- test02 1624351658000 댓글 458753의 댓글6의 댓글10
- test03 1624351653000 댓글 458753의 댓글1의 댓글7
- test04 1624351654000 댓글 458753의 댓글1의 댓글8

## [콘솔 확인 화면]



The screenshot shows the Chrome DevTools Console tab. The sidebar on the left shows the following message counts:

- 5 messages
- 5 user messages
- No errors
- No warnings
- 5 info
- No verbose

The main area displays the following log entries:

- getCommentList()가 전달받은 bno: 458753 (mycomment.js:18)
- getCommentList()가 전달받은 page: 1 (mycomment.js:19)
- getCommentList() 함수의 ajax 처리 시작..... (mycomment.js:20)
- 서버로부터 전달된 pageNum(replyPagingCreator.myRelyPaging.pageNum): 1 (detail?rowAmountPerPage=10&bno=458753:613)
- 폼에 저장된 페이지번호 pageNum(): 1 (detail?rowAmountPerPage=10&bno=458753:619)

☞ 이 후부터, 현재 웹 브라우저에 표시된 게시물 상세 페이지를 새로고침 하면서, 코드가 추가될 때마다, 개발자 도구에서 오류 확인 및 화면의 표시, 버튼 등의 기능 동작을 스스로 확인하시기 바랍니다.

→ 추가 코드- 3: 페이지 번호 클릭 이벤트(새로운 댓글 표시) 함수 추가

→ showCmtPagingNums() 함수 밑에 다음 코드 추가

```
//선택된 페이지 번호의 게시물목록 가져오는 함수: 이벤트 전파 이용
$("#showCmtPagingNums").on("click", "ul li a", function(e){

    e.preventDefault();

    var targetPageNum = $(this).attr("href");
    console.log("targetPageNum: " + targetPageNum);

    showCmtList(targetPageNum);

});
```

→ 추가 코드- 4: mycomment.js 파일에 날짜 표시 함수 추가 및 showCmtList() 함수의 날짜 표시 부분 수정

→ mycomment.js 파일에, 다음의 함수를 기준 함수 밑에 추가

```
//날짜시간 표시형식 설정 (서버와 상관없음)
//JSON 날짜시간을 그대로 표시하면 1594169682000 이렇게 표시됩니다.
//일반적인 날짜 시간 표시 형식으로 표시,
function showDatetime(datetimeValue) {

    var dateObj = new Date(datetimeValue); //전달된 댓글의 수정 날짜시간 값 저장
    var str = "";

    var yyyy = dateObj.getFullYear(); //YYYY
    var mm = dateObj.getMonth() + 1; // getMonth() is zero-based
    var dd = dateObj.getDate();
    var hh = dateObj.getHours();
    var mi = dateObj.getMinutes();
    var ss = dateObj.getSeconds();

    str= [yyyy, '/',
        (mm > 9 ? '' : '0') + mm, '/',
        (dd > 9 ? '' : '0') + dd, ' ',
        (hh > 9 ? '' : '0') + hh, ':',
        (mi > 9 ? '' : '0') + mi, ':',
        (ss > 9 ? '' : '0') + ss].join(''); //배열값으로 하나의 문자열로 합침

    return str ;
}

return {
    getCmtList : getCmtList,           //댓글 목록
    registerCmt : registerCmt,         //댓글 등록
    registerReply : registerReply,       //답글 등록
    getCmtReply : getCmtReply,          //댓글-답글 조회
    modifyCmtReply : modifyCmtReply,     //댓글-답글 수정
    removeCmtReply : removeCmtReply,      //댓글-답글 삭제(실제삭제)
    showDatetime : showDatetime         //날짜시간 표시형식 지정
};

})(); //즉시 실행 함수
```

→ showCmtList() 함수의 기준 코드 중 다음의 부분을 찾아서 빨간색 코드를 추가

```
str += replyPagingCreator.replyList[i].rwriter
+ '          </strong>
+ '          <span>&ampnbsp</span>
+ '          <small class="text-muted">수정일: '
+ '              myCommentClsr.showDatetime(replyPagingCreator.replyList[i].rmodDate)
+ '          </small>';
```

## [웹 브라우저 테스트]

☞ 브라우저 종류를 바꿔서 테스트 해야합니다.

☞ 지금까지 사용한 구글 크롬 브라우저에서 테스트 시에는, 수정된 mycomment.js 내용이 반영되지 않음  
톰캣 재기동, 오류만 발생됨.

☞ 톰캣 재기동, 브라우저 종료 후 새로운 크롬브라우저에서 테스트를 해도,

이전에 메모리에 로드 된 mycomment.js 파일로 실행되어 오류발생

→ 톰캣 서버 기동 → 엣지 브라우저(크롬이 아닌 다른 웹 브라우저) 실행

→ <http://localhost:8080/mypro00/myboard/list> URL로 접속

→ 댓글 많은 게시물 번호(458753, 게시물 목록에서 제일 첫 게시물)의 게시물 상세 페이지 이동

→ 댓글/답글에 표시된 날짜 시간 확인 및 페이징 이동 가능 확인

→ 웹 브라우저의 개발자 도구(F12키)의 Console 페이지에서도 오류표시가 없어야 합니다.

The screenshot shows a web browser window titled "My Admin Board". The URL in the address bar is <http://localhost:8080/mypro00/myboard/detail?rowAmountPerPage=10&scope=&keyword=&pageNum=1&rowAm...>. The page displays a list of comments:

- user3** 수정일: 2021/06/22 17:47:14  
458753의 댓글3  
[답글 작성](#)
- user5** 수정일: 2021/06/22 17:47:16  
458753의 댓글5  
[답글 작성](#)
- replyer** 수정일: 2021/06/22 20:33:21  
매퍼 댓글 입력 테스트  
[답글 작성](#)
- test** 수정일: 2021/06/23 09:49:35  
서비스 게시물에 대한 댓글 등록 테스트  
[답글 작성](#)
- testJson** 수정일: 2021/06/23 15:41:34  
컨트롤러-게시물에 대한 댓글 등록 테스트: JSON입력테스트  
[답글 작성](#)

At the bottom right of the comment list, there is a pagination control with the number 2 highlighted in blue.

-- 댓글 목록 표시 완료

### (3) 댓글 추가 화면 및 기능 구현(detail.jsp 파일에 다음 코드 추가)

#### ☞ 추가 코드- 5: 다음의 기능 구현

- 댓글 작성 버튼 클릭 → 댓글 등록 버튼, 취소 버튼 표시, 댓글 입력창 활성화 기능 구현
- 댓글 등록 버튼 클릭 → 서버에 댓글 저장 후, 목록화면 표시(추가된 댓글(목록의 맨 마지막에 표시) 페이지 표시)
  - ReplyServiceImpl 수정
  - 댓글 목록 표시 시, 버튼 및 입력창 표시 초기화(댓글 작성 버튼만 표시)
- 취소 버튼 클릭 → 동일 댓글 화면에서 댓글 작성 버튼만 표시, 댓글 입력창 읽기전용으로 변경

→ 다음의 함수들을 순서에 맞추어, 페이징 번호 클릭 이벤트 실행문 밑(제일 마지막 함수)에 작성.

(주의) 제일 마지막의 \$(document).ready(function(){}); 실행문 위에 아래의 함수들이 정의되어야 합니다.

```
<%--댓글 추가 요소 초기화 함수 --%>
function chgBeforeCmtBtn() {
    $("#btnChgCmtReg").attr("style", "display:inline-block");
    $("#btnRegCmt").attr("style", "display:none");
    $("#btnCancelRegCmt").attr("style", "display:none");
    $(".txtBoxCmt").val("");
    $(".txtBoxCmt").attr("readonly", true);
}

<%--댓글 작성 버튼 클릭 - 댓글 등록 버튼으로 변경, 댓글 입력창 활성화--%>
$("#btnChgCmtReg").on("click", function(){
    chgBeforeCmtBtn();

    $(this).attr("style", "display:none");
    $("#btnRegCmt").attr("style", "display:inline-block; margin-right:2px");
    $("#btnCancelRegCmt").attr("style", "display:inline-block");
    $(".txtBoxCmt").attr("readonly", false);

});

<%--댓글등록 취소 클릭 --%>
$("#btnCancelRegCmt").on("click", function(){
    if(!confirm("댓글 입력을 취소하시겠습니까?")){
        return ;
    }
    chgBeforeCmtBtn();
});

<%--댓글등록 버튼 클릭 이벤트 처리 --%>
$("#btnRegCmt").on("click", function(){

    var loginUser = "user9";
    var txtBoxCmt = $(".txtBoxCmt").val();
    var comment = { bno: bnoValue,
                    rcontent: txtBoxCmt,
                    rwriter: loginUser};

    console.log("댓글등록: 서버전송 객체내용: " + comment);
});
```

```

myCommentClsr.registerCmt(
    comment,
    function(serverResult){
        $(".txtBoxCmt").val("");
        chgBeforeCmtBtn();

        alert("댓글이 등록되었습니다");

        showCmtList(-1); <%--댓글이 추가된 맨 마지막 페이지 표시--%>
    }
);
});

```

○ ReplyServiceImpl.java 클래스 수정

☞ -1 값인 pageNum 이 전달되었을 때, 댓글 목록의 마지막 페이지가 표시될 수 있도록 수정

→ src/main/java/com.spring5213.mypro00.service. ReplyServiceImpl.java 클래스를 코드 작성 뷰에 오픈

→ getReplyListByBno() 메소드를 다음처럼 수정, 수정할 내용이 많으므로, 다시 작성하는 것이 빠릅니다.

```

//특정 게시물에 대한 댓글 목록 조회(페이징-전체댓글 수 고려)
@Override
public MyReplyPagingCreatorDTO getReplyListByBno(MyReplyPagingDTO myReplyPaging) {
    log.info("댓글-서비스-게시물에 대한 댓글 목록 조회 - 전달된 MyReplyPagingDTO: " + myReplyPaging);

    int replyTotalByBno = myReplyMapper.selectReplyTotalByBno(myReplyPaging);

    Integer pageNum = myReplyPaging.getPageNum();

    if (replyTotalByBno == 0) {
        myReplyPaging.setPageNum(1);
        log.info("댓글-서비스- 댓글이 없는 경우, pageNum은 1: 수정된 myReplyPaging: " + myReplyPaging);

        MyReplyPagingCreatorDTO myReplyPagingCreator =
            new MyReplyPagingCreatorDTO(replyTotalByBno, myReplyPaging, null);

        return myReplyPagingCreator ;
    } else {
        if (pageNum == -1) {
            pageNum = (int) Math.ceil(replyTotalByBno/(myReplyPaging.getRowAmountPerPage()*1.0));
            myReplyPaging.setPageNum(pageNum);
            log.info("댓글-서비스-댓글추가 후, 마지막 댓글 페이지로 이동(myReplyPaging): " + myReplyPaging);
        }

        MyReplyPagingCreatorDTO myReplyPagingCreator =
            new MyReplyPagingCreatorDTO(
                myReplyMapper.selectReplyTotalByBno(myReplyPaging),
                myReplyPaging,
                myReplyMapper.selectMyReplyList(myReplyPaging));
        log.info("댓글-서비스-게시물에 대한 댓글 목록 조회 - 생성된 myReplyPagingCreatorDTO: " + myReplyPagingCreator);
        log.info("댓글-서비스-게시물에 대한 댓글 목록 조회 - myReplyPagingCreator가 컨트롤러로 전달됨");

        return myReplyPagingCreator;
    }
}

```

[웹 브라우저 테스트] - 웹 브라우저에서 댓글 등록을 스스로 테스트 합니다.

#### (4) 댓글에 대한 답글 추가 화면 및 기능 구현

##### ☞ 추가 코드- 6: 다음의 기능 구현

- 답글 작성 버튼 클릭 → 답글 등록 버튼, 취소 버튼 표시 기능 구현
- 답글 등록 버튼 클릭 → 서버에 답글 저장 후, 목록화면 표시(답글이 추가된 페이지 표시)  
→ 댓글 목록 표시 시, 버튼 표시 초기화(답글 작성 버튼만 표시)
- 취소 버튼 클릭 → 동일 댓글 화면에서 답글 추가 버튼만 표시

→ 다음의 함수들을 순서에 맞추어, 앞에서 추가한 코드의 맨 마지막에 작성(현재 제일 마지막).

(주의) 제일 마지막의 `$(document).ready(function(){});` 실행문 위에 아래의 함수들이 정의되어야 합니다.

```
<%--답글 관련 화면 상태 초기화--%>
function chgBeforeReplyBtn(){
    $(".btnRegReply").remove();
    $(".btnCancelRegReply").remove();
    $(".txtBoxReply").remove();
    $(".btnChgReplyReg").attr("style", "display:inline-block");

}

<%--답글 작성 버튼 클릭 이벤트--%>
<%--JSP 코드에 없는 생성된 요소, 이벤트 전파 사용 --%>
$(".chat").on("click", ".commentLi .btnChgReplyReg" ,function(){

    $("p").attr("style", "display:inline-block");

    chgBeforeCmtBtn();
    chgBeforeReplyBtn();

    var strTxtBoxReply =
        "<textarea class='form-control txtBoxReply' name='rcontent' style='margin-bottom:10px;'"
        + "placeholder='답글작성을 원하시면, &#10;답글 작성 버튼을 클릭해주세요.'"
        + "></textarea>"
        + "<button type='button' class='btn btn-warning btn-xs btnRegReply'>답글 등록</button>"
        + "<button type='button' class='btn btn-danger btn-xs btnCancelRegReply'"
        + "style='margin-left:4px;'>취소</button>";

    $(this).after(strTxtBoxReply);
    $(this).attr("style", "display:none"); //답글 작성 버튼 감춤
})

<%--답글등록 취소 클릭--%>
$(".chat").on("click", ".commentLi .btnCancelRegReply" ,function(){
    if(!confirm("답글 입력을 취소하시겠습니까?")){
        return ;
    }
    chgBeforeReplyBtn();

});

<%--답글 등록 버튼 클릭 이벤트 처리: 답글이 달린 댓글이 있는 페이지 표시--%>
$(".chat").on("click", ".commentLi .btnRegReply" ,function(){
    var loginUser = "test8";
    var pageNum = frmCmtPagingValue.find("input[name='pageNum']").val();
    console.log("답글 추가가 발생된 댓글 페이지 번호: "+ pageNum);

    var txtBoxReply = $(this).prev().val();
```

```

console.log("txtBoxReply: " + txtBoxReply);

var prnoVal = $(this).closest("li").data("rno");
console.log("prnoVal: " + prnoVal);

var reply = { bno: bnoValue,
    rcontent: txtBoxReply,
    rwriter: loginUser,
    prno: prnoVal };

console.log("답글등록: 서버전송 객체내용: " + reply);

myCommentClsr.registerReply(
    reply,
    function(serverResult){
        alert("답글이 등록되었습니다");

        showCmtList(pageNum);<%--댓글이 추가된 페이지 표시--%>
    }
);
}
)

```

## (5) 댓글-답글 수정/삭제 화면 및 기능 구현

### ☞ 추가 코드- 7: 다음의 기능 구현

- 목록의 댓글-답글 글 내용 클릭 → 수정/삭제 화면 표시
- 수정 버튼 클릭 → 서버에 댓글-답글 저장 후, 목록화면 표시(수정된 댓글-답글이 있던 댓글 목록페이지 표시)
  - 댓글 목록 표시 시, 버튼 표시 초기화(수정/삭제 요소 삭제)
- 삭제 버튼 클릭 → 서버에서 삭제 후, 목록화면 표시(삭제된 댓글-답글이 있던 댓글 목록 페이지 표시)
  - 댓글 목록 표시 시, 버튼 표시 초기화(수정/삭제 요소 삭제)
- 취소 버튼 클릭 → 동일 댓글 목록 화면 표시

→ 다음의 함수들을 순서에 맞추어, 앞에서 추가한 코드의 맨 마지막에 작성(현재 제일 마지막).

(주의) 제일 마지막의 \$(document).ready(function(){}); 실행문 위에 아래의 함수들이 정의되어야 합니다.

```

<%--댓글/답글 수정-삭제-취소-입력창 삭제 함수--%>
function chgBeforeCmtRepBtns(){

    $("p").attr("style","display:inline-block;");

    //답글처리관련버튼
    $(".btnModCmt").remove();
    $(".btnDelCmt").remove();
    $(".btnCancelCmt").remove();
    $(".txtBoxMod").remove();
}

<%--댓글-답글 수정/삭제 화면 요소 표시: p 태그 클릭 이벤트 --%>
$(".chat").on("click", ".commentLi p", function(){
    chgBeforeCmtBtn();<%--댓글 등록 상태 초기화--%>
    chgBeforeReplyBtn();<%--다른 답글 등록 상태 초기화--%>
})

```

```

chgBeforeCmtRepBtns(); <%-- 다른 답글/댓글 수정 상태 초기화--%>

$(this).parents("li").find(".btnChgReplyReg").attr("style", "display:none");

var rcontent = $(this).text();
console.log("선택된 댓글내용: " + rcontent);

var strTxtBoxReply =
    "<textarea class='form-control txtBoxMod' name='rcontent' style='margin-bottom:10px;' "
    + "placeholder='답글작성을 원하시면,&#10;답글 작성 버튼을 클릭해주세요.'"
    + "></textarea>"
    + "<button type='button' class='btn btn-warning btn-sm btnModCmt'>수정</button> "
    + "<button type='button' class='btn btn-danger btn-sm btnDelCmt'>삭제</button> "
    + "<button type='button' class='btn btn-info btn-sm btnCancelCmt' style='margin-left: 4px;'>취소</button>";

$(this).after(strTxtBoxReply);
$(".txtBoxMod").val(rcontent);
$(this).attr("style", "display:none");

});

<%-- 댓글-답글 수정/삭제의 취소 버튼 클릭 이벤트 --%>
$(".chat").on("click", ".commentLi .btnCancelCmt", function(){
    chgBeforeCmtBtn();
    chgBeforeReplyBtn()
    chgBeforeCmtRepBtns()
});

<%-- 댓글-답글 수정 처리: 수정 버튼 클릭 이벤트 --%>
$(".chat").on("click", ".commentLi .btnModCmt", function(){

    <%--작성자 변수에 저장--%>
    var rwriterVal = $(this).siblings("p").data("rwriter");
    var pageNum = frmCmtPagingValue.find("input[name='pageNum']").val();
    console.log("댓글/답글 수정이 페이지 번호: " + pageNum);

    var txtBoxComment = $(this).prev().val();
    console.log("txtBoxComment: " + txtBoxComment);

    var rnoVal = $(this).closest("li").data("rno");
    console.log("rnoVal: " + rnoVal);

    var comment = { bno: bnoValue,
                    rno: rnoVal,
                    rcontent: txtBoxComment,
                    rwriter: rwriterVal };

    console.log("답글등록: 서버전송 객체내용: " + comment);

    myCommentClsr.modifyCmtReply(
        comment,
        function(serverResult){
            alert("수정되었습니다");

            showCmtList(pageNum); <%-- 답글이 추가된 페이지 표시 --%>
        }
    );
});

<%-- 댓글-답글 삭제 처리: 삭제 버튼 클릭 이벤트 --%>
$(".chat").on("click", ".commentLi .btnDelCmt", function(){

    <%--작성자 변수에 저장--%>
    var rwriterVal = $(this).siblings("p").data("rwriter");
    var delConfirm = confirm('삭제하시겠습니까?');


```

```

if(!delConfirm){
    alert('삭제가 취소되었습니다.');
    return ;
}

var pageNum = frmCmtPagingValue.find("input[name='pageNum']").val();
console.log("답글 삭제가 발생된 댓글 페이지 번호: " + pageNum);

var rnoVal = $(this).closest("li").data("rno");
console.log("rnoVal: " + rnoVal);

var myComment = { bno: bnoValue,
                  rno: rnoVal,
                  rwriter: rwriterVal };

console.log("답글삭제: 서버전송 객체내용: " + myComment);

myCommentClsr.removeCmtReply(
    myComment,
    function(serverResult){<%--서버에서 댓글저장 성공 시, 브라우저에서 실행될 콜백함수--%>
        alert("삭제되었습니다.");
        showCmtList(pageNum);
    }
);
});
});

```

- 화면 테스트는 스스로 !!!!

#### (6) 버튼 동작 시, 불필요한 화면 정리

→ 다음의 지시대로 기존 함수에 버튼 및 입력요소를 초기화하는 코드 추가

→

```
<%-- 댓글작성 버튼 클릭 이벤트 함수에 다음의 코드 추가 --%>

<%--댓글 작성 버튼 클릭 - 댓글 등록 버튼으로 변경, 댓글 입력창 활성화--%>
$("#btnChgCmtReg").on("click", function(){

    chgBeforeCmtRepBtns(); ←추가
    chgBeforeCmtBtn();
    chgBeforeReplyBtn() ←추가

    $(this).attr("style", "display:none");
    $("#btnRegCmt").attr("style", "display:inline-block; margin-right:2px");
    $("#btnCancelRegCmt").attr("style", "display:inline-block");
    $("#txtBoxCmt").attr("readonly", false);

});
```

```
<%-- 답글 작성 버튼 클릭 이벤트 함수에 다음의 코드 추가 --%>
<%--답글 작성 버튼 클릭 이벤트--%>
<%--JSP 코드에 없는 생성된 요소, 이벤트 전파 사용 --%>
$(".chat").on("click", ".commentLi .btnChgReplyReg" ,function(){

    $("p").attr("style", "display:inline-block;");
```

```
chgBeforeCmtBtn();  
chgBeforeReplyBtn();  
chgBeforeCmtRepBtns(); ←추가
```

```
var strTxtBoxReply =
```

```
...(생략)...
```

-- 댓글-답글 화면처리 완료

#### (7) 댓글추가/삭제에 따른 게시물의 댓글 수 변경(수정하는 파일이 많으므로 잘 수행합니다)

- ☞ 새로운 댓글/답글이 book\_ex.tbl\_myReply 테이블에 등록으로 저장될 때, 게시물의 댓글 수(book\_ex.tbl\_myboard.breplyCnt 컬럼)의 값에 +1 변경이 수행되어야 합니다. 또한, 기존 댓글/답글 삭제가 발생된 경우에도, 게시물의 댓글수에 대하여 -1 변경이 수행되어야 합니다.
- ☞ 위의 댓글/답글 등록과 댓글 수 변경 작업은, tbl\_myReply 테이블과 tbl\_myBoard 테이블의 데이터에 대한 변경작업들이 하나의 단위로 구성된 업무로 처리되어야 합니다.
  - 이 때, 두 테이블에 대한 변경 작업 중, 하나라도 오류가 되면, 변경 작업이 완료된 테이블의 데이터도 포함하여 모두 변경 전 상태가 되어야 합니다.
- ☞ 댓글 등록의 경우처럼, 댓글 등록이라는 업무를 구현할 때, 같이 고려되어 처리해야 하는 작업(댓글 수 증가)도 업무에 포함시켜 고려해야 합니다. 이렇게 하나의 업무를 구성하는 하나 이상의 처리작업을 한 그룹으로 묶어서 "하나의 트랜잭션"이라고 합니다. 참고로, 3명의 사용자가 동시에 댓글을 등록한다면, 3개의 댓글등록 업무가 동시에 처리되는 것이고, 이는 3개의 트랜잭션이 동시에 처리 중이라고 합니다.
- ☞ 여러가지 처리가 하나의 업무를 구성하는 경우, 즉, 여러가지 처리가 하나의 트랜잭션을 구성하는 경우, 이를 프로그램으로 구현하는 것은 매우 어려운 작업이 될 수 있습니다. 즉, 여러 처리가 모두 정상적으로 성공하면 업무 처리 후의 상태를 유지해야 되고, 여러 처리 중, 하나라도 오류가 발생되어 실패한 경우, 다른 구성 단위업무도 업무가 처리되기 전상태로 되돌리는 것을 구현하는 것은 어려운 작업입니다.
- ☞ 스프링 프레임워크에서는 이를 쉽게 처리할 수 있도록 스프링 AOP와 스프링 트랜잭션이라는 기능을 제공합니다. 이 두 기능을 이용하면, 하나의 트랜잭션으로 처리해야 되는 메서드에 @Transactional 어노테이션을 이용하여 쉽게 트랜잭션으로 구성되도록 구현할 수 있습니다(스프링 트랜잭션 기능은 스프링 AOP 기능에 의해 제어됨).

#### (7-1) 스프링 트랜잭션 기능 사용을 위한 라이브러리 설치(이미 설치되어 있음, pom.xml 파일 확인만 수행)

```
<properties>  
    <java-version>1.8</java-version>  
    <org.springframework-version>5.2.13.RELEASE</org.springframework-version>  
    <org.aspectj-version>1.9.6</org.aspectj-version><!--1.6.10-->1.9.6 -->  
    <org.slf4j-version>1.7.30</org.slf4j-version><!-- 1.6.6 -->1.7.30 -->  
</properties>  
<dependencies>  
    <!-- 새로 추가: aspectjweaver AOP-->  
<dependency>  
    <groupId>org.aspectj</groupId>  
    <artifactId>aspectjweaver</artifactId>  
    <version>1.9.6</version>
```

```

</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency><!-- 기본 구성됨 -->
    <groupId>org.aspectj</groupId>

    <artifactId>aspectjrt</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>

```

- ☞ Spring Transaction 기능(spring-tx)을 사용하려면, Spring AOP 관련 라이브러리(aspectjweaver, aspectjrt) 및 Spring-JDBC 라이브러리를 spring-tx 라이브러리와 같이 설치해야 합니다.

#### (7-2) 스프링 트랜잭션 라이브러리 기능 사용 구성

- ☞ servlet-context.xml 파일에 구성할 내용: 스프링 AOP 기능 활성화
- ☞ mybatis-context.xml 파일에 구성할 내용: (1) 스프링 트랜잭션 기능 활성화 (2) 스프링 트랜잭션 관리 빈 사용 구성 설정

##### (7-2-1) Spring AOP 기능 활성화

- src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml 파일을 코드 작성 뷰에 오픈
- 하단의 Namespaces 탭 클릭 → aop Namespace 항목 체크 → 저장!!! → Source 탭 클릭(AOP 관련 네임스페이스가 상단에 추가됨)
- <aop:aspectj-autoproxy/> 코드 추가 → 저장!!!!

```

<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc https://www.springframework.org/schema/mvc/spring-
mvc.xsd
    http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

    <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->
    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven/>

    <!-- Enables Spring AOP -->
    <aop:aspectj-autoproxy/> ← 추가 : 스프링 AOP 기능 활성화
    ...
    ... (생략)...

```

##### (7-2-2) 스프링 트랜잭션 기능 활성화 및 스프링의 DataSourceTransactionManager 빈 사용 구성

- ☞ 스프링 트랜잭션 기능은, 데이터베이스에 접속에 사용되는 dataSource 빈을 주입받아야 하므로, dataSource 빈이 설정된 스프링 빈 구성 설정 파일에 기능을 활성화 합니다(문서에서는 mybatis-context.xml 파일).

☞ 스프링 트랜잭션 기능 활성화 구성 후, org.springframework.jdbc.datasource.DataSourceTransactionManager 빈이 자동으로 생성되도록 구성합니다.

→ src/main/webapp/WEB-INF/spring/mybatis-context.xml 파일을 코드 작성 뷰에 오픈

→ 하단의 Namespaces 탭 클릭 → tx Namespace 항목 체크 → 저장!!! → Source 탭 클릭

→ 스프링 트랜잭션 관련 네임스페이스가 상단에 추가됨

→ <tx:annotation-driven/> 코드 및 빈 자동 구성 추가 → 저장!!!!

```
<?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:context="http://www.springframework.org/schema/context"
    xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/mybatis-spring.xsd
        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-4.3.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">

<!-- Connection Pool 관련 Bean 구성: HikariCP -->
<!-- 1. jdbc.properties 파일에서 읽어 들이는 설정 -->
<context:property-placeholder location="classpath:jdbcConfig/jdbc.properties"/>

<!-- 2. HikariConfig 빈을 통해 읽어들인 정보를 설정: 설정값 형식이 JSTL 표현식과 유사 -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <!-- for Server Deploy --><!--
    <property name="driverClassName" value="${jdbcHikari.driverClassName}"></property>
    <property name="jdbcUrl" value="${jdbcHikari.jdbcUrl}"></property>
    <property name="username" value="${jdbcHikari.username}"></property>
    <property name="password" value="${jdbcHikari.password}"></property> -->

    <!-- for Development log4jdbc-log4j2 라이브러리-->
    <property name="driverClassName" value="${jdbcHikariLog.driverClassName}"></property>
    <property name="jdbcUrl" value="${jdbcHikariLog.jdbcUrl}"></property>
    <property name="username" value="${jdbcHikariLog.username}"></property>
    <property name="password" value="${jdbcHikariLog.password}"></property>
</bean>

<!-- 3. HikariDataSource 클래스를 이용한 dataSource 빈 생성 -->
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close">
    <constructor-arg ref="hikariConfig" />
</bean>

<!-- 4. mybatis-spring 연동 (HikariDataSource 빈 주입 받음) -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mapperLocations">
        <list>
            <value>classpath:com/spring5213/mypro00/mapper/MyBoardMapper.xml</value>
            <value>classpath:com/spring5213/mypro00/mapper/MyReplyMapper.xml</value>
            <!-- <value>classpath:com/spring5213/mypro00/mapper/MyBoardMapperDAO.xml</value> -->
        </list>
    </property>
</bean>

<!-- 5. mybatis-spring 연동 (SqlSessionFactoryBean 빈 주입 받음) -->
<!-- SqlSessionFactory 보다 쓰레드에 안전, DAO 타입 클래스 사용 시는 세션 생성을 위해 반드시 사용해야 함 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>

<!-- 매퍼인터페이스 자동 빈 생성 -->
<!-- 설정된 패키지에 DAO클래스를 대신하는 매퍼-인터페이스가 있습니다. -->
```

```

<mybatis-spring:scan base-package="com.spring5213.mypro00.mapper"/>

<!-- 스프링 트랜잭션 기능 활성화 -->
<tx:annotation-driven/>

<!-- Spring Transaction 관리자 빈(dataSource 빈을 주입받음) -->
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
</bean>

</beans>

```

### (7-3) 게시물에 대한 댓글 수 변경 처리 구현

- ☞ 게시물에 대한 댓글 수 처리는 많이 고려되기 때문에, 이미 데이터베이스의 TBL\_MYBOARD 테이블과 MyBoardVO에 관련 컬럼과 필드는 구현되어 있습니다.
- ☞ 댓글/답글 등록 및 댓글/답글 삭제 처리에 동반되는 게시물의 댓글 수 변경은 TBL\_MYREPLY 테이블이 아닌 TBL\_MYBOARD 테이블에서 일어납니다. 따라서, TBL\_MYBOARD 테이블에 작업과 관련된 MyBoardMapper 인터페이스와 MyBoardMapper.xml SQL 매핑 파일에 필요한 SQL문과 메서드를 구현합니다.

→ src/main/resource/com/spring5213/mypro00/mapper/MyBoardMapper.xml 파일을 코드 작성부에 오픈

→ 다음의 UPDATE문을 제일 마지막의 </mapper> 태그 위에 추가합니다.

```

<!-- 게시물의 댓글 개수 수정: 댓글추가 시에 #{amount}에 1, 댓글삭제 시 #{amount}에 -1 이 각각 전달됨 -->
<update id="updateBReplyCnt">
  UPDATE book_ex.tbl_myboard
  SET breplyCnt = breplyCnt + #{amount}
  WHERE bno = #{bno} AND bdelFlag = 0
</update>

</mapper>

```

→ src/main/java/com.spring5213.mypro00.mapper.MyBoardMapper.java 인터페이스를 코드 작성부에 오픈

→ 다음의 updateBReplyCnt() 메서드를 추가합니다.

```

// 특정 게시물 댓글/답글 수 변경: 삭제 시 -1, 추가 시 +1
public void updateBReplyCnt(@Param("bno") Long bno, @Param("amount") int amount);

```

### (7-4) 댓글/답글 등록과 삭제 시에 게시물에 대한 댓글 수 변경 처리 구현

: 둘 이상의 데이터베이스 수정이 포함된 서비스 메서드에 스프링 트랜잭션 적용(@Transaction 어노테이션 추가)

- ☞ 하나의 업무 처리 로직은 비즈니스 계층인 서비스 클래스에 구현합니다.

댓글 등록(TBL\_MYREPLY 테이블)과 함께 수행되는 댓글 수 변경(TBL\_MYBOARD 테이블)이 하나의 업무를 구성하는 처리들이고, 이 중 댓글 등록이 매인 처리이므로, MyReplyServiceImpl 구현 클래스에 구현된 댓글/답글의 등록 및 삭제 메서드에 게시물의 댓글 수 변경 작업이 같이 처리되도록 구현합니다.

→ src/main/java/com.spring5213.mypro00.MyReplyServiceImpl.java 클래스를 코드 작성부에 오픈

→ 다음의 빨간색 코드 추가.

```
import org.springframework.transaction.annotation.Transactional;
import com.spring5213.mypro00.mapper.MyBoardMapper;

@Service
@Log4j
//@AllArgsConstructor
public class MyReplyServiceImpl implements MyReplyService {

    private MyReplyMapper myReplyMapper;
    private MyBoardMapper myBoardMapper; ←추가

    public MyReplyServiceImpl(MyReplyMapper myReplyMapper, MyBoardMapper myBoardMapper) { ←추가
        this.myReplyMapper = myReplyMapper;
        this.myBoardMapper = myBoardMapper; ←추가
    }

    ... (생략) ...

    // 특정 게시물에 대한 댓글 등록: rno 반환
    @Transactional ←추가
    @Override
    public long registerReplyForBoard(MyReplyVO myReply) {
        log.info("댓글-서비스-게시물에 대한 댓글 등록 시 처음 전달된 myReplyVO: " + myReply);

        // 게시물 댓글 수 증가
        myBoardMapper.updateBReplyCnt(myReply.getBno(), 1); ←추가

        // 게시물에 대한 댓글 등록 처리
        myReplyMapper.insertMyReplyForBoard(myReply);
        log.info("댓글-서비스 - 댓글 등록: 등록된 myReply: " + myReply);
        log.info("댓글-서비스 - 댓글 등록: 등록된 rno(byGetter): " + myReply.getRno());
        log.info("댓글-서비스 - 댓글 등록: 등록된 rno가 컨트롤러로 전달됨 " );

        return myReply.getRno();
    }

    // 댓글의 답글 등록: rno 반환
    @Transactional ←추가
    @Override
    public long registerReplyForReply(MyReplyVO myReply) {
        log.info("댓글-서비스-답글 등록 시 myReplyVO: " + myReply);

        // 게시물 댓글 수 증가
        myBoardMapper.updateBReplyCnt(myReply.getBno(), 1); ←추가

        // 댓글에 대한 답글 등록 DB 저장
        myReplyMapper.insertMyReplyForReply(myReply);

        log.info("댓글-서비스 - 답글 등록: 등록된 myReply: " + myReply);
        log.info("댓글-서비스 - 답글 등록: 등록된 rno(ByGetter): " + myReply.getRno() );
        log.info("댓글-서비스 - 답글 등록: 등록된 rno가 컨트롤러로 전달됨 " );

        return myReply.getRno();
    }

    ... (생략) ...

    // 특정 댓글 삭제: 삭제된 행수인 1 반환
    @Transactional ←추가
    @Override
    public int removeReply(long bno, long rno) {
        log.info("댓글-서비스-댓글-삭제시 전달된 bno: " + bno);
        log.info("댓글-서비스-댓글-삭제시 전달된 rno: " + rno);

        // 댓글 삭제: 댓글 개수 감소
```

```

myBoardMapper.updateBReplyCnt(bno, -1);    ←추가

Integer delCnt = myReplyMapper.deleteMyReply(bno, rno);
log.info("댓글-서비스-삭제된 댓글 개수: " + delCnt);

return delCnt;
}

}

```

(7-5) SQL\*Developer에서 다음의 문장을 실행하여, 현재 실습용 데이터 상태에 맞추어, 댓글 개수 수정.

```

UPDATE book_ex.tbl_myBoard a
SET breplyCnt = (SELECT count(*)
                  FROM book_ex.tbl_myreply
                 WHERE bno = a.bno ) ;

COMMIT ;

```

(7-6) 댓글 수 표시 화면처리

☞ 게시물 목록에 댓글 수가 제목 옆에 표시되도록 list.jsp 페이지를 수정합니다.

→ src/main/webapp/WEB-INF/views/myboard/list.jsp 페이지를 코드 뷰에 오픈

→ 다음의 코드 추가

```

...(생략)...

<td style="text-align:left;">
  <c:out value="${board.btitle}" />
  <small>[댓글수: <strong><c:out value="${board.breplyCnt}" /></strong>]</small>
</td>
...(생략)...

```

[스프링 트랜잭션 작동 테스트]

☞ MyReplyMapper.xml 파일을 열고 오류 상황이 발생되도록 수정합니다.

**테스트 후에 원래대로 수정합니다.!!!!**

☞ 게시물의 댓글 개수 증가 후, 댓글 등록을 하는 INSERT 문에 오류가 나도록 값을 수정

→ 증가된 처리가 다시 원래의 값으로 되돌아 가는지 확인 → **테스트 후 XML 매퍼 파일의 변경 내용을 원래의 내용으로 수정함!!!!.**

→ src/main/resources/com/sprin5213/mypro00/mapper/MyReplyMapper.xml 파일을 코드 뷰에 오픈

→ 댓글 등록을 처리하는 insertMyReplyForBoard 아이디의 INSERT 문에 제일 마지막 DEFAULT를 "a"로 수정(빨간색 코드)

```

<!-- 특정 게시물에 대한 댓글 등록-->
<insert id="insertMyReplyForBoard">
  <selectKey keyProperty="rno" order="BEFORE" resultType="long">
    SELECT book_ex.seq_myreply.NEXTVAL FROM dual
  </selectKey>

```

```

INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno)
VALUES (#{bno}, #{rno}, #{rcontent}, #{rwriter}, DEFAULT, DEFAULT, 'a')
</insert>

```

☞ 맨 마지막 DEFAULT 키워드로 값이 입력되는 컬럼은 숫자 값이 저장되는 prno 컬럼입니다. 해당 컬럼에 a 가 저장될 수 없으니 오라클 데이터베이스 엔진이 오류를 발생시킵니다.

☞ 데이터베이스에 대한 처리 순서가 댓글 수 변경 후에 → 댓글 등록이 수행되도록 구현했으므로, 댓글 수가 증가 했지만, 댓글 등록 시의 오류에 의해서 댓글 수 증가도 원래의 댓글 수로 되돌아 가야합니다.

#### [테스트 실행]

톰캣 서버 재기동 → <http://localhost:8080/mypro00/myboard/list> URL로 접속 → 게시물 목록 화면 표시됨

→ 게시물의 목록 정보에서 현재 댓글 개수를 확인

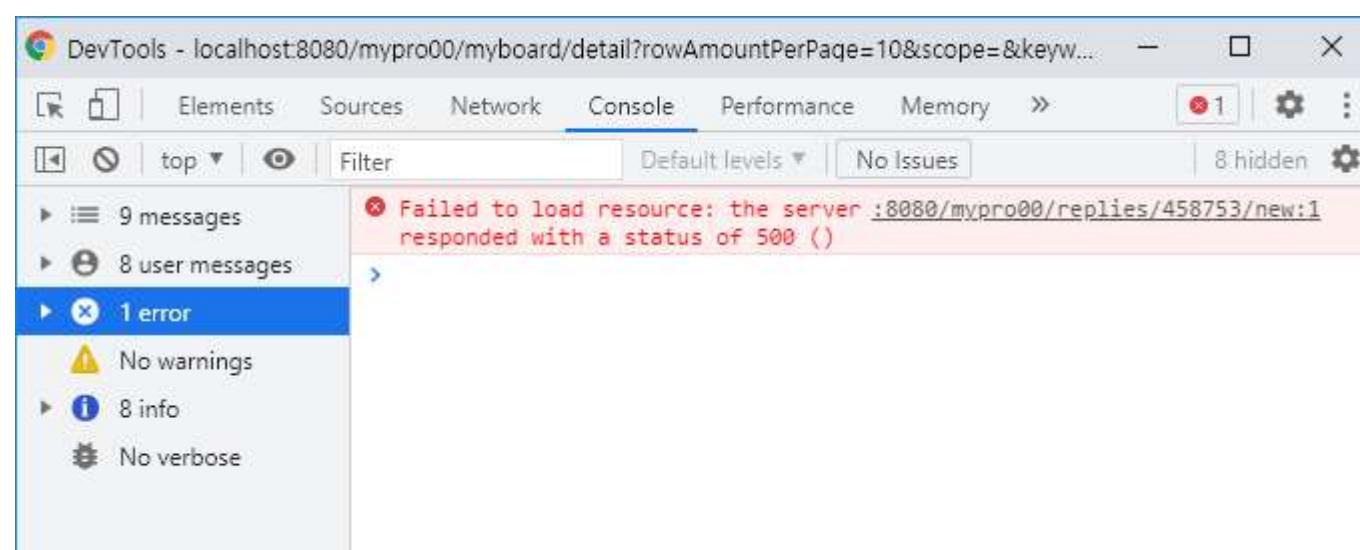
→ 제일 첫 번째 게시물의 게시물 상세 화면 이동

→ 게시물 상세 페이지에서 댓글등록 버튼 클릭

→ 댓글 등록 시도 → 오류로 동작 안됨 → 취소 버튼 클릭

→ 목록 버튼으로 게시물 목록 화면 이동

→ 게시물의 댓글 개수 확인 → 0 개로 표시되어야 함



My Admin Board

localhost:8080/mypro00/myboard/list?pageNum=1&r...

번호	제목	작성자	작성일	수정일	조회수
458753	화면 새글등록 테스트1 [댓글수: 18]	test	2021/06/21	2021/06/21 22:33:07	27
458752	테스트제목5 [댓글수: 1]	test5	2021/06/21	2021/06/21 19:22:10	7
458751	테스트제목4 [댓글수: 1]	test4	2021/06/21	2021/06/21 08:28:13	0
458750	테스트제목3 [댓글수: 0]	test3	2021/06/21	2021/06/21 08:28:13	0

← 댓글 수 변화 없음.

#### → 실습 시 콘솔 표시 메시지 일부(댓글 등록과 관련된 로그만 표시함)

```

INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러-게시물에 대한 댓글 등록-URI 추출 bno: 458753
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러-게시물에 대한 댓글 등록-myReply.getBno: 458753
INFO : com.spring5213.mypro00.controller.MyReplyController - 댓글-컨트롤러-게시물에 대한 댓글 등록-서비스로 전달되는 MyReplyVO: MyReplyVO(bno=458753, rno=0, rcontent=댓글 등록 트랜잭션 테스트 오류가 발생되어 처리가 않습니다, rwriter=user9, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : com.spring5213.mypro00.service.MyReplyServiceImpl - 댓글-서비스-게시물에 대한 댓글 등록 시 처음 전달된 myReplyVO: MyReplyVO(bno=458753, rno=0, rcontent=댓글 등록 트랜잭션 테스트 오류가 발생되어 처리가 않습니다, rwriter=user9, rregDate=null, rmodDate=null, prno=0, lvl=0)
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET breplyCnt = breplyCnt + 1 WHERE bno = 458753 AND bdelFlag =
0 ← 현재의 댓글 개수에 +1 이 처리되어 증가됨.

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET breplyCnt = breplyCnt + 1 WHERE bno = 458753 AND bdelFlag =
0
{executed in 1 msec}
INFO : jdbc.sqlonly - SELECT book_ex.seq_myreply.NEXTVAL FROM dual

INFO : jdbc.sqltiming - SELECT book_ex.seq_myreply.NEXTVAL FROM dual
{executed in 1 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|28    |
|-----|

INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 28, '댓글 등록 트랜잭션 테스트 오류가 발생되어 처리가 않습니다', 'user9', DEFAULT, DEFAULT, 'a')

ERROR: jdbc.audit - 1. PreparedStatement.execute() INSERT INTO book_ex.tbl_myreply(bno, rno, rcontent, rwriter, rregDate, rmodDate, prno) VALUES
(458753, 28, '댓글 등록 트랜잭션 테스트 오류가 발생되어 처리가 않습니다', 'user9', DEFAULT, DEFAULT, 'a')

java.sql.SQLSyntaxErrorException: ORA-01722: invalid number ← 오류 발생, 입력작업 오류로, 댓글 수가 원래의 값으로 변경됨.

...(생략)...

```

→ 실습 정리: MyReplyMapper.xml 파일의 댓글 등록 INSERT문에 'a' → DEFAULT로 다시 수정, 저장 후,

→ 텡캣 서버를 재기동하여, 동일한 댓글 등록 작업을 수행하여, 댓글 등록 후, 댓글 수가 정상적으로 변경되는지 확인합니다.

The screenshots show a web application interface with two main pages:

- Board Detail Page:** Shows a comment form with a count of 18 comments. Below it is a list of comments by user1.
- Comment List Page:** Shows a table of comments with columns: 번호 (Number), 제목 (Title), 작성자 (Author), 작성일 (Created Date), 수정일 (Modified Date), and 조회수 (View Count). The table contains four rows of test data.

### [스스로 해결해야 하는 문제]

☞ 게시물 상세 화면의 댓글 작성 버튼 옆에 표시되는 댓글 개수가 댓글 등록 후에 변화가 없습니다. 왜 변화가 없을까요 ?

[힌트] detail.jsp 파일의 소스를 보면서, 댓글 작성 버튼 옆의 댓글 개수는 언제 값이 정해지는지 생각해봅니다.

그리고, 그것과 댓글 등록의 처리 방법을 다시 고려하면서 왜 댓글 개수가 변화가 없는지 알아봅니다.

☞ 그러면, 댓글 등록 시, 댓글 작성 버튼 옆의 댓글 개수가 현재 브라우저의 댓글 등록 시에 잘 반영되도록 할 수 있을지 방법을 생각해서 직접 구현해 봅니다.

힌트 댓글 등록 후, 다시 목록 정보가 표시될 때, ajax가 전달 받는 데이터를 잘 확인해 보시면,

이미 중요한 것은 모두 구현되었고, 화면에 표시만 하도록 추가 구현하면 되는 것을. . . . 호호호!!!

-- 스프링 REST-API와 Ajax를 이용한 댓글 처리 기능 완료

## [11] 파일 업로드 처리 - 파일 업로드 방식

☞ 게시물의 첨부파일(이미지 파일이나 문서파일 등)을 서버에 전송할 때, 대부분의 경우 <form> 태그나 또는 Ajax를 이용하는 방법이 사용됩니다.

### ○ <form> 태그를 이용하는 방법 : 브라우저의 제한이 없어야 하는 경우에 사용됩니다.

- 일반적으로 페이지 이동과 동시에 첨부파일이 업로드 됩니다.
- <iframe>을 이용해서 화면의 이동 없이 첨부파일을 처리할 수도 있습니다.

### ○ Ajax를 이용하는 방법 : 첨부파일을 별도로 처리하는 방식

- <input type = 'file'>을 이용하고, Ajax로 처리됩니다.
- HTML5의 Drag And Drop 기능이나 jQuery 라이브러리를 이용해서 처리하는 방식으로 구현됩니다.

☞ 업로드 파일에 대한 브라우저 상에서의 구현하는 방식은 다양하지만, 서버상에서는 유사한 방식으로 처리됩니다.

즉, 전송된 업로드 파일을 WAS 서버에 저장한 후, 결과를 HTML 또는 JSON으로 브라우저에게 전송하도록 구현됩니다.

☞ 서버에서 첨부파일을 처리할 때 다음과 같은 API 들이 주로 사용됩니다.

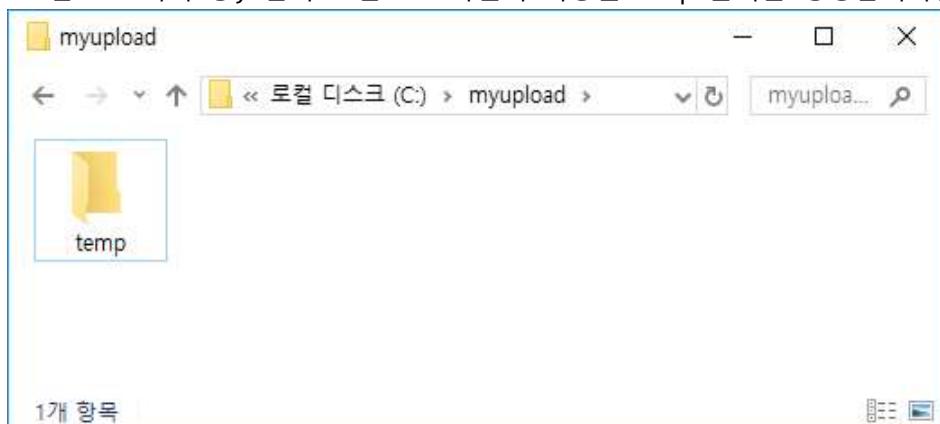
- cos.jar : 2002년도 이후에 개발이 종료되었으므로, 더 이상 사용하는 것을 권장하지 않습니다.
- commons-fileupload : 가장 일반적으로 많이 활용되고, 서블릿 스펙이 3.0 이전인 경우에도 사용될 수 있습니다.
- 서블릿 3.0 이상 : 서블릿 3.0 버전 이후부터 파일 업로드 처리를 위한 API를 내장하고 있습니다.

☞ 위의 방식에서 commons-fileupload가 일반적으로 많이 사용 되지만, 예제에서는 서블릿 4.0에 내장된 API를 이용하여 파일업로드를 구현합니다.

### 11-1. 파일 업로드 처리를 위한 프로젝트 구성

#### (1) 업로드 파일 저장 폴더 생성

☞ 첨부파일은 실제 WAS-서버(예, 톰캣)가 동작하는 시스템의 저장장치 공간의 폴더에 업로드 시켜야 하므로, Windows 운영체제 기반의 개발환경에서는 C: 드라이브 밑에 myupload 폴더를 생성한 후, myupload 폴더로 이동하여 업로드 처리 중, 임시로 업로드 파일이 저장될 temp 폴더를 생성합니다(아래 그림 참고).



## (2) 의존성 라이브러리 설치 구성

☞ XML 설정파일을 사용하는 Spring5 환경에서, servlet 4.0 을 이용하는 파일 업로드를 구현하기 위하여 pom.xml 파일에 다음의 의존성 라이브러리를 포함시킵니다.

☞ 앞의 [프로젝트 준비] 파트을 수행한 경우, 이미 구성되어 있으므로, pom.xml 파일을 오픈한 후, 내용만 확인합니다.

→ mypro00 프로젝트/pom.xml 파일을 코드 작성 뷰에 오픈

→ 다음의 빨간색 코드를 추가(이미 추가되어 있음), 아래처럼 기존 servlet-api 와 jsp-api 를 주석처리

```
<!-- 새로추가: javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
<!-- 새로추가: javax.servlet.jsp-api -->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.3</version>
    <scope>provided</scope>
</dependency>
<!-- Servlet -->
<!--기존 servlet-api 라이브러리는 주석처리 함 --> <여기서부터 아래 남색 내용 주석처리
<!--
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>
-->
<!-- jsp-api 버전 변경: 2.1 -> 2.2 -->
<!--
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.2</version>
    <scope>provided</scope>
</dependency>
-->
<!-- JSON관련, jackson-databind Libary 추가-->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<!-- JSON관련, jackson-databind -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.12.2</version>
</dependency>
```

```

<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20210307</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.6</version>
</dependency>
<!-- JSON관련, jackson-databind Library END-->

<!-- 이미지에 대한 Thumbnail 처리 라이브러리 -->
<!-- https://mvnrepository.com/artifact/net.coobird/thumbnailator -->
<dependency>
    <groupId>net.coobird</groupId>
    <artifactId>thumbnailator</artifactId>
    <version>0.4.14</version>
</dependency>
<!-- END - 이미지에 대한 Thumbnail 처리 라이브러리 -->

```

### (3) 프로젝트에 대한 4.0 servlet 모듈 버전 설정(이미 설정되어 있음, 확인만 함)

- ☞ 다음 섹션에서 web.xml 파일에 설정되는 web-app 의 서블릿 버전(4.0)을 프로젝트에 정상적으로 적용시키기 위하여 프로젝트의 서블릿 버전 관련 Dynamic Web Module 설정을 설정하려는 버전값으로 먼저 설정합니다.
  
  
  
- 프로젝트 탐색부에서 mypro00 프로젝트를 마우스 오른쪽 버튼으로 클릭 → 속성(Properties) 클릭 → 프로젝트 속성창 표시됨
- 프로젝트 속성창의 왼쪽에서 Project Facets 항목 클릭 → 오른쪽에서 Dynamic Web Module의 버전을 4.0 으로 변경
- Apply 클릭 → Apply and Close 클릭

### (4) web.xml 파일 수정: XML 스키마 정의(web-app 버전 포함) 변경 및 첨부파일에 대한 설정 추가

- ☞ web-app 요소의 version 속성 값과 XML 스키마 정의 파일 버전을 변경하고, 스프링의 Dispatcher Servlet에 파일 업로드와 관련된 설정을 추가합니다.
  - Spring Legacy Project 의 기본 서블릿 버전이 2.5 버전이므로 이를 4.0 버전으로 수정하고, XML 스키마 정의 파일도 web-app\_4\_0.xsd 로 수정
  - 스프링의 Dispatcher Servlet의 구성설정 부분에 multipart-config 요소를 이용하여 업로드 파일의 최대크기등을 설정합니다.
- ☞ 이 작업이 수행되어야지만, 서블릿 내장 파일 업로드 API 를 정상적으로 사용할 수 있습니다.

→ src/main/webapp/WEB-INF/web.xml 파일을 코드 작성부에 오픈하고, 다음의 빨간색 코드를 추가 또는 수정

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 기존 web-app 태그 전체 내용을 주석 처리 후, 아래의 빨간색 코드로 변경.
   xml 정의 스카마 참조 URL 이 변경되었습니다(2021년5월 이 후).
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee https://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
-->

<web-app version="4.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee
  http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_4_0.xsd">

...(생략)...

<!-- Processes application requests -->
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>

  <!-- 파일 업로드 관련 추가 내용 -->
  <multipart-config>
    <location>C:\\\\myupload\\\\temp</location><!-- 임시 업로드 경로 --><!-- 앞에서 생성한 경로를 설정
    <max-file-size>20971520</max-file-size><!-- 20MB: 업로드가 허용되는 한 개 파일의 최대크기 -->
    <max-request-size>41943040</max-request-size><!-- 40MB: 한 번의 업로드 요청에서 허용되는 최대 파일 크기 -->
    <file-size-threshold>20971520</file-size-threshold><!-- 20MB --><!-- 메모리 제한 크기 -->
  </multipart-config>
  <!-- 파일 업로드 관련 추가 내용 끝 -->
</servlet>

<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

☞ <multipart-config> 요소에 설정된 각 하부요소의 기능은 다음과 같습니다.

설정 요소	기능
<file-size-threshold>	하나의 파일을 업로드 시에 사용하도록 허용된 메모리 제한 크기 일반적으로 <max-file-size> 요소의 설정값과 동일하거나 크게 설정합니다.
<max-file-size>	업로드가 허용되는 한 개 파일의 최대 크기
<max-request-size>	한 번의 업로드 요청에서 허용되는 최대 파일크기 업로드 파일이 여러 개일 경우, 업로드 파일들 모두의 크기 합계가 요소에 설정된 크기값을 넘을 수 없음
<location>	임시 업로드 경로를 설정하며, <file-size-threshold>요소에 설정된 크기 보다 업로드 파일의 크기가 큰 경우, 메모리에 있는 먼저 업로드 된 파일의 일부를 설정된 임시 경로에 저장하여 메모리를 비우고 파일의 다음부분을 업로드 처리하면서 업로드를 처리합니다.

(5) 톰캣 9.0 서버의 context.xml 파일 수정: 첨부파일에 대한 설정 추가

☞ 앞에서 스프링의 DispatcherServlet에 설정한 업로드 처리에 허용된 메모리 크기 제한 값(<file-size-threshold>요소)이

톰캣에서 허용한 최대 캐시 크기 범위보다 작거나 같아야 파일 업로드가 정상적으로 처리됩니다.

- 이클립스의 프로젝트 탐색 뷰의 Servers 프로젝트 더블클릭 → 확장됨
- Tomcat v9.0 Server at localhost-config 를 더블클릭 → 확장됨
- context.xml 파일을 코드 작성뷰에 오픈
- 기존 <context> 태그에 다음의 빨간색 코드를 추가합니다.

```
<context allowCasualMultipartParsing="true" path="/" >
    <Resources cachingAllowed="true" cacheMaxSize="41943040" />
```

- ☞ cacheMaxSize 값을, web.xml 의 Dispatcher 서블릿 설정에 추가한 <file-size-threshold> 설정에 지정된 값보다 크게 설정해야 합니다.

#### (6) servlet-context.xml 수정

- ☞ 파일 업로드를 위한 MultipartResolver 빈과 JSON 처리를 위한 메시지 컨버터 빈 등록
- ☞ 스프링에서 업로드 처리는 MultipartResolver라는 빈을 이용하므로 이를 빈으로 생성되도록 등록합니다.

- src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml 파일을 코드 작성뷰에 오픈

- 다음의 코드를 마지막에 있는 </beans:beans> 태그 위에 추가

```
<!-- JSON 처리시 메시지 컨버터 빈: 실습에서는 사용 않함 -->
<!--
<beans:bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">
    <beans:property name="messageConverters">
        <beans:list>
            <beans:bean class="org.springframework.http.converter.StringHttpMessageConverter">
                <beans:property name="supportedMediaTypes">
                    <beans:list>
                        <beans:value>text/html; charset=UTF-8</beans:value>
                        <beans:value>application/json; charset=UTF-8</beans:value>
                    </beans:list>
                </beans:property>
            </beans:bean>
            <beans:bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"/>
        </beans:list>
    </beans:property>
</beans:bean>
-->

<!-- 멀티파티 업로드 구성을 위한 MultiPartResolver 빈 설정 -->
<beans:bean id="multipartResolver"
    class="org.springframework.web.multipart.support.StandardServletMultipartResolver"/>

</beans:beans>
```

- ☞ 첨부파일을 처리하는 StandardServletMultipartResolver 클래스의 빈을 설정할 때, id 로 'multipartResolver'를 지정합니다.

## 11-2. 기본적인 파일 업로드 구현 실습: <form> 방식의 파일 업로드 구현

### (1) 업로드 요청 JSP 페이지(form) 생성

→ src/main/webapp/WEB-INF/views 폴더에 사용자가 업로드 요청을 할 수 있는 fileUploadForm.jsp 페이지를 생성하고 다음의 내용을 작성합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>File Upload Form Page</title>
</head>
<body>
    <!-- 다중파일 업로드 방법1: form 방식의 파일업로드 -->
    <form action="${contextPath}/fileUploadFormAction" method="post" enctype="multipart/form-data">
        <input id="inputFile" type="file" name="uploadFiles" multiple>
        <button>Submit</button>
    </form>
</body>
</html>
```

☞ <form> 태그 내에, 전송될 파일을 입력할 <input type = 'file'> 태그를 추가합니다.

☞ 파일 전송을 위하여 form 태그에 다음의 속성을 명시된 값으로 반드시 설정합니다.

- enctype 속성은 반드시 'multipart/form-data' 값으로 지정합니다.
- method 속성은 반드시 'post' 값으로 지정합니다.

☞ 여러 개의 파일을 전송하도록 구현하려면, input 태그에 multiple 속성을 지정합니다.

- multiple 속성은 HTML5에서 새로 추가된 불리언(boolean) 속성이며, 해당 속성을 명시하지 않으면 자동으로 false 값을 가지게 되며, 명시하면 true 값을 가지게 됩니다. 이 속성은 브라우저의 버전에 따라 지원 여부가 다릅니다.

[multiple 속성을 지원하는 브라우저 및 버전]

브라우저명						
지원버전	6.0 이후	지원함	10.0 이후	3.6 이후	5.0 이후	11.0 이후

## (2) 업로드 결과를 표시하는 JSP 페이지(form) 생성

→ src/main/webapp/WEB-INF/views 폴더에 사용자 브라우저에 업로드 결과를 표시하는 fileUploadFormAction.jsp 페이지를 생성하고 다음의 내용을 작성합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>File Upload Form Result Page</title>
</head>
<body>
    <h2>파일업로드를 성공했습니다.</h2>
</body>
</html>
```

## (3) 파일 업로드 요청을 처리할 컨트롤러 구성

→ src/main/java/com.spring5213.mypro00.common 패키지에 fileupload 패키지를 생성한 후, 생성된 fileupload 패키지에 FileUploadControllerForm.java 컨트롤러 클래스를 생성하고 아래의 내용을 작성합니다.

```
package com.spring5213.mypro00.common.fileupload;

import java.io.File;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.multipart.MultipartFile;

@Controller
public class FileUploadControllerForm {
    private static final Logger logger = LoggerFactory.getLogger(FileUploadControllerForm.class);

    //저장경로 설정 (Windows 환경이므로 경로구분자를 \\로 지정)
    private String uploadFileRepoDir = "C:\\\\myupload" ;

    //다중파일 업로드 방법1: form 방식의 파일업로드
    //파일 업로드 요청 JSP 페이지 호출
    @GetMapping("/fileUploadForm")
    public void callFileUploadForm() {
        logger.info("upload Form =====");
    }

    //다중파일 업로드 방법1: form 방식의 파일업로드
    //Model이용, 업로드 파일 저장
    @PostMapping("/fileUploadFormAction")
    public void fileUploadActionPost(MultipartFile[] uploadFiles, Model model ) {
        logger.info("====FileUpload With Form =====");

        for(MultipartFile multipartUploadFile : uploadFiles) {
```

```

Logger.info("=====");
Logger.info("Upload File Name: "+ multipartUploadFile.getOriginalFilename());
Logger.info("Upload File Size: "+ multipartUploadFile.getSize());

//업로드 파일의 리소스(저장폴더와 파일이름)가 설정된 File 객체 생성
File saveUploadFile = new File(uploadFileRepoDir, multipartUploadFile.getOriginalFilename());

try {
    //서버에 파일객체를 이용하여 업로드 파일 저장
    multipartUploadFile.transferTo(saveUploadFile);
} catch (Exception e) {
    Logger.error(e.getMessage());
}
} // End-for
}
}

```

- ☞ transferTo() 메소드에 의해 파라미터로 지정된 java.io.File 클래스 객체의 설정대로 업로드 된 파일이 저장됩니다.
- ☞ 파일 업로드 구현 시에, 스프링-MVC 에는 제공하는 MultipartFile 클래스를 이용해서 파일업로드 처리를 쉽게 구현할 수 있습니다.
- ☞ 업로드 요청 JSP 에서 file 유형 input 요소의 name 속성에 지정한 uploadFiles 값과 fileUploadActionPost 메서드에서 파일 정보를 전달받는 매개변수 이름을, 동일하게 uploadFiles 로 지정합니다.
- ☞ 요청 시, 여러 개의 파일을 업로드 할 수 있도록 multiple 속성을 명시했으므로, 이를 처리하기 위하여 배열형태로 uploadfiles 매개변수의 타입을 지정합니다.

☞ 다음은 스프링의 MultipartFile 클래스가 제공하는 메서드들에 대한 간단한 기능요약입니다.

반환타입 메소드이름	기능요약
String getName()	파라미터의 이름(<input> 태그의 이름) 반환
String getOriginalFileName()	업로드 되는 파일의 이름
boolean isEmpty()	파일이 존재하지 않는 경우 true
long getSize()	업로드 되는 파일의 크기
byte[] getBytes()	byte[]로 파일 데이터 반환
InputStream getInputStream()	파일데이터와 연결된 InputStream 을 반환
transferTo(File file)	파일 저장

#### (4) FileUploadControllerForm.java 클래스 자동 빈 구성 설정

→ src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml 파일에 다음의 설정을 추가합니다.

```

...(생략)...
<context:component-scan base-package="com.spring5213.mypro00.service" />
<context:component-scan base-package="com.spring5213.mypro00.controller" />
<context:component-scan base-package="com.spring5213.mypro00.common.fileupload" /> ←추가
...(생략)...

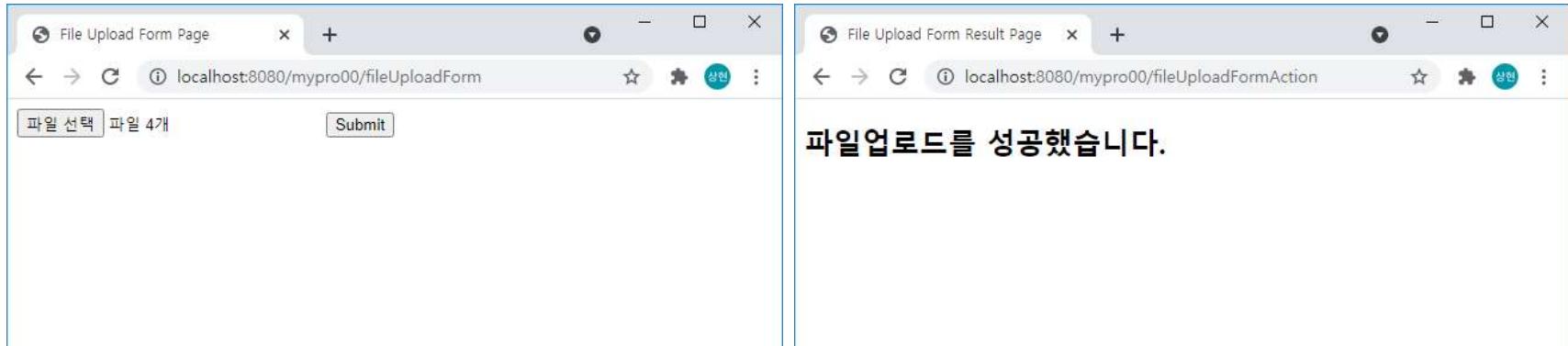
```

[테스트: Form 을 통한 파일 업로드 테스트]

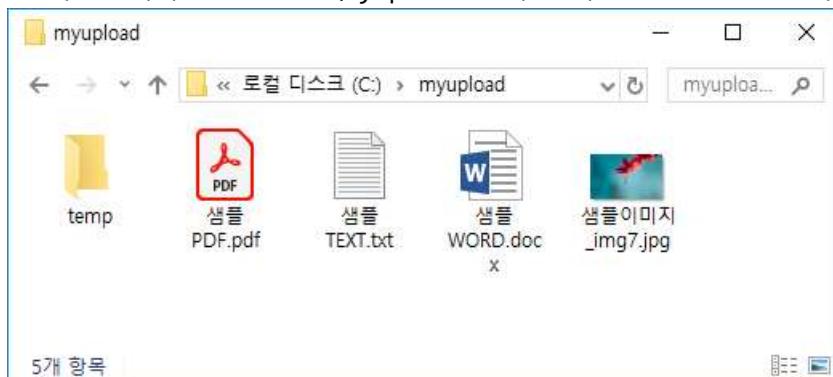
→ 톰캣 서버를 기동

→ 크롬-브라우저에서 <http://localhost:8080/mypro00/fileUploadForm> URL로 접속.

→ 파일 선택 버튼 클릭 → 업로드할 파일을 선택 → Submit 클릭 → 응답 페이지가 표시됨.



→ 파일탐색기 실행 → C:\myupload 폴더로 이동 → 업로드 된 파일 확인



☞ 다음은 업로드 시에 이클립스 콘솔에 표시된 내용입니다.

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - upload Form =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - ===FileUpload With Form ====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 282604
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: 샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 436
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: 샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 23952
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 158112
```

→ 동일한 업로드 테스트를 인터넷 익스플로러와 Edge 브라우저에서도 수행한 후, → 이클립스 콘솔에 기록된 로그를 확인합니다.

☞ 다음은 인터넷 익스플로러에서 동일한 파일 업로드 테스트 시, 이클립스 콘솔에 표시된 내용입니다(오류 발생).

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - upload Form =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - ===FileUpload With Form ====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: D:\Temp\샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 282604
ERROR: com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - java.io.FileNotFoundException: C:\myupload\D:\Temp\샘플PDF.pdf (파일 이름, 디렉터리 이름 또는 볼륨 레이블 구문이 잘못되었습니다)
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: D:\Temp\샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 436
ERROR: com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - java.io.FileNotFoundException: C:\myupload\D:\Temp\샘플TEXT.txt (파일 이름, 디렉터리 이름 또는 볼륨 레이블 구문이 잘못되었습니다)
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: D:\Temp\샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 23952
ERROR: com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - java.io.FileNotFoundException: C:\myupload\D:\Temp\샘플WORD.docx (파일 이름, 디렉터리 이름 또는 볼륨 레이블 구문이 잘못되었습니다)
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: D:\Temp\샘플이미지_img7.jpg
```

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 158112
ERROR: com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - java.io.FileNotFoundException: C:\myupload\D:\Temp\샘플이미지_img7.jpg
(파일 이름, 디렉터리 이름 또는 볼륨 레이블 구문이 잘못되었습니다)
```

- ☞ 엣지 브라우저의 경우, 버전에 따라 위와 동일한 오류가 발생되기도 하고(42.17133.1.0), 정상적으로 업로드가 수행(91.0.864.59)되기도 합니다.
- ☞ 인터넷 익스플로러나 Edge 브라우저(일부 버전)에서 파일업로드 시, 파일이름에 원본 파일의 경로가 포함되어 있으므로, 실제로 파일업로드가 수행되지 않습니다.

→ 이를 수정하기 위하여 다음의 지시대로 `FileUploadControllerForm.java` 클래스 파일의 코드를 수정합니다(빨간색 코드).

```
//업로드 파일의 리소스(저장폴더와 파일이름) 설정
//File saveUploadFile = new File(uploadFileRepoDir, multipartUploadFile.getOriginalFilename()); ← 주석처리

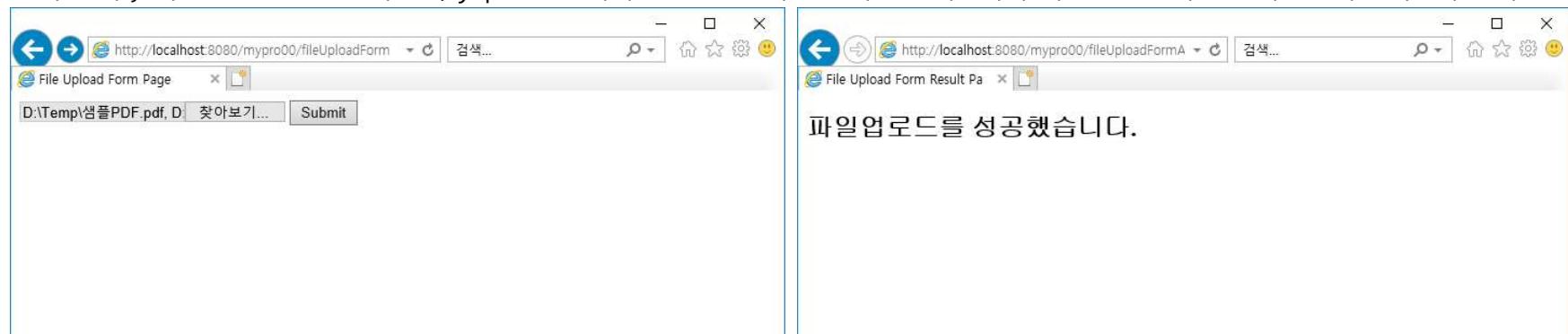
//위의 코드를 아래의 코드로 대체합니다.
String strUploadFileName = multipartUploadFile.getOriginalFilename();

//[Edge, IE 오류 해결] multipartUploadFile.getOriginalFilename()에서 업로드 파일이름만 추출
//파일이름만 있는 경우, 파일이름만 추출됨
strUploadFileName = strUploadFileName.substring(strUploadFileName.lastIndexOf("\\") + 1);
logger.info("수정된 파일이름(strUploadFileName): " + strUploadFileName);

//업로드 정보(저장폴더와 파일이름 문자열)의 파일객체 생성
File saveUploadFile = new File(uploadFileRepoDir, strUploadFileName);
```

→ 수정 및 저장 후 → 인터넷 익스플로러와 Edge 브라우저에서 파일업로드 테스트를 다시 수행.

→ 수정 후, 이클립스 콘솔 로그와 C:\myupload 폴더에 업로드 된 파일을 확인을 확인하여 파일 업로드가 정상적으로 수행되는지를 확인.



☞ 다음은 코드 수정 후, 인터넷 익스플로러에서 테스트 시 이클립스 콘솔에 표시된 로그 내용입니다.

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - upload Form =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====FileUpload With Form =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: D:\Temp\샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 282604
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - 수정된 파일이름(strUploadFileName): 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: D:\Temp\샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 436
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - 수정된 파일이름(strUploadFileName): 샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: D:\Temp\샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 23952
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - 수정된 파일이름(strUploadFileName): 샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Name: D:\Temp\샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - Upload File Size: 158112
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerForm - 수정된 파일이름(strUploadFileName): 샘플이미지_img7.jpg
```

### 11-3. 기본적인 파일 업로드 구현 실습: Ajax를 이용한 파일 업로드 구현

#### (1) 업로드 요청 JSP 페이지 생성

→ src/main/webapp/WEB-INF/views 폴더에 사용자가 업로드 요청을 할 수 있는 fileUploadAjax.jsp 페이지를 생성하고 다음의 내용 전체를 작성합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>File Upload Ajax Page</title>
</head>
<body>
    <h1>Upload With Ajax</h1>
    <div class="uploadDiv"><%-- form0 없습니다 --%>
        <input id="inputFile" type="file" name="uploadFiles" multiple><br>
    </div>
    <button id="btnFileUpload">File Upload With Ajax</button>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script><%-- JQuery 라이브러리 --%>
<script>
//파일업로드 처리
$("#btnFileUpload").on("click", function(e) {
    //FormData() Ajax 파일 전송 시에 사용되는 Web API 클래스의 생성자
    var formData = new FormData();

    //uploadFiles 이름의 file 유형 input 요소를 변수에 저장
    var inputFiles = $("input[name='uploadFiles']");

    //inputFiles에 저장된 파일들을 files 변수에 저장.
    //inputFiles[0]은 첫번째 input 요소를 의미(input 요소가 하나만 있더라도 [0]을 명시해야 함).
    var files = inputFiles[0].files;
    console.log(files);

    //formdata 객체에 파일추가
    for(var i = 0; i < files.length ; i++) {

        //uploadFiles 파라미터로 file 정보 추가
        formData.append("uploadFiles", files[i]);
    }

    //url 키에 명시된 주소의 컨트롤러에게 formData 객체를 POST 방식으로 전송.
    $.ajax({
        url: '${contextPath}/fileUploadAjaxAction',
        processData: false, //contentType에 설정된 형식으로 data를 처리하지 않음.
        contentType: false, //contentType에 MIME 타입을 지정하지 않음.
        data: formData,
        type: 'post',
        success: function(uploadResult){
            alert("첨부파일의 업로드가 정상적으로 완료되었습니다...");}
    })
});</script>
</body>
</html>
```

☞ Ajax 를 이용하여 파일 업로드를 처리하는 경우에는, form 요소 대신, Web API 로 제공되는 FormData 라는 객체를 이용합니다.

☞ 브라우저의 버전에 따라 FormData 객체의 지원 여부가 다르므로, 이를 확인할 필요가 있습니다.

다음은 FormData 객체를 지원하는 대표적인 브라우저들의 버전입니다.

브라우저명						
지원버전	7 이후	12 이후	10.0 이후	4 이후	5.0 이후	12.0 이후

☞ 인터넷 익스플로러는 formData() 생성자와 append() 메서드만 지원하며, 나머지 메소드들은 지원하지 않습니다.

☞ FormData 클래스에 대한 자세한 사항은 <https://developer.mozilla.org/ko/docs/Web/API/FormData> 의 내용을 참고합니다.

(주의) 위의 mozilla 문서 상에 FormData 를 인터페이스라고 명시되어 있지만, 이는 자바의 인터페이스를 의미하지 않습니다.

☞ form 요소는 URL 을 통해 처리 후, JSP 페이지 호출에 의한 페이지 전환이 발생됩니다.

FormData 는 페이지 전환 없이 폼 데이터만을 제출하고 싶을 때 form 대신 사용됩니다.

☞ FormData 는 주로 Ajax 를 이용하는 파일 업로드에서 필요한 파라미터를 서버로 전송하기 위하여 사용됩니다.

즉, FormData() 생성자를 이용하여 객체를 생성한 후, append 메소드를 이용하여, input 요소로 입력된 값이나

업로드 되는 파일을 FormData 객체에 추가합니다. 이러한 FormData 객체가 Ajax 를 통해 서버로 전송되도록 구현합니다.

☞ 위의 실습 코드에서 Ajax 코드에서 FormData 객체를 이용하여 파일을 서버로 전송 시, 다음의 사항을 주의합니다.

○ processData 키와 contentType 키는 반드시 'false'로 지정해야 합니다.

○ 전송 방식은 반드시 POST 전송방식을 사용해야만 합니다.

- contentType 을 False 로 지정하면, 전송타입을 설정하지 않겠다는 의미입니다.

- processData 키를 False 로 지정하면, contentType 에 지정된 전송형식으로의 데이터 변환처리를 수행하지 않도록 설정됩니다.

## (2) 업로드 결과를 표시하는 JSP 페이지(form) 생성

☞ 테스트에서 Ajax 로 처리되기 때문에 이 JSP 페이지는 웹 브라우저로 전송되지 않습니다.

단 서버에서 파일 업로드를 처리하는 REST API 컨트롤러가 아니므로, 이 JSP 페이지가 없으면, HTTP 상태 404 오류(찾을 수 없음)가 웹브라우저에 전달되므로 이를 방지하기 위하여 생성합니다.

→ src/main/webapp/WEB-INF/views 폴더에 사용자 브라우저에 업로드 결과를 표시하는 fileUploadAjaxAction.jsp 페이지를 생성하고

다음의 내용을 작성합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
```

```

<title>File Upload Ajax Result Page</title>
</head>
<body>
    <h2>Ajax에 의한 파일업로드를 성공했습니다.</h2>
</body>
</html>

```

### (3) 파일 업로드 요청을 처리할 컨트롤러 구성

→ src/main/java/com.spring5213.mypro00.common.fileupload 패키지에 FileUploadControllerAjax.java 컨트롤러 클래스를 생성하고 아래의 내용 전체를 작성합니다

```

package com.spring5213.mypro00.common.fileupload;

import java.io.File;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.multipart.MultipartFile;

@Controller
public class FileUploadControllerAjax {

    private static final Logger logger = LoggerFactory.getLogger(FileUploadControllerAjax.class);

    //저장경로 (Windows 환경이므로 경로구분자를 \\로 지정)
    private String uploadFileRepoDir = "C:\\\\myupload" ;

    //파일 업로드 요청 JSP 페이지 호출
    @GetMapping("/fileUploadAjax")
    public void callFileUploadAjax() {
        logger.info("upload Ajax =====");
    }

    //업로드 요청 파일-저장 및 결과 메시지 전송
    @PostMapping("/fileUploadAjaxAction")
    public void fileUploadActionPost(MultipartFile[] uploadFiles) { //Ajax사용 시 Model 필요없음
        logger.info("====FileUpload With Ajax =====");

        for(MultipartFile multipartUploadFile : uploadFiles) {

            logger.info("=====");
            logger.info("Upload File Name: " + multipartUploadFile.getOriginalFilename());
            logger.info("Upload File Size: " + multipartUploadFile.getSize());

            //업로드파일이름 원본문자열
            String strUploadFileName = multipartUploadFile.getOriginalFilename();

            //#[Edge, IE 오류 해결] multipartUploadFile.getOriginalFilename()에서 업로드 파일이름만 추출
            //파일이름만 있는 경우, 파일이름만 추출됨
            strUploadFileName = strUploadFileName.substring(strUploadFileName.lastIndexOf("\\") + 1);
            logger.info("수정된 파일이름(strUploadFileName): " + strUploadFileName);

            //업로드 정보(저장폴더와 파일이름 문자열)의 파일객체 생성
            File saveUploadFile = new File(uploadFileRepoDir, strUploadFileName);
            logger.info("저장시 사용되는 파일이름(saveUploadFile, 경로포함): " + saveUploadFile);

            try {
                //서버에 파일객체를 이용하여 업로드 파일 저장
                multipartUploadFile.transferTo(saveUploadFile);
            } catch (Exception e) {

```

```

        logger.error(e.getMessage());
    }
} // End-for
}
}

```

- ☞ form에서와 동일하게 MultipartFile[] 타입의 매개변수를 이용해서 업로드 파일을 처리하여 저장합니다.
- ☞ form을 사용하지 않고, Ajax 통해 파일 데이터만 전송되므로, Model 타입의 파라미터가 필요없습니다.
- ☞ transferTo() 메소드에 의해 파라미터로 지정된 java.io.File 클래스 객체의 설정대로 업로드 된 파일이 저장됩니다.
- ☞ 파일 업로드 시에, 스프링-MVC에는 제공하는 MultipartFile 클래스를 이용해서 파일업로드 처리를 쉽게 구현할 수 있습니다.
- ☞ 업로드 요청 JSP에서 file 유형 input 요소의 name 속성에 지정한 uploadFiles 값과 fileUploadActionPost 메서드에서 파일 정보를 전달받는 매개변수 이름을, 동일하게 uploadFiles로 지정합니다.
- ☞ 요청 시, 여러 개의 파일을 업로드 할 수 있도록 multiple 속성을 명시했으므로, 이를 처리하기 위하여 배열형태로 uploadfiles 매개변수의 타입을 지정합니다.
- ☞ 다음은 스프링의 MultipartFile 클래스가 제공하는 메서드들에 대한 간단한 기능요약입니다.

반환타입 메소드이름	기능요약
String getName()	파라미터의 이름(<input> 태그의 이름) 반환
String getOriginalFileName()	업로드 되는 파일의 이름
boolean isEmpty()	파일이 존재하지 않는 경우 true
long getSize()	업로드 되는 파일의 크기
byte[] getBytes()	byte[]로 파일 데이터 반환
InputStream getInputStream()	파일데이터와 연결된 InputStream을 반환
transferTo(File file)	파일 저장

#### (4) FileUploadControllerAjax.java 클래스 자동 빈 구성 설정(이미 앞의 실습으로 구성됨, 확인만)

→ src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml 파일에 다음의 설정을 추가합니다.

```

...(생략)...
<context:component-scan base-package="com.spring5213.mypro00.service" />
<context:component-scan base-package="com.spring5213.mypro00.controller" />
<context:component-scan base-package="com.spring5213.mypro00.common.fileupload" /> ←추가되어야 합니다.
...(생략)...

```

#### [테스트: Ajax를 통한 파일 업로드 테스트]

- ☞ 문서에서는 크롬 브라우저, 엣지 브라우저, 인터넷 익스플로러 브라우저에서 테스트를 수행합니다.  
이 외의 브라우저에서 테스트를 수행해도 괜찮습니다.

☞ 개발자 도구(F12 키)를 오픈 시켜놓고 웹 브라우저 상에서의 오류를 확인하면서, 테스트 합니다.

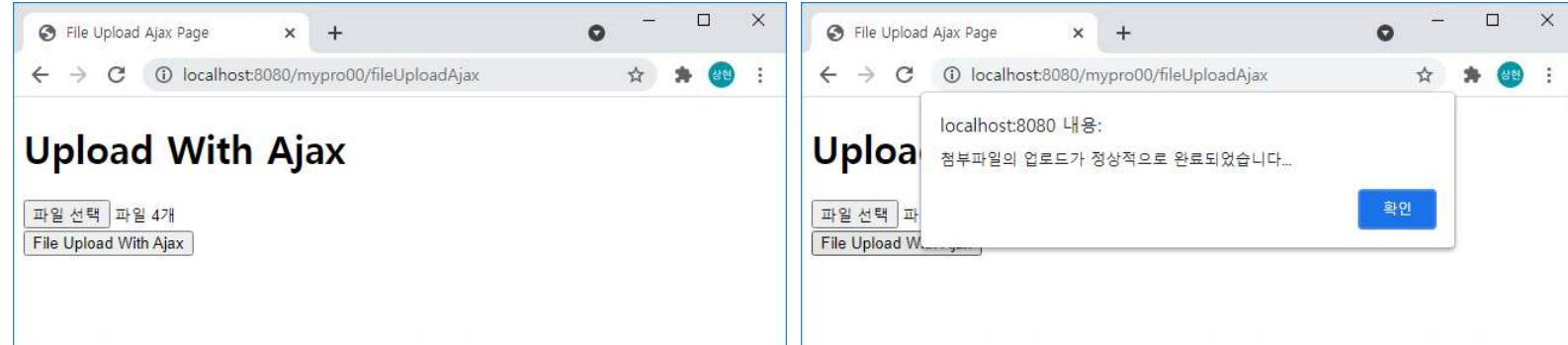
☞ 테스트 결과는 크롬 브라우저의 것만 캡쳐했습니다.

→ 톰캣 서버 기동

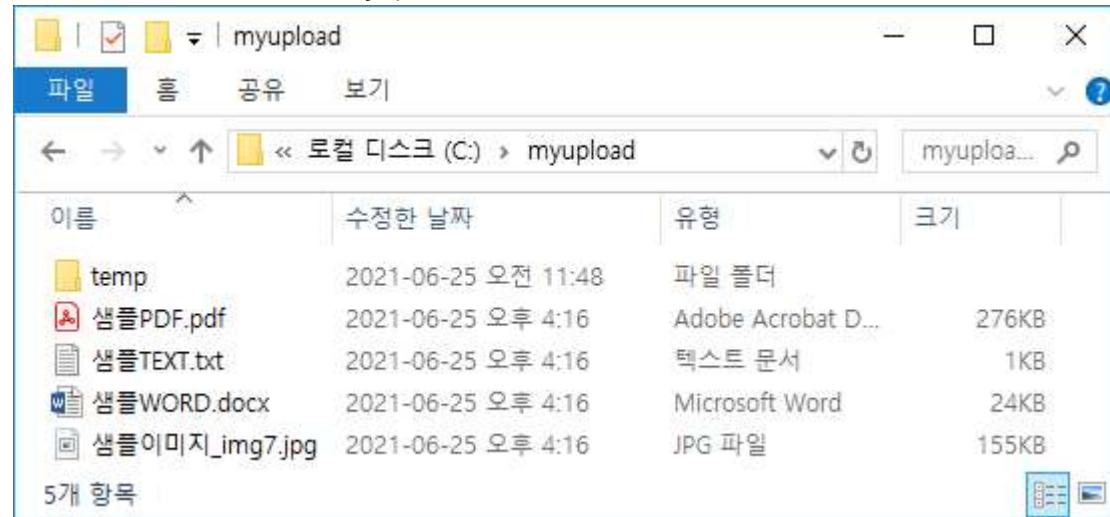
→ 브라우저에서 <http://localhost:8080/mypro00/fileUploadAjax> URL로 접속.

→ 파일 선택 버튼 클릭 → 업로드할 파일을 선택 → File Upload With Ajax 버튼 클릭

→ 업로드 후, 알림창 표시, 확인 클릭



→ 파일탐색기 실행 → C:\myupload 폴더로 이동 → 업로드 된 파일 확인



☞ 다음은 업로드 시에 이를립스 콘솔에 표시된 내용입니다.

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - ===FileUpload With Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 282604
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함): C:\myupload\샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 436
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 23952
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 158112
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\샘플이미지_img7.jpg
```

☞ 다음은 File Upload With Ajax 버튼 클릭 시, 개발자 도구의 Console 창에 표시된 로그입니다.

The screenshot shows the Google Chrome DevTools Console tab. The left sidebar displays various log categories: 3 messages, 2 user messages, No errors, No warnings, 2 info, fileUploadAjax 2, and 1 verbose. The main area shows a log entry under the fileUploadAjax:37 label:

```
fileUploadAjax:37
FileList {0: File, 1: File, 2: File, 3: File, length: 4}
  0: File
    lastModified: 1606113308406
    lastModifiedDate: Mon Nov 23 2020 15:35:08 GMT+0900 (대한민국 ...)
    name: "샘플PDF.pdf"
    size: 282604
    type: "application/pdf"
    webkitRelativePath: ""
  1: File
    lastModified: 1618746184055
    lastModifiedDate: Sun Apr 18 2021 20:43:04 GMT+0900 (대한민국 ...)
    name: "샘플TEXT.txt"
    size: 436
    type: "text/plain"
    webkitRelativePath: ""
  2: File
    lastModified: 1607080428965
    lastModifiedDate: Fri Dec 04 2020 20:13:48 GMT+0900 (대한민국 ...)
    name: "샘플WORD.docx"
    size: 23952
    type: "application/vnd.openxmlformats-officedocument.wordproc..."
    webkitRelativePath: ""
  3: File
    lastModified: 1521919011571
    lastModifiedDate: Sun Mar 25 2018 04:16:51 GMT+0900 (대한민국 ...)
    name: "샘플이미지_img7.jpg"
    size: 158112
    type: "image/jpeg"
    webkitRelativePath: ""
length: 4
__proto__: FileList
```

## 11-4. 파일 업로드 구현 실습: 파일 업로드 상세 처리

☞ 파일 업로드(첨부파일을 서버에 전송하고 저장)를 구현할 때, 상황에 따라 다음의 사항들을 고려해야 합니다.

- 기존 파일과 동일한 이름의 파일이 업로드 되었을 때, 기존 파일이 사라지는 상황 고려
- 이미지파일의 경우에 썸네일 이미지의 생성여부를 결정
- 이미지파일과 일반파일의 처리 구분(일반파일은 다운로드, 이미지파일은 표시를 구분해서 요청 페이지에서 처리할 상황)
- 첨부파일을 통한 서버 공격에 대비하여, 업로드 파일에 대한 확장자 제한여부

☞ Ajax 를 이용한 파일업로드 소스를 이용하여 위의 고려 사항들을 구현합니다.

### (1) 파일의 "확장자 및 최대 크기 제한" 처리

☞ 최근 포털에서도 특정한 확장자가 제외되도록 파일 업로드를 제한하는 경우가 많습니다. 이는 첨부파일을 이용한 웹 공격을 막기 위해 행해지는 조치입니다. 실습에서는 첨부파일의 확장자가 'exe, sh, zip, alz' 인 경우에는 업로드를 제한하고, 특정 크기 이상의 파일은 업로드할 수 없도록 제한하는 처리를 구현하겠습니다.

☞ 업로드 요청 JSP 파일에 다음의 검사를 수행하는 JavaScript 함수를 추가합니다.

- 업로드 파일의 크기가, 최대 허용 크기(1MB)를 넘지 않을 때만 업로드됩니다.
- 업로드 파일의 확장자가 업로드가 허용되지 않는 확장자가 아닐 때만 업로드됩니다.

☞ 업로드 요청 시, 서버에 요청이 전송되기 전에 업로드 파일에 대하여 추가된 함수가 실행되어, 검사에 성공한 경우에만 서버로 파일 업로드 요청이 전송되고, 검사에 실패한 경우에는 서버로 업로드 요청이 전송되지 않도록 구현합니다.

→ src/main/webapp/WEB-INF/views/fileUploadAjax.jsp 페이지를 코드 뷰에 오픈

→ <script> 태그 밑에 checkUploadfile() 함수를 정의하고, #btnFileUpload 아이디의 버튼 클릭 이벤트 처리 jQuery 코드에 checkUploadfile() 함수가 업로드 파일에 대하여 검사를 수행하도록 빨간색 코드를 추가합니다.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script><%-- JQuery 라이브러리 --%>
<script>

//업로드 파일의 확장자 및 최대 파일 크기 검사 함수
function checkUploadfile(fileName, fileSize) {
    // 확장자에 대한 정규식 및 최대 허용크기(1MB) 저장변수
    var maxSizeAllowed = 1048576;
    var regExpForFileExtention = /(.*).(exe|sh|zip|alz)$/i ;

    //최대 허용 크기 제한 검사
    if (fileSize >= maxSizeAllowed) {
        alert("업로드 파일의 제한된 크기(1MB)를 초과했습니다.");
        return false;
    }

    //업로드파일의 확장자 검사:
    if (regExpForFileExtention.test(fileName)) {
        alert("해당 종류(exe/sh/zip/alz)의 파일은 업로드할 수 없습니다.");
        return false;
    }
    return true;
}


```

```

//파일업로드 처리
$("#btnFileUpload").on("click", function(e) {

    //Ajax 파일 전송 시에 사용되는 클래스
    var formData = new FormData();

    //uploadFiles 이름의 file 유형 input 요소를 변수에 저장
    var inputFiles = $("input[name='uploadFiles']");

    //inputFiles(file 유형 input 요소)에 저장된 파일들을 files 변수에 저장.
    var files = inputFiles[0].files;
    console.log(files);

    for(var i = 0; i < files.length ; i++) {

        if (!checkUploadfile(files[i].name, files[i].size)) { //파일 확장자 및 최대크기검사 실행
            return false; //검사를 만족하지 못하면 false를 반환하고 파일업로드 중지
        }

        formData.append("uploadFiles", files[i]); //FormData 객체에 uploadFiles 파라미터로 file 정보 추가
    }

    //url 키에 명시된 주소의 컨트롤러에게 formData 객체를 POST 방식으로 전송.
    $.ajax({
        url: '${contextPath}/fileUploadAjaxAction',
        processData: false,
        contentType: false,
        data: formData,
        type: 'post',
        success: function(uploadResult){
            alert("첨부파일의 업로드가 정상적으로 완료되었습니다....");
        }
    })
});

</script>

```

☞ File Upload With Ajax 버튼을 클릭하여 업로드를 시도하면, 코드의 for 루프에서 파일마다 checkUploadfile() 함수가 호출되어 파일의 확장자와 파일의 크기를 검사합니다.

#### [테스트 - 업로드 파일 확장자 및 파일 크기 제한 테스트]

☞ 다음의 순서대로 스스로 웹브라우저에서 테스트를 수행합니다.

- 톰캣 서버가 중지된 경우, 톰캣 서버를 기동.
- 브라우저에서, <http://localhost:8080/mypro00/fileUploadAjax> URL로 접속
- 파일 선택 버튼 클릭
- 확장자가 zip/alz/sh/exe 인 파일 추가
- File Upload With Ajax 버튼 클릭 시, 알림창이 표시되고, 업로드가 수행되지 않아야 합니다.

- 다시 파일 선택 버튼 클릭
- zip/alz/sh/exe 확장자가 아닌 1MB 이상의 파일크기를 가지는 파일을 추가
- File Upload With Ajax 버튼 클릭 시, 알림창이 표시되고 업로드가 수행되지 않아야 합니다.
  
- 다시 파일 선택 버튼 클릭
- zip/alz/sh/exe 확장자가 아닌 1MB 보다 작은 파일크기를 가지는 파일을 추가

→ File Upload With Ajax 버튼 클릭 시, 정상적인 업로드가 수행되어야 합니다.

☞ 함수의 검사를 통과하지 못하면, 업로드 자체가 수행되지 않으므로, 이클립스 콘솔에 표시되는 내용은 없습니다.

## (2) 톰캣 서버에 저장된 파일과 업로드 파일의 이름이 동일한 경우의 처리

☞ 기본적으로 서버에 저장된 기존 파일과 동일한 이름의 파일이 업로드 되면, 업로드 되는 파일로 교체됩니다.

따라서, 기존 파일 삭제를 방지하기 위하여 업로드 되는 파일들의 이름이 기존 파일과 중복되지 않도록 처리해야 되는 상황이 필요한 경우에는, 업로드 되는 파일에 대하여 **밀리세컨트가 포함된 현재 시간이나 문자화 된 UUID를 파일이름에 포함시켜서** 서버에 저장시키는 방법이 많이 사용됩니다.

☞ 업로드에 의해서 서버에 저장되는 파일이름이 중복되지 않도록 처리기 위하여 문서에서는 UUID를 이용하여 업로드 되는 파일이름에 UUID 가 추가되어 저장되도록 처리하는 방법을 실습합니다.

→ src/main/java/com.spring5213.mypro00.common.fileupload/FileUploadControllerAjax.java 파일을 코드 작성부에 오픈하고 fileUploadActionAjaxPost 메소드에 다음의 빨간색 코드를 추가합니다.

```
import java.util.UUID;      ← Ctrl + Shift + O 키로 자동 추가됨
...(생략)...
//업로드 요청 파일-저장 및 결과 메시지 전송
@PostMapping("/fileUploadAjaxAction")
public void fileUploadActionPost(MultipartFile[] uploadFiles) { //Ajax사용 시 Model 필요없음
    logger.info("=====FileUpload With Ajax =====");

    for(MultipartFile multipartUploadFile : uploadFiles) {
        logger.info("=====");
        logger.info("Upload File Name: " + multipartUploadFile.getOriginalFilename());
        logger.info("Upload File Size: " + multipartUploadFile.getSize());

        //업로드파일이름 원본문자열
        String strUploadFileName = multipartUploadFile.getOriginalFilename();

        //#[Edge, IE 오류 해결] multipartUploadFile.getOriginalFilename()에서 업로드 파일이름만 추출
        strUploadFileName = strUploadFileName.substring(strUploadFileName.lastIndexOf("\\") + 1);
        logger.info("수정된 파일이름(strUploadFileName): " + strUploadFileName);

        //UUID를 이용한 고유한 파일이름 적용
        //랜덤한 UUID값을 가진 UUID 객체 생성
        UUID uuid = UUID.randomUUID(); ← 추가

        //파일이름에 UUID 문자열 추가(파일 확장자 때문에 UUID를 앞에다 추가해야 함
        strUploadFileName = uuid.toString() + "_" + strUploadFileName ; ← 추가
        logger.info("UUID처리된파일이름: "+strUploadFileName); ← 추가

        //업로드 정보(저장폴더와 파일이름 문자열)의 파일객체 생성
        File saveUploadFile = new File(uploadFileRepoDir, strUploadFileName);
        logger.info("저장시 사용되는 파일이름(saveUploadFile, 경로포함): " + saveUploadFile);

        try {
            multipartUploadFile.transferTo(saveUploadFile); //서버에 파일객체를 이용하여 업로드 파일 저장
        } catch (Exception e) {
            logger.error(e.getMessage());
        }
    } // End-for
}
...(생략)...
```

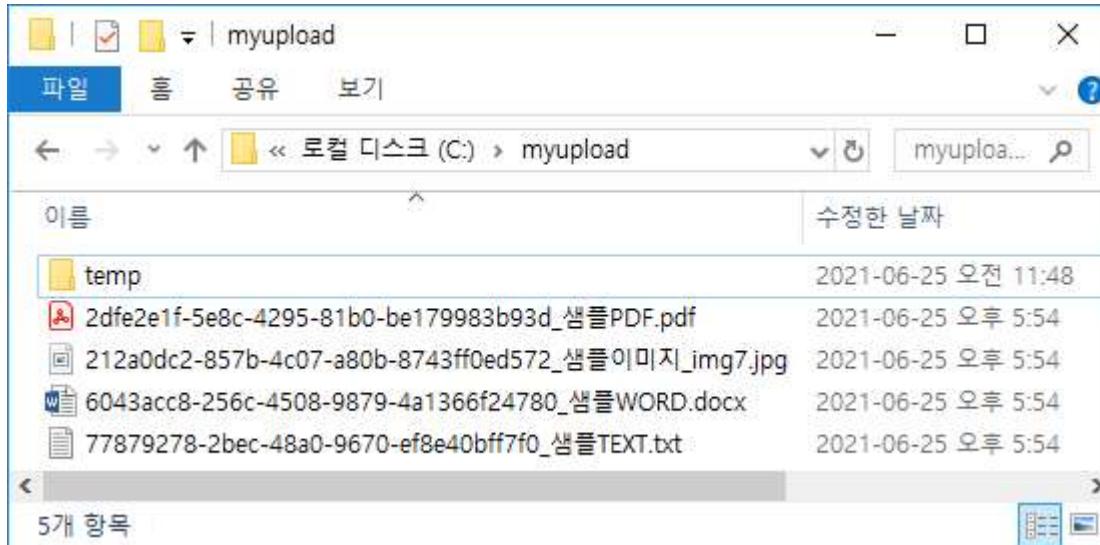
## [테스트 - UUID 적용 파일이름 변경 테스트]

- 톰캣 서버 재 기동
- 크롬브라우저와 Edge 브라우저에서, <http://localhost:8080/mypro00/fileUploadAjax> 페이지를 요청/표시한 후, 파일 업로드를 수행하고, 파일 탐색기에서 C:\myupload 폴더에 업로드된 파일 이름과 이클립스의 콘솔 로그를 확인합니다.

☞ 다음은 톰캣 서버에 업로드 시에 이클립스 콘솔에 표시된 로그입니다.

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - ===FileUpload With Ajax ====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 282604
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 2dfe2e1f-5e8c-4295-81b0-be179983b93d_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함): C:\myupload\2dfe2e1f-5e8c-4295-81b0-be179983b93d_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: 샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 436
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 77879278-2bec-48a0-9670-ef8e40bff7f0_샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함): C:\myupload\77879278-2bec-48a0-9670-ef8e40bff7f0_샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: 샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 23952
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 6043acc8-256c-4508-9879-4a1366f24780_샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함): C:\myupload\6043acc8-256c-4508-9879-4a1366f24780_샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 158112
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 212a0dc2-857b-4c07-a80b-8743ff0ed572_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함): C:\myupload\212a0dc2-857b-4c07-a80b-8743ff0ed572_샘플이미지_img7.jpg
```

☞ 다음은 파일탐색기에서 C:\myupload 폴더에 업로드된 파일이름을 확인 한 것입니다.



### (3) 업로드파일의 저장 경로 구성

- ☞ 컴퓨터의 한 폴더에 너무 많은 파일들이 저장되어 있으면, 해당 폴더에서 수행되는 작업들의 처리 성능이 저하되는 현상이 발생됩니다. 이는 컴퓨터의 전체적인 처리 성능을 저하시키는 원인이 됩니다. 따라서, 톰캣 서버에 저장되는 사용자들의 업로드 파일이 하나의 폴더에 지나치게 많이 저장되지 않도록 구성할 필요가 있습니다.
- ☞ 일반적으로 사용자 ID 별 폴더 또는 년/월/일 구조의 디렉토리 경로를 생성해서 업로드 되는 파일이 서버에 저장되도록 구성합니다.

☞ 문서에서는 년/월/일 구조의 디렉토리 패스를 생성해서 업로드파일이 저장되도록 처리하는 방법을 실습합니다.

☞ 폴더를 생성하는 작업은 java.io.File 클래스의 mkdirs()를 이용하여, 한 번에 서브디렉토리까지 생성되도록 처리합니다.

☞ 서버에 파일이 저장될 때, 디렉토리 구조가 생성되야 하므로, 컨트롤러의 메서드에 다음의 기능을 구현합니다.

○ FileUploadControllerAjax 클래스에 년/월/일 형식의 문자열을 생성하는 getDatefmtPathName() 메서드를 추가

○ getDatefmtPathName() 메서드를 이용하여 폴더를 생성한 후, 파일이 업로드 되도록 fileUploadActionAjaxPost() 메소드를 수정.

→ src/main/java/com.spring5213.mypro00.common.fileupload/FileUploadControllerAjax.java 파일을 코드 작성부에 오픈하고 다음의 빨간색 코드를 추가 및 수정합니다.

```
import java.text.SimpleDateFormat;           ← import는 Ctrl + Shift + O 키로 자동 추가됨
import java.util.Date;
...(생략)...

//Step1- 날짜 형식 경로 문자열 생성 메소드
private String getDatefmtPathName() {    ← 메서드 추가
    //날짜 형식 지정
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy/MM/dd");
    //날짜 생성
    Date date = new Date();
    //날짜 형식이 지정된 날짜문자열(yyyy/MM/dd) 생성
    String strDatefmtPathName = simpleDateFormat.format(date);
    //경로구분자를 운영체제에 맞도록 변경한 문자열을 반환
    return strDatefmtPathName.replace("/", File.separator);
}

//업로드 요청 파일-저장 및 결과 메시지 전송
@PostMapping("/fileUploadAjaxAction")
public void fileUploadActionPost(MultipartFile[] uploadFiles) { //Ajax 사용 시 Model 필요없음
    logger.info("=====FileUpload With Ajax =====");

    //날짜 형식(yyyy/MM/dd)의 폴더구조 생성
    // File(String Parent, String Child)
    // 부모경로(레포지토리 경로) 문자열에 자식경로(날짜형식 경로) 문자열이 더해진 File 객체 생성
    // 날짜형식 경로를 생성하기 위하여 앞에서 생성한 getDatefmtPathName() 메서드 호출
    File fileUploadPath = new File(uploadFileRepoDir, getDatefmtPathName());    ← 추가
    logger.info("upload path: " + fileUploadPath);                                ← 추가

    //경로가 존재하지 않으면 전체 폴더 구조 생성, 기존 폴더 구조중 일부가 있으면 없는 부분을 포함하여 전체를 생성
    if (fileUploadPath.exists() == false) {    ← 추가
        fileUploadPath.mkdirs();            ← 추가
    }

    for(MultipartFile multipartUploadFile : uploadFiles) {
        logger.info("=====");
        logger.info("Upload File Name: " + multipartUploadFile.getOriginalFilename());
        logger.info("Upload File Size: " + multipartUploadFile.getSize());

        //업로드파일이름 원본문자열
        String strUploadFileName = multipartUploadFile.getOriginalFilename();

        // [Edge, IE 오류 해결] 경로 제거된 파일이름만 추출
        strUploadFileName = strUploadFileName.substring(strUploadFileName.lastIndexOf("\\") + 1);
        logger.info("수정된 파일이름(strUploadFileName): " + strUploadFileName);

        //UUID를 이용한 고유한 파일이름 적용
        //랜덤한 UUID값을 가진 UUID 객체 생성
        UUID uuid = UUID.randomUUID();

        //파일이름에 UUID 문자열 추가(파일 확장자 때문에 UUID를 앞에다 추가해야 함)
        strUploadFileName = uuid.toString() + "_" + strUploadFileName ;
        logger.info("UUID처리된파일이름: " + strUploadFileName);

        //업로드 정보(저장폴더와 파일이름 문자열)의 파일객체 생성
        //File saveUploadFile = new File(uploadFileRepoDir, strUploadFileName); ← 주석처리
        //logger.info("저장시 사용되는 파일이름(saveUploadFile, 경로포함): " + saveUploadFile); ← 주석처리
    }
}
```

```

//날짜형식폴더구조가 포함된 File 객체를 사용하도록 수정됨
File saveUploadFile = new File(fileUploadPath, strUploadFileName);           ← 추가
logger.info("저장시 사용되는 파일이름(saveUploadFile, 경로포함): " + saveUploadFile); ← 추가

try {
    //서버에 파일객체를 이용하여 업로드 파일 저장
    multipartUploadFile.transferTo(saveUploadFile);
} catch (Exception e) {
    logger.error(e.getMessage());
}
} // End-for
}
...(생략)...

```

### [테스트 - 날짜형식 업로드 폴더 구조 생성 테스트]

- 템켓 서버 재 기동
- 크롬브라우저와 Edge 브라우저에서, <http://localhost:8080/mypro00/fileUploadAjax> 페이지를 요청/표시한 후, 파일 업로드를 수행합니다.
- 파일 탐색기에서 C:\myupload 폴더에 날짜구조의 폴더가 생성되어 파일이 저장되는지 확인
- 이클립스의 콘솔 로그를 확인합니다.

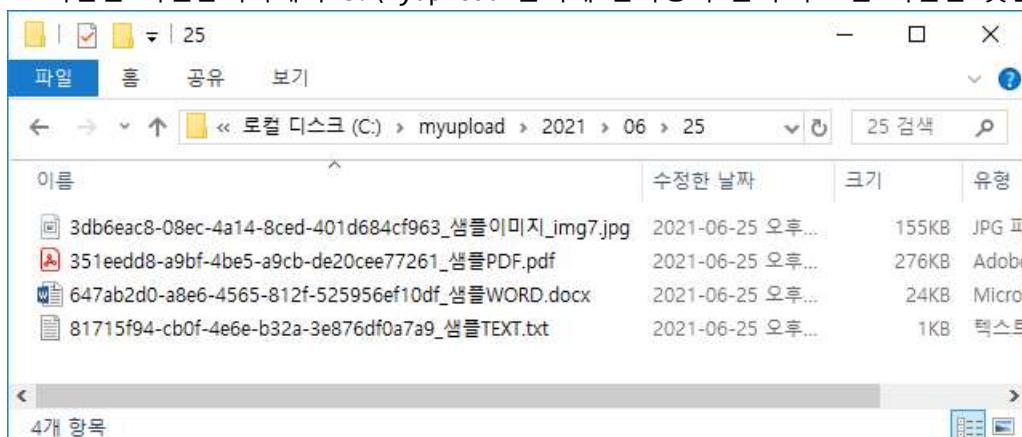
### ☞ 다음은 템켓 서버에 업로드 시에 이클립스 콘솔에 표시된 로그입니다.

```

INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====FileUpload With Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload path: C:\myupload\2021\06\25
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 282604
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 351eedd8-a9bf-4be5-a9cb-de20cee77261_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\25\351eedd8-a9bf-4be5-a9cb-de20cee77261_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: 샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 436
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 81715f94-cb0f-4e6e-b32a-3e876df0a7a9_샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\25\81715f94-cb0f-4e6e-b32a-3e876df0a7a9_샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: 샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 23952
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 647ab2d0-a8e6-4565-812f-525956ef10df_샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\25\647ab2d0-a8e6-4565-812f-525956ef10df_샘플WORD.docx
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 158112
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 3db6eac8-08ec-4a14-8ced-
401d684cf963_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\25\3db6eac8-08ec-4a14-8ced-401d684cf963_샘플이미지_img7.jpg

```

### ☞ 다음은 파일탐색기에서 C:\myupload 폴더에 날짜형식 폴더 구조를 확인한 것입니다.



#### (4) 썸네일 이미지 생성

- ☞ 썸네일 파일은, 이미지파일에 대한 아주 작은용량의 이미지 복사본입니다. 사용자는 이 썸네일을 통해, 이미지파일을 다운로드 받기 전에 간단히 이미지를 확인할 수 있습니다.
- ☞ 웹브라우저나 컴퓨터/모바일의 파일탐색기에서 이미지파일 미리보기 같은 기능을 사용할 때나 또는 일반파일과 이미지 파일을 구분하는 경우에 썸네일이 사용됩니다. 만약 썸네일이 없다면, 이미지 파일(썸네일 보다 용량이 큽니다)을 사용해야 하므로 처리가 늦어집니다.
- ☞ 썸네일을 제작하는 여러 가지 방법 중, JDK1.4 부터 기본 내장된 `ImageIO` 를 이용하는 방법은 이미지 크기나 해상도를 직접 조절하는 작업을 추가적으로 수행해야 하므로 일반적으로, `ImgScalr` 나 `Thumbnailator` 와 같은 별도의 라이브러리를 이용하여 썸네일을 생성합니다.
- ☞ 문서에서는 웹 개발 시 많이 사용되는 `Thumbnailator` 라이브러리를 이용해서 썸네일 이미지를 생성합니다.
- ☞ 사용자가 업로드하는 파일이 이미지 파일인 경우만 썸네일을 생성해서 이미지파일과 같이 서버에 저장되도록 구현하는 것이 일반적이며 따라서, 이 기능은 서버의 컨트롤러에 구현합니다.

##### (4-1) `Thumbnailator` 라이브러리 설치(이미 설치되어 있음)

→ mypro00 프로젝트의 `pom.xml` 파일을 코드 작성부에 오픈 후, 다음을 추가합니다

```
<!-- Thumbnail 이미지 생성 라이브러리 -->
<!-- https://mvnrepository.com/artifact/net.coobird/thumbnailator -->
<dependency>
    <groupId>net.coobird</groupId>
    <artifactId>thumbnailator</artifactId>
    <version>0.4.14</version>
</dependency>
```

##### (4-2) `FileUploadControllerAjax.java` 수정

- ☞ 이미지 파일이 서버에 저장될 때 썸네일도 생성되어 같이 저장되어야 하므로, 컨트롤러의 메서드에 다음의 기능을 구현합니다.
  - `FileUploadControllerAjax` 클래스에 업로드 파일이 이미지 파일인지 검사하는 `checkIsImageForUploadFile()` 메서드를 추가
  - `checkIsImageForUploadFile()` 메서드를 이용하여 업로드 파일을 검사한 후,  
업로드 파일이 이미지 파일일 때, 썸네일이 생성되어 같이 저장되도록 `fileUploadActionAjaxPost()` 메소드를 수정.

→ `src/main/java/com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax.java` 파일을 코드 작성부에 오픈하고  
다음의 빨간색 코드를 `fileUploadActionAjaxPost()` 메소드에 추가합니다.

```
import java.io.FileOutputStream;           ←import는 Ctrl + Shift + O 키로 자동 추가됨
import java.io.IOException;
import java.nio.file.Files;
import net.coobird.thumbnailator.Thumbnailator;

...(생략)...

//썸네일 이미지 생성
//Step1: 업로드파일에 대한 이미지 파일여부 검사 메소드
```

```

private boolean checkIsImageForUploadFile(File uploadFile) {      ← 메서드 추가
    try {
        String strContentType = Files.probeContentType(uploadFile.toPath());
        logger.info("업로드파일의 ContentType: " + strContentType);

        return strContentType.startsWith("image");
    } catch (IOException e) {
        e.printStackTrace();
    }

    return false;
}

//업로드 요청 파일-저장 및 결과 메시지 전송
@PostMapping("/fileUploadAjaxAction")
public void fileUploadActionPost(MultipartFile[] uploadFiles) {
    logger.info("====FileUpload With Ajax =====");

    //날짜 형식(yyyy/MM/dd)의 폴더구조 생성
    File fileUploadPath = new File(uploadFileRepoDir, getDatefmtPathName());
    logger.info("upload path: " + fileUploadPath);

    if (fileUploadPath.exists() == false) {
        fileUploadPath.mkdirs();
    }

    //각각의 업로드파일 이름 구성 후, 저장경로와 파일이름으로 생성된 파일객체를 서버에 저장
    for(MultipartFile multipartUploadFile : uploadFiles) {
        logger.info("=====");
        logger.info("Upload File Name: " + multipartUploadFile.getOriginalFilename());
        logger.info("Upload File Size: " + multipartUploadFile.getSize());

        //업로드파일이름 원본문자열
        String strUploadFileName = multipartUploadFile.getOriginalFilename();

        //#[Edge, IE 오류 해결] 경로 제거된 파일이름만 추출
        strUploadFileName = strUploadFileName.substring(strUploadFileName.lastIndexOf("\\") + 1);
        logger.info("수정된 파일이름(strUploadFileName): " + strUploadFileName);

        //UUID를 이용한 고유한 파일이름 적용
        UUID uuid = UUID.randomUUID();
        strUploadFileName = uuid.toString() + "_" + strUploadFileName ;
        logger.info("UUID처리된파일이름: "+strUploadFileName);

        //최종 저장 정보를 가진 파일 객체(서버레포경로 + 날짜형식 폴더 + UUID적용 파일이름)
        File saveUploadFile = new File(fileUploadPath, strUploadFileName);
        logger.info("저장시 사용되는 파일이름(saveUploadFile, 경로포함): " + saveUploadFile);

        try {
            //서버에 파일객체를 이용하여 업로드 파일 저장
            multipartUploadFile.transferTo(saveUploadFile);

            //업로드 파일에 대하여 이미지파일 여부 확인 -> 썸네일이미지 생성, 이미지파일이 아닌 경우, if 문 처리 없음.
            if (checkIsImageForUploadFile(saveUploadFile)) { //이미지 파일 인 경우, ←if 문 전체 추가
                //썸네일 생성경로와 파일이름이 설정된 파일객체를 전송 보내는 FileOutputStream 객체 생성
                FileOutputStream outputStreamForThumbnail =
                    new FileOutputStream( new File(fileUploadPath, "s_" + strUploadFileName) );
                //FileOutput-스트림으로 보내진 파일객체를 서버에 저장(input)하여, 20x20 크기(px)의 썸네일 생성
                Thumbnailator.createThumbnail(multipartUploadFile.getInputStream(), outputStreamForThumbnail, 20, 20);
                //OUT 스트림리소스 닫기
                outputStreamForThumbnail.close();
            }
        } catch (Exception e) {
            logger.error(e.getMessage());
        }
    }
}

```

```
    } // End-for  
}
```

### [테스트 - 이미지파일에 대한 썸네일 생성 테스트]

→ C:\myupload 폴더의 기존 내용을 모두 삭제합니다. 업로드 파일(이미지 파일과 이미지가 아닌 파일)을 준비합니다.

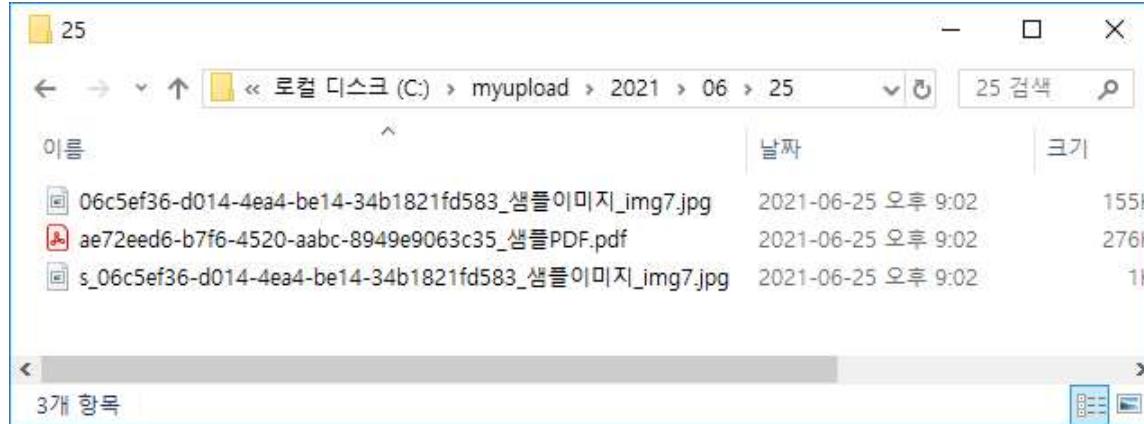
→ 톰캣 서버를 재기동

→ 웹 브라우저에서, <http://localhost:8080/mypro00/fileUploadAjax> 페이지를 요청/표시한 후,  
이미지 파일과 이미지가 아닌 파일을 업로드 합니다. → 이미지 파일인 경우에만 썸네일 파일이 생성되는지 확인합니다.

☞ 다음은 인터넷 익스플로러를 이용하여 톰캣 서버에 업로드 시 이클립스 콘솔에 표시된 로그입니다.

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload Ajax =====  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====FileUpload With Ajax =====  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload path: C:\myupload\2021\06\25  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플PDF.pdf  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 282604  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플PDF.pdf  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: ae72eed6-b7f6-4520-aabc-8949e9063c35_샘플PDF.pdf  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):  
C:\myupload\2021\06\25\ae72eed6-b7f6-4520-aabc-8949e9063c35_샘플PDF.pdf  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: application/pdf  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플이미지_img7.jpg  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 158112  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플이미지_img7.jpg  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 06c5ef36-d014-4ea4-be14-  
34b1821fd583_샘플이미지_img7.jpg  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):  
C:\myupload\2021\06\25\06c5ef36-d014-4ea4-be14-34b1821fd583_샘플이미지_img7.jpg  
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: image/jpeg
```

☞ 다음은 파일탐색기에서 C:\myupload\날짜형식폴더에서 썸네일 생성을 확인한 것입니다.



### (5) 업로드 요청 처리 후, 브라우저에 표시하는 피드백 메시지 구성(선택)

☞ Ajax를 통한 파일 업로드 요청을 서버에서 처리한 후, 파일 업로드를 요청한 동일한 웹브라우저의 페이지(페이지 전환없음)에 업로드 성공/실패 메시지 외에 필요한 경우, 업로드 처리에 대한 추가적인 정보를 전해서 표시되도록 구성할 수 있습니다.

☞ 문서에서는, 성공/실패 메시지와 함께 피드백으로 다음의 추가적인 내용이 전달되도록 구성합니다.

- 업로드로 저장된 파일에 대한 원본 파일의 이름과 파일이름에 추가된 문자열(예에서는 UUID 값)
- 파일이 저장된 년/월/일 경로
- 업로드된 파일이 이미지 파일인지의 여부

- ☞ 위의 정보들은 실제로 관리자 페이지에서 파일 관리 시에 필요한 정보로 활용될 수도 있습니다
- ☞ 업로드 성공/실패 메시지와 함께 위의 정보들이 웹 브라우저의 파일 업로드 요청 페이지에 전달되도록 구현하는 방법은 다음 2 가지를 고려할 수 있습니다.
  - 업로드 된 경로가 포함된 파일 이름을 반환하는 방식: 이 방식은 브라우저의 화면에서 해야 할 일이 많습니다.
  - **별도의 데이터 전달 객체를 생성해서 서버에서 정보를 제공하는 방식**

- ☞ 문서에서는 **별도의 데이터 전달 객체를 생성해서 서버에서 정보를 제공하는 방식**으로 구현합니다.

(5-1) JSON 처리 라이브러리 설치(이미 설치되어 있습니다)

- ☞ JSP 페이지 호출 없이 JSON 형식의 데이터를 보내기 위하여 서버에 JSON 및 jackson-databind 관련 라이브러리를 설치합니다.

→ mypro00 프로젝트의 pom.xml 파일을 코드 작성부에 오픈 후, 다음의 코드를 추가합니다.

```
<!-- JSON관련, jackson-databind -->
<!-- 데이터를 JSON 포맷의 문자열로 변환시켜 브라우저에 전송 -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.12.2</version>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.12.2</version>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.12.2</version>
</dependency>
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20210307</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.6</version>
</dependency>
```

## (5-2) AttachFileDTO 클래스 생성

☞ 브라우저로 첨부파일 관련 데이터를 전달할 저장할 AttachFileDTO 를 생성합니다

→ src/main/java/com.spring5213.mypro00.common.fileupload 패키지에 AttachFileDTO 클래스를 생성하고 다음의 내용을 작성합니다.

```
package com.spring5213.mypro00.common.fileupload;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class AttachFileDTO {
    private String fileName;      //원본파일이름
    private String uploadPath;   //업로드 경로 : yyyy/MM/dd 형식의 문자열이 저장됨
    private String uuid;         //uuid
    private String fileType;     //파일유형(이미지:Y, 이미지아님:N)
    private String repoPath = "C:\\myupload"; //서버레포지토리경로(C:\\myupload) 저장
}
```

☞ AttachFileDTO 클래스는 원본 파일의 이름, 업로드 경로(yyyy/MM/dd), UUID 값, 이미지여부 데이터를 브라우저로 전달하는 용도로 사용됩니다.

## (5-3) FileUploadControllerAjax.java 수정

☞ 업로드를 처리하는 fileUploadActionAjaxPost() 메소드가 AttachFileDTO 객체의 리스트를 반환하도록 수정합니다.

→ src/main/java/com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax.java 파일을 코드 작성부에 오픈하고 다음의 빨간색 코드를 fileUploadActionAjaxPost() 메소드에 추가합니다.

```
import java.util.ArrayList;                                     ← import는 Ctrl + Shift + O 키로 자동 추가됨
import java.util.List;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ResponseBody;

...(생략)...

@Controller
public class FileUploadControllerAjax {

    private static final Logger logger = LoggerFactory.getLogger(FileUploadControllerAjax.class);

    //저장경로 (Windows 환경이므로 경로구분자를 \\\로 지정)
    private String uploadFileRepoDir = "C:\\\\myupload" ;

    ...(생략)...

    //업로드 요청 파일-저장 및 결과 메시지 전송
    @PostMapping(value="/fileUploadAjaxAction", produces = "application/json; charset=UTF-8") ← 추가
    @ResponseBody                                         ← 추가
    public ResponseEntity<List<AttachFileDTO>> fileUploadActionPost(MultipartFile[] uploadFiles) { ← 수정
        logger.info("=====FileUpload With Ajax =====");
    }

    //업로드 파일 각각에 대한 피드백정보(AttachInfoDTO)를 담을 리스트 객체
    List<AttachFileDTO> listAttachInfo = new ArrayList<AttachFileDTO>(); ← 추가
```

```

//날짜 형식(yyyy/MM/dd)의 폴더구조 생성
//File fileUploadPath = new File(uploadFileRepoDir, getDatefmtPathName());      ← 주석처리 또는 삭제
//logger.info("upload path: " + fileUploadPath);      ← 주석처리 또는 삭제

//날짜 형식 폴더경로 문자열 객체(사용자에게 전달될 각 업로드파일의 저장경로로 날짜구조 폴더만 전달)
String strDatefmtPathName = getDatefmtPathName();      ← 추가

//전체 업로드 경로 파일객체 생성
File fileUploadPath = new File(uploadFileRepoDir, strDatefmtPathName);      ← 추가
logger.info("upload path: " + fileUploadPath);      ← 추가

if (fileUploadPath.exists() == false) {
    fileUploadPath.mkdirs();
}

//각각의 업로드파일 이름 구성 후, 저장경로와 파일이름으로 생성된 파일객체를 서버에 저장
for(MultipartFile multipartUploadFile : uploadFiles) {
    logger.info("=====");
    logger.info("Upload File Name: " + multipartUploadFile.getOriginalFilename());
    logger.info("Upload File Size: " + multipartUploadFile.getSize());

    //업로드파일 각각에 대한 피드백 정보가 저장될 AttachInfoDTO 객체 생성
    AttachFileDTO attachInfo = new AttachFileDTO();      ← 추가

    //attachInfoDTO에 repository 경로 저장
    attachInfo.setRepoPath(uploadFileRepoDir.toString());      ← 추가
    logger.info("attachInfoDTO.repoPath: " + attachInfo.getRepoPath());      ← 추가

    //attachInfoDTO에 날짜형식 경로 저장
    attachInfo.setUploadPath(strDatefmtPathName.toString());      ← 추가
    logger.info("attachInfoDTO.uploadPath: " + attachInfo.getUploadPath());      ← 추가

    //업로드파일이름 원본문자열
    String strUploadFileName = multipartUploadFile.getOriginalFilename();

    //#[Edge, IE 오류 해결] multipartUploadFile.getOriginalFilename()에서 업로드 파일이름만 추출
    //파일이름만 있는 경우, 파일이름만 추출됨
    strUploadFileName = strUploadFileName.substring(strUploadFileName.lastIndexOf("\\\\")+1);
    logger.info("수정된 파일이름(strUploadFileName): " + strUploadFileName);

    //attachInfoDTO에 원본업로드파일이름 저장
    attachInfo.setFileName(strUploadFileName);      ← 추가
    logger.info("attachInfoDTO.fileName: " + attachInfo.getFileName());      ← 추가

    //UUID를 이용한 고유한 파일이름 적용
    //랜덤한 UUID값을 가진 UUID 객체 생성
    UUID uuid = UUID.randomUUID();

    //attachInfoDTO에 UUID 문자열 저장
    attachInfo.setUuid(uuid.toString());      ← 추가
    logger.info("attachInfoDTO.uuid: " + attachInfo.getUuid());      ← 추가

    //파일이름에 UUID 문자열 추가(파일 확장자 때문에 UUID를 앞에다 추가해야 함)
    strUploadFileName = uuid.toString() + "_" + strUploadFileName ;
    logger.info("UUID처리된파일이름: "+strUploadFileName);

    //최종 저장 정보를 가진 파일 객체(서버레포경로 + 날짜형식 폴더 + UUID적용 파일이름)
    File saveUploadFile = new File(fileUploadPath, strUploadFileName);
    logger.info("저장시 사용되는 파일이름(saveUploadFile, 경로포함): " + saveUploadFile);

    try {
        //서버에 파일객체를 이용하여 업로드 파일 저장
        multipartUploadFile.transferTo(saveUploadFile);

        //업로드 파일에 대하여 이미지파일 여부 확인 -> 썸네일이미지 생성, 이미지파일이 아닌 경우, if 문 처리 없음.
        if (checkIsImageForUploadFile(saveUploadFile)) { //이미지 파일 인 경우,
            //attachInfoDTO.fileType에 "I" 저장
        }
    }
}

```

```

        attachInfo.setFileType("I");    ← 추가
        logger.info("attachInfoDTO.fileType: " + attachInfo.getFileType());    ← 추가
        //썸네일 생성경로와 파일이름이 설정된 파일객체를 전송 보내는 FileOutputStream 객체 생성
        FileOutputStream outputStreamForThumbnail =
            new FileOutputStream( new File(fileUploadPath, "s_" + strUploadFileName) );
        //FileOutput-스트림으로 보내진 파일객체를 서버에 저장(input)하여, 20X20 크기(px)의 썸네일 생성
        Thumbnailator.createThumbnail(multipartUploadFile.getInputStream(), outputStreamForThumbnail, 20, 20);
        //OUT 스트림리소스 닫기
        outputStreamForThumbnail.close();
    } else {//이미지파일이 아닌 경우           ← else {} 전체 추가
        //attachInfoDTO.fileType 에 "F" 저장
        attachInfo.setFileType("F");
        logger.info("attachInfoDTO.fileType: " + attachInfo.getFileType());
    }
} catch (Exception e) {
    logger.error(e.getMessage());
}
//List 객체에 attachInfoDTO 추가
listAttachInfo.add(attachInfo);   ← 추가

} // End-for

return new ResponseEntity<List<AttachFileDTO>> (listAttachInfo, HttpStatus.OK) ; ← 추가
}

```

☞ 업로드가 수행되면, 첨부파일 정보가 JSON 스트링 객체로 사용자 브라우저로 전달됩니다.

→ src/main/webapp/WEB-INF/views/ 폴더에 있는 fileUploadAjax.jsp 파일(업로드 요청 JSP 파일)의 Ajax 코드에서 서버로 부터 전송받을 데이터유형을 지정하는 dataType 옵션을 다음처럼 추가합니다(아래 코드의 빨간색부분 추가). 이 옵션을 명시하지 않아도 자동으로 유추해서 JSON 형식으로 데이터를 받지만 명시하는 것을 권장합니다.

```

//url 키에 명시된 주소의 컨트롤러에게 formData 객체를 POST 방식으로 전송.
$.ajax({
    url: '${contextPath}/fileUploadAjaxAction',
    processData: false, //contentType에 설정된 형식으로 data를 처리하지 않음.
    contentType: false, //contentType에 MIME 타입을 지정하지 않음.
    data: formData,
    type: 'POST',
    dataType: 'json',   ← 추가
    success: function(uploadResult){
        alert("첨부파일의 업로드가 정상적으로 완료되었습니다...\"");
    }
})

```

#### [테스트 - 이미지파일에 대한 썸네일 생성 테스트]

- C:\myupload 폴더의 기존 내용을 모두 삭제합니다. 업로드 파일(이미지 파일과 이미지가 아닌 파일)을 준비합니다.
- 톰캣 서버를 재기동
- 웹 브라우저에서, F12 키를 이용하여 개발자 도구의 네트워크 탭 페이지를 표시합니다.
- 웹 브라우저에서, http://localhost:8080/mypro00/fileUploadAjax 페이지를 요청/표시한 후,
- 이미지 파일과 이미지가 아닌 파일을 업로드 합니다.
- 개발자 도구의 네트워크 탭에 표시된 파일 목록에서 fileUploadAjaxAction 을 클릭하여, 본문(또는 Preview) 탭에 서버로부터 전달된 데이터가 표시되는지 확인합니다.

☞ 다음은 인터넷 익스플로러를 이용하여 템켓 서버에 업로드 시 이클립스 콘솔에 표시된 로그입니다.

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - ===FileUpload With Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload path: C:\myupload\2021\06\26
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 282604
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.repoPath: C:\myupload
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uploadPath: 2021\06\26
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileName: 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uuid: b9651772-b0f3-4d59-8e1b-d1ecbedf9d8c
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: b9651772-b0f3-4d59-8e1b-d1ecbedf9d8c_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\26\b9651772-b0f3-4d59-8e1b-d1ecbedf9d8c_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: application/pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileType: F
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 436
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.repoPath: C:\myupload
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uploadPath: 2021\06\26
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileName: 샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uuid: 3a531500-04b3-46ab-9dd9-a7df311bf8e8
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 3a531500-04b3-46ab-9dd9-a7df311bf8e8_샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\26\3a531500-04b3-46ab-9dd9-a7df311bf8e8_샘플TEXT.txt
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: text/plain
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileType: F
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 158112
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.repoPath: C:\myupload
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uploadPath: 2021\06\26
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileName: 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uuid: 95a41b9b-8942-4221-9a23-b6762fd576a4
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: 95a41b9b-8942-4221-9a23-
b6762fd576a4_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\26\95a41b9b-8942-4221-9a23-b6762fd576a4_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: image/jpeg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileType: I
```

☞ 다음은 인터넷 익스플로러의 개발자 도구에서 파일 업로드 시에 네트워크 탭 페이지에 표시된 fileUploadAjaxAction 이름의 본문 내용을 확인한 것입니다.

The screenshot shows the F12 developer tools Network tab with the following details:

- Request URL: fileUploadAjax - F12 개발자 도구
- Method: POST
- Status: 200
- Content-Type: application/json
- Request Headers:
  - Content-Type: application/json
  - Content-Length: 136
  - Host: localhost:8080
  - Connection: keep-alive
  - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36
  - Accept: \*/\*
  - Referer: http://localhost:8080/mypro00/fileUploadAjax
  - Origin: http://localhost:8080
  - Sec-Fetch-Site: same-origin
  - Sec-Fetch-Mode: cors
  - Sec-Fetch-Dest: empty
  - Accept-Encoding: gzip, deflate
  - Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
- Response Headers:
  - Content-Type: application/json
  - Content-Length: 136
  - Date: Mon, 28 Jun 2021 10:45:26 GMT
  - Server: Apache/2.4.41 (Ubuntu)
- Request Body (JSON):

```
[{"fileName": "샘플PDF.pdf", "uploadPath": "2021\\06\\26", "uuid": "b9651772-b0f3-4d59-8e1b-d1ecbedf9d8c", "fileType": "F", "repoPath": "C:\\myupload"}, {"fileName": "샘플TEXT.txt", "uploadPath": "2021\\06\\26", "uuid": "3a531500-04b3-46ab-9dd9-a7df311bf8e8", "fileType": "F", "repoPath": "C:\\myupload"}, {"fileName": "샘플이미지_img7.jpg", "uploadPath": "2021\\06\\26", "uuid": "95a41b9b-8942-4221-9a23-b6762fd576a4", "fileType": "I", "repoPath": "C:\\myupload"}]
```
- Response Body (JSON):

```
[{"fileName": "샘플PDF.pdf", "uploadPath": "2021\\06\\26", "uuid": "b9651772-b0f3-4d59-8e1b-d1ecbedf9d8c", "fileType": "F", "repoPath": "C:\\myupload"}, {"fileName": "샘플TEXT.txt", "uploadPath": "2021\\06\\26", "uuid": "3a531500-04b3-46ab-9dd9-a7df311bf8e8", "fileType": "F", "repoPath": "C:\\myupload"}, {"fileName": "샘플이미지_img7.jpg", "uploadPath": "2021\\06\\26", "uuid": "95a41b9b-8942-4221-9a23-b6762fd576a4", "fileType": "I", "repoPath": "C:\\myupload"}]
```

☞ 크롬 브라우저와 엣지 브라우저에서도 동일한 방법으로 업로드를 테스트 합니다.

## 11-4. 파일 업로드 실습: 파일 업로드 결과 메시지 화면 처리

- 업로드 구현 시에 업로드 요청 페이지에 대하여 고려해야 할 사항으로 다음과 같은 작업이 중요합니다.
  - 업로드 후에 업로드 파일을 선택시에 사용한 `<input type='file'>` 을 초기화 시키는 작업
  - 결과 데이터를 이용해서 요청 페이지의 화면에 파일 정보나 셜네일 또는 파일 이미지를 보여주는 작업

### (1) 업로드 후, 요청페이지 화면 처리: `<input type='file'>` 초기화

- 파일 업로드를 요청할 때, 업로드할 파일을 선택한 `<input type='file'>` 요소를, 업로드가 끝난 후에는 초기화하여 첨부파일을 다시 추가할 수 있도록 합니다.

- `<input type='file'>`은 다른 DOM 요소들과 다르게, 사용자가 `input` 요소에 선택된 파일 내용을 변경할 수 없기 때문에 파일을 선택하기 전에 비어있는 `input` 을 복사하여 변수에 저장한 후, 업로드 후에 복사된 빈 `input` 으로, 파일이 선택된 `input` 을 교체하여 초기화를 구현하는 것이, 요소를 삭제하고 새로 추가하는 것보다 쉬운 방법입니다.

→ src/main/webapp/WEB-INF/views/fileUploadAjax.jsp 페이지를 코드 작성부에 오픈

→ 다음의 빨간색 코드를 추가 또는 수정합니다

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>//업로드 파일의 확장자 및 최대 파일 크기 검사 함수

...(생략)...

//input 초기화를 위해 div 요소의 비어있는 input 요소를 복사해서 저장함.
var cloneInputFile = $(".uploadDiv").clone() ; ← 파일업로드 처리 함수위에 추가

//파일업로드 처리
$("#btnFileUpload").on("click", function(e) {
    var formData = new FormData(); //Ajax 파일 전송 시에 사용되는 Web API 클래스
    var inputFiles = $("input[name='uploadFiles']");
    var files = inputFiles[0].files; //uploadFiles 이름의 file 유형 input 요소를 변수에 저장.
    console.log(files);

    for(var i = 0; i < files.length ; i++) { //formdata 객체에 파일추가
        if (!checkUploadfile(files[i].name, files[i].size)) { //파일 확장자 제한, 최대허용파일크기검사
            return false;
        }

        formData.append("uploadFiles", files[i]); //uploadFiles 파라미터로 file 정보 추가
    }

    $.ajax({ //서버로 formData 객체를 POST 방식으로 전송.
        url: '${contextPath}/fileUploadAjaxAction',
        processData: false,
        contentType: false,
        data: formData,
        type: 'post',
        dataType: 'json',
        success: function(uploadResult){
            alert("첨부파일의 업로드가 정상적으로 완료되었습니다..."); //파일이름이 선택된 기존 input을 초기화
            $(".uploadDiv").html(cloneInputFile.html()); //다른 방법 브라우저 버전따라 실행이 않을 수 있음(권장않함)
        }
    }) //end ajax
}); //end ajax
</script>
```

## [테스트 - input 초기화 테스트]

- 브라우저에서, <http://localhost:8080/mypro00/fileUploadAjax> 페이지를 요청/표시한 후, `input` 초기화가 정상적으로 수행되는지 확인합니다.

input 초기화 구현 전	input 초기화 코드 적용 후

☞ 화면에서 업로드 작업이 끝난 후, 다시 업로드를 할 수 있도록 파일 선택 부분이 초기화 된 것이 확인됩니다(오른쪽 마지막 그림).

## (2) 업로드 후의 요청페이지 화면 처리(이미지가 아닌 파일) - 첨부파일 아이콘과 파일이름 표시

☞ 이미지가 아닌 업로드 파일에 대하여, 화면에 첨부파일 아이콘과 파일이름을 표시해주도록 구성합니다.

→ 결과에 표시할 첨부파일 ICON 이미지를, 구글에서 attach.png 로 검색하여 PNG 형식의 무료 아이콘을 다운로드 합니다.

→ src/main/webapp/resources/img 폴더(img 폴더가 없으면 생성)에 다운로드 받은 첨부파일 아이콘 이미지파일을 복사해서 icon-attach.png 이름으로 붙여넣기 합니다.

→ src/main/webapp/WEB-INF/views/fileUploadAjax.jsp 페이지를 코드 작성 뷰에 오픈하고, 다음의 사항을 구현합니다.

- 파일이름이 표시될 <ul> 태그가 포함된 <div> 요소를 추가.
- 자바스크립트 showUploadedFiles() 함수 생성: <ul> 태그 내에 <li> 태그를 추가하여 업로드 파일 아이콘과 파일이름을 표시
- 파일 업로드 ajax 성공 시 실행되는 함수에 showUploadedFiles() 함수 실행코드 추가.

→ File Upload With Ajax 버튼 밑에 파일이름이 표시될 <ul> 태그가 포함된 <div> 요소를 추가합니다.

```
...(생략)...
<button id="btnFileUpload">File Upload With Ajax</button>
<div class="fileUploadResult">
    <ul>
        <%-- 업로드 후 처리결과가 표시될 영역 --%>
    </ul>
</div>
...(생략)...
```

→ checkUploadfile() 함수 밑에 showUploadedFiles() 함수 코드를 추가.

```
...(생략)...
//업로드 결과 표시 함수
function showUploadedFiles(uploadResult) {
    console.log("showUploadedFiles 실행: 화면 표시 시작=====");
    //ul 태그 변수화
    var fileUploadResult = $(".fileUploadResult ul");
    var str = "";
    //전달받은 배열형식 데이터 각각에 대하여
    $(uploadResult).each(function(i, obj) {
        if (obj.fileType=="F") { //이미지가 아닌 경우, 아이콘 이미지 및 원본 파일이름 표시

            str += "<li>" +
                "    <img src='${contextPath}/resources/img/icon-attach.png'" +
                "        alt='No Icon' style='height: 18px; width: 18px;'>" +
                "        obj.fileName
                "</li>";
        } else if (obj.fileType=="I") { //이미지파일인 경우 파일이름만 표시(임시)
            str += "<li>" + obj.fileName + "</li>";
        }
    });
    //기존 페이지에 결과를 HTML로 추가
    fileUploadResult.append(str);
}

//input 초기화를 위해 div 요소의 비어있는 input 요소를 복사해서 저장함.
var cloneInputFile = $(".uploadDiv").clone() ;

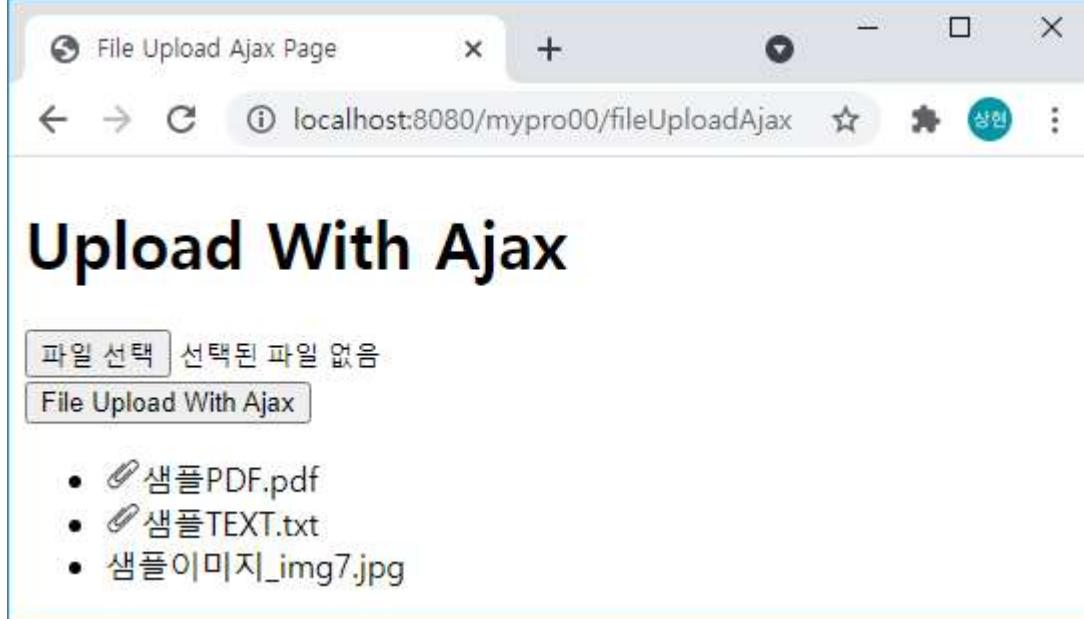
...(생략)...
```

→ 파일 업로드 ajax 성공 시 실행되는 함수에 `showUploadedFiles()` 함수 실행코드를 추가합니다.

```
//파일업로드 처리
$("#btnFileUpload").on("click", function(e) {
    ...생략...
    $.ajax({
        url: '${contextPath}/fileUploadAjaxAction',
        processData: false,
        contentType: false,
        data: formData,
        type: 'POST',
        dataType: 'json',
        success: function(uploadResult){
            alert("첨부파일의 업로드가 정상적으로 완료되었습니다....");
            $(".uploadDiv").html(cloneInputFile.html());
            showUploadedFiles(uploadResult); ←추가
        }
    })
})
```

#### [테스트 - 이미지가 아닌 파일의 결과 표시 테스트]

→ 웹 브라우저에서 업로드 수행 후, 표시된 결과 화면을 확인합니다.



#### (3) 업로드 후의 요청페이지 화면 처리(이미지 파일) - 썸네일을 이용하여 이미지 표시

☞ 업로드 파일이 이미지 파일인 경우, 서버의 컨트롤러에서 업로드 수행 후에, 사용자의 웹브라우저(요청페이지)로 전송하는 결과 메시지에, 썸네일 파일이 GET 방식으로 전달(즉, URI 뒤에 파일이름이 추가되어 전송)되도록 구현합니다.

☞ 전송된 결과를 전달받은 사용자의 요청페이지에, 썸네일 이미지가 `img` 태그를 이용하여 표시되도록 구현합니다.

☞ 현재 서버에 저장된 이미지파일에 대한 썸네일 파일의 이름은 '파일의 경로' + 's\_' + 'uuid' + 원본파일이름.확장자'입니다.

☞ 주의해야 할 사항은, 경로나 파일 이름에 한글 혹은 공백 등의 문자가 들어가면 문제가 발생할 수 있으므로 자바스크립트의 encodeURIComponent() 함수를 이용해서, 문제가 없도록 처리합니다.

→ src/main/java/com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax.java 파일을 코드 작성부에 오픈  
→ 이미지 파일 이름을 받아서 썸네일 이미지 파일 데이터를 GET 방식으로 전송하는 sendThumbNailFile() 메서드를 기존 메서드 밑에 생성합니다.

```
import org.springframework.http.HttpHeaders;           ← import는 Ctrl + Shift + O 로 자동 완성 됨
import org.springframework.util.FileCopyUtils;
import net.coobird.thumbnailator.Thumbnailator;

...(생략)...

//썸네일 이미지 파일 다운로드 ← 메서드 전체 추가
@GetMapping("/displayThumbnailFile")
@ResponseBody
public ResponseEntity<byte[]> sendThumbNailFile(String fileName) {

    File file = new File(fileName);
    logger.info("썸네일파일이름(경로포함): " + file);

    ResponseEntity<byte[]> result = null;

    try {
        HttpHeaders header = new HttpHeaders();

        //HttpHeader 객체에 썸네일이미지파일의 Content-Type 추가
        header.add("Content-Type", Files.probeContentType(file.toPath()));

        //복사된 썸네일 파일을 HttpHeaders에 추가된 Content-Type과 상태값을 가지고 ResponseEntity<byte[]> 객체 생성
        result = new ResponseEntity<byte[]>(FileCopyUtils.copyToByteArray(file), header, HttpStatus.OK);

    } catch (IOException e) {
        e.printStackTrace();
    }

    return result;      //ResponseEntity<byte[]> 객체 반환
}
```

☞ getThumbNailFile() 메서드는, 파일의 경로가 포함된 fileName 문자열을 파라미터로 받아서, 스프링의 FileCopyUtils.copyToByteArray() 메서드를 이용하여, 이미지 파일의 복사본에 대한 byte[]를 전송합니다.

byte[]로 이미지 파일의 데이터를 전송할 때, 파일의 종류에 따라 MIME 타입이 변하지 않도록 probeContentType()을 이용해서 파일에 대한 MIME 타입 값을 Http의 헤더 메시지에 포함하여 ResponseEntity 객체를 통해 전달합니다.

☞ 브라우저의 업로드 요청페이지에서 위의 컨트롤러 메소드를 호출하여 썸네일 파일이 표시될 수 있도록 showUploadedFiles() 함수에 구현합니다. GET 방식으로 썸네일 파일을 요청하여 받을 때, 파일 이름에 포함된 공백 문자나 한글 이름 등이 포함된 경우를 고려해서 파일이름이 포함된 URI에 대하여 encodeURIComponent()를 이용해서 URI 호출에 적합한 문자열로 인코딩합니다.

→ src/main/webapp/WEB-INF/views/fileUploadAjaxAction.jsp 페이지를 코드 작성부에 오픈 후

showUploadedFiles() 함수를 다음처럼 빨간색 코드를 추가합니다.

```
//업로드 결과 표시 함수: 이미지 파일은 썸네일과 원본 파일이름 표시, 이미지가 아닌 파일은 첨부파일 아이콘과 원본 파일이름 표시
//서버로부터 전달받은 JSON 데이터가 매개변수로 사용됨.
function showUploadedFiles(uploadResult) {
    console.log("showUploadedFiles 실행: 화면 표시 시작=====");
```

```

//ul 태그 변수화
var fileUploadResult = $(".fileUploadResult ul");
var str = "";
//전달받은 배열형식 데이터 각각에 대하여
$(uploadResult).each(function(i, obj) {
    if (obj.fileType=="F") { //이미지가 아닌 경우, 아이콘 이미지 및 원본 파일 이름 표시
        str += "<li>" +
            "    <img src='${contextPath}/resources/img/icon-attach.png' alt='No Icon' " +
            "        style='height: 18px; width: 18px;'>" +
            "    obj.fileName" +
            "</li>";
    } else if (obj.fileType=="I") { //이미지파일인 경우 썸네일 및 원본 파일 이름 표시
        //str += "<li>" + obj.fileName + "</li>"; //기존코드 삭제
        //경로명이 포함된 썸네일 파일 이름 저장: encodeURIComponent로 처리
        var thumbnailFilePath =
            encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/s_" + obj.uuid + "_" + obj.fileName);
        console.log("encodeURIComponent 처리 파일이름: " + thumbnailFilePath);

        str += "<li>" +
            "    <img src='${contextPath}/displayThumbnailFile?fileName=" + thumbnailFilePath + "'"
            "        alt='No Icon' style='height: 18px; width: 18px;'>" +
            "    obj.fileName" +
            "</li>";
    }
});
});

fileUploadResult.append(str); //fileUploadResult 클래스의 div에 있는 ul에 결과를 HTML로 추가
}

```

#### [테스트 - 이미지가 아닌 파일의 결과 표시 테스트]

→ 웹 브라우저에서, <http://localhost:8080/mypro00/fileUploadAjax> 페이지를 요청한 후,  
한글이 포함된 파일들을 업로드합니다. → 브라우저에 피드백 결과가 정상적으로 표시되는지 확인합니다.



☞ 다음은 인터넷 익스플로러를 이용하여 톰캣 서버에 업로드 시 이클립스 콘솔에 표시된 로그입니다.

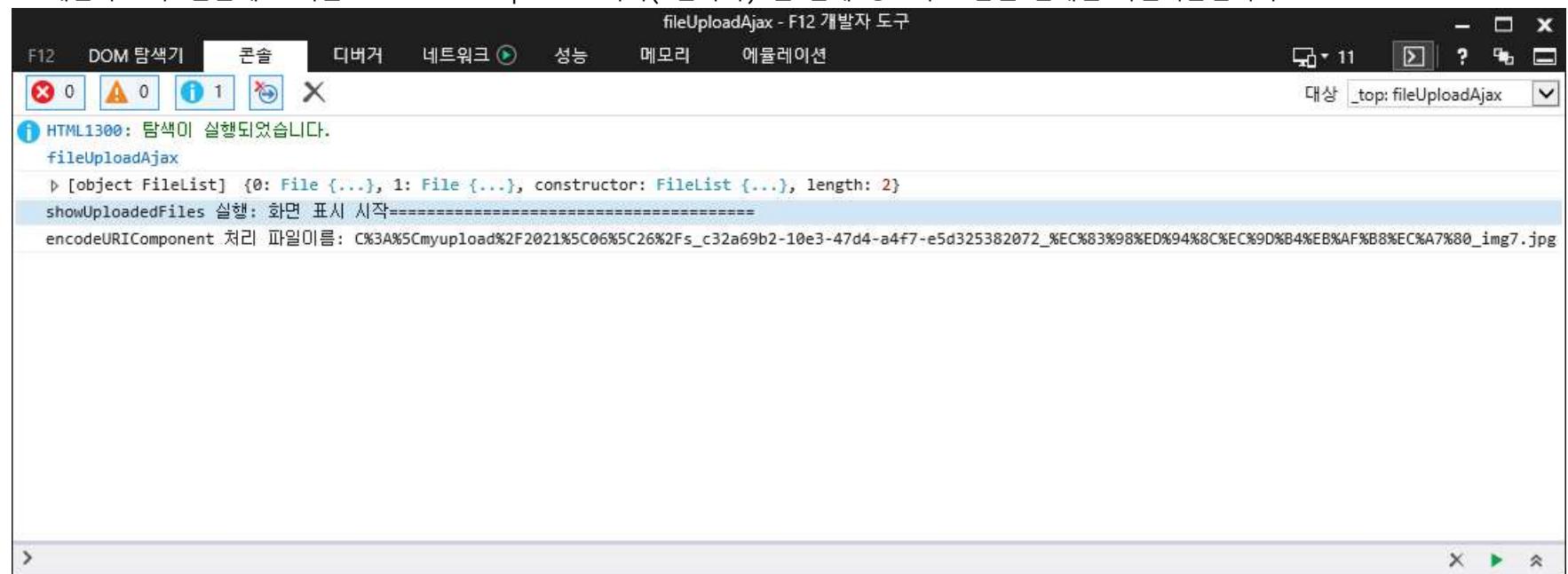
```

INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - ===FileUpload With Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload path: C:\myupload\2021\06\26
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 282604
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.repoPath: C:\myupload
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uploadPath: 2021\06\26
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileName: 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uuid: ee8daad2-aae9-4cdf-a58e-693cd43fb36
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: ee8daad2-aae9-4cdf-a58e-693cd43fb36_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\26\ee8daad2-aae9-4cdf-a58e-693cd43fb36_샘플PDF.pdf

```

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: application/pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileType: F
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 158112
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.repoPath: C:\myupload
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uploadPath: 2021\06\26
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileName: 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uuid: c32a69b2-10e3-47d4-a4f7-e5d325382072
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: c32a69b2-10e3-47d4-a4f7-e5d325382072_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함): C:\myupload\2021\06\26\c32a69b2-10e3-47d4-a4f7-e5d325382072_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: image/jpeg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileType: I
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 썸네일파일이름(경로포함): C:\myupload\2021\06\26\s_c32a69b2-10e3-47d4-a4f7-e5d325382072_샘플이미지_img7.jpg
```

☞ 개발자 도구 콘솔에 표시된 encodeURIComponent 처리(%문자화) 된 전체 경로가 포함된 썸네일 파일이름입니다.



## 11-4. 파일 다운로드 실습: 첨부파일의 다운로드 및 원본 보여주기

- ☞ 브라우저에서 보이는 첨부파일(실제 파일은 서버에 저장되어 있음)은 크게 이미지 파일과 일반 파일로 구분됩니다
- ☞ 첨부파일이 이미지 파일인 경우에는 썸네일 이미지를 클릭했을 때, 브라우저에 원본 파일을 크게 보여주는 형태로 처리되어야 합니다.  
이 경우는 브라우저에서 새로운 <div>등을 생성해서 처리하는 방식을 이용하는데 이를 흔히 "light-box"라고 합니다.  
"light-box"는 jQuery를 이용하는 많은 플러그인들이 있으므로 이를 이용하거나 직접 구현할 수 있습니다.  
예제는 직접 구현하는 방식으로 합니다.
- ☞ 첨부파일이 일반 파일인 경우에, 기본적인 처리는 다운로드입니다. 사용자가 파일을 선택하면 다운로드가 실행되면서 해당 파일의 이름으로 다운로드가 가능해야 합니다. 이 때, 브라우저 종류에 따른 한글 처리 등의 이슈를 해결해야 합니다.
- ☞ 이미지가 아닌 일반 파일의 다운로드는 서버에서 파일의 MIME 타입을 다운로드 타입으로 지정하고, 적절한 헤더 메시지를 통해서 다운로드 이름이 지정되도록 처리합니다.

### [참고] Content-Disposition Header

- Response-Body의 데이터(파일)를 브라우저가 어떻게 처리할지를 설정하는 Header입니다.
- 설정값이 inline인 경우, 웹페이지 화면에 표시되고, attachment인 경우에는 다운로드됩니다.  
설정예: Content-Disposition: inline  
설정예: Content-Disposition: attachment; filename='파일이름.확장자'
- 다운로드되길 원하는 파일은 attachment로 설정하고, filename 옵션으로 파일명을 지정합니다.

### [참고] application/octet-stream MIME 타입

- 육텟 스트림은, 8비트 단위의 이진 데이터를 의미하며, 특별히 표현할 수 있는 프로그램이 존재하지 않는 이진 파일 데이터의 경우, 기본값으로 octet-stream을 사용합니다. 실제로 잘 알려지지 않은 이진 파일을 의미하므로, 브라우저는 보통 자동으로 실행하지 않거나 실행해야 할지 묻기도 합니다.
- 보통 Content-Disposition 헤더를 attachment로 설정하여, 해당 데이터를 수신받은 브라우저가 파일을 저장 또는 다른이름으로 저장 여부를 지정하도록 합니다.

☞ MIME 타입과 Content-Disposition 헤더에 대한 보다 자세한 사항은 구글에서 검색하거나 다음의 문서사이트를 이용합니다.

- MIME 타입: [https://developer.mozilla.org/ko/docs/Web/HTTP/Basics\\_of\\_HTTP/MIME\\_types](https://developer.mozilla.org/ko/docs/Web/HTTP/Basics_of_HTTP/MIME_types)
- Content-Disposition : <https://developer.mozilla.org/ko/docs/Web/HTTP/Headers/Content-Disposition>

☞ 예제에서의 다운로드 구현은 2 단계로 수행됩니다.

- 다운로드 요청을 처리하는 컨트롤러 생성
- 업로드 결과 페이지에서 다운로드 요청 구현

## (1) 다운로드 컨트롤러 구현.

☞ 서버에서 다운로드 요청을 처리하는 컨트롤러는 다음의 사항을 고려해서 구현합니다.

- 서버에서 확인된 파일은, ReponseEntity<>를 이용하여 다운로드 되도록 처리합니다. 이 때, ResponseEntity<>의 타입으로 byte[] 보다 **org.springframework.core.io.Resource** 타입을 이용하면 다운로드를 보다 간단히 처리할 수 있습니다.
- 서버에서 브라우저로 보내는 첨부파일에 대한 **MIME 타입은** 이진파일을 다운로드를 할 수 있도록 '**application/octet-stream**'으로 **지정**합니다.
- 다운로드 시 **저장되는 파일이름은 'Content-Disposition' Http-헤더를 이용해서 전달하여 지정**합니다.  
이 때, 파일 이름에 한글이 포함될 경우 저장할 때 깨지는 문제를 막기 위해서, **파일이름-문자열을 UTF-8로 인코딩**하고 이를 **HTTP 기본 인코딩 설정으로 디코딩할 수 있도록 처리**합니다.
- 파일 업로드 시에 서버에 저장되는 파일의 이름이 중복되는 것을 방지하기 위하여 **파일이름에 추가된 UUID를 다운로드 되는 파일이름에서는 제거**해줍니다.
- 일단 다운로드 테스트는 브라우저에서 C:\myupload 폴더에 있는 파일의 이름과 확장자로 '/download?fileName=xxxx' URI로 호출하면 브라우저는 자동으로 해당 파일을 다운로드 되도록 구현합니다.
- 단, **IE 계열 및 Edge 브라우저에서는 다운로드 파일에 한글이 포함된 경우, 다운로드가 수행 되지 않습니다.**  
이것은 IE 계열 및 Edge 브라우저에서, 'Content-Disposition' HTTP-헤더에 설정된 파일이름을 처리하는 인코딩 방식이 다르기 때문입니다. Edge 브라우저와 IE-계열도 인코딩 방식이 차이가 납니다. 따라서, **IE/Edge/Chrome** 브라우저에 대하여 동시에 서비스해야 한다면 HttpServletRequest에 포함된 헤더 정보들을 이용해서 요청이 발생한 브라우저의 종류에 따라 파일이름의 인코딩이 수행될 수 있도록 구현해야 합니다. HTTP 헤더들 중 'User-Agent' 헤더의 메시지를 이용하여, 브라우저의 종류와 모바일/데스크톱 용 여부, **브라우저 프로그램의 종류를 확인**합니다.

→ src/main/java/ 폴더의 com.spring5213.mypro00.common 패키지에 filedownload 패키지를 생성한 후,

FileDownloadControllerAjax 클래스를 생성하고, 위의 사항을 고려한 다음의 코드를 작성합니다.

```
package com.spring5213.mypro00.common.filedownload;

import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class FileDownloadControllerAjax {

    private static final Logger logger = LoggerFactory.getLogger(FileDownloadControllerAjax.class);

    //private static final String uploadFileRepoDir = "C:\\\\upload" ;
    //크롬/IE/Edge 브라우저에서 다운로드 가능하도록 구현, 다운로드 파일이름에서 UUID 제거 해야 함
    //서버에서 보내는 파일에 대한 MIME타입(produces 속성)을 application/octet-stream 또는
    //{MediaType.APPLICATION_OCTET_STREAM_VALUE} 으로 설정해야 함
```

```

@GetMapping(value = "/fileDownloadAjax", produces = {"application/octet-stream"})
@ResponseBody //일반 컨트롤러에서 REST 방식으로 사용자 브라우저에 데이터만 전송하는 REST-API 어노테이션
public ResponseEntity<Resource> fileDownloadActionAjax(
    //사용자 브라우저를 확인하기 위해 User-Agent 헤더를 매개변수로 설정
    @RequestHeader("User-Agent") String userAgent, String fileName){
    logger.info("브라우저로 요청에서 전달된 User-Agent 헤더정보: " + userAgent);
    logger.info("처리 전 파일이름: " + fileName);

    //톰캣 서버 운영체제에 있는 파일을 액세스할 수 있는 Resource 객체를 생성하기 위하여
    //스프링 Resource 인터페이스를 통해 FileSystemResource 구현객체를 생성합니다.
    Resource resource = new FileSystemResource(fileName);
    logger.info("resource :" + resource);

    //파일이 존재하지 않으면, 오류전송 후, 메소드 종료
    if(resource.exists() == false) {
        return new ResponseEntity<Resource>(HttpStatus.NOT_FOUND);
    }
    //파일이 존재하면 아래의 내용 수행
    //확인된 파일이름을 변수에 저장
    String resourceFileName = resource.getFilename();
    logger.info("UUID 제거 전 resourceFileName :" + resourceFileName);

    //UUID가 제거된 파일이름
    resourceFileName = resourceFileName.substring(resourceFileName.indexOf("_") + 1);
    logger.info("UUID 제거 후 resourceFileName :" + resourceFileName);

    //스프링의 HttpHeaders 객체 생성
    HttpHeaders httpHeaders = new HttpHeaders();

    try {
        String downloadName = null;
        if (userAgent.contains("Trident") || userAgent.contains("MSIE") ||
            userAgent.contains("Edge") || userAgent.contains("Edg")) {
            logger.info("IE 또는 엣지 브라우저입니다");
            downloadName = URLEncoder.encode(resourceFileName, "UTF8");
            logger.info("IE에서의 파일이름: " + downloadName);

        } else {
            logger.info("Chrome 브라우저입니다");
            //UTF-8로 인코딩한 문자열의 바이트를, HTML기본 인코딩(ISO-8859-1)으로 인코딩 된 문자열 생성
            //ISO-8859-1 대신 UTF-8을 지정하면, 웹브라우저에 전달된 파일이름이 _____.jpg로 표시됩니다.
            downloadName = new String(resourceFileName.getBytes("UTF-8"), "ISO-8859-1");
            //콘솔(UTF-8 디코딩)을 사용하는데, 브라우저에 맞추어 ISO-8859-1 인코딩, 따라서 한글이 이상하게 표시됨(정상)
            logger.info("Chrome에서의 파일이름: " + downloadName);

        }
        //생성된 HttpHeaders 객체에 파일을 다운로드 받을 수 있도록 Content-Disposition 헤더 설정
        //HTML기본 인코딩(ISO-8859-1)으로 인코딩된 파일이름 문자열을 지정
        httpHeaders.add("Content-Disposition", "attachment; filename=" + downloadName);

    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }

    //ResponseEntity<Resource>에 서버의 파일리소스, HttpHeaders 객체, 상태를 담아서 반환
    return new ResponseEntity<Resource>(resource, httpHeaders, HttpStatus.OK);
}
}

```

☞ 파일의 종류에 상관없이 다운로드가 가능하도록 produces 속성으로 서버에서 브라우저로 보내는 파일의 MIME 타입()을 application/octet-stream 또는 MediaType.APPLICATION\_OCTET\_STREAM\_VALUE로 설정합니다.

☞ fileDownloadActionAjax() 메서드는 @RequestHeader를 이용해서 'User-Agent' HTTP 헤더의 정보를 파악하고, User-Agent 헤더의 설정값에 'MSIE'나 'Trident' 또는 'Edge'나 'Edg'가 포함된 경우(IE 또는 엣지 브라우저)는

URLEncoder.encode()로 다운로드 파일의 경로와 파일이름을 처리하고, 크롬 브라우저의 경우에는 웹 브라우저(HTTP) 기준의 경로와 파일이름에 대한 인코딩을 각각 처리하도록 구현되었습니다.

☞ 구글 크롬 브라우저가 오픈 소스로 공개된 이 후, 엣지 브라우저가 크롬 오픈 소스 기반의 브라우저로 변경되어 배포됩니다.

따라서 엣지 브라우저에 대한 User-Agent에 설정된 값이, 크롬 브라우저와 동일한 설정값에 **Edg/91.0.864.59**만 추가되어 있습니다.

○ 크롬 브라우저의 User-Agent 헤더 설정값

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
```

○ 엣지 브라우저의 User-Agent 헤더 설정값

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36  
Edg/91.0.864.59
```

→ src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml 파일에 다음의 설정을 추가하여,

생성한 FileDownloadControllerAjax 클래스가 빈으로 생성되도록 구성합니다.

```
...(생략)...  
<context:component-scan base-package="com.spring5213.mypro00.service" />  
<context:component-scan base-package="com.spring5213.mypro00.controller" />  
<context:component-scan base-package="com.spring5213.mypro00.common.fileupload" />  
<context:component-scan base-package="com.spring5213.mypro00.common.filownload" /> ←추가  
...(생략)...
```

## (2) 브라우저에서의 파일 다운로드 요청 및 원본 이미지 표시 요청 구현

☞ 업로드 후, 업로드 요청 페이지에 표시된 결과를 이용하여 다운로드 혹은 원본 이미지 표시가 가능하도록

fileUploadAjax.jsp 페이지에 다음의 작업을 구현합니다.

○ 이미지 파일은 썸네일 이미지를 클릭하면, 이미지 파일이 다운로드 되어 웹브라우저에 표시되도록 구현합니다.

○ 이미지가 아닌 일반 파일들에 대해서, icon-attach.png 를 클릭하면, 파일이 다운로드 되도록 구현합니다.

○ 파일 다운로드가 가능하도록, 다운로드에 필요한 경로와 UUID 가 붙은 파일 이름을 이용해서

'\${contextPath}/fileDownloadAjax?fileName=xxxx' 부분을 <a> 태그로 추가하고

다운로드 된 파일들은 UUID 가 제거된 이름으로 저장되도록 구현합니다.

→ src/main/webapp/WEB-INF/views/fileUploadAjax.jsp 페이지를 코드 작성 뷰에 오픈, 다음의 작업들을 구현

→ 이미지 파일이 표시될 <div> 영역을 <h1> 밑에 추가(위치는 크게 상관없습니다).

```
...(생략)...  
<h1>Upload With Ajax</h1>  
<div class='bigImageWrapper'>  
    <div class='bigImage'>  
        <!-- 이미지파일이 표시되는 DIV -->  
    </div>  
</div>  
...(생략)...
```

→ 원본 이미지를 다운로드 받아 표시하는 showImage()함수를 파일업로드 처리함수 밑(제일 마지막)에 추가

```
//이미지 표시
//display 속성의 flex 옵션을 이용하면, 이미지가 웹브라우저의 중앙에 표시됩니다.
function showImage(calledPathImagefileName){
    $(".bigImageWrapper").css("display","flex").show();

    $(".bigImage")
        .html("<img src='${contextPath}/fileDownloadAjax?fileName=" + encodeURI(calledPathImagefileName)+"'>")
        .animate({width: '100%', height: '100%'}, 1000);
}
```

☞ 이미지파일의 이름을 자바스크립트 encodeURI()함수로 처리해주어야 브라우저에 상관없이 파일이 다운로드되어 표시됩니다.

☞ 작성된 함수는 showUploadedFiles() 함수에서 호출되어 사용됩니다.

→ 표시된 원본 이미지를 클릭했을 때, 이미지가 사라지고, 업로드 JSP 페이지를 다시 표시하는 jQuery 클릭 이벤트 함수 추가

```
//원본 이미지 표시 DIV 클릭 이벤트처리: 클릭 시 1초 후에 이미지 사라짐.
$(".bigImageWrapper").on("click", function(e){

    $(".bigImage").animate({width:'0%', height: '0%'}, 1000);
    //setTimeout(() => {$(this).hide();}, 1000);
    setTimeout(function () {
        $(".bigImageWrapper").hide();
    }, 1000);
});
```

→ showUploadedFiles() 함수를 파일 다운로드가 가능하도록 <a> 태그를 추가하여, 다음처럼 수정(빨간색 부분).

```
//업로드 결과 표시 함수
function showUploadedFiles(uploadResult) {
    console.log("showUploadedFiles 실행: 화면 표시 시작-----");
    //ul 태그 변수화
    var fileUploadResult = $(".fileUploadResult ul");
    var str = "";
    //전달받은 배열형식 데이터 각각에 대하여
    $(uploadResult).each(function(i, obj) {
        if (obj.fileType=="F") { //이미지가 아닌 경우, 아이콘 이미지 및 원본 파일이름 표시
            var calledPathFileName = encodeURIComponent( obj.repoPath + "/" + obj.uploadPath + "/" +
                obj.uuid + "_" + obj.fileName );      ← 추가
            console.log("호출된 파일이름: " + calledPathFileName);      ← 추가

            str += "<li>
                + "    <a href='${contextPath}/fileDownloadAjax?fileName=" + calledPathFileName + "'>"      ← 추가
                + "        <img src='${contextPath}/resources/img/icon-attach.png' alt='No Icon' "
                + "            style='height: 18px; width: 18px;'>
                + "            obj.fileName
                + "        </a>"           ← 추가
                + "</li>";
        } else if (obj.fileType=="I") { //이미지파일인 경우 썸네일 및 원본 파일이름 표시
            //전체 경로명이 포함된 썸네일 파일이름을 encodeURIComponent로 처리
            var thumbnailFilePath =
                encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/s_" + obj.uuid + "_" + obj.fileName);
            console.log("encodeURIComponent 처리 파일이름: " + thumbnailFilePath);

            //원본이미지 파일이름(경로포함)
            var originPathImageFileName = obj.uploadPath + "\\\" + obj.uuid + "_" + obj.fileName;      ← 추가
            originPathImageFileName = originPathImageFileName.replace(new RegExp(/\\"/g),"/");      ← 추가

            str += "<li>
                + "    <a href=\"javascript:showImage('" + originPathImageFileName + "')\">"      ← 추가
                + "        <img src='${contextPath}/displayThumbnailFile?fileName=" + thumbnailFilePath + "'"
                + "            alt='No Icon' style='height: 18px; width: 18px;'>
                + "</li>";
        }
    });
}
```

```

        + "      obj.fileName
        + "    </a>"   ← 추가
        + "</li>";
    }

    fileUploadResult.append(str);
}

```

☞ 이미지 파일에 대한 <a> 작성 시에 명시된 javascript:showImage() 는 앞에서 작성한 이미지를 표시하는 함수입니다.

→ 해상도가 높은 원본 이미지가 웹브라우저에 너무 크게 표시되므로 되는 것을 방지하기 위하여

다음의 css 코드를 <head>와 </head> 사이에 추가합니다. 그리고 업로드 결과가 표시되는 영역에 대한 css 코드도 같이 추가합니다.

```

<head>
<meta charset="UTF-8">
<title>File Upload Ajax Page</title>

<%-- 파일 업로드 표시 영역에 대한 CSS: 문서 실습에서는 사용하지 않으므로 주석처리함 --%><%--
<style>
    .fileUploadResult {width: 100%; background-color: lightgrey; }
    .fileUploadResult ul { display: flex; flex-flow: row; justify-content: center; align-items: center; }
    .fileUploadResult ul li { list-style: none; padding: 5px; }
    .fileUploadResult ul li img { height: 50px; width: auto; max-height: 100px; overflow: hidden" }
</style>--%>

<%-- 다운로드 이미지 표시 영역에 대한 CSS --%>
<style>
    .bigImageWrapper {
        position: absolute;
        display: none;
        justify-content: center;
        align-items: center;
        top:0%;
        width: 100%;
        height: 100%;
        background-color: lightgray;
        z-index: 100;
    }
    .bigImage {
        position: relative;
        display:flex;
        justify-content: center;
        align-items: center;
    }
    .bigImage img { height: 100%; max-width: 100%; width: auto; overflow: hidden }
</style>
</head>

```

☞ 위의 css 코드에 대한 내용은 스스로 인터넷을 찾아 학습해 보시기 바랍니다(어렵지 않습니다).

#### [테스트: 다운로드 및 원본 이미지 표시 테스트]

- servlet-context.xml 변경사항 적용을 위해 이클립스에서 직접 톰캣 서버 재기동
- 웹 브라우저에서, <http://localhost:8080/mypro00/fileUploadAjax> URL 을 요청
- 파일이름에 한글이 포함된 파일들을 업로드 한 후, 브라우저에 표시된 파일아이콘과 썸네일 이미지를 클릭하여, 파일 다운로드 및 원본 이미지 표시가 정상적으로 수행되는지 확인합니다.

다음은 크롬 브라우저에서 테스트 시에 브라우저 표시 내용을 캡처한 그림입니다.

File Upload Ajax Page  
localhost:8080/mypro00/fileUploadAjax

## Upload With Ajax

파일 선택 선택된 파일 없음  
File Upload With Ajax

- 샘플PDF.pdf
- 샘플이미지 img7.jpg

localhost:8080/mypro00/fileDownloadAjax?fileName=C%3A%5Cmyupload%2F2021%5C06%5C26%2F656da10f-976e-402b-bcba-819b1c031ff2\_샘플PDF...

☞ 각 파일의 링크에 마우스를 올리면 브라우저 왼쪽 하단에 링크주소가 표시됩니다.

File Upload Ajax Page  
localhost:8080/mypro00/fileUploadAjax

## Upload With Ajax

파일 선택 선택된 파일 없음  
File Upload With Ajax

- 샘플PDF.pdf
- 샘플이미지 img7.jpg

Javascript:showImage('C:/myupload/2021/06/26/cb21dd00-5a4b-4b22-8384-85999b756c74\_샘플이미지\_img7.jpg')

☞ 샘플 PDF.pdf 의 첨부파일 아이콘을 마우스로 클릭 시에 원본 파일 이름으로 다운로드 됩니다.

File Upload Ajax Page  
localhost:8080/mypro00/fileUploadAjax

## Upload With Ajax

파일 선택 선택된 파일 없음  
File Upload With Ajax

- 샘플PDF.pdf

샘플PDF (12).pdf

☞ 이미지 파일의 썸네일을 클릭하면 원본 이미지가 표시되며, 이를 저장하면, 원본 파일 이름이 정상적으로 자동입력되어 있는 것이 확인됩니다.

File Upload Ajax Page  
localhost:8080/mypro00/fileUploadAjax

File Upload Ajax Page  
localhost:8080/mypro00/fileUploadAjax

## Upload With Ajax

새 탭에서 이미지 열기  
이미지를 다른 이름으로 저장...  
이미지 복사  
이미지 주소 복사  
이 이미지의 QR 코드 생성  
Google에서 이미지 검색  
검사 Ctrl+Shift+I

샘플이미지 img7.jpg

☞ 브라우저에 표시된 원본이미지를 다시 클릭하면, 처음 업로드의 결과화면이 표시됩니다.

☞ Edge/IE 브라우저에서도 동일한 순서로 파일 다운로드 및 원본 이미지 파일 표시를 테스트합니다.

☞ 다음은 다운로드 및 이미지 표시 시에 이클립스 콘솔에 표시된 내용입니다.

#### - 파일다운로드 테스트(크롬브라우저)

```
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 브라우저로 요청에서 전달된 User-Agent 해더정보: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 처리 전 파일이름: C:\myupload\2021\06\26\656da10f-976e-402b-bcba-819b1c031ff2_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - resource :file [C:\myupload\2021\06\26\656da10f-976e-402b-bcba-819b1c031ff2_샘플PDF.pdf]
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 전 resourceFileName :656da10f-976e-402b-bcba-819b1c031ff2_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 후 resourceFileName :샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - Chrome 브라우저입니다
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - Chrome에서의 파일이름: i í PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 브라우저로 요청에서 전달된 User-Agent 해더정보: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 처리 전 파일이름: C:/myupload/2021/06/26/cb21dd00-5a4b-4b22-8384-85999b756c74_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - resource :file [C:\myupload\2021\06\26\cb21dd00-5a4b-4b22-8384-85999b756c74_샘플이미지_img7.jpg]
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 전 resourceFileName :cb21dd00-5a4b-4b22-8384-85999b756c74_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 후 resourceFileName :샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - Chrome 브라우저입니다
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - Chrome에서의 파일이름: i í i é_,í§ _img7.jpg
```

#### - 파일다운로드 테스트(인터넷 익스플로러)

```
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 브라우저로 요청에서 전달된 User-Agent 해더정보: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 처리 전 파일이름: C:/myupload/2021/06/26/a7a1d58b-0f19-4655-849d-8ad66277be87_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - resource :file [C:\myupload\2021\06\26\ a7a1d58b-0f19-4655-849d-8ad66277be87_샘플이미지_img7.jpg]
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 전 resourceFileName :a7a1d58b-0f19-4655-849d-8ad66277be87_샘플이미지_img7.jpg
```

```
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 후 resourceFileName :샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE 또는 엣지 브라우저입니다
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE에서의
파일이름: %EC%83%98%ED%94%8C%EC%9D%B4%EB%AF%B8%EC%A7%80_img7.jpg
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 브라우저로 요청에서 전달된 User-Agent 해더정보: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 처리 전 파일이름: C:\myupload\2021\06\26\589c3a3a-3193-4062-8ddd-bea3f3985d23_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - resource :file [C:\myupload\2021\06\26\589c3a3a-3193-4062-8ddd-bea3f3985d23_샘플PDF.pdf]
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 전 resourceFileName :589c3a3a-3193-4062-8ddd-bea3f3985d23_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 후 resourceFileName :샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE 또는 엣지 브라우저입니다
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE에서의 파일이름: %EC%83%98%ED%94%8CPDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 브라우저로 요청에서 전달된 User-Agent 해더정보: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 처리 전 파일이름: C:\myupload\2021\06\26\589c3a3a-3193-4062-8ddd-bea3f3985d23_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - resource :file [C:\myupload\2021\06\26\589c3a3a-3193-4062-8ddd-bea3f3985d23_샘플PDF.pdf]
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 전 resourceFileName :589c3a3a-3193-4062-8ddd-bea3f3985d23_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 후 resourceFileName :샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE 또는 엣지 브라우저입니다
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE에서의 파일이름: %EC%83%98%ED%94%8CPDF.pdf

- 파일다운로드 테스트(엣지 브라우저)
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 브라우저로 요청에서 전달된 User-Agent 해더정보: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36 Edg/91.0.864.59
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 처리 전 파일이름: C:\myupload\2021\06\26\29b5296c-6a15-418b-98c6-020fd12ff243_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - resource :file [C:\myupload\2021\06\26\29b5296c-6a15-418b-98c6-020fd12ff243_샘플PDF.pdf]
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 전 resourceFileName :29b5296c-6a15-418b-98c6-020fd12ff243_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 후 resourceFileName :샘플PDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE 또는 엣지 브라우저입니다
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE에서의 파일이름: %EC%83%98%ED%94%8CPDF.pdf
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 브라우저로 요청에서 전달된 User-Agent 해더정보: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36 Edg/91.0.864.59
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - 처리 전 파일이름: C:/myupload/2021/06/26/02e4339f-ab4a-4455-819f-25bb1353a276_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - resource :file [C:\myupload\2021\06\26\02e4339f-ab4a-4455-819f-25bb1353a276_샘플이미지_img7.jpg]
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 전 resourceFileName :02e4339f-ab4a-4455-819f-25bb1353a276_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - UUID 제거 후 resourceFileName :샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE 또는 엣지 브라우저입니다
INFO : com.spring5213.mypro00.common.filownload.FileDownloadControllerAjax - IE에서의
파일이름: %EC%83%98%ED%94%8C%EC%9D%B4%EB%AF%B8%EC%A7%80_img7.jpg
```

[참고] HTML 이스케이프문자 <https://namu.wiki/w/URL%20escape%20code>

## 11-5. 서버에 저장된 첨부파일의 삭제

☞ 예제에서는 업로드 후, 표시된 결과에서 업로드 된 파일을 삭제하는 것을 구현합니다.  
따라서, 삭제 관련 기능은 업로드 기능을 수행하는 컨트롤러와 JSP 페이지에 구현합니다.

- ☞ 서버에 업로드된 첨부파일의 삭제를 구현할 때는 다음의 상황을 고려해야 합니다.
  - 이미지 파일의 경우에는 썸네일까지 같이 삭제되어야 합니다.
  - 업로드 된 첨부파일을 삭제한 후에는 결과에 표시된 썸네일이나 파일 아이콘 등의 정보가 삭제되어야 합니다.
  - 비정상적으로 브라우저가 종료되었을 때 업로드 된 파일을 어떻게 처리할 지 결정해야 합니다.
- ☞ 업로드 된 첨부파일은 Ajax 또는 <form> 태그를 이용하여 삭제할 수 있습니다.
- ☞ 이미지가 아닌 일반파일은 업로드된 파일만 삭제하면 되지만,  
이미지 파일의 경우에는 같이 저장된 썸네일 파일과 원본 파일을 같이 삭제해야 합니다.
- ☞ 삭제 요청을 받은 컨트롤러는, 삭제하려는 파일의 유형을 검사해서 일반 파일인지  
이미지 파일인지를 파악하여 파일을 삭제시키도록 구현합니다.

### (1) 파일 삭제 처리 컨트롤러 메서드 구현

→ src/main/java/com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax.java 클래스를 코드 작성 뷰에 오픈  
제일 마지막에 파일 삭제를 처리하는 다음의 deleteFile() 메서드를 작성 후 저장합니다.

```
import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;

//서버에 업로드 된 파일 삭제
@PostMapping("/deleteUploadedFile")
@ResponseBody
public ResponseEntity<String> deleteFile(String fileName, String fileType) {
    logger.info("deleteFileName: " + fileName);
    logger.info("deleteFileType: " + fileType);

    //File 객체 생성
    File delFile;

    try {
        //전체 경로명이 포함된 파일이름을 UTF-8로 디코딩하여 파일 객체를 생성합니다.
        delFile = new File(URLDecoder.decode(fileName, "UTF-8"));
        logger.info("decoded deleting fileName: " + delFile);

        //파일객체의 delete()메서드로 파일(썸네일파일과 일반파일)을 삭제합니다.
        delFile.delete();

        //삭제파일의 유형이 image 값인 경우
        if (fileType.equals("I")) {
            //삭제된 파일의 파일객체에서 s_가 삭제된 파일이름을 얻음
            String originalImageFileName = delFile.getAbsolutePath().replace("s_", "");
            logger.info("largeFileName: " + originalImageFileName);

            //s_ 가 삭제된 파일이름으로 이미지파일 이름의 객체를 얻어서
            delFile = new File(originalImageFileName);
            //원본 이미지파일 삭제합니다.
            delFile.delete();
        }
    }
```

```

} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
    return new ResponseEntity<Object>(HttpStatus.NOT_FOUND);
}

// 삭제 후, 파일이 삭제되었다는 메시지를 전달합니다.
return new ResponseEntity<String>("SuccessDeletingFile", HttpStatus.OK);
}

```

☞ 위의 `deleteFile()` 메서드는 파라미터를 통해 파일이름과 파일의 종류를 텍스트로 전달받아, 썸네일과 일반파일의 경우에는 파일 삭제를 수행하고, 삭제되는 파일이 썸네일인 경우, 추가적으로 `s_`를 제거한 파일이름으로 원본 이미지파일을 삭제하도록 구현되었습니다.

## (2) 업로드 된 첨부파일 삭제 처리에 대한 피드백 반영(JSP 페이지)

☞ 첨부파일 삭제 요청 및 피드백은, 업로드 결과 화면에서 처리됩니다. 따라서, `fileUploadAjax.jsp` 페이지에 다음의 요구사항대로 코드를 수정 및 추가합니다.

→ `src/main/webapp/WEB-INF/views/fileUploadAjax.jsp` 페이지를 코드 작성부에 오픈  
 → 업로드 된 결과를 표시하는 `showUploadedFiles()` 함수의 `li` 요소에 [삭제] 버튼을 추가합니다.  
 이 때, HTML 의 `data-xxx` 속성을 추가하여, 문자열로 파일이름(경로포함)과 파일타입을 지정합니다(빨간색 코드 추가).

```

//업로드 결과 표시 함수
function showUploadedFiles(uploadResult) {
    console.log("showUploadedFiles 실행: 화면 표시 시작=====");

    //서버로부터 전달된 파일업로드 결과내용이 없으면, 함수 종료.
    if(!uploadResult || uploadResult.length == 0){
        return ;
    }
    //ul 태그 변수화
    var fileUploadResult = $(".fileUploadResult ul");
    var str = "";
    //전달받은 배열형식 데이터 각각에 대하여
    $(uploadResult).each(function(i, obj) {
        if (obj.fileType=="F") { //이미지가 아닌 경우, 아이콘 이미지 및 원본 파일이름 표시
            var calledPathFileName = encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/" +
                obj.uuid + "_" + obj.fileName);
            console.log("호출된 파일이름: " + calledPathFileName);

            str += "<li>" +
                "    <a href='${contextPath}/fileDownloadAjax?fileName=" + calledPathFileName + "'>" +
                "        <img src='${contextPath}/resources/img/icon-attach.png' alt='No Icon' " +
                "            style='height: 18px; width: 18px;'>" +
                "        obj.fileName
                "    </a>" +
                "    //data-filename:경로포함 파일이름, data-filetype: 'F' 설정
                "    <span data-filename='" + calledPathFileName + "' data-filetype='F'>[삭제]</span>" ←추가
                "    </li>";
        } else if (obj.fileType=="I") { //이미지파일인 경우 썸네일 및 원본 파일이름 표시
            //전체 경로명이 포함된 썸네일 파일이름을 encodeURIComponent로 처리
            var thumbnailFilePath =
                encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/s_" + obj.uuid + "_" + obj.fileName);
            console.log("encodeURIComponent 처리 파일이름: " + thumbnailFilePath);
        }
    });
}

```

```

//원본이미지 파일이름(경로포함)
var originPathImageFileName = obj.repoPath + "/" + obj.uploadPath + "/" + obj.uuid + "_" + obj.fileName;
originPathImageFileName = originPathImageFileName.replace(new RegExp(/\//g), "/");
console.log("originPathImageFileName: " + originPathImageFileName);

str += "<li>
+ "    <a href=\"javascript:showImage('" + originPathImageFileName + "')\">
+ "        <img src='${contextPath}/displayThumbnailFile?fileName=" + thumbnailFilePath + "'"
+ "            alt='No Icon' style='height: 18px; width: 18px;'>
+ "            obj.fileName
+ "    </a>
//data-filename:경로포함 파일이름, data-filetype: 'I' 설정
+ "    <span data-filename='" + thumbnailFilePath + "' data-filetype='I'>[삭제]</span>" ←추가
+ "</li>";
}
});

fileUploadResult.append(str); //fileUploadResult 클래스의 div에 있는 ul에 결과를 HTML로 추가
}

```

→ 앞에서 추가한 업로드 파일의 [삭제] 링크 클릭 이벤트 함수를 제일 마지막에 추가합니다.

```

//파일 삭제:이벤트 위임을 이용하여, 서버의 파일 삭제 및 브라우저 항목 삭제.
$(".fileUploadResult").on("click", "span", function(e){
    //this: span
    var targetFileName = $(this).data("filename");
    var targetType = $(this).data("filetype");
    console.log(targetFileName);
    console.log(targetType);

    //span이 포함된 li 변수에 저장
    var parentLi = $(this).parent();
    //var targetLi = $(this).closest("li");

    $.ajax({
        url: '${contextPath}/deleteUploadedFile',
        data: {fileName: targetFileName, fileType: targetType},
        dataType: 'text',
        type: 'post',
        success: function(result){
            if (result == "SuccessDeletingFile"){
                alert("파일이 삭제되었습니다.");
                //이벤트 위임을 이용하여, 삭제된 파일의 항목을 브라우저의 HTML 문서에서 삭제합니다.
                //이 때, $(this).parent().remove(); 와 $(this).closest("li").remove(); 는 항목 삭제가 되지 않습니다.
                parentLi.remove();
                //targetLi.remove();

            } else {
                alert("오류: 파일 삭제 실패.");
            }
        }
    }); //End $.ajax
});

```

☞ [삭제] 클릭(<span>요소 안에 있음) 시, 파일 삭제 요청을 컨트롤러에게 보내고, 수행된 결과를 받아서 경고창을 표시한 후, 화면에서 삭제된 파일 정보를 삭제합니다.

☞ 업로드 된 파일을 삭제하는 것은, 업로드 후 새로 생성된 화면 요소를 통해 수행되므로, '이벤트 위임'방식으로 처리합니다. 이벤트 위임(Event Delegation) 이란, 동적으로 생성된 노드를 삭제할 때, 각 노드에 대해 이벤트를 추가 하지 않고, 상위 노드에서 하위 노드의 이벤트를 제어 하는 방식입니다.

▶ 따라서, 클릭 이벤트는 선택된 `<div>` 내의 `<span>` 이 `<div>`의 클릭 이벤트를 위임받아 처리되어, `span` 요소의 `data-file`, `data-type` 속성에 설정된 파일이름(경로포함)과 파일유형 값이 Ajax를 통해 컨트롤러에 전달됩니다.

### [테스트: 첨부파일 삭제 테스트]

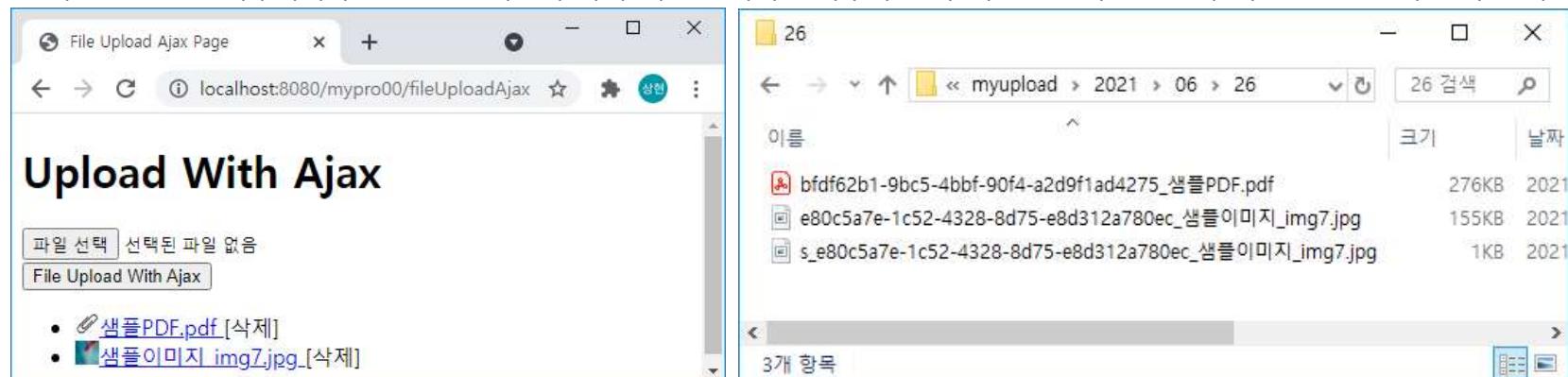
▶ 웹브라우저에서, `http://localhost:8080/mypro00/fileUploadAjax` URL 요청

한글이 포함된 파일들을 업로드 합니다. 그리고 파일탐색기를 실행하여 업로드된 파일이 저장된 디렉토리를 표시합니다.

각 파일의 삭제 버튼을 클릭하면서, 파일 탐색기에서 파일삭제 여부를 확인하고, 브라우저에서 해당 항목이 사라지는지 확인합니다. 이미지 파일의 경우, 썸네일이미지도 같이 삭제 되어야 합니다.

▶ 동일한 테스트를 Edge 브라우저와 IE 브라우저에서도 수행하여 정상적으로 동작하는지 확인합니다.

▶ 다음은 크롬 브라우저에서 업로드 된 파일의 삭제 테스트 시에 브라우저 표시 내용을 캡처한 그림과 이클립스 콘솔에 표시된 내용입니다.



File Upload Ajax Page

localhost:8080/mypro00/fileUploadAjax

## Upload With Ajax

파일 선택 선택된 파일 없음

File Upload With Ajax

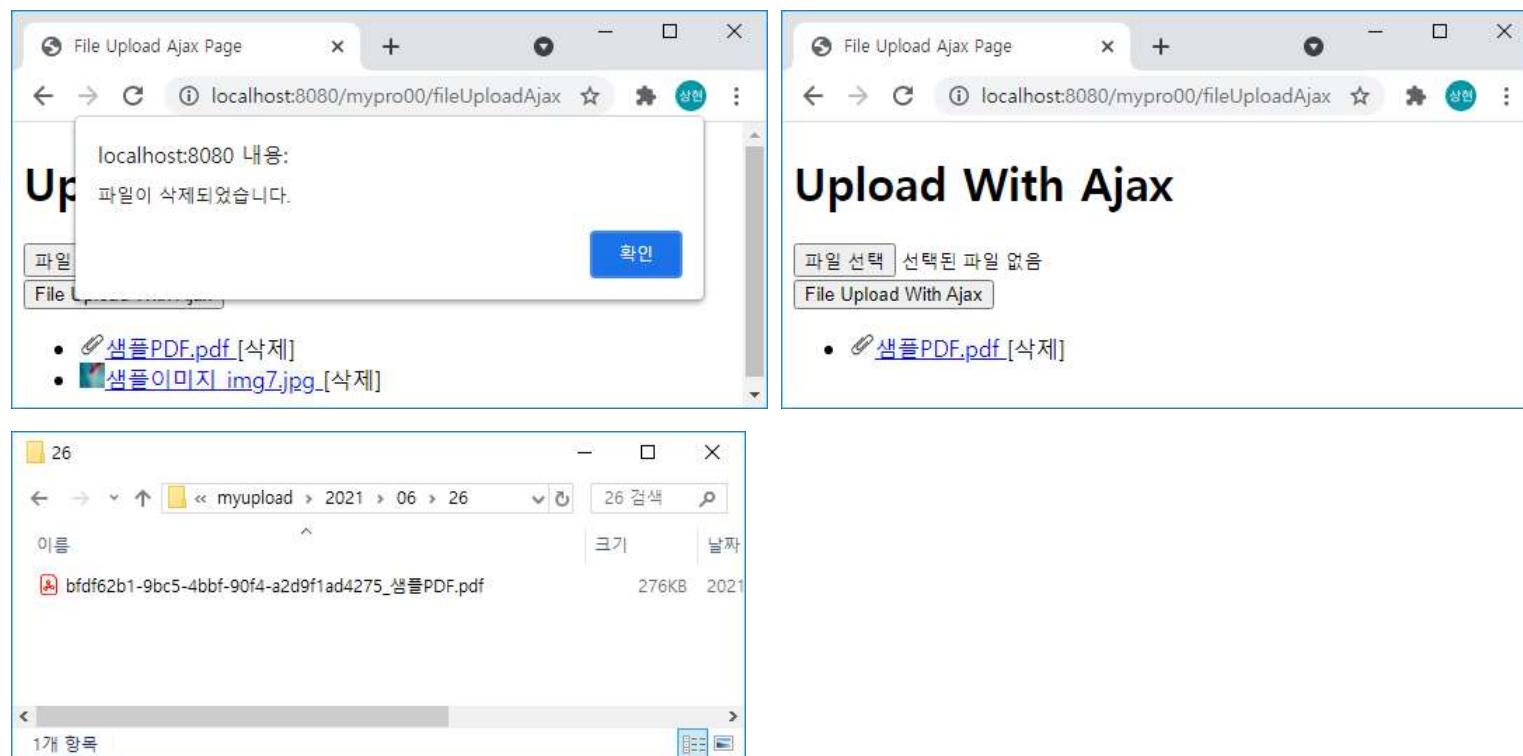
- 샘플PDF.pdf [삭제]
- 샘플이미지 img7.jpg [삭제]

File Explorer:

- bfd62b1-9bc5-4bbf-90f4-a2d9f1ad4275\_샘플PDF.pdf (276KB)
- e80c5a7e-1c52-4328-8d75-e8d312a780ec\_샘플이미지\_img7.jpg (155KB)
- s\_e80c5a7e-1c52-4328-8d75-e8d312a780ec\_샘플이미지\_img7.jpg (1KB)

Eclipse Console:

```
rm -rf myupload/26/*
```



File Upload Ajax Page

localhost:8080/mypro00/fileUploadAjax

## Upload With Ajax

파일 선택 선택된 파일 없음

File Upload With Ajax

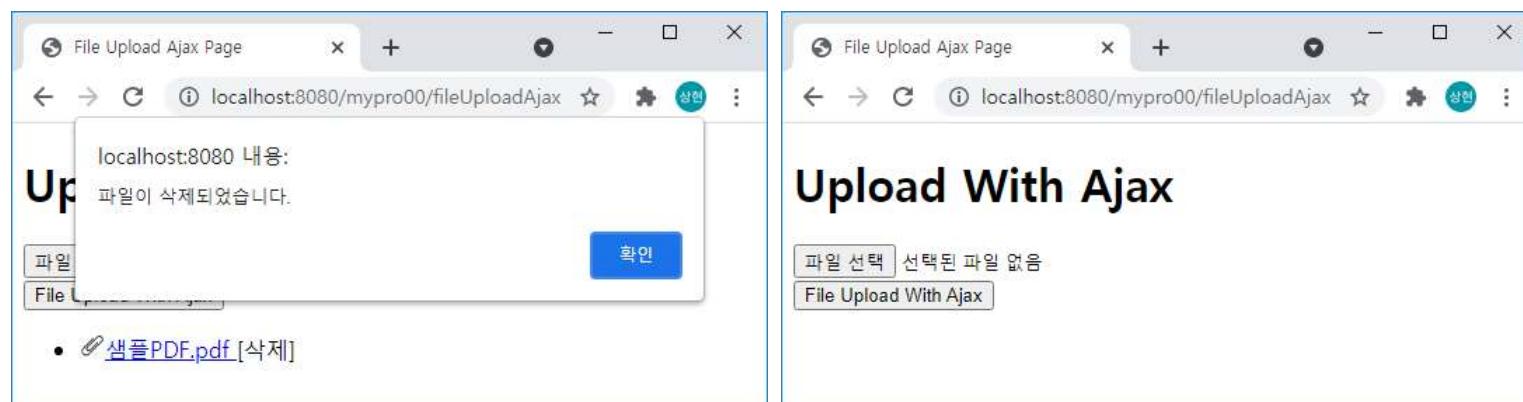
- 샘플PDF.pdf [삭제]

File Explorer:

- bfd62b1-9bc5-4bbf-90f4-a2d9f1ad4275\_샘플PDF.pdf (276KB)

Eclipse Console:

```
rm -rf myupload/26/*
```



File Upload Ajax Page

localhost:8080/mypro00/fileUploadAjax

## Upload With Ajax

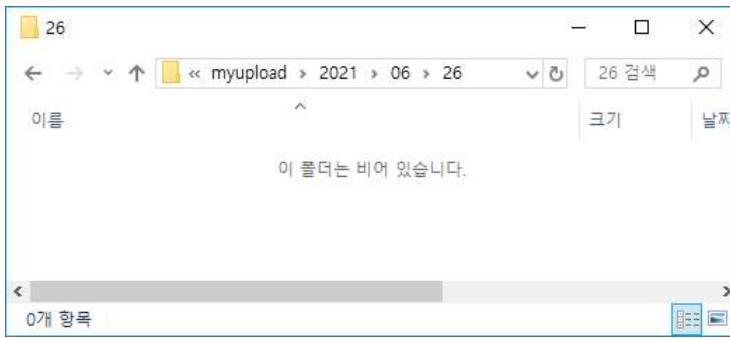
파일 선택 선택된 파일 없음

File Upload With Ajax

- 샘플PDF.pdf [삭제]

Eclipse Console:

```
rm -rf myupload/26/*
```



☞ 다음은 파일 삭제 시, 이클립스 콘솔 창에 표시된 로그기록입니다.

```
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - deleteFileName: C%3A%5Cmyupload%2F2021%5C06%5C26%2Fs_e80c5a7e-1c52-4328-8d75-e8d312a780ec_%EC%83%98%ED%94%8C%EC%9D%B4%EB%AF%B8%EC%A7%80_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - deleteFileType: I
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - decoded deleting fileName: C:\myupload\2021\06\26\s_e80c5a7e-1c52-4328-8d75-e8d312a780ec_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - largeFileName: C:\myupload\2021\06\26\e80c5a7e-1c52-4328-8d75-e8d312a780ec_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - deleteFileName: C%3A%5Cmyupload%2F2021%5C06%5C26%2Fbfdf62b1-9bc5-4bbf-90f4-a2d9f1ad4275_%EC%83%98%ED%94%8CPDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - deleteFileType: F
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - decoded deleting fileName: C:\myupload\2021\06\26\bfdf62b1-9bc5-4bbf-90f4-a2d9f1ad4275_샘플PDF.pdf
```

#### 11-6. 첨부파일 구현 실습: 프로젝트에 적용 - 게시물 등록 시의 첨부파일 처리

☞ 기존 mypro00 프로젝트에 첨부파일 기능을 추가하는 작업을 진행합니다.

첨부파일은 게시물의 등록, 조회, 및 수정/삭제 화면에서 처리할 필요가 있으므로, 각 단계마다 각각 구현을 진행합니다.

### (1) 영속 계층(DB 및 DB Access 계층) 구현: 테이블 생성

☞ 첨부파일과 관련된 데이터들이 저장될 테이블을 데이터베이스에 생성합니다.

`book_ex` 계정으로 데이터베이스에 접속된 `SQL*Developer`에서 다음의 문장을 실행하여 테이블을 생성합니다

```
--첨부파일 정보 저장을 위한 테이블 생성  
CREATE TABLE book_ex.tbl_myAttachFiles (  
    uuid VARCHAR2(300) NOT NULL,          --첨부파일 이름에 포함된 UUID(기본키 컬럼)  
    uploadPath VARCHAR2(250) NOT NULL,        --첨부파일이 업로드 되어 저장된 경로  
    fileName VARCHAR2(100) NOT NULL,           --첨부파일 이름  
    filetype CHAR(1) DEFAULT 'I',            --파일유형(I: 이미지파일, F:일반파일)  
    bno NUMBER(10,0)                            --첨부파일과 관련된 게시물번호  
) TABLESPACE users;
```

-- 기본키 제약조건 추가

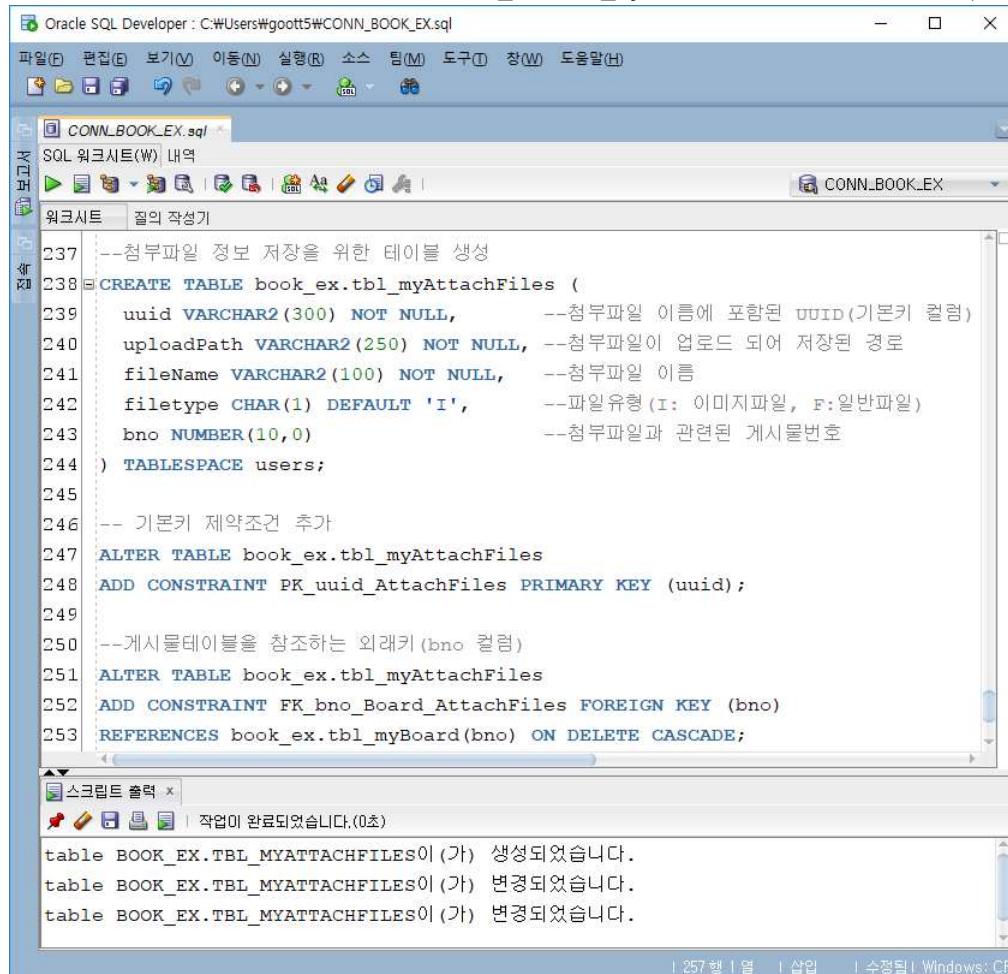
```
ALTER TABLE book_ex.tbl_myAttachFiles  
ADD CONSTRAINT PK_uuid_AttachFiles PRIMARY KEY (uuid) ;
```

--게시물테이블을 참조하는 외래키(bno 컬럼)

```
ALTER TABLE book_ex.tbl_myAttachFiles  
ADD CONSTRAINT FK_bno_Board_AttachFiles FOREIGN KEY (bno)  
REFERENCES book_ex.tbl_myBoard(bno) ON DELETE CASCADE ;
```

☞ 시퀀스 값을 기본키로 사용하는 다른 테이블(tbl\_myBoard/tbl\_myreply)과 달리, book\_ex.tbl\_myAttachFiles 테이블에서는 첨부파일의 처리 중에 생성되는 고유한 UUID를 기본키로 사용하도록 구성합니다.

특정 게시물에 대한 첨부파일이 존재하므로, book\_ex.tbl\_myAttachFiles 테이블에 해당 게시물을 참조할 bno 컬럼을 정의하고, bno 컬럼에 외래키를 정의하여 book\_ex.tbl\_myBoard 테이블의 bno 컬럼(PK 컬럼)을 참조하도록 구성합니다.



## (2) 영속 계층(DB 및 DB Access 계층) 구현: MyBoardAttachFileVO 클래스 생성

→ src/main/java/com.spring5213.mypro00.domain 패키지에 book\_ex.tbl\_myAttachFiles 테이블의 데이터를 처리할 MyBoardAttachFileVO 클래스를 생성하고, 다음의 코드를 작성합니다.

```
package com.spring5213.mypro00.domain;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class MyBoardAttachFileVO {
    private String uuid;
    private String uploadPath;
    private String fileName;
    private String fileType;
    private Long bno;
    private String repoPath= "C:\\myupload"; //서버의 첨부파일 기본 저장경로(C:\\myupload)
                                            //데이터베이스에는 저장되지 않습니다.

}
```

## (3) 영속 계층(DB 및 DB Access 계층) 구현: MyBoardVO 클래스 수정

☞ 기존 MyBoardVO 에 게시물 등록 시에 첨부파일들의 등록도 같이 처리될 수 있도록 첨부파일 데이터를 저장할

List<MyBoardAttachFileVO> 유형의 필드를 추가합니다.

→ com.spring5213.mypro00.domain 패키지의 MyBoardVO 클래스에 위의 내용을 반영하여 다음처럼 수정합니다.

```
package com.spring5213.mypro00.domain;

import java.sql.Timestamp;
import java.util.Date;
import java.util.List;

import lombok.Data;

@Data
public class MyBoardVO {
    private long bno ;
    private String btitle ;
    private String bcontent ;
    private String bwriter ;
    private int bviewsCnt ;
    private int breplyCnt ;
    private int bdelFlag ; //1: 삭제요청됨, 0: 유지
    private Date bregDate ;
    private Timestamp bmodDate ;
    private List<MyBoardAttachFileVO> attachFileList ; ← 새로 추가

}
```

#### (4) 영속 계층(DB 및 DB Access 계층) 구현: SQL 매퍼 XML 파일 생성

→ book\_ex.tbl\_MyAttachFiles 테이블에 대한 MyBoardAttachFileMapper.xml SQL 매퍼 파일을

src/main/resources/com/spring5213/mypro00/mapper 폴더에 생성하고, 다음의 SQL 문들을 작성합니다.

- 첨부파일을 추가하는 INSERT 문
- uuid 를 이용하여 특정 첨부파일 하나를 삭제하는 DELETE 문
- 특정 게시물의 모든 첨부파일을 조회하는 SELECT 문
- 특정 게시물의 모든 첨부파일을 삭제하는 DELETE 문

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<!-- MapperInterface 사용 -->
<mapper namespace="com.spring5213.mypro00.mapper.MyBoardAttachFileMapper">

<!--첨부파일 추가 -->
<insert id="insertAttachFile">
  <![CDATA[
    INSERT INTO book_ex.tbl_myAttachFiles
    VALUES ( #{uuid}, #{uploadPath}, #{fileName}, #{fileType}, #{bno} )
  ]]>
</insert>

<!--특정 첨부파일 삭제 -->
<delete id="deleteAttachFile">
  <![CDATA[
    DELETE FROM book_ex.tbl_myAttachFiles WHERE uuid = #{uuid}
  ]]>
</delete>

<!-- 특정 게시물의 모든 첨부파일 조회 -->
<!-- 서버의 첨부파일 기본저장 경로는 리터럴 문자열로 처리 -->
<select id="selectAttachFilesByBno" resultType="com.spring5213.mypro00.domain.MyBoardAttachFileVO">
  <![CDATA[
    SELECT uuid, uploadPath, fileName, fileType, bno
    FROM book_ex.tbl_myAttachFiles
    WHERE bno = #{bno}
  ]]>
</select>

<!-- 특정 게시물의 모든 첨부파일 삭제 -->
<delete id="deleteAttachFilesByBno">
  <![CDATA[
    DELETE FROM book_ex.tbl_myAttachFiles WHERE bno = #{bno}
  ]]>
</delete>

</mapper>
```

☞ 첨부파일에 대해서는 새로운 첨부파일 추가와 기존 첨부파일에 대한 조회 및 삭제 작업만 처리합니다.

게시물의 첨부파일 수정은, 새로운 첨부파일을 추가하거나 또는 기존 첨부파일을 삭제하거나 또는 기존 첨부파일을 삭제하고 새로운 첨부파일을 추가하는 작업으로 처리됩니다. 즉, 첨부파일에 대해서는 UPDATE 라는 처리가 없습니다.

## (5) 영속 계층(DB 및 DB Access 계층) 구현: Mapper-인터페이스 생성

☞ MyBoardAttachFileMapper.xml SQL 매퍼 파일의 SQL 문들을 사용할 수 있는 MyBoardAttachFileMapper 매퍼 인터페이스를 생성합니다.

→ src/main/java/com.spring5213.mypro00.mapper 패키지에 MyBoardAttachFileMapper 인터페이스를 생성하고,  
다음의 코드를 작성합니다.

```
package com.spring5213.mypro00.mapper;

import java.util.List;

import com.spring5213.mypro00.domain.MyBoardAttachFileVO;

public interface MyBoardAttachFileMapper {

    public void insertAttachFile(MyBoardAttachFileVO boardAttachFile);

    public void deleteAttachFile(String uuid);

    public List<MyBoardAttachFileVO> selectAttachFilesByBno(Long bno);

    public void deleteAttachFilesByBno(Long bno);

}
```

## (6) 영속 계층(DB 및 DB Access 계층) 구현: mybatis-context.xml 파일 설정

☞ 새로 추가한 MyBoardAttachFileMapper.xml SQL 매퍼파일의 경로와 이름을 SqlSessionFactoryBean 빈 생성 정의에 있는  
mapperLocations에 추가하고, <mybatis-spring:scan> 태그의 base-package 속성에,  
추가한 MyBoardAttachFileMapper 인터페이스가 존재하는 패키지를 설정합니다(이미 설정되어 있음).

→ src/main/webapp/WEB-INF/spring/mybatis-context.xml 파일을 코드 작성부에 오픈 → MyBoardAttachFileMapper.xml SQL 매퍼  
파일과 MyBoardAttachFileMapper 인터페이스가 사용될 수 있도록 다음의 빨간색 코드를 추가합니다

...(생략)...

```
<!-- 4. mybatis-spring 연동 (HikariDataSource 빈 주입 받음) -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mapperLocations">
        <list>
            <value>classpath:com/spring5213/mypro00/mapper/MyBoardMapper.xml</value>
            <value>classpath:com/spring5213/mypro00/mapper/MyReplyMapper.xml</value>
            <value>classpath:com/spring5213/mypro00/mapper/MyBoardAttachFileMapper.xml</value> ←추가
        </list>
    </property>
</bean>

<!-- 5. mybatis-spring 연동 (SqlSessionFactoryBean 빈 주입 받음) -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>

<!-- 매퍼인터페이스 자동 빈 생성 -->
<mybatis-spring:scan base-package="com.spring5213.mypro00.mapper"/> ← 이미 설정되어 있음.

<!-- 스프링 트랜잭션 기능 활성화 -->
<tx:annotation-driven/>
```

```

<!-- Spring Transaction 관리자 빈(dataSource 빈을 주입받음) -->
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
</bean>

</beans>

```

## (7) 첨부파일 등록을 위한 화면 처리(register.jsp 수정)

- ☞ 게시물 등록 시에 첨부파일도 같이 추가할 수 있도록 필요한 요소들을 추가합니다.
- ☞ 첨부파일의 업로드는 Ajax를 이용하며, 첨부파일에 대한 정보는 게시물의 등록 시에 같이 처리됩니다.  
구체적으로, 첨부파일을 <input type='file'> 요소에서 선택하면, 선택과 동시에 Ajax를 통해 첨부파일은 서버에 업로드(저장)되고 서버로부터 첨부파일 정보가 브라우저에 전송되면, 이 첨부파일 정보를 이용해 등록화면에 첨부파일 정보가 표시됩니다.  
게시물의 등록 버튼을 클릭했을 때 게시물 정보가 설정된 form에, 이미 서버에 업로드 된 각 첨부파일의 정보를 각각의 <input type='hidden'> 요소에 설정하고, 이렇게 설정된 input 요소를 게시물 등록 form에 추가하여, 게시물 정보와 함께 서버로 한 번에 전송하는 방식을 사용합니다.
- ☞ 구현 시에, 앞 단원에서 작성한 fileUploadAjax.jsp 파일의 내용을 이용합니다(복사→ 붙여넣기→ 수정).

→ src/main/webapp/WEB-INF/views/myboard/register.jsp 페이지를 코드 작성부에 오픈 → 다음의 순서대로 코드를 작성합니다.

### (7-1) 첨부파일 등록을 위한 HTML 요소 추가

→ 기존 새글 등록의 <div class='row'> 요소 부분의 밑(</div><!-- /#page-wrapper --> 태그 위)에 다음의 빨간색 코드 추가.

...(생략)...

```

<div class="row">
  <div class="col-lg-12">
    <div class="panel panel-default">
      <div class="panel-heading">파일첨부</div>
      <div class="panel-body" id="panelFileBody" style="background-color: #f9f9f9;">← id 추가
        <div class="form-group uploadDiv">
          <input id="inputFile" type="file" name="uploadFiles" multiple><br>
        </div>
        <div class="form-group fileUploadResult">
          <ul style="list-style-type: none; padding-left: 0; margin: 0; border: 1px solid #ccc; border-radius: 5px; background-color: #fff; width: fit-content; margin-left: auto; margin-right: auto; padding: 5px; font-size: 0.8em; font-weight: bold; color: #007bff; text-decoration: none; text-align: center; position: relative; z-index: 1; opacity: 0.9; transition: all 0.3s ease; ">
            <!-- 업로드 후, 업로드 처리결과가 표시될 영역 -->
          </ul>
        </div>
      </div><!-- /.panel-body -->
    </div><!-- /.panel -->
  </div><!-- /.col-lg-12 -->
</div><!-- /.row -->

</div><!-- /#page-wrapper -->

<%@ include file="../myinclude/myfooter.jsp" %>

```

## (7-2) 업로드 후, 표시되는 처리결과에 대하여 CSS 스타일 적용(문서 실습에서는 구현하지 않음)

☞ CSS 코드는, 스타일이 적용되는 다른 HTML 요소들 보다 먼저 선언되어야 하므로 JSP 파일의 윗부분에 작성하는 것을 권장합니다.

→ 아래의 파란색 코드(fileUploadAjax.jsp 파일의 <head> 태그에 작성한 CSS-코드 중 일부)를 page-wrapper 아이디의 div 요소 위에 추가(문서실습에서는 사용하지 않음).

...(생략)...

```
<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<%@ include file="../myinclude/myheader.jsp" %>

<%-- 파일 업로드 표시 역역에 대한 CSS, 결과 표시 CSS 스타일은 실습에서 사용 않음 --%><%--
<style>
.fileUploadResult {width: 100%; background-color: lightgrey; }
.fileUploadResult ul { display: flex; flex-flow: row; justify-content: center; align-items: center; }
.fileUploadResult ul li { list-style: none; padding: 5px; }
.fileUploadResult ul li img { height: 50px; width: auto; max-height: 100px; overflow: hidden" }
</style>--%>

<div id="page-wrapper">

...(생략)...
```

## (7-3) JavaScript 함수 및 jQuery 이벤트 함수 추가

☞ 이 후에 작성되는 코드들은, 앞 단원에서 작성한 fileUploadAjax.jsp 파일의 자바스크립트(jQuery) 코드를 복사하여 register.jsp 페이지의 제일 마지막에 있는 include 지시자 위에 <script> </script>를 생성하여 붙여넣고, 적절하게 수정하여 작성합니다. jQuery 라이브러리는 참조는 myheader.jsp 에 명시되어 있으므로 따로 추가할 필요가 없습니다.

☞ 게시물 등록 시에 첨부파일 등록하는 작업에서는 첨부파일 다운로드나 이미지 확대 보기 같은 기능은 필요하지 않으므로, 해당 부분을 제거하고 코드를 수정해서 구현합니다.

### (7-3-1) 업로드 되는 파일의 크기 및 확장자 제한 검사 함수 추가

→ checkUploadfile() 함수를 복사하여 붙여넣고, 업로드 파일의 최대 파일 크기를 5MB(5242880 Bytes)로 수정합니다.

...(생략)...

```
</div><!-- /.row -->
<script>

//업로드 파일의 확장자 및 최대 파일 크기 검사 함수
function checkUploadfile(fileName, fileSize) {
    // 확장자에 대한 정규식 및 최대 허용크기(5MB)
    var maxSizeAllowed = 5242880;
    var regExpForFileExtention = /(.*\.(exe|sh|zip|alz))$/i ;

    //최대 허용 크기 제한 검사
    if (fileSize >= maxSizeAllowed) {
        alert("업로드 파일의 제한된 크기(5MB)를 초과했습니다.");
        return false;
    }
}
```

```

//업로드파일의 확장자 검사:
if (regExpForFileExtention.test(fileName)) {
    alert("해당 종류(exe/sh/zip/alz)의 파일은 업로드할 수 없습니다.");
    return false;
}
return true;
}

</script>

</div><!-- /#page-wrapper -->

<%@ include file="../myinclude/myfooter.jsp" %>

```

☞ 이 후에 작성되는 함수들은 checkUploadfile() 함수 밑에 하나씩 추가합니다.

#### (7-3-2) 업로드 처리 결과를 표시하는 함수 추가

☞ 게시물 등록 시에는 첨부파일에 대한 다운로드 및 이미지 표시 기능은 필요없으므로, <a> 요소는 삭제합니다.

→ showUploadedFiles() 함수를 복사하여 붙여넣고, 아래의 빨간색 코드처럼 수정합니다.

```

var bnoValue = '<c:out value="${board.bno}" />'; ← 추가

//업로드 후, 첨부파일 정보 표시 함수
function showUploadedFiles(uploadResult) {
    console.log("showUploadedFiles 실행: 화면 표시 시작=====");

    if(!uploadResult || uploadResult.length == 0){ //첨부파일이 없으면, 함수 종료.
        return ;
    }

    //화면표시 시작
    var fileUploadResult = $(".fileUploadResult ul");
    var str = "";

    $(uploadResult).each(function(i, obj) {
        if (obj.fileType=="F") { //이미지가 아닌 경우, 아이콘 이미지 및 원본 파일이름 표시
            var calledPathFileName = encodeURIComponent( obj.repoPath + "/" + obj.uploadPath + "/" +
                obj.uuid + "_" + obj.fileName);
            console.log("호출된 파일이름: " + calledPathFileName);

            str += "<li>" //<a> 와 </a> 태그만 삭제
                + "    <a href='${contextPath}/fileDownloadAjax?fileName=" + calledPathFileName + "'>"
                + "        <img src='${contextPath}/resources/img/icon-attach.png' alt='No Icon' "
                + "            style='height: 18px; width: 18px;'>"
                + "        obj.fileName
                + "    </a>"
                + "    <span data-filename='" + calledPathFileName + "' data-filetype='F'>[삭제]</span>"
                + "</li>";

        } else if (obj.fileType=="I") { //이미지파일인 경우 썸네일 및 원본 파일이름 표시
            //encodeURIComponent로 처리된 전체 경로명이 포함된 썸네일 파일이름
            var thumbnailFilePath =
                encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/s_" + obj.uuid + "_" + obj.fileName);
            console.log("encodeURIComponent 처리 파일이름: " + thumbnailFilePath);

            //전체 경로명이 포함된 원본 이미지 파일이름, 주석처리
            //var originPathImageFileName = obj.repoPath + "/" + obj.uploadPath + "/" + obj.uuid + "_" + obj.fileName;
            //originPathImageFileName = originPathImageFileName.replace(new RegExp(/\//g), "/");
            //console.log("originPathImageFileName: " + originPathImageFileName);
        }
    });
}

```

```

        str += "<li> //<a> 와 </a> 태그만 삭제
        + "     <a href=\"javascript:showImage('" + originPathImageFileName + "')\">
        + "         <img src='${contextPath}/displayThumbnailFile?fileName=" + thumbnailFilePath + "'"
        + "             alt='No Icon' style='height: 18px; width: 18px;'>
        + "             obj.fileName
        + "     </a>
        + "     <span data-filename='" + thumbnailFilePath + "' data-filetype='I'>[삭제]</span>
        + "</li>";
    }
});
fileUploadResult.append(str);
}

```

### (7-3-3) 파일 업로드 요청 구현 및 <input type='file'> 초기화

☞ 첨부파일의 업로드는 별도의 버튼 없이 <input type='file'> 의 내용이 변경되면 자동으로 업로드 되도록, 이벤트 함수로 처리합니다.

→ fileUploadAjax.jsp 파일의 #btnFileUpload 이벤트 함수 코드를 이용하여 다음의 빨간색 코드 추가 및 수정

```

//빈 input 요소를 복사(초기화를 위함)
var cloneInputFile = $(".uploadDiv").clone() ;      ← 추가

//파일 업로드 처리: 파일 input 요소의 내용이 바뀌면 업로드가 자동으로 수행되도록 수정
//업로드 후, .uploadDiv가 복사된 것으로 교체되어 input이 초기화되므로,
//화면에서의 파일 수정(파일 추가, 삭제)은 이벤트 위임 방식을 사용하여 구현
$("#panelFileBody").on("change", "input[type='file']", function(e) { ← 수정

    var formData = new FormData(); //전송될 파일을 저장할 객체 ← 여기서부터 동일하게 추가
    var inputFiles = $("input[name='uploadFiles']");
    var files = inputFiles[0].files; //input 요소의 파일을 변수에 저장
    console.log(files);

    for(var i = 0; i < files.length ; i++) { //FormData 객체에 파일 추가(for문)
        if (!checkUploadfile(files[i].name, files[i].size)) { //업로드 파일 요구조건 검사
            return false;
        }
        formData.append("uploadFiles", files[i]); //uploadFiles 파라미터로 file 정보 추가
    }

    $.ajax({ //파일 전송 및 서버로부터 첨부파일 데이터를 받아, 화면 결과 표시
        url: '${contextPath}/fileUploadAjaxAction',
        processData: false,
        contentType: false,
        data: formData,
        type: 'post',
        dataType: 'json',
        success: function(uploadResult){
            alert("첨부파일의 업로드가 정상적으로 완료되었습니다...");
            $(".uploadDiv").html(cloneInputFile.html()); //input 초기화
            showUploadedFiles(uploadResult); //업로드 결과 표시
        }
    }) // End .ajax
});
```

#### [참고] 최종적인 게시물 등록/수정 요청 시에 첨부파일의 업로드가 같이 수행되도록 구현하지 않는 이유

사용자의 화면에서 업로드할 파일을 선택하는 `<input type="file">` 요소는, 파일 선택이 완료된 상태에서 선택된 파일 정보를 수정(일부 선택 취소)하도록 프로그램으로 구현할 수 없습니다(파일 `input`은 내용 수정이 불가함). 따라서 파일이 선택될 때, Ajax를 통해 선택된 파일을 게시물 등록/수정 보다 먼저 서버에 업로드하여 저장한 후, 서버에 저장된 파일의 정보를 가져와서, 화면에 선택된 파일로서 표시하고, `<input type="file">` 요소가 초기화된 후, 파일의 재선택이 가능하도록 구현함으로써, 사용자가 게시물 등록/수정화면에 표시된 첨부파일의 선택을 취소(선택 파일 삭제→서버의 파일도 삭제됨)할 수 있도록 합니다.

#### (7-3-4) 게시물 등록 전, 첨부파일 삭제 처리

☞ 사용자가 게시물 등록 전, 일부 첨부파일의 [삭제] 클릭 시, 파일을 삭제하고 업로드 결과에서 지우는 이벤트를 처리합니다

→ `fileUploadAjax.jsp` 파일의 `.fileUploadResult` 클릭 이벤트 jQuery 함수 코드를 이용하여, 다음의 코드를 추가합니다.

필요없는 `console.log`를 제외하고, `parentLi`를 `targetLi`를 이용하는 코드로 대체했습니다.

```
//서버의 파일 삭제 및 브라우저 항목 삭제: 이벤트 위임을 이용
$(".fileUploadResult").on("click", "span", function(e){
    var targetFileName = $(this).data("filename");
    var targetType = $(this).data("filetype");

    //var parentLi = $(this).parent("li");
    var targetLi = $(this).closest("li");

    $.ajax({
        url: '${contextPath}/deleteUploadedFile',
        data: {fileName: targetFileName, fileType: targetType},
        dataType:'text',
        type: 'post',
        success: function(result){
            if (result == "SuccessDeletingFile"){
                //alert("파일이 삭제되었습니다.");
                //이벤트 위임을 이용하여, 삭제된 파일의 항목을 브라우저의 HTML 문서에서 삭제합니다.
                //parentLi.remove()
                targetLi.remove();
            } else {
                alert("오류: 파일 삭제 실패.");
            }
        }
    });
}); //End $.ajax
});
```

#### (8) 게시물 및 첨부파일 등록

☞ `register.jsp` 수정 → `MyBoardServiceImpl.java` 순으로 수정합니다.

☞ 첨부파일 정보를 `MyBoardVO`에 필드추가를 통해 구현했으므로, 실제로 컨트롤러에서의 추가 구현은 없습니다.

문서에서는 사용자 브라우저로부터 첨부파일 정보가 컨트롤러로 잘 전달되는지 확인하기 위하여 필요한 로그를 콘솔에 기록하는 코드를 `MyBoardController.java`에 추가합니다. 최종 구현 후, 추가된 코드는 삭제됩니다.

☞ 첨부파일의 업로드는 Ajax를 이용하여 이미 완료된 상태이므로, 게시물의 등록을 위한 `submit` 버튼이 클릭 될 때, 게시물 및 첨부파일의 정보(파일이름, uuid, 저장경로 등)가 컨트롤러에게 전달되도록 구현합니다.

## (8-1) 화면에서의 전송 데이터 준비

첨부파일 결과가 표시되는 <li> 요소에 HTML의 data-xxx 속성들을 이용하여, 업로드 후에 서버로부터 AttachInfoDTO 객체들의 리스트를 통해 전달된 첨부파일 정보들이 li 요소에 저장되도록 지정합니다.

→ src/main/webapp/WEB-INF/views/myboard/register.jsp 페이지를 코드 작성부에 오픈

→ showUploadedFiles() 함수에 다음의 빨간색 코드를 추가합니다.

```
//업로드 후, 첨부파일 정보 표시 함수
function showUploadedFiles(uploadResult) {
    console.log("showUploadedFiles 실행: 화면 표시 시작=====");

    if(!uploadResult || uploadResult.length == 0){
        return ;
    }

    var fileUploadResult = $(".fileUploadResult ul");
    var str = "";

    $(uploadResult).each(function(i, obj) {

        if (obj.fileType=="F") { //이미지가 아닌 경우, 아이콘 이미지 및 원본 파일이름 표시
            var calledPathFileName = encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/" +
                obj.uuid + "_" + obj.fileName);
            console.log("호출된 파일이름: " + calledPathFileName);

            str += "<li data-uploadpath='" + obj.uploadPath + "' data-uuid='" + obj.uuid + "'"
                + " data-filename='" + obj.fileName + "' data-filetype='" + obj.fileType + "'>" ← 추가
                + " <img src='${contextPath}/resources/img/icon-attach.png' alt='No Icon' "
                + " style='height: 18px; width: 18px;'>"
                + " " + obj.fileName
                + " <span data-filename='" + calledPathFileName + "' data-filetype='F'>[삭제]</span>"
                + "</li>";

        } else if (obj.fileType=="I") { //이미지파일인 경우 썸네일 및 원본 파일이름 표시

            var thumbnailFilePath =
                encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/s_" + obj.uuid + "_" + obj.fileName);
            console.log("encodeURIComponent 처리 파일이름: " + thumbnailFilePath);

            //전체 경로명이 포함된 원본 이미지 파일이름, 주석처리
            //var originPathImageFileName = obj.repoPath + "/" + obj.uploadPath + "/" + obj.uuid + "_" + obj.fileName;
            //originPathImageFileName = originPathImageFileName.replace(new RegExp(/\\"/g), "/");
            //console.log("originPathImageFileName: " + originPathImageFileName);

            str += "<li data-uploadpath='" + obj.uploadPath + "' data-uuid='" + obj.uuid + "'"
                + " data-filename='" + obj.fileName + "' data-filetype='" + obj.fileType + "'>" ← 추가
                + " <img src='${contextPath}/displayThumbnailFile?fileName=" + thumbnailFilePath + "'"
                + " alt='No Icon' style='height: 18px; width: 18px;'>"
                + " " + obj.fileName
                + " <span data-filename='" + thumbnailFilePath + "' data-filetype='I'>[삭제]</span>"
                + "</li>";
        }
    });
    //end $(uploadResult).each(function(){}
    fileUploadResult.append(str);
}
```

☞ 다음은 게시물 등록 버튼(submit 버튼) 클릭 이벤트 jQuery 함수를 추가합니다. 추가된 함수는 다음의 기능이 처리됩니다.

- 기존 submit 버튼의 전송 기능이 실행되지 않도록 기능을 해제시킵니다.
- 첨부파일 표시 결과가 담긴 <li> 요소의 data-xxx 속성들의 값을, 게시물 정보가 담긴 <form> 요소에 <input type='hidden'>으로 추가합니다.
- 첨부파일 정보가 추가된 <form> 요소를 서버의 컨트롤러에게 전달합니다.

→ 다음의 코드를 register.jsp 페이지의 제일 마지막에 추가합니다.

```
//게시물 및 첨부파일 정보 등록
$("button[type='submit']").on("click", function(e){

    e.preventDefault(); //submit 버튼의 원래 동작 막음
    var formObj = $("form[role='form']");
    var strInputHidden = "";

    //업로드 결과의 li 요소 각각에 대하여 다음을 처리(이벤트 위임 이용)
    $(".fileUploadResult ul li").each(function(i, obj){
        var objLi = $(obj);
        //name 속성 값은 MyBoardVO의 첨부파일 목록이 저장되는 attachFileList 필드명(List 객체)으로 지정합니다.
        strInputHidden
            += " <input type='hidden' name='attachFileList["+i+"].uuid' value='"+objLi.data("uuid")+"'>"
            + " <input type='hidden' name='attachFileList["+i+"].uploadPath' value='"+objLi.data("uploadpath")+"'>"
            + " <input type='hidden' name='attachFileList["+i+"].fileName' value='"+objLi.data("filename")+"'>"
            + " <input type='hidden' name='attachFileList["+i+"].fileType' value='"+objLi.data("filetype")+"'>";
    });

    console.log(strInputHidden); //테스트 후, 주석처리할 것
    formObj.append(strInputHidden);
    formObj.submit();

});

</script>

<%@ include file="../myinclude/myfooter.jsp" %>
```

☞ 함수를 통해 전송된 <form>의 값들(새로운 게시물과 첨부파일 데이터들)은 컨트롤러로 전달됩니다(MyBoardVO)

첨부파일의 데이터는, MyBoardVO에 새로 추가된 List<MyBoardAttachFileVO> 타입의 attachFileList 필드를 통해 컨트롤러에게 전달됩니다.

#### (8-2) MyBoardController 수정(첨부파일 정보 전달 상태 확인 로그 추가)

☞ <form>을 통해 게시물 정보와 함께 전송된 첨부파일의 정보는, MyBoardController의 registerNewBoard() 메소드에 전달되어 처리됩니다. 다음은 기존 BoardControllerImpl의 registerArticle() 메소드입니다.

```
//게시물 등록 - 처리
@PostMapping("/register")
public String registerNewBoard(MyBoardVO myBoard, RedirectAttributes redirectAttr) {
    log.info("컨트롤러 - 게시물 등록: " + myBoard);

    long bno = myBoardService.registerBoard(myBoard);
    log.info("등록된 게시물의 bno: " + bno );

    redirectAttr.addFlashAttribute("result", bno);
    return "redirect:/myboard/list"; //리다이렉트 방식으로 URL 호출
}
```

→ 위의 메소드에 다음처럼 로그기록 코드(빨간색 코드)를 추가하여, MyboardVO 매개변수로 게시물과 첨부파일 정보가 정상적으로 전달되었는지 확인합니다.

```
//게시물 등록 - 처리
@PostMapping("/register")
public String registerNewBoard(MyBoardVO myBoard, RedirectAttributes redirectAttr) {

    log.info("컨트롤러 - 게시물등록(전달된 VO): " + myBoard.toString()); ←수정

    System.out.println("===== attachInfo =====");
    if (myBoard.getAttachFileList() != null) {
        myBoard.getAttachFileList()
            .forEach(attachFile -> System.out.println("첨부 파일 정보:" + attachFile.toString()));
    }
    System.out.println("===== ===== =====");

    long bno = myBoardService.registerBoard(myBoard);
    log.info("등록된 개시물의 bno: " + bno);

    redirectAttr.addFlashAttribute("result", bno);

    return "redirect:/myboard/list";
}
```

### (8-3) MyBoardServiceImpl 설정

☞ 게시물 등록 작업은 하나지만, 내부적으로 2 개의 데이터베이스 테이블에 대한 입력 및 저장 작업이 포함됩니다.

- TBL\_MYBOARD 테이블에 게시물 정보 입력 및 저장
- TBL\_MYATTACHFILES 테이블에 게시물과 관련된 첨부파일 정보 입력 및 저장

☞ 댓글구현에서처럼 게시물 등록 작업을 처리하는 비즈니스 계층의 서비스 구현 클래스에 트랜잭션으로 처리되도록 필요한 처리를 구현해야 합니다.

☞ MyBoardAttachFileMapper 인터페이스 유형의 필드를 정의하고 자동 주입되도록 구현한 후, 게시물 등록을 처리하는 registerBoard() 메서드에, book\_ex.TBL\_MYBOARD 테이블과 book\_ex.TBL\_MYATTACHFILES 테이블에 대한 입력작업이 트랜잭션 단위로 수행되도록 수정합니다.

→ src/main/java/com.spring5213.mypro00.service.MyBoardServiceImpl.java 구현클래스를 코드 작성뷰에 오픈

→ 다음의 빨간색 코드를 추가

```
package com.spring5213.mypro00.service;

import org.springframework.transaction.annotation.Transactional; ← Ctrl + Shift + O로 자동 완성됨.
import com.spring5213.mypro00.mapper.MyBoardAttachFileMapper;

...(생략)...

@Service
@AllArgsConstructor
@Log4j
public class MyBoardServiceImpl implements MyBoardService {

    //MyBoard 매퍼 인터페이스 주입: 생성자를 이용한 자동 주입
    private MyBoardMapper myBoardMapper;

    //MyBoardAttachFileMapper 매퍼 인터페이스 주입: 생성자를 이용한 자동 주입
}
```

```

private MyBoardAttachFileMapper myBoardAttachFileMapper ; ← 추가

...(생략)...

//게시물 등록:selectKey이용
@Transactional ← 추가
@Override
public long registerBoard(MyBoardVO myBoard) {
    log.info("MyBoardService.registerBoard()에 전달된 MyBoardVO: " + myBoard);

    myBoardMapper.insertMyBoardSelectKey(myBoard);
    System.out.println("MyBoardService에서 등록된 게시물의 bno: " + myBoard.getBno());

    //첨부파일이 없는 경우, 메서드 종료
    if (myBoard.getAttachFileList() == null || myBoard.getAttachFileList().size() <= 0) {
        return myBoard.getBno() ;
    } ← 추가

    //첨부파일이 있는 경우, myBoard의 bno 값을 첨부파일 정보 vo에 저장 후, tbl_myAttachFiles 테이블에 입력
    myBoard.getAttachFileList().forEach(attachFile -> {
        attachFile.setBno(myBoard.getBno());
        myBoardAttachFileMapper.insertAttachFile(attachFile);
    });
    ← 추가

    return myBoard.getBno();
}

```

...(생략)...

☞ 게시물의 등록 작업은 tbl\_myBoard 테이블과 tbl\_myAttachFiles 테이블 양쪽 모두 insert 가 진행되어야 하기 때문에 트랜잭션으로 처리됩니다(@Transactional 어노테이션 추가).

☞ 새 게시물 정보가 tbl\_myBoard 테이블에 입력된 후, boardVO 의 게시물번호(bno 컬럼값)가 BoardAttachFileVO (attachFile)에 설정된 후, tbl\_myAttachFiles 테이블에 첨부파일 정보가 첨부파일 수만큼 입력됩니다.

[테스트 - 첨부파일을 포함한 게시물 등록 테스트]

→ c:\myupload 폴더에서 temp 폴더를 제외한 모든 다른 이전 실습으로 생성된 폴더와 파일들을 삭제합니다.

→ 템켓 서버를 기동하고, 브라우저에서 http://localhost:8080/mypro00/myboard/list URL 을 요청합니다.

→ 게시물 목록 페이지에서 [새글 등록] 버튼을 클릭하여 첨부파일이 포함된 새로운 게시물을 등록합니다.

→ 테스트 시에 다음의 사항을 확인합니다.

- 파일 탐색기로 업로드 파일의 생성 확인
- SQL\*Developer 에서 book\_ex.tbl\_myattachFiles 테이블에 조회하여 첨부파일 정보 확인

☞ 다음은, 인터넷 익스플로리에서 게시물 등록 페이지이동 → 첨부파일 2 개 추가 → 첨부파일 이미지 삭제 → 다시 첨부파일 1 개 추가 후, → 게시물 등록을 수행했을 때, 이클립스 콘솔에 표시된 로그의 일부 내용입니다.

#### 등록페이지 호출

```

INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물 등록 페이지 호출

첨부파일 2개 선택 → 자동 업로드
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====FileUpload With Ajax =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload path: C:\myupload\2021\06\27
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 282604
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.repoPath: C:\myupload
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uploadPath: 2021\06\27

```

```

INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileName: 샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uuid: f993f26e-bbc3-4c32-b7da-6a00b2ff20c0
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: f993f26e-bbc3-4c32-b7da-6a00b2ff20c0_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\27\f993f26e-bbc3-4c32-b7da-6a00b2ff20c0_샘플PDF.pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: application/pdf
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileType: F
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 158112
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.repoPath: C:\myupload
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uploadPath: 2021\06\27
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileName: 샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uuid: bfdaf3f4-cf51-47b4-a236-b569c3e4cb5b
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: bfdaf3f4-cf51-47b4-a236-
b569c3e4cb5b_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\27\bfda..._샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: image/jpeg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileType: I
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 썸네일파일이름(경로포함): C:\myupload\2021\06\27\s_bfdaf3f4-cf51-47b4-
a236-b569c3e4cb5b_샘플이미지_img7.jpg

```

#### 이미지 첨부파일 삭제 → 서버의 이미지 및 썸네일까지 삭제 확인

```

INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - deleteFileName: C%3A%5Cmyupload%2F2021%5C06%5C27%2Fs_bfdaf3f4-cf51-
47b4-a236-b569c3e4cb5b_%EC%83%98%ED%94%8C%EC%9D%B4%EB%AF%8B%EC%A7%80_img7.jpg ←썸네일 삭제
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - deleteFileType: I
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - decoded deleting fileName: C:\myupload\2021\06\27\s_bfdaf3f4-cf51-47b4-
a236-b569c3e4cb5b_샘플이미지_img7.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - largeFileName: C:\myupload\2021\06\27\bfda..._샘플이미지_img7.jpg ← 이미지 삭제

```

#### 다른 이미지 첨부파일 1개 추가 → 자동 업로드

```

INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - ===FileUpload With Ajax ====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - upload path: C:\myupload\2021\06\27
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - =====
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Name: D:\Temp\샘플이미지104.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - Upload File Size: 364030
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.repoPath: C:\myupload
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uploadPath: 2021\06\27
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 수정된 파일이름(strUploadFileName): 샘플이미지104.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileName: 샘플이미지104.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.uuid: c11cf24d-2dfe-4e42-8594-48933408c3a0
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - UUID처리된파일이름: c11cf24d-2dfe-4e42-8594-
48933408c3a0_샘플이미지104.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 저장시 사용되는 파일이름(saveUploadFile, 경로포함):
C:\myupload\2021\06\27\c11cf24d-2dfe-4e42-8594-48933408c3a0_샘플이미지104.jpg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 업로드파일의 ContentType: image/jpeg
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - attachInfoDTO.fileType: I
INFO : com.spring5213.mypro00.common.fileupload.FileUploadControllerAjax - 썸네일파일이름(경로포함): C:\myupload\2021\06\27\s_c11cf24d-2dfe-4e42-
8594-48933408c3a0_샘플이미지104.jpg

```

#### 게시물 등록(게시물 및 첨부파일 정보 데이터베이스 저장)

```

INFO : com.spring5213.mypro00.controller.MyBoardController - 컨트롤러 - 게시물등록(전달된 VO): MyBoardVO(bno=0, btitle=첨부파일 포함 게시물 등록테스트,
bcontent=첨부파일 포함 게시물 등록테스트
첨부파일 추가 삭제도 테스트 함. 잘됨, bwriter=user1, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null,
attachFileList=[MyBoardAttachFileVO(uuid=f993f26e-bbc3-4c32-b7da-6a00b2ff20c0, uploadPath=2021\06\27, fileName=샘플PDF.pdf, fileType=F, bno=null),
MyBoardAttachFileVO(uuid=c11cf24d-2dfe-4e42-8594-48933408c3a0, uploadPath=2021\06\27, fileName=샘플이미지104.jpg, fileType=I, bno=null)])
===== attachInfo =====
첨부 파일 정보:MyBoardAttachFileVO(uuid=f993f26e-bbc3-4c32-b7da-6a00b2ff20c0, uploadPath=2021\06\27, fileName=샘플PDF.pdf, fileType=F, bno=null)
첨부 파일 정보:MyBoardAttachFileVO(uuid=c11cf24d-2dfe-4e42-8594-48933408c3a0, uploadPath=2021\06\27, fileName=샘플이미지104.jpg, fileType=I,
bno=null)
=====

```

```

INFO : com.spring5213.mypro00.service.MyBoardServiceImpl - MyBoardService.registerBoard()에 전달된 MyBoardVO: MyBoardVO(bno=0, btitle=첨부파일 포함
게시물 등록테스트, bcontent=첨부파일 포함 게시물 등록테스트
첨부파일 추가 삭제도 테스트 함. 잘됨, bwriter=user1, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null,
attachFileList=[MyBoardAttachFileVO(uuid=f993f26e-bbc3-4c32-b7da-6a00b2ff20c0, uploadPath=2021\06\27, fileName=샘플PDF.pdf, fileType=F, bno=null),
MyBoardAttachFileVO(uuid=c11cf24d-2dfe-4e42-8594-48933408c3a0, uploadPath=2021\06\27, fileName=샘플이미지104.jpg, fileType=I, bno=null)])
INFO : jdbc.sqlonly - SELECT book_ex.seq_myboard.nextval FROM dual

```

```

INFO : jdbc.sqltiming - SELECT book_ex.seq_myboard.nextval FROM dual
{executed in 3 msec}
INFO : jdbc.resultsettable -
|-----|
|nextval |
|-----|
|458761 |
|-----|

```

```

INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myboard VALUES (458761, '첨부파일 포함 게시물 등록테스트', '첨부파일 포함 게시물 등록테스트 첨부파일
추가 삭제도 테스트 함. 잘됨', 'user1', DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT) ←게시물 입력

```

```

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myboard VALUES (458761, '첨부파일 포함 게시물 등록테스트', '첨부파일 포함 게시물 등록테스트 첨부파일
추가 삭제도 테스트 함. 잘됨', 'user1', DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT)
{executed in 32 msec}

```

MyBoardService에서 등록된 게시물의 bno: 458761

```

INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myAttachFiles VALUES ('f993f26e-bbc3-4c32-b7da-6a00b2ff20c0', '2021\06\27',
'샘플PDF.pdf', 'F', 458761 ) ←첨부파일 입력 저장

```

```

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_myAttachFiles VALUES ('f993f26e-bbc3-4c32-b7da-6a00b2ff20c0', '2021\06\27',
'샘플PDF.pdf', 'F', 458761 )
{executed in 18 msec}
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_myAttachFiles VALUES ('c11cf24d-2dfe-4e42-8594-48933408c3a0', '2021\06\27',

```

'샘플이미지104.jpg', 'I', 458761 ) ←첨부파일 입력 저장

INFO : jdbc.sqltiming - INSERT INTO book\_ex.tbl\_myAttachFiles VALUES ( 'c11cf24d-2dfe-4e42-8594-48933408c3a0', '2021\06\27', '샘플이미지104.jpg', 'I', 458761 )  
{executed in 1 msec}

INFO : com.spring5213.mypro00.controller.MyBoardController - 등록된 개시물의 bno: 458761

...(생략)...

## 11-7. 첨부파일 구현 실습: 프로젝트에 적용 - 게시물 조회 시의 첨부파일 처리

- ☞ 목록 페이지에서 특정 게시물을 조회할 때, 게시물 상세 페이지에서 게시물의 정보 및 첨부파일의 아이콘(또는 썸네일)과 파일이름을 표시해주고, 해당 첨부파일을 다운로드하거나 이미지의 파일의 원본을 확인할 수 있도록 기능을 추가합니다.
- ☞ 특정 게시물의 첨부파일 삭제 및 새로운 첨부파일 추가는 게시물 수정 페이지에 구현됩니다.
- ☞ 게시물의 정보는 `tbl_myBoard` 테이블에 기록되어 있고, 첨부파일의 정보는 `tbl_myAttachFiles` 테이블에 저장되어 있기 때문에 화면에서 두 테이블에 있는 정보를 모두 표시해주어야 합니다.
- ☞ 이를 처리하는 한 방법으로 Ajax를 활용하는 방법이 있습니다. 사용자 화면에 표시하는 부분을 구성하기 위하여 복잡한 자바스크립트를 작성해야 하는 부담이 있지만, 구현이 쉽고 REST-API를 활용하여 서버에 부담을 덜 주기 때문에 많이 활용되는 방법입니다.
- ☞ 다른 방법은 SQL-JOIN SELECT를 이용하여 데이터베이스에 대한 SQL 요청 회수를 줄이며 처리하는 방법이 있습니다. 현재 구성된 `MyBoardVO`에 JOIN SELECT 결과가 저장될 수 있게끔 Mybatis에 resultMap을 추가하고 이를 이용하도록 SQL 매퍼파일에 SELECT 문을 작성하여 구현합니다.
- ☞ 문서 실습에서는 첨부파일 정보를 Ajax를 이용해서 전달받는 방법(REST-API 실습)을 이용합니다.
- ☞ 첨부파일 조회 시에는 업로드 없이, 데이터베이스 정보를 이용하여 조회페이지에 첨부파일 정보가 표시되어야 합니다. 이 때문에 특정 게시물의 첨부파일을 조회하는 SELECT 문에 `C:\myupload` 문자열을 상수처럼 추가하여, 반환되도록 앞에서 SQL 매퍼 파일을 구현했습니다.

### (1) MyBoardService/MyBoardServiceImpl 수정

- ☞ 특정 게시물을 조회할 때, 첨부파일 정도도 같이 조회되어 화면에 표시될 수 있도록 구현합니다. 첨부파일 정보를 Ajax로 브라우저로 전달하려면, 서버에서 JSON 데이터를 만들어서 화면에 올바르게 전송하는 작업을 먼저 구현합니다.
- ☞ `MyBoardAttachFileMapper.xml` 과 `MyBoardAttachFileMapper` 인터페이스에 게시물번호(`bno`)를 이용하여 첨부파일 정보를 `MyBoardAttachFileVO` 타입으로 반환하는 `selectAttachFilesByBno()` 메서드가 정의된 상태입니다.
- ☞ 첨부파일에 대하여 구성된 매퍼를 추가로 이용하여 게시물의 정보와 첨부파일의 정보를 가져오도록 비즈니스 계층의 서비스 클래스에 관련 메서드를 추가합니다.

→ `src/main/java/com.spring5213.mypro00.service` 패키지의 `MyBoardService.java` 인터페이스와 `MyBoardServiceImpl.java` 클래스에 각각 첨부파일 정보를 가져오도록, `selectAttachFilesByBno()` 메서드를 추가합니다.

→ `MyBoardService.java` 인터페이스

```
import com.spring5213.mypro00.domain.MyBoardAttachFileVO; ← Ctrl + Shift + O로 자동 완성됨
```

```
//게시물의 첨부파일 조회  
public List<MyBoardAttachFileVO> getAttachFilesByBno(Long bno);
```

→ MyBoardServiceImpl.java 구현 클래스: myBoardAttachFileMapper 객체는 게시물(첨부파일) 등록 구현 시에 이미 추가했음

```
import com.spring5213.mypro00.domain.MyBoardAttachFileVO; ← Ctrl + Shift + O로 자동 완성됨

//게시물의 첨부파일 조회
@Override
public List<MyBoardAttachFileVO> getAttachFilesByBno(Long bno) {
    log.info("MyBoardService.getAttachFilesByBno()에 전달된 bno:" + bno);
    return myBoardAttachFileMapper.selectAttachFilesByBno(bno);
}
```

## (2) MyBoardController.java 설정

↳ MyBoardService.getAttachFilesByBno() 메서드가 반환한 게시물의 첨부파일 정보(MyBoardAttachFileVO의 리스트)를 JSON 형식으로 브라우저에 전달되도록 구현합니다.

→ src/main/java/com.spring5213.mypro00.controller.MyBoardController.java 클래스에 다음의 메서드를 추가합니다.

```
#####
특정 게시물의 첨부파일 정보를 JSON으로 전달 #####
@GetMapping(value = "/getFiles", produces = {"application/json;charset=UTF-8"})
@ResponseBody
public ResponseEntity<List<MyBoardAttachFileVO>> showAttachFiles(Long bno) {
    log.info("컨트롤러: 조회할 첨부파일의 게시물번호(bno): " + bno);

    return new ResponseEntity<List<MyBoardAttachFileVO>>(myBoardService.getAttachFilesByBno(bno),
        HttpStatus.OK);
}
```

↳ 위의 메서드는 @ResponseBody REST-API 어노테이션을 이용해서 JSON 데이터를 반환하도록 처리합니다.

이 때, 첨부파일 목록과 Http 처리 상태를 ResponseEntity 객체를 이용해서 전달되도록 구현합니다.

## (3) 게시물 상세 화면에서의 첨부파일 표시(detail.jsp 파일 수정)

↳ 게시물 정보가 표시되는 detail.jsp 페이지에 첨부파일 정보가 표시될 수 있는 <div>와 CSS <style> 을 fileUploadAjax.jsp 페이지의 내용을 이용하여 추가합니다. <div> 추가 위치는 댓글 표시 영역 위에 추가하고, CSS <style> 요소는 myheader.jsp를 호출하는 include 지시자 아래(페이지 윗부분)에 추가합니다.  
작업 시에, 기존 댓글 관련 CSS <style> 태그의 위치도 윗부분으로 이동시켜 놓습니다.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지를 코드 작성부에 오픈하고 다음의 지시대로 코드를 추가합니다.

→ 다음의 첨부파일 표시와 원본 이미지 표시를 위한 CSS <style> 추가

...(생략)...

```
<%@ include file="../myinclude/myheader.jsp" %>

<%-- 댓글 CSS style 태그 --%><%-- 아래에서 이동 시킨 댓글 관련 CSS <style> 요소 --%>
<style>
    .txtBoxCmt, .txtBoxComment { overflow: hidden; resize: vertical; min-height: 100px; color: black; }
</style>

<%-- 파일 업로드 표시 영역에 대한 CSS 스타일, 실습에서 사용 않함 --%><%-->
<style>
    .fileUploadResult { width: 100%; background-color: lightgrey; }
```

```

.fileUploadResult ul { display: flex; flex-flow: row; justify-content: center; align-items: center; }
.fileUploadResult ul li { list-style: none; padding: 5px; }
.fileUploadResult ul li img { height: 50px; width: auto; max-height: 100px; overflow: hidden" }
</style>--%
<%-- 다운로드 이미지 표시 영역에 대한 CSS --%>
<style>
.bigImageWrapper {
  position: absolute;
  display: none;
  justify-content: center;
  align-items: center;
  top:0%;
  width: 100%;
  height: 100%; /* background-color: lightgray; */
  z-index: 100;
}
.bigImage {
  position: relative;
  display:flex;
  justify-content: center;
  align-items: center;
}
.bigImage img { height: 100%; max-width: 100%; width: auto; overflow: hidden }
</style>

<div id="page-wrapper">
...(생략)...

```

→ 첨부파일 표시 <div> 추가: register.jsp 의 첨부파일 결과 표시 및 원본 이미지 표시 <div>를 복사하여 기존 게시물 정보 표시 <div class="row">의 종료태그 밑에 붙여넣습니다 이 때, input 요소 div 는 삭제합니다.  
그리고 원본 이미지 표시 div 를 추가합니다(빨간색 코드).

...(생략)...

```

</div><%-- /.row --%>

<!-- 첨부파일 표시 -->
<div class="row">
  <div class="col-lg-12">
    <div class="panel panel-default">
      <div class="panel-heading">첨부파일</div> ← 수정
      <div class="panel-body"> ←id 속성 제거
        <div class='form-group bigImageWrapper'><%-- 원본 이미지 표시 div --%> ←div 전체 내용 추가
          <div class='bigImage'>
            <%-- 이미지파일 크게 표시되는 영역: 썸네일 클릭 시, 첨부파일 패널 영역에만 원본이미지가 표시됨.--%>
          </div>
        </div><%-- 원본이미지 div 끝 --%>
        <div class="form-group fileUploadResult"><%-- 첨부파일 목록 표시 div --%> ←div 전체 내용 추가
          <ul>
            <%-- 첨부파일 목록이 표시될 영역 --%>
          </ul>
        </div>
      </div><%-- /.panel-body -->
    </div><%-- /.panel -->
  </div><%-- /.col-lg-12 -->
</div><%-- /.row --%>

<form id="frmSendValue"><%-- 폼을 추가 --%>
...(생략)...

```

다음은, detail.jsp 페이지가 웹 브라우저에 표시될 때(로딩 시), 자동으로 JSON 형식으로 게시물의 첨부파일(들) 정보를 가져와서, <div> 영역에 표시되도록 jQuery를 구현합니다. 코드 작성 시에, register.jsp 페이지의 showUploadedFiles() 함수의 코드를 이용하고, 삭제 기능은 필요없으므로, 업로드 된 파일이름과 삭제버튼은 주석처리합니다. 주석처리하는 이유는, modify.jsp에서 detail.jsp 파일내용 사용 시에 다시 필요하기 때문입니다.

→ detail.jsp 페이지 로딩 시, 첨부파일 정보를 가져와서 표시하는 jQuery 코드 추가

- 파일 하단의 \$(document).ready(function(){}); 실행문의 위에 showUploadedFiles() 함수 정의 코드를 추가
- \$(document).ready(function(){}); 실행문에 Ajax 실행코드를 추가합니다.

```
//업로드 후, 첨부파일 정보 표시 함수
function showUploadedFiles(uploadResult) {
    //console.log("showUploadedFiles 실행: 화면 표시 시작=====");
    if(!uploadResult || uploadResult.length == 0){
        return ;
    }

    var fileUploadResult = $(".fileUploadResult ul");
    var str = "";

    $(uploadResult).each(function(i, obj) {
        if (obj.fileType=="F") { //이미지가 아닌 경우, 아이콘 이미지 및 원본 파일이름 표시
            //var calledPathFileName = encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/" +
            //                                              obj.uuid + "_" + obj.fileName); ←주석처리
            //
            //console.log("호출된 파일이름: " + calledPathFileName); ←주석처리

            //아래의 li 링크에 data-repopath 추가
            str += "<li data-repopath='" + obj.repoPath + "' data-uploadpath='" + obj.uploadPath + "'"
                + " data-uuid='" + obj.uuid + "' data-filename='" + obj.fileName + "'"
                + " data-filetype='" + obj.fileType + "'>"
                + " <img src='${contextPath}/resources/img/icon-attach.png' alt='No Icon' "
                + " style='height: 18px; width: 18px;'>" + obj.fileName
            // + " <span data-filename='" + calledPathFileName + "' data-filetype='F'>[삭제]</span>" ←주석처리
            + "</li>";
        } else if (obj.fileType=="I") { //이미지파일인 경우 썸네일 및 원본 파일이름 표시
            var thumbnailFilePath =
                encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/s_" + obj.uuid + "_" + obj.fileName);
            console.log("encodeURIComponent 처리 파일이름: " + thumbnailFilePath);

            //아래의 li 링크에 data-repopath 추가
            str += "<li data-repopath='" + obj.repoPath + "' data-uploadpath='" + obj.uploadPath + "'"
                + " data-uuid='" + obj.uuid + "' data-filename='" + obj.fileName + "'"
                + " data-filetype='" + obj.fileType + "'>"
                + " <img src='${contextPath}/displayThumbnailFile?fileName=" + thumbnailFilePath + "'"
                + " alt='No Icon' style='height: 18px; width: 18px;'>" + obj.fileName
            // + " <span data-filename='" + thumbnailFilePath + "' data-filetype='I'>[삭제]</span>" ←주석처리
            + "</li>";
        }
    });
    fileUploadResult.append(str);
}

$(document).ready(function(){//페이지 로딩 시 함수 실행, 전체 JavaScript 내용 중 제일 마지막에 위치해야 함
    $.ajax({ ← Ajax 코드 전체 추가
        url: '${contextPath}/myboard/getFiles',
        data: {bno: bnoValue}, //bnoValue는 상단에 정의(댓글구현 시 사용)된 것 사용
        type: 'get',
        dataType: 'json',
        success: function(fileList){
            showUploadedFiles(fileList);
        }
    }) //ajax end
    showCmtList(1); //댓글목록 표시, 자동 실행문
});
```

#### (4) 첨부파일 클릭 시, 이벤트 처리

☞ fileUploadAjax.jsp 파일의 내용을 이용하여 다음의 jQuery 함수를 구현합니다(추가 또는 수정, 빨간색 코드).

- 썸네일 클릭 시, 첨부파일 표시 영역에 원본이미지 표시가 표시되도록 showImage() 함수를 추가하고 표시된 원본 이미지를 클릭 시, 다시 원래의 결과목록이 표시되는 jQuery 이벤트 함수 추가합니다.
- 일반파일은 파일 다운로드가 수행될 jQuery 이벤트 함수를 추가합니다.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 페이지를 코드 작성부에 오픈하고

→ 필요한 함수들을 showUploadedFiles() 함수 밑에 작성합니다.

```
//이미지 표시
//display 속성의 flex 옵션을 이용하면, 이미지가 웹브라우저의 중앙에 표시됩니다.
function showImage(calledPathImagefileName){
    $(".bigImageWrapper").css("display","flex").show();
    $(".bigImage").html("<img src='${contextPath}/fileDownloadAjax?fileName=" + calledPathImagefileName + "'>")
        .animate({width:'100%', height: '100%'}, 1000);
}

//DIV 클릭 이벤트처리: 클릭 시 1초 후에 이미지 사라짐.
$(".bigImageWrapper").on("click", function(e){
    $(".bigImage").animate({width:'0%', height: '0%'}, 1000);
    //setTimeout(() => {$(this).hide();}, 1000);
    setTimeout(function () {
        $(".bigImageWrapper").hide();
    }, 1000);
});

//첨부파일 다운로드 및 원본 이미지 표시(새로 추가)
$(".fileUploadResult ul").on("click","li", function(e){

    var objLi = $(this);

    var downloadedPathFileName = encodeURIComponent(objLi.data("repopath") + "/" + objLi.data("uploadpath") + "/" +
        objLi.data("uuid")+"_"+ objLi.data("filename"));

    if(objLi.data("filetype") == "I"){
        //이미지파일일 경우, 원본이미지를 다운로드 하여 결과표시 영역에 표시
        showImage(downloadedPathFileName.replace(new RegExp(/\//g),"/"));

    } else { // else if(objLi.data("filetype") == "F") {
        //이미지가 이닐 경우, 다운로드 수행
        self.location ="${contextPath}/fileDownloadAjax?fileName=" + downloadedPathFileName ;
    }
});
```

[테스트 - 첨부파일 다운로드 및 원본 이미지 표시 테스트]

☞ 테스트 시 웹브라우저의 개발자 도구를 활용하여 jsp 파일의 정적 요소들의 잘못된 코드를 찾아낼 수 있으므로 같이 사용합니다.

☞ 크롬/엣지/IE 브라우저에서 동일한 테스트를 수행합니다.

→ 톰캣 서버를 기동하고 웹 브라우저에서 http://localhost:8080/mypro00/myboard/list URL로 접속

- 앞 단원 실습으로 추가된 첨부파일이 있는 게시물의 상세 정보 확인 페이지로 이동
- 표시화면에 첨부파일이 잘 표시되어야 하며, 각 첨부파일을 클릭했을 때, 원본 이미지가 표시되고, 파일이 다운로드 되는지 확인합니다.

## 11-8. 첨부파일 구현 실습: 프로젝트에 적용 - 게시물 수정 - 삭제 시의 첨부파일 처리

☞ 게시물 수정 중에 발생될 수 있는 첨부파일의 수정은, 다음의 3 가지 작업 중 하나입니다.

- 새로운 첨부파일의 추가
- 기존 첨부파일 삭제
- 기존 첨부파일 삭제 후, 새로운 첨부파일 추가

☞ 위의 경우를 정리하면, 게시물 수정페이지에서 해당 게시물의 첨부파일을 삭제/추가할 수 있는 기능이 구현되어야 합니다.

☞ 이러한 첨부파일 추가와 삭제는 데이터베이스에 저장된 관련정보도 수정해주는 작업과 서버에 저장된 첨부파일의 삭제도 같이 수행되어야 합니다.

### (1) 게시물 수정 - 첨부파일 수정: 화면처리(src/main/webapp/WEB-INF/views/myboard/modify.jsp 파일)

☞ 게시물의 첨부파일에 대한 수정(첨부파일 추가/삭제)을 할 수 있도록 다음의 요소들을 추가합니다.

- 첨부파일(들)이 표시되는 영역(register.jsp에서와 동일)
- 첨부파일을 삭제할 수 있는 요소
- 첨부파일을 추가할 수 있는 요소

☞ 게시물 수정 작업 중에는 첨부파일 다운로드와 이미지 원본 파일을 표시하는 기능은 게시물 조회페이지에서 제공되므로 중복해서 구현하지 않습니다.

#### (1-1) 게시물 수정 페이지: 첨부파일 목록 표시 (modify.jsp 수정)

☞ 결과 표시 영역의 DIV는 register.jsp 파일과 동일하게 작성합니다(복사/붙여넣기 합니다)

☞ 결과 표시 영역에 대한 CSS-Style은 실습에서는 사용하지 않지만, register.jsp 파일과 동일하게, 주석처리하여 추가합니다. detail.jsp 와는 달리, 이미지 확대를 위한 CSS는 필요없습니다.

→ CSS-Style 추가

...(생략)...

```
<%@ include file="../myinclude/myheader.jsp" %>

<%-- 파일 업로드 표시 역역에 대한 CSS 스타일, 실습에서 사용 않함 --%><%--
<style>
    .fileUploadResult { width: 100%; background-color: lightgrey; }
    .fileUploadResult ul { display: flex; flex-flow: row; justify-content: center; align-items: center; }
    .fileUploadResult ul li { list-style: none; padding: 5px; }
    .fileUploadResult ul li img { height: 50px; width: auto; max-height: 100px; overflow: hidden" }
</style>--%>

<div id="page-wrapper">
    ... (생략)...
```

→ 첨부파일 표시 <div> 추가 및 첨부파일 추가 file 유형의 input 추가

```
...(생략)...
</div><%-- /.row --%>

<%-- 첨부파일 표시 --%>
<div class="row">
    <div class="col-lg-12">
        <div class="panel panel-default">
            <div class="panel-heading">파일첨부</div>
            <div class="panel-body" id="panelFileBody">
                <div class="form-group uploadDiv">
                    <input id="inputFile" type="file" name="uploadFiles" multiple><br>
                </div>
                <div class="form-group fileUploadResult">
                    <ul>
                        <!-- 업로드 후 결과처리 로직이 표시될 영역 -->
                    </ul>
                </div>
            </div><%-- /.panel-body -->
        </div><%-- /.panel -->
    </div><%-- /.col-lg-12 -->
</div><%-- /.row -->

</div><%--/#page-wrapper --%>
```

...(생략)...

→ 첨부파일을 표시하기 위한 showUploadFiles() 함수코드를 detail.jsp 의 함수의 코드를 사용하여 작성합니다.

→ 첨부파일 삭제는 버튼으로 교체합니다.

→ 페이지로 로딩 시, ajax로 첨부파일 정보를 가져와서 showUploadFiles() 함수가 자동 실행되도록. detail.jsp 의 \$(document).ready(function(){}); 코드를 복사하여 기존 함수의 제일 마지막에 추가합니다.

(주의) 요구사항의 구현 코드를 modify.jsp 페이지에 있는 기존 자바스크립트 코드의 제일 마지막에 추가합니다.

```
<script><%-- 첨부파일 표시 처리 --%>

var bnoValue = '<c:out value="${board.bno}" />';

//첨부파일 정보 표시 함수
function showUploadedFiles(uploadResult) {
    if(!uploadResult || uploadResult.length == 0){
        return ;
    }

    var fileUploadResult = $(".fileUploadResult ul");
    var str = "";

    $(uploadResult).each(function(i, obj) {
        if (obj.fileType=="F") { //이미지가 아닌 경우

            //encodeURIComponent로 처리된 첨부파일(전체 경로 포함, 삭제 시 필요):
            var calledPathFileName = encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/" +
                obj.uuid + "_" + obj.fileName);
            console.log("전체 경로포함 파일이름: " + calledPathFileName);

            //아이콘 이미지 및 uuid 없는 파일이름 표시
            str += "<li data-repopath='" + obj.repoPath + "' data-uploadpath='" + obj.uploadPath + "'"
                + " data-uuid='" + obj.uuid + "' data-filename='" + obj.fileName + "'"
                + " data-filetype='F' style='height:25px;'>" + obj.fileName + "<img src='${contextPath}/resources/img/icon-attach.png' alt='No Icon' "
                + " style='height: 18px; width: 18px;'" + "&nbsp;&nbsp;" + "<span data-filename='"
                + calledPathFileName + "' data-filetype='F'>[삭제]</span>"/> //버튼으로 교체
        }
    })
}

//
```

```

+ "     &nbsp;<button type='button' class='btn btn-danger btn-xs' "
+ "             data-filename='" + calledPathFileName + "' data-filetype='F'>X</button>"
+ "</li>";

} else if (obj.fileType=="I") { //이미지파일인 경우

//encodeURIComponent로 처리된 썸네일파일(전체 경로 포함):
var thumbnailFilePath =
    encodeURIComponent(obj.repoPath + "/" + obj.uploadPath + "/s_" + obj.uuid + "_" + obj.fileName);
console.log("encodeURIComponent 처리된 썸네일이름: " + thumbnailFilePath);

//업로드 된 파일이름(uuid 포함된 이름, 삭제 시 필요)
//var originPathImageFileName = obj.uploadPath + "\\\" + obj.uuid + "_" + obj.fileName;
//originPathImageFileName = originPathImageFileName.replace(new RegExp(/\\"/g),"/");

//썸네일 및 uuid 없는 파일이름 표시
str += "<li data-repopath='" + obj.repoPath + "' data-uploadpath='" + obj.uploadPath + "'"
+ " data-uuid='" + obj.uuid + "' data-filename='" + obj.fileName + "'"
+ " data-filetype='I' style='height:25px;'>" ← 수정
+ " <img src='${contextPath}/displayThumbnailFile?fileName=" + thumbnailFilePath + "'"
+ " alt='No Icon' style='height: 18px; width: 18px;'>&nbsp;&nbsp;" ← 추가
+ " obj.fileName
// + " <span data-filename='" + thumbnailFilePath + "' data-filetype='I'>[삭제]</span>"//버튼으로 교체
+ " &nbsp;<button type='button' class='btn btn-danger btn-xs'"
+ "             data-filename='" + thumbnailFilePath + "' data-filetype='I'>X</button>"
+ "</li>";
}
});

fileUploadResult.append(str);
}

$(document).ready(function(){//페이지 로딩 시 함수 실행, 전체 JavaScript 내용 중 제일 마지막에 위치해야 함

$.ajax({
    url: '${contextPath}/myboard/getFiles',
    data: {bno: bnoValue}, //bnoValue는 상단에 정의된 것 사용
    type: 'get',
    dataType: 'json',
    success: function(fileList){
        showUploadedFiles(fileList);
    }
}) //ajax end

});

</script>

<%@ include file="../myinclude/myfooter.jsp" %>

```

#### [테스트]

- 템켓 서버를 기동 후, 웹브라우저에서 게시물 목록 → 게시물 상세 → 게시물 수정 페이지 호출 후,
- 첨부파일 정보가 표시되는지 확인합니다.

## (1-2) 게시물 수정 페이지: 첨부파일 변경 - modify.jsp 페이지에서의 새로운 첨부파일 추가

☞ 추가되는 함수들은 기존 코드의 밑에 추가하되 단, `$(document).ready(function{ 자동실행내용 })` 또는 `$(function{ 자동실행내용 })` 함수의 위에 추가해야 합니다.

→ 첨부파일 결과 표시 div에 있는 `<input type="file">`에 대하여, `checkUploadfile()` 함수와 Ajax로 첨부파일을 서버로 전송하는 jQuery 코드를 `register.jsp` 파일에서 복사하여 `showUploadedFiles()` 함수 아래에 붙여넣기 합니다.

(주의) 추가되는 코드는 `$(document).ready(function(){})`; 실행문보다 위에 작성되어야 합니다.

```
//업로드 파일의 확장자 및 최대 파일 크기 검사 함수
function checkUploadfile(fileName, fileSize) {
    // 확장자에 대한 정규식 및 최대 허용크기(5MB)
    var maxSizeAllowed = 5242880;
    var regExpForFileExtention = /(.*).(exe|sh|zip|alz)$/i ;

    //최대 허용 크기 제한 검사
    if (fileSize >= maxSizeAllowed) {
        alert("업로드 파일의 제한된 크기(5MB)를 초과했습니다.");
        return false;
    }

    //업로드파일의 확장자 검사:
    if (regExpForFileExtention.test(fileName)) {
        alert("해당 종류(exe/sh/zip/alz)의 파일은 업로드할 수 없습니다.");
        return false;
    }
    return true;
}

//빈 input 요소를 복사(초기화를 위함)
var cloneInputFile = $(".uploadDiv").clone() ;

//파일 업로드 처리: 버튼 클릭이 아닌, input의 내용이 바뀌면 업로드가 자동으로 수행되도록 수정
//업로드 후, .uploadDive가 복사된 것으로 교체되어 input이 초기화되므로, 파일 수정이 되도록 이벤트 위임 방식을 사용
$("#panelFileBody").on("change", "input[type='file']", function(e) {

    var formData = new FormData(); //전송될 파일을 저장할 객체
    var inputFiles = $("input[name='uploadFiles']");
    var files = inputFiles[0].files; //input 요소의 파일을 변수에 저장
    console.log(files);

    for(var i = 0; i < files.length ; i++) { //formdata 객체에 파일 추가(for문)
        if (!checkUploadfile(files[i].name, files[i].size)) { //업로드 파일 요구조건 검사
            return false;
        }
        formData.append("uploadFiles", files[i]); //uploadFiles 파라미터로 file 정보 추가
    }

    $.ajax({ //파일 전송 및 서버로부터 첨부파일 데이터를 받아, 화면 결과 표시
        url: '${contextPath}/fileUploadAjaxAction',
        processData: false,
        contentType: false,
        data: formData,
        type: 'post',
        dataType: 'json',
        success: function(uploadResult){
            //alert("첨부파일의 업로드가 정상적으로 완료되었습니다...");
            $(".uploadDiv").html(cloneInputFile.html()); //input 초기화
            showUploadedFiles(uploadResult); //업로드 결과 표시
        }
    }) // End .ajax
});

$(document).ready(function(){
```

...(생략)...

### (1-3) 게시물 수정 페이지: 첨부파일 변경 - modify.jsp 페이지에서의 기존 첨부파일 삭제

#### [참고] 기존 첨부파일 삭제 관련 고려사항

- ☞ 게시물 수정 시에, 새로 추가된 첨부파일은 서버에 파일이 이미 업로드 된 상태에서  
게시물 수정 시에, 새로 추가된 첨부파일의 정보를 데이터베이스에 저장되도록 구현됩니다.
- ☞ 게시물 수정페이지에서 표시된 기존 첨부파일이 삭제될 때 서버에 저장된 파일도 삭제되면,  
즉, **게시물 수정 요청보다 앞서서 기존 파일이 서버에서 삭제되면, 데이터베이스에 파일 정보는  
존재하고, 파일은 없는 상태가 됩니다.**
- ☞ 위의 상황은, 사용자가 이미 있던 첨부파일 중의 일부를 삭제한 후,  
정상적으로 게시물을 수정하지 않고 게시물 수정 삭제 화면을 빠져나가는 경우에도 발생됩니다.  
위의 상황이 문제가 되는 이유는, 데이터베이스에 정보는 존재하는데 파일은 존재하지 않는 경우,  
데이터베이스에서 존재하지 않는 파일에 대한 정보를 삭제하는 것이 구현하기 어려운 작업이기 때문입니다.
- ☞ 따라서, 존재하지 않는 파일과 관련된 데이터베이스의 정보가 생성되지 않도록 하려면,  
사용자가 특정 첨부파일을 삭제했을 때(데이터베이스 정보와 업로드 파일이 존재),  
화면에서만 해당 항목을 삭제하고, 서버의 파일은 삭제되지 않도록 합니다. 이 부분이 게시물 등록에서와 다릅니다.

#### [참고] 불필요한 첨부파일(데이터베이스에 관련된 정보가 없는 파일)의 삭제

실제 파일의 삭제는 주기적으로 데이터베이스의 정보를 이용하여 확인 후,  
정보가 없는 파일을 배치성 작업을 통해 삭제합니다. 이 후 단원에서 배치성 작업을 통해  
주기적으로, 필요없이(데이터베이스의 정보없이) 존재하는 파일을 삭제하는 작업을 구현합니다.

→ modify.jsp 페이지에, 사용자가 파일 삭제 버튼(X)를 클릭할 때, 사용자의 확인을 거쳐 화면에서만 항목을 삭제시키는 다음의 함수를,  
기존 파일 업로드 실행문 밑에 추가합니다.

```
//파일 삭제(수정화면): 브라우저 표시 항목만 삭제.  
//$(".fileUploadResult").on("click","li span", function(e){  
    $(".fileUploadResult").on("click", "button", function(e){  
        //this: span 또는 button  
        console.log("파일삭제(화면에서 항목만 삭제)");  
  
        if (confirm("이 파일을 삭제하시겠습니까?")){  
            var targetLi = $(this).closest("li");  
            targetLi.remove();  
            alert("파일이 삭제되었습니다")  
        } else {  
            alert("파일 삭제가 취소되었습니다.")  
        }  
    });  
});
```

#### (1-4) 게시물 수정(첨부파일 변경 포함) 처리 - modify.jsp 의 게시물에 대한 수정버튼 클릭 처리

##### (1-4-1) modify.jsp 파일 수정 (화면 처리)

☞ 게시물 수정 버튼 클릭 시에, 첨부파일의 변경 정보도 서버에 전달되도록 버튼 클릭 이벤트 함수에 다음의 기능을 추가합니다.

- 클릭된 버튼의 기본 기능이 실행되지 않도록 지정(버튼의 type 속성을 button으로 지정하여 해결함).
- 첨부파일 표시 li 요소에 설정된 첨부파일 정보를 <input type="hidden"> 요소에 설정하여 form(이미 게시물 정보와 페이징 데이터가 지정되어 있음)에 추가.
- 게시물 정보/페이징 데이터/첨부파일 정보가 지정된 form 을 서버로 전달.

→ 다음은 modify.jsp 파일의 기존 [글 수정/삭제/취소] 버튼의 클릭이벤트 함수 코드입니다(참고만 합니다).

```
//form의 수정/삭제/목록보기 버튼에 따른 동작 제어
var frmModify = $("#frmModify");

$('button').on("click", function(e){
    var operation = $(this).data("oper"); //각 버튼의 data-xxx 속성에 설정된 값을 저장

    if(operation == "modify"){ //게시물 수정 요청
        frmModify.attr("action", "${contextPath}/myboard/modify");

    } else if(operation == "remove"){ //게시물 삭제 요청
        frmModify.attr("action", "${contextPath}/myboard/delete");

    } else if(operation == "list"){ //게시물 목록 화면 요청

        var pageNumInput = $("input[name='pageNum']").clone();
        var rowAmountInput = $("input[name='rowAmountPerPage']").clone();
        var scopeInput = $("input[name='scope']").clone();
        var keywordInput = $("input[name='keyword']").clone();

        frmModify.empty();

        frmModify.attr("action","${contextPath}/myboard/list").attr("method","get");
        //복사된 input 요소를 form에 추가
        frmModify.append(pageNumInput);
        frmModify.append(rowAmountInput);
        frmModify.append(scopeInput);
        frmModify.append(keywordInput);
    }
    frmModify.submit() ; //요청 전송
});
```

→ 브라우저 화면에 표시된 수정, 삭제, 취소 버튼에 btn-frmModify 클래스를 추가합니다.

```
...(생략)...
<button type="button" class="btn btn-default btn-frmModify" id="btnModify" data-oper="modify">수정</button>
<button type="button" class="btn btn-danger btn-frmModify" id="btnRemove" data-oper="remove">삭제</button>
<button type="button" class="btn btn-info btn-frmModify" id="btnList" data-oper="list">취소</button>
<%-- 검색 페이징 데이터가 저장되는 hidden 유형의 input --%>
<input type='hidden' name='pageNum' value='${myBoardPagingDTO.pageNum}'>
<input type='hidden' name='rowAmountPerPage' value='${myBoardPagingDTO.rowAmountPerPage}'>
<input type='hidden' name='scope' value='${myBoardPagingDTO.scope}'>
<input type='hidden' name='keyword' value='${myBoardPagingDTO.keyword}'>
</form>
...(생략)...
```

→ 기존 form 전송 버튼 클릭 이벤트 처리 jQuery 함수를 다음처럼 수정합니다(빨간색 코드 수정 및 추가).

```
<script>
//form의 수정/삭제/목록보기 버튼 클릭 이벤트 처리
var frmModify = $("#frmModify");

$(".btn-frmModify").on("click", function(e){ ← 수정

    var operation = $(this).data("oper"); //각 버튼의 data-xxx 속성에 설정된 값을 저장

    if(operation == "modify"){ //게시물 수정 요청

        var strFilesInputHidden = ""; ← 여기서부터는 추가 시작
        //업로드 결과의 li 요소 선택하여 각각에 대하여 다음을 처리
        $(".fileUploadResult ul li").each(function(i, obj){
            var objLi = $(obj);
            strFilesInputHidden
                += " <input type='hidden' name='attachFileList["+i+"] uuid' "
                + " value='" + objLi.data("uuid") + "'>
                + " <input type='hidden' name='attachFileList["+i+"] uploadPath' "
                + " value='" + objLi.data("uploadpath") + "'>
                + " <input type='hidden' name='attachFileList["+i+"] fileName' "
                + " value='" + objLi.data("filename") + "'>
                + " <input type='hidden' name='attachFileList["+i+"] fileType' "
                + " value='" + objLi.data("filetype") + "'> ";
        });
        console.log(strFilesInputHidden); //테스트 후, 주석처리할 것
        frmModify.append(strFilesInputHidden); //form에 추가 ← 여기까지 추가 끝

        frmModify.attr("action", "${contextPath}/myboard/modify");

    } else if(operation == "remove"){ //게시물 삭제 요청
        frmModify.attr("action", "${contextPath}/myboard/delete"); //bdelFlag만 업데이트
        //frmModify.attr("action", "${contextPath}/myboard/remove"); //실제 행 삭제

    } else if(operation == "list"){ //게시물 목록 화면 요청
        //기존 페이지 데이터 input 요소 복사
        var pageNumInput = $("input[name='pageNum']").clone();
        var rowAmountInput = $("input[name='rowAmountPerPage']").clone();
        var scopeInput = $("input[name='scope']").clone();
        var keywordInput = $("input[name='keyword']").clone();

        frmModify.empty();

        frmModify.attr("action", "${contextPath}/myboard/list").attr("method", "get");
        //복사된 input 요소를 form에 추가
        frmModify.append(pageNumInput);
        frmModify.append(rowAmountInput);
        frmModify.append(scopeInput);
        frmModify.append(keywordInput);

    }

    frmModify.submit(); //요청 전송
});
</script>
```

#### (1-4-2) 게시물 수정 시의 서버 처리 - MyBoardServiceImpl 클래스 수정

- ☞ 게시물 수정 시에 발생된 첨부파일의 변경(기존 첨부파일 삭제 및 새로운 첨부파일 추가)는 최종적으로 게시물 수정페이지에서 요청된 첨부파일을 기준으로, 데이터베이스에 저장된 게시물의 수정 전 첨부파일 정보를 삭제하고, 게시물 수정 요청 시에, 전달된 첨부파일 정보로 저장되도록 구현하면 편리합니다.
- ☞ 게시물 수정 버튼이 클릭되면, 게시물 수정 정보와 변경된 첨부파일 정보가 담긴 form이 MyBoardController의 modifyBoard 메서드에 전달되어 MyBoardVO 객체에 저장됩니다(파일은 이미 업로드되어 저장되므로 폼을 통해 전달되는 파일은 없습니다). 컨트롤러의 메서드는, 게시물 정보와 첨부파일 정보가 담긴 MyBoardVO 객체를 MyBoardService의 modifyBoard 메서드에게 전달합니다. 서비스의 메소드는 Mapper-인터페이스와 SQL-Mapper XML 파일을 호출하여 데이터베이스에 작업과 관련된 데이터 변경사항을 요청합니다.
- ☞ 따라서 게시물 수정과 첨부파일 수정 작업은 MyBoardService의 modifyBoard() 메서드에서 MyBoardMapper 와 MyBoardAttachFileMapper 를 호출하여 데이터변경 작업들이 하나의 트랜잭션으로 수행되도록 구현합니다.

→ src/main/java/com.spring5213.mypro00.service.MyBoardServiceImpl.java 클래스를 코드 작성부에 오픈

→ modifyBoard() 메서드의 내용을 모두 주석처리 또는 삭제 후, 다음의 내용으로 교체합니다

```
//게시물 수정 처리
@Transactional ← 추가
@Override
public boolean modifyBoard(MyBoardVO myBoard){
    log.info("BoardService.modify()에 전달된 MyBoardVO: " + myBoard);

    Long bno = myBoard.getBno();

    //게시물 변경 시, 기존 첨부파일을 모두 삭제되어 전달된 첨부파일 정보가 없는 경우도 있으므로
    //첨부파일 삭제는 무조건 처리합니다.
    myBoardAttachFileMapper.deleteAttachFilesByBno(bno);

    //게시물 수정: tbl_myboard 테이블에 수정
    boolean boardModifyResult = myBoardMapper.updateMyBoard(myBoard) == 1;

    //게시물 수정이 성공하고, 첨부파일이 있는 경우에만 다음작업 수행
    //첨부파일 정보 저장: tbl_myAttachFiles 테이블에 저장
    if (boardModifyResult && myBoard.getAttachFileList().size() > 0) {
        myBoard.getAttachFileList().forEach(
            attachFile -> {
                attachFile.setBno(bno);
                myBoardAttachFileMapper.insertAttachFile(attachFile);
            }
        );
    }

    return boardModifyResult;
}
```

## (1-5) 게시물 삭제 시의 서버처리

- ☞ 게시물 삭제는 데이터베이스의 정보도 삭제되고, 게시물의 실제 첨부파일도 서버에서 삭제되어야 합니다.
- ☞ 데이터베이스 정보는 MyBoardServiceImpl 서비스의 메소드에 bno 값을 전달하여 삭제되도록 구성되었습니다. 서버의 파일 삭제를 위해 필요한 첨부파일 정보는 화면에서 컨트롤러의 메서드에 전달됩니다. 따라서, 서버의 첨부파일 삭제는 컨트롤러의 관련 메서드에서 수행되도록 구현합니다.
- ☞ 데이터베이스 정보는 게시물 정보와 첨부파일 정보가 2 개의 테이블로부터 함께 삭제되어야 하므로, 이를 하나의 트랜잭션으로 구성합니다. 이 때, 서버 상의 파일 삭제도 같이 수행되어야 하므로, 데이터베이스의 첨부파일 정보가 삭제되기 전에, 먼저 첨부파일 정보를 변수에 저장해 놓은 후, 데이터베이스에서의 삭제가 수행되도록 구현합니다.
- ☞ MyBoardController 에 대해서 게시물 삭제 시 처리되는 작업의 순서를 정리하면 다음과 같습니다.
  - (1) 해당 게시물의 첨부파일 정보 저장
  - (2) 데이터베이스 상에서 해당 게시물과 첨부파일 데이터 삭제: 서비스클래스의 메서드 이용
  - (3) 저장된 첨부파일 목록을 이용해서 해당 폴더에서 첨부파일(이미지파일에 대한 썸네일 포함)을 삭제.
- ☞ 기존에 com.spring5213.mypro00.common.fileupload.FileUploadController 에 구현한 deleteFile() 메서드는 fileName 과 fileType 을 매개변수를 이용하여, 하나의 (이미지파일의 경우, 썸네일 포함) 첨부파일이 삭제되도록 구현되어 있으므로, 파일목록을 매개변수로 받아 하나 이상의 첨부파일 삭제를 처리할 수 있는 메서드를 구현합니다.
- ☞ 실습 문서에서는 게시물 삭제 처리 시 다음의 3 가지 방법을 구현했고 그 중 1 방법을 이용하여 게시물 삭제를 화면에서 적용했습니다.
  1. 게시물 삭제 처리 시, 데이터베이스의 bdelFlag 를 1 로 수정만 함(실제 게시물 삭제 없음)
  2. 실제 데이터베이스에서 게시물 데이터 삭제(화면에서는 사용 않함)
  3. 삭제 요청된(bdelFlag 가 1 인) 모든 게시물 삭제(화면에서는 사용 않함)따라서, 현재의 실습 구성으로는, 화면에서의 게시물 삭제 요청으로 데이터베이스 정보가 삭제되지 않으므로, 첨부파일도 삭제되면 됩니다.
- ☞ 게시물 삭제 시에 첨부파일 삭제처리는 위의 2 의 경우에 대하여 구현합니다. 그리고 화면에서의 적용은 간단히 버튼을 추가하여 처리합니다.
- ☞ 실습은, 서비스 클래스 수정 → 컨트롤러 수정 → 화면(JSP 페이지 수정) 순으로 진행합니다.

### (1-5-1) MyBoardServiceimpl 에서의 게시물 삭제 처리

- src/main/java/com.spring5213.mypro00.service.MyBoardServiceImpl.java 클래스를 코드 작성부에 오픈
- removeBoard() 메서드를 다음처럼 수정합니다.

```
//게시물 삭제 - 실제 삭제 발생
@Transactional ← 추가
@Override
public boolean removeBoard(Long bno){
    log.info("MyBoardService.removeBoard()에 전달된 bno: " + bno);
    //데이터베이스 첨부파일 정보 삭제
    //첨부파일이 없어도 SQL은 정상처리됨(0개 행이 삭제)
    //테이블의 외래키(FK)에 ON DELETE CASCADE 옵션이 사용된 경우,
    //첨부파일 정보 삭제는 데이터베이스에 의해 자동으로 처리되므로
```

```

//아래의 실행문은 필요없습니다.
myBoardAttachFileMapper.deleteAttachFilesByBno(bno); ← 추가

//게시물정보 삭제가 정상처리되면, true 반환
return myBoardMapper.deleteMyBoard(bno) == 1;
}

```

### (1-5-2) MyBoardController 에서의 게시물 삭제 처리

→ src/main/java/com.spring5213.mypro00.controller.MyBoardController.java 클래스를 코드 작성부에 오픈

→ removeAttachFiles() 메서드를 추가합니다.

```

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import org.zerock.ex00.domain.MyBoardAttachFileVO;

...(생략)...

//게시물의 모든 첨부파일을 서버에서 삭제
private void removeAttachFiles(List<MyBoardAttachFileVO> attachFileList) {

    if(attachFileList == null || attachFileList.size() == 0) {
        return;
    }

    System.out.println("첨부파일 삭제 시작.....");
    System.out.println("삭제되는 첨부파일 목록: " + attachFileList.toString());

    attachFileList
        .forEach(attachFile -> {
            try {
                Path file = Paths.get(attachFile.getRepoPath() + "/" + //C:\upload\ +
                    attachFile.getUploadPath() + "/" +
                    attachFile.getUuid() + "_" +
                    attachFile.getFileName());
                Files.deleteIfExists(file);

                if(Files.probeContentType(file).startsWith("image")) {
                    Path thumbNail = Paths.get(attachFile.getRepoPath() + "/" + //C:\upload\ +
                        attachFile.getUploadPath() + "/s_" +
                        attachFile.getUuid() + "_" +
                        attachFile.getFileName());
                    Files.delete(thumbNail);
                }
            } catch(Exception e) {
                System.out.println("파일삭제 오류 발생" + e.getMessage());
            }
        });
    } //End 의명블록, forEach
} //End Method

```

☞ 위의 메소드가 실행되었을 때 첨부파일이 삭제되는 과정은 다음과 같습니다.

Path 인터페이스 타입으로 Paths final 클래스의 static get() 메서드를 통해, 첨부파일의 경로와 파일이름을 file 객체에 저장  
 → Files final 클래스의 deleteIfExists() 메서드로 첨부파일을 먼저 삭제  
 → 삭제된 파일이 이미지 파일인 경우,  
 → 동일한 방법으로 썸네일의 Path 객체를 생성한 후, Files.delete() 메서드로 썸네일 삭제

☞ java.nio.file 패키지의 Files/Paths/Path 에 대한 자세한 사항은 JAVA-API 문서를 참고합니다.

☞ 위의 메서드는 com.spring5213.mypro00.common.fileupload 패키지의 FileUploadController에 구현할 수도 있지만 게시물과 관련된 첨부파일 목록을 처리하므로, MyBoardController에 구현합니다.

→ removeBoard() 메서드를 다음처럼 수정합니다.

```
//특정 게시물 실제 삭제
@PostMapping("/remove")
public String removeBoard(@RequestParam("bno") Long bno,
                         RedirectAttributes redirectAttr,
                         MyBoardPagingDTO myBoardPagingDTO ){
    log.info("MyBoardController.removeBoard에 전달된 bno: " + bno);
    log.info("MyBoardController.removeBoard에 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    //첨부파일 정보를 저장할 리스트 객체 생성
    List<MyBoardAttachFileVO> attachFileList = myBoardService.getAttachFilesByBno(bno); ←추가

    //게시물 삭제가 성공한 후,
    if(myBoardService.removeBoard(bno)) {
        //첨부파일 삭제(MyBoardController에 새로 추가된 메서드)
        removeAttachFiles(attachFileList); ← 추가
        redirectAttr.addFlashAttribute("result", "successRemove");

    } else { ← else 절 추가
        redirectAttr.addFlashAttribute("result", "failRemove");
    }
    //처리 결과 메시지 추가
    redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
    redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
    redirectAttr.addAttribute("scope", myBoardPagingDTO.getScope());
    redirectAttr.addAttribute("keyword", myBoardPagingDTO.getKeyword());
    log.info("화면으로 전달될 redirectAttr: " + redirectAttr); //추가

    return "redirect:/myboard/list";
}
```

☞ MyBoardController의 removeBoard() 메서드는, 게시물의 첨부파일 목록을 저장 후(attachFileList) 목록페이지로 리다이렉트 됩니다.

### (1-5-3) modify.jsp 파일 수정

→ modify.jsp 파일에 게시물을 실제로 삭제하는 요청이 가능하도록 ,

기존 삭제 버튼의 클릭이벤트 jQuery 함수의 전송 요청 URL을 다음처럼 수정합니다.

```
//form의 수정/삭제/목록보기 버튼에 따른 동작 제어
var frmModify = $("#frmModify");

$('.btn-frmModify').on("click", function(e){
    //e.preventDefault(); //버튼 유형이 submit가 아니므로 설정할 필요 없음
    var operation = $(this).data("oper"); //각 버튼의 data-xxx 속성에 설정된 값을 저장
    //alert("operation: " + operation);

    if(operation == "modify"){ //게시물 수정 요청
        ... (생략) ...
    } else if(operation == "remove"){ //게시물 삭제 요청
        //frmModify.attr("action", "${contextPath}/myboard/delete");
        frmModify.attr("action", "${contextPath}/myboard/remove"); ← 기존라인 주석처리 후, 추가
    }
});
```

```

} else if(operation == "list"){ //게시물 목록 화면 요청
...(생략)...
}

frmModify.submit() ; //요청 전송
});

</script>

```

### [테스트 - 게시물 수정 시의 첨부파일 수정 및 게시물 삭제 시의 첨부파일 삭제 테스트] - 아래의 절차대로 스스로 수행해서 확인!!!

- ☞ 웹 브라우저에서의 테스트를 아래의 절차대로 스스로 수행하십시오.
- ☞ 테스트 시에 웹브라우저의 개발자 도구를 실행해 놓고, 테스트를 수행하십시오. 개발자 도구를 통해 JSP 페이지의 HTML 요소 오류나 자바스크립트 오류를 쉽게 찾을 수 있으므로, 오류 조치 시에 편리합니다
- ☞ 이클립스 콘솔에서의 로그도 체크합니다. 컨트롤러, 서비스에서의 처리상황이나 오류를 탐지할 수 있습니다.
- 테스트를 원활히 진행하기 위해 이전의 테스트 내용을 모두 삭제한 후 테스트를 수행하면, 결과 확인이 편리합니다.
  - SQL\*Developer로 book\_ex 계정으로 데이터베이스에 접속하여 TBL\_MYATTACHFILES 테이블에 저장된 이전 실습 데이터를 모두 삭제합니다.
  - 파일 탐색기에서 서버의 업로드 폴더(C:\myupload)에 업로드 된 기존 첨부파일을 모두 삭제합니다(temp 폴더는 유지).
- 테스트 수행
  - 톰캣 서버를 기동하고, 웹 브라우저에서 <http://localhost:8080/mypro00/myboard/list> URL 요청
  - 첨부파일이 있는 게시물 상세 페이지로 이동
  - 게시물 상세 페이지에서 수정 버튼을 클릭 → 게시물 수정-삭제 페이지로 이동
  - 게시물 수정-삭제 페이지에서 다음을 수행
    - 첨부파일 변경: 기존 첨부파일 삭제/새로운 첨부파일 추가 → 동작이 잘되는지 확인(폴더의 내용 확인)
      - 새로운 첨부파일 추가 시, 서버 업로드 폴더(C:\myupload)에, 날짜형식 폴더 생성 후, UUID가 포함된 파일이 정상적으로 저장되는지 파일 탐색기로 확인, 이미지 파일의 경우, 썸네일도 같이 생성되는지 확인
      - 첨부파일 삭제, 화면에서 항목이 삭제되는지 확인(실제 파일은 삭제되지 않음)
    - 게시물 내용 수정 수행
  - 수정 버튼 클릭 → 게시물 상세 페이지로 이동 → 첨부파일 정보 및 수정된 사항이 잘 표시되는지 확인
  - 게시물 상세 페이지에서 → 수정 버튼을 클릭 → 게시물 수정-삭제 페이지로 다시 이동
    - 삭제 버튼을 클릭 → 게시물 삭제

- 게시물 삭제 시, 파일 탐색기에서 업로드 된 파일도 삭제(썸네일 파일 포함)되는지 확인
- 게시물 삭제 후 이동된 게시물 목록에서 게시물 정보가 삭제되어 표시되지 않는지 확인

[주의] MS-IE에서 테스트 시 첨부파일 변경이 확인되지 않을 때 조치

제어판 → 인터넷 옵션(IE-웹브라우저의 [도구 메뉴]의 인터넷 옵션)을 실행

- "일반" 탭에 있는 "웹 사이트 데이터 설정" 페이지를 실행
- "저장된 페이지의 새 버전 확인" 항목을 "웹 페이지를 열 때마다" 값으로 변경합니다.
- "고급" 탭에 있는 "Internet Explorer 설정 및 데이터 동기화 사용"을 체크합니다.

위의 설정을 웹브라우저에 수행해야만, 게시물의 첨부파일을 수정 후, 동일 브라우저에서 다시 게시물 조회 페이지를 확인했을 때 수정된 내용이 확인됩니다. **위의 사항을 구성하지 않으면, 브라우저에 캐싱된 이전 이미지의 썸네일이 표시되므로 변경된 사항이 확인되지 않습니다.** 크롬이나 MS-Edge에서는 위와 같은 문제가 발생되지 않습니다.

## 11-9. 첨부파일 구현 실습: 프로젝트에 적용 - 필요 없는 첨부파일 파일 처리(@Scheduled)

### [부제] 데이터베이스에 정보 없이 서버에 저장된 불필요한 업로드 파일의 처리

☞ 앞의 구현 작업을 통해, 게시물 등록 페이지에서 등록할 게시물을 작성할 때 첨부할 첨부파일을 선택하면 첨부파일은, 자바스크립트의 Ajax를 통해 이미 서버에 전송되어 저장된 상태가 됩니다. 이 구현 방식은 게시물 등록 작업 중의 이미지 확인이 가능하고 게시물 등록 작업의 로드 분산이라는 장점이 있지만, 다음과 같은 상황이 문제가 될 수 있습니다.

- 게시물 등록 중에 첨부파일만 등록되고, 게시물을 등록하지 않은 경우.
- 게시물의 수정 시에, 데이터베이스의 첨부파일의 정보만 삭제된 경우.

☞ 위의 상황은 게시물 등록/수정 작업 중에 첨부파일의 추가/수정만 해 놓은 상태에서 사용자가 게시물 등록/수정을 완료하지 않고 웹브라우저를 종료할 경우 발생됩니다.

☞ 위의 상황이 발생되면, 데이터베이스에는 해당 정보가 없는 첨부파일이 레포지토리 폴더에서 존재하게 됩니다.

☞ 게시물 등록/수정을 마치지 않고 등록 페이지를 빠져 나가거나 또는 비정상적으로 브라우저가 종료되는 경우가 빈번한 경우, 서버 상의 업로드 파일 저장공간이 필요없이 소모되는 결과가 됩니다.  
따라서, 서비스 운영 중에, 정상적으로 업로드 된 파일 정보(데이터베이스에 파일 정보가 존재함)와 저장된 파일 정보를 비교하여, 데이터베이스에 해당 정보가 없이 저장된 첨부파일들을 삭제해주어야 합니다.

[비교] 위의 상황과 다르게 데이터베이스 정보는 있는데 실제 첨부파일이 없는 경우는, 최대한 발생되지 않도록 주의해야 합니다. 이유는 게시물 조회 서비스 상에 오류가 발생되기 때문입니다. 이러한 상황이 발생될 수 있는 경우는, 게시물 삭제 시에, 정보보다 첨부파일이 먼저 삭제된 상황에서 웹브라우저와 서버의 접속이 비 정상적으로 종료되었을 때 발생될 수 있습니다. 이를 방지하기 위하여 정보를 먼저 삭제하고 첨부파일이 삭제되도록 구현하게 됩니다.

### [참고] 데이터베이스 정보가 없는 첨부파일의 정리

☞ 정상적인 게시물 등록이나 수정 시에 서버에 저장된 첨부파일의 정보는, 본 문서에서는 데이터베이스의 `tbl_myAttachFiles` 테이블에 저장됩니다. 이 정보들과 서버의 업로드 저장 레포지토리 폴더의 파일을 비교하여, 필요없이 업로드 된 파일들을 찾아 삭제합니다. 이 삭제작업은 일정을 잡아 주기적으로 반복해서 수행됩니다.

☞ 불필요한 업로드 파일들을 찾을 때에는, 오늘 날짜가 아닌 과거 날짜의 파일들만들 대상으로 해야합니다. 즉, 지금 현재 새로운 게시물을 등록하기 위하여 작성중이거나 또는 수정 중인 게시물의 첨부파일들이 정리를 위한 삭제작업 중에, 삭제될 수도 있기 때문에, 보통 현재가 포함된 오늘 날짜는 제외시킵니다.

☞ 매일, 매주, 매월 등 주기적으로 동일하게 수행되는 작업을 보통 "스케줄링-작업"이라고 하며, 이러한 주기적인 수행 작업은, 주로 시스템 관리자에 의하여 운영체제 명령어(at, crond)를 이용하여 처리됩니다.

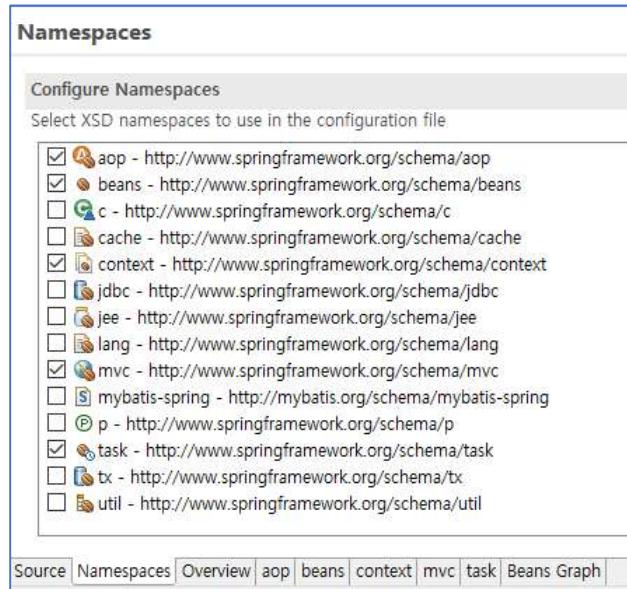
☞ 개발자들은 Spring-Framework 를 기반으로 간단히 사용할 수 있는 Quartz 라이브러리를 이용하여, 웹서비스 운영 시에, 프로그램적으로 주기적인 스케줄링-작업을 구성했습니다.

- 문서에서는 Quartz 라이브러리를 사용하지 않고, Spring-Framework 에 기본적으로 포함되어 있는 org.springframework.scheduling.annotation.Scheduled 인터페이스를 이용하여 주기적인 스케줄링 작업을 구성합니다.

## (1) 스프링 @Scheduled 어노테이션을 사용하기 위한 구성

(1-1) 스케줄링-작업을 위한 어노테이션을 이용하기 위한 스프링 DispatcherServlet 의 빈 설정파일에 task 네임스페이스 추가

- src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml 파일을 코드 작성부에 오픈
- 코드 작성 창 하단의 Namespaces 탭을 클릭 후, 스프링의 task 네임스페이스 추가 → 저장. → Source 탭 클릭



다음은 task 네임스페이스를 추가한 후의 servlet-context.xml 파일의 변경된 부분입니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:task="http://www.springframework.org/schema/task"
  xsi:schemaLocation="http://www.springframework.org/schema/mvc
    https://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task-4.3.xsd
    http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
```

## (1-2) task 스케줄러 빈 등록

→ servlet-context.xml Source 탭을 클릭 후, 다음의 내용을 기준 내용 밑에 추가합니다(빨간색 코드).

```
<!-- 스케줄 아이디: clearUploadRepoJobScheduler, ThreadPool 개수: 10개 -->
<task:scheduler id="clearUploadRepoJobScheduler" pool-size="10" />
<task:annotation-driven scheduler="clearUploadRepoJobScheduler" />

</beans:beans>
```

## (2) 불필요한 첨부파일 삭제를 위한 자동 실행 작업 구성

☞ 현재를 기준으로 하루 전 날짜 폴더에서 필요없는 첨부파일이 자동으로 삭제되도록 구성합니다.

### (2-1) 현재를 기준으로 하루 전 날짜의 모든 첨부파일 목록 확인

☞ 첨부파일 관련 SQL 매퍼 XML 파일에 데이터베이스에서 어제(하루 전)에 등록된 모든 첨부파일의 목록이 조회하는 SQL을 추가합니다.

→ src/main/resources/com/spring5213/mypro00/mapper/MyBoardAttachFileMapper.xml 파일을 코드 작성부에 오픈

→ 다음 SELECT 문을 </mapper> 위에 추가합니다(sysdate에서 1을 빼서 하루 전 날짜를 처리).

```
<!-- 하루 전의 첨부파일 정보 조회 -->
<select id="selectAttachFilesBeforeOneDay" resultType="com.spring5213.mypro00.domain.MyBoardAttachFileVO">
    <![CDATA[
        SELECT * FROM book_ex.tbl_myAttachFiles
        WHERE uploadpath = TO_CHAR(sysdate - 1, 'yyyy\mm\dd')
    ]]>
</select>

</mapper>
```

☞ uploadpath 컬럼은 실제 날짜가 아니라 날짜형식의 디렉토리 문자열입니다. 위의 SELECT 문은 하루 전 날짜의 폴더값을 가진 첨부파일들의 정보를 반환합니다. 만약 WHERE 절의 = 을 <= 로 변경하면, 어떤 데이터가 조회될지 스스로 예측해 보세요. 어제(하루 전) 이전의 모든 첨부파일 정보가 조회되는 것으로 바람직하지 않습니다.

→ src/main/java/com.spring5213.mypro00.mapper.MyBoardAttachFileMapper.java 인터페이스를 코드 작성부에 오픈

→ MyBoardAttachFileMapper.xml 파일에 추가된 SELECT 문을 실행하는 메서드를 추가합니다.

```
public List<MyBoardAttachFileVO> selectAttachFilesBeforeOneDay();
```

### (2-2) 자동 실행 작업이 구현된 클래스 생성

☞ 하루 전 날짜 폴더에서 필요없는 파일을 삭제하는 실행 코드를 가진 클래스를 생성합니다.

이 클래스를 스프링 스캐줄러가 자동으로 실행하여 합니다.

→ src/java/main/com.spring5213.mypro00.common 패키지에 task 패키지를 생성하고,

task 패키지에 ClearMyUploadFileRepo 클래스를 생성한 후, 다음의 내용을 구현합니다. 작성 시에 주석은 제외하고 작성하세요.

```
package com.spring5213.mypro00.common.task;
```

```
import java.io.File;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;

import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
```

```

import com.spring5213.mypro00.domain.MyBoardAttachFileVO;
import com.spring5213.mypro00.mapper.MyBoardAttachFileMapper;

import lombok.AllArgsConstructor;

@Component
@AllArgsConstructor
public class ClearMyUploadFileRepo {

    private MyBoardAttachFileMapper myBoardAttachFileMapper;

    //하루 전 폴더 문자열 생성 메서드
    private String getStrOfYesterdayFolder() {
        //문자열 형식 지정
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy/MM/dd");
        //달력객체(년/월/일)생성
        Calendar calendar = Calendar.getInstance();
        //달력에서의 하루전 날짜를 생성해서 달력 객체에 추가
        calendar.add(Calendar.DATE, -1);
        //달력객체에 설정된 밀리세컨트 값을 가져와서 날짜형식 문자열로 변환
        String strYesterdayFolder = simpleDateFormat.format(calendar.getTime());
        //문자열의 날짜구분자를 운영체제 디렉토리 구분자로 변경
        return strYesterdayFolder.replace("/", File.separator);
    }

    //하루 전 날짜 폴더에 있는 파일 중, DB에 정보가 없는 필요없는 파일 삭제
    //spring-task 의 스케줄 기능으로 자동으로 실행되는 메서드
    //매일 새벽 3시에 자동으로 실행,
    @Scheduled(cron = "0 0 3 * * *") // 테스트 시에 "0 분 * * * *" 로 분을 수정하세요. 테스트 후에는 주석처리
    public void clearNeedlessFiles() throws Exception {

        String uploadFileRepoDir = "C:\\\\myupload" ;

        System.out.println("파일 검사 작업.....");
        System.out.println("오늘 날짜: " + new Date());

        //데이터베이스에 저장된 하루 전 첨부파일정보목록 생성(삭제하면 않되는 파일들)
        List<MyBoardAttachFileVO> doNotDeleteFileList = myBoardAttachFileMapper.selectAttachFilesBeforeOneDay();
        //데이터베이스의 파일목록을 자바의 Stream 반복자로 처리하여 파일경로의 리스트 객체를 생성
        //지우면 않되는 첨부파일들의 경로목록
        List<Path> doNotDeleteFilePathList
            = doNotDeleteFileList
                .stream() //자바의 Stream 기능 사용
                .map(myBoardAttachFile -> Paths.get(myBoardAttachFile.getRepoPath(), //uploadFileRepoDir,
                                                    myBoardAttachFile.getUploadPath(),
                                                    myBoardAttachFile.getUuid() + " " +
                                                    myBoardAttachFile.getFileName()))
        //Paths객체들을, List 컬렉션 객체로 변환
        .collect(Collectors.toList());
        //썸네일 파일 정보 추가
        doNotDeleteFileList
            .stream()
            //파일정보객체 중, 이미지파일 정보만 골라냄
            .filter(myBoardAttachFile -> myBoardAttachFile.getFileType().equals("I"))
            //이미지파일boardAttachFileVO 객체를 이미지의 썸네일 경로객체로 교체
            .map(myBoardAttachFile -> Paths.get(myBoardAttachFile.getRepoPath(), //uploadFileRepoDir,
                                                myBoardAttachFile.getUploadPath(),
                                                "s_" + myBoardAttachFile.getUuid() +
                                                "_" + myBoardAttachFile.getFileName()))
        //각각의 썸네일 경로객체를 경로 목록에 추가
        .forEach(doNotDeleteFilePath -> doNotDeleteFilePathList.add(doNotDeleteFilePath));
    }
}

```

```

System.out.println("=====");
//최종 지우면 않되는 파일(썸네일 포함)의 경로목록
doNotDeleteFilePathList
//각 경로를 콘솔에 하나씩 출력(배포시엔 주석처리)
.forEach(doNotDeleteFilePath -> System.out.println(doNotDeleteFilePath));

//하루 전 날짜경로가 저장된 파일 객체
File dirBeforeOneDay =
    Paths.get(uploadFileRepoDir, getStrOfYesterdayFolder()) //하루 전 날짜 경로 객체(Path 객체) 가져와서
    .toFile(); //Path 객체를 File 객체로 변환

//하루 전 날짜폴더에 있는 모든 파일들을 DB로 부터 가져온 파일정보와 비교하여
//정보가 없는 파일들의 File 객체가 저장된 배열 생성
File[] needlessFileArray
= dirBeforeOneDay //하루 전 날짜폴더 파일객체
.listFiles(
    //각파일을 Path 객체로 변환하여
    //디렉토리의 각 파일의 Path 객체를
    //doNotDeleteFilePathList(DB에 정보가 있는 파일) 목록의
    //Path 객체와 비교하여 DB에 없는(false) 디렉토리 파일들의 File객체를
    //needlessFileArray File[] 배열에 저장
    eachFile -> doNotDeleteFilePathList.contains(eachFile.toPath()) == false);
System.out.println("-----");

//필요없는 파일 삭제
if (needlessFileArray == null) {
    System.out.println("===== 삭제할 파일이 없습니다.=====");
    return ;
} else {
    for (File needlessFile : needlessFileArray) {

        //삭제파일 정보 표시(DB에 정보가 없는 파일)
        System.out.println("=====다음의 파일들이 삭제됩니다.=====");
        System.out.println("필요없는 파일 이름: " + needlessFile.getAbsolutePath());
        //필요없는 파일 삭제
        needlessFile.delete();
    }
}
}

}

```

☞ 작성된 ClearMyUploadFileRepo.clearNeedlessFiles() 메서드의 작업의 순서는 다음과 같습니다.

- 1) 오늘을 기준으로 데이터베이스로부터 하루 전 첨부파일들의 데이터를 가져와서, 첨부파일들의 목록(doNotDeleteFilePathList)을 생성합니다. 이 때 가져온 목록은 MyBoardAttachFileVO 타입 객체들의 목록입니다.
- 2) 생성된 MyBoardAttachFileVO 타입 객체들의 목록을 변환하여, 삭제하면 안되는 파일의 경로(파일이름 포함)들의 목록 (Path 타입 객체들의 목록, doNotDeleteFilePathList)을 생성하며, 이 때, 썸네일파일에 대한 경로도 포함시킵니다.
- 3) 하루 전 날짜 경로가 포함된 전체 경로 정보를 가진 File 유형 객체(dirBeforeOneDay)를 생성합니다.
- 4) 스텝 3에서 생성된 File 유형 객체(dirBeforeOneDay)에 있는 파일들을 지우면 않되는 파일들의 경로목록(doNotDeleteFilePathList)으로 확인하여, 필요없는 파일들의 경로(파일이름 포함)의 목록(needlessFileArray)을 생성합니다.
- 5) 필요없는 파일들의 경로 목록의 파일을 삭제합니다.

☞ ClearMyUploadFileRepo 클래스는 스프링에 의해서 자동으로 빈이 생성되며(@Component 어노테이션), clearNeedlessFiles() 메서드는 @Scheduled 어노테이션의 cron 속성에 의하여 매일 새벽 3시에 자동으로 실행되어, 필요없는 파일들을 서버에서 삭제합니다.

(2-3) 자동 실행 클래스(ClearMyUploadFileRepo 클래스)의 빈 생성 등록

→ src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml 파일을 코드 작성창에 오픈,

다음의 빨간색 코드를 추가합니다.

...(생략)...

```
<context:component-scan base-package="com.spring5213.mypro00.service" />
<context:component-scan base-package="com.spring5213.mypro00.controller" />
<context:component-scan base-package="com.spring5213.mypro00.common.fileupload" />
<context:component-scan base-package="com.spring5213.mypro00.common.filownload" />
<context:component-scan base-package="com.spring5213.mypro00.common.task" /> ← 추가
```

...(생략)...

[참고] 기본적인 @Scheduled 어노테이션 사용법

☞ @Scheduled 어노테이션이 명시된 메서드는, 반환값과 매개변수를 가질 수 없습니다.

☞ 다음과 같은 방법으로 실행 주기를 설정합니다.

방법	설명
@Scheduled(fixedDelay = 1000)	- fixedDelay 는 앞에 실행된 작업의 종료시간부터 정의된 시간(1초)만큼 지난 후, 뒤의 작업이 실행됩니다.
@Scheduled(fixedRate = 1000)	- fixedRate 는 앞에 실행된 작업의 시작시간부터 정의된 시간(1초)만큼 지난 후, 뒤의 작업이 실행됩니다.
<b>@Scheduled(cron="값들")</b>	- 가장 많이 사용되는 방법이며, 아래의 설명을 참고합니다.

☞ @Scheduled 어노테이션의 cron 속성에는 다음의 형식으로 자동 실행될 주기를 설정합니다.

[형식] @Scheduled(cron = "초 분 시 일 월曜일 년도") //년도 필드만 선택적이며 생략될 수 있습니다.

☞ 각 필드는 빈칸(스페이스 바)로 구분되며, 다음의 방법으로 값을 지정합니다.

필드	기본적인 설정값	필드	기본적인 설정값
초 와 분	0 ~ 59 사이의 정수값	요일	SUN 또는 0 또는 7 또는 MON 또는 1 또는 TUE 또는 2 또는 WED 또는 3 또는 THU 또는 4 또는 FRI 또는 5 또는 SAT 또는 6
시간	0 ~ 23 사이의 정수값		
일	1 ~ 31 사이의 정수값		
월	1 ~ 12 사이의 정수값	년도	정수 4 자리(YYYY)

☞ 각 필드에 다음의 표현식도 사용할 수 있습니다.

기호	의미
*	해당 필드의 모든 값
?	일(Day)과 요일에 주로 사용되며, 특정한 값을 지정하지 않음
-(하이픈)	범위를 지정 3-6 : 3부터 6 의미
,(콤마)	여러 개의 값 설정 TUE,FRI
값 1/값 2	예) 시간필드가 */2로 설정되면, 2시간마다가 됨.
L	일(Day)과 요일에 주로 사용되어 지정할 수 있는 범위의 마지막 값

☞ 표 계속

기호	의미
W	일(Day)에 사용되어, 주말을 제외한 평일(월~금요일)이나 지정된 일에 가장 가까운 월요일이나 금요일 예) 10W로 지정되었을 때 -10 일이 평일이면, 10일에 실행 -10 일이 토요일이면, 가장 가까운 평일인 금요일(9일)에 실행 -10 일이 일요일이면, 가장 가까운 평일인 월요일(11일)에 실행
#	요일에 사용되어 몇째 주 인지를 지정(요일번호#주의순서) 예) 3#2 : 2째 주 수요일에 실행

☞ 다음의 위의 기호들을 사용한 간단한 설정 예제입니다.

- 매월 10일 오전 1시 : 0 0 1 10 \* \*
- 매일 오후 2시 5분 0초: 0 5 14 \* \* \*
- 매시 10분마다(10분 0초, 20분 0초,...): 0 \*/10 \* \* \*
- 매시 10분, 11분, 12분, 13분에 실행: 0 10-13 \* \* \* 또는 0 10,11,12,13 \* \* \*

[테스트 준비]

→ 톰캣 서버를 기동한 후, 첨부파일이 있는 새로운 게시물을 등록합니다.

→ 추가된 게시물의 첨부파일을 수정(기존 첨부파일 삭제 및 새로운 첨부파일 추가)합니다.

→ 게시물 수정 시에, 첨부파일 변경을 위해 기존 첨부 파일을 삭제하면, 나중에 게시물 수정 시에 데이터베이스 정보는 삭제되지만, 실제 서버의 업로드 폴더에 파일은 삭제되지 않습니다.

→ 즉, 이렇게 남아있는 파일들을 자동으로 삭제하도록 구현한 것을 테스트 하는 것입니다.

→ 첨부파일 변경 후, 반드시 게시물 수정 버튼을 클릭해야, 변경사항이 데이터베이스에 반영됩니다.

→ 첨부파일들을 하루 전 첨부파일 상태로 변경해 주기 위하여,

추가된 첨부파일들의 Day 폴더를 하루 전 Day 의 폴더로 변경합니다.

실습에서는 새로 생성된 첨부파일이 2021/06/29 폴더에 저장되어 있습니다.

해당 Day 폴더 이름을 **28**로 변경합니다. 이미 28 폴더가 있다면, **29** 폴더에 있는 모든 파일을 28 폴더로 이동시킵니다.

→ 그런 후, 28 폴더에 다른 파일 몇 개 더 붙여넣기 합니다. 이 파일들도 같이 삭제되는지 확인합니다.

→ SQL\*Developer 로 book\_ex 로 접속하여 다음 준비 작업은 데이터의 변경입니다. 현재 추가된 게시물의 오늘날짜 폴더(2020\08\06)에 저장되어 있는 정보를 하루전 날짜 폴더로 변경합니다.

→ SQL\*Developer 에서 다음작업을 수행합니다.

```
UPDATE book_ex.tbl_myAttachFiles  
SET uploadpath = '2021\06\28'  
WHERE uploadpath = '2021\06\29';  
  
COMMIT ;
```

☞ 데이터베이스의 uploadpath 컬럼 값이 하루 전 폴더로 (2021\06\28) 변경되었습니다.

☞ 테스트 준비가 완료되었습니다.

→ 이클립스에서 ClearMyUploadFileRepo 클래스의 clearNeedlessFiles() 메서드에 설정된 @Scheduled(cron = "0 0 3 \* \* \*") 설정의 분 필드(빨간색 숫자)의 값을, 현재 시간의 2분뒤(실습에서는 @Scheduled(cron = "0 40 \* \* \*") )로 설정 후, 저장합니다.

→ 2분 뒤에 필요없는 파일들이 자동으로 삭제되는지 확인합니다.

→ 이 때, 게시물에 등록되어 데이터베이스에 첨부파일 정보가 있는 파일들(첨부파일과 썸네일파일)은 유지되고, 데이터베이스에 정보가 없는 파일들만 삭제되어야 합니다.

☞ 다음은 필요없는 업로드파일이 자동으로 삭제될 때, 이클립스 콘솔에 표시된 내용입니다.

```
파일 검사 작업 시작.....  
오늘 날짜: Tue Jun 29 09:40:00 KST 2021  
INFO : jdbc.sqlonly - SELECT uuid, uploadPath, fileName, fileType, bno FROM book_ex.tbl_myAttachFiles WHERE uploadpath  
= TO_CHAR(sysdate -1 , 'yyyy\mm\dd')  
INFO : jdbc.sqltiming - SELECT uuid, uploadPath, fileName, fileType, bno FROM book_ex.tbl_myAttachFiles WHERE uploadpath  
= TO_CHAR(sysdate -1 , 'yyyy\mm\dd')  
{executed in 282 msec}  
INFO : jdbc.resultsettable -  
|-----|-----|-----|-----|-----|  
|uuid|uploadpath|filename|filetype|bno|  
|-----|-----|-----|-----|-----|  
|cbca220c-d2aa-4780-9350-d996ef50be11|2021\06\28|img105.jpg|I|458763|  
|4e2dece2-8998-46be-a6b4-fc735a7efab2|2021\06\28|샘플PDF.pdf|F|458763|  
|ce862f93-dbdf-4679-a68a-2aecb95ae5cc|2021\06\28|샘플이미지104.jpg|I|458763|  
|-----|-----|-----|-----|-----|  
=====  
C:\myupload\2021\06\28\cbca220c-d2aa-4780-9350-d996ef50be11_img105.jpg  
C:\myupload\2021\06\28\4e2dece2-8998-46be-a6b4-fc735a7efab2_샘플PDF.pdf  
C:\myupload\2021\06\28\ce862f93-dbdf-4679-a68a-2aecb95ae5cc_샘플이미지104.jpg  
C:\myupload\2021\06\28\s_cbca220c-d2aa-4780-9350-d996ef50be11_img105.jpg  
C:\myupload\2021\06\28\s_ce862f93-dbdf-4679-a68a-2aecb95ae5cc_샘플이미지104.jpg  
=====  
=====다음의 파일들이 삭제됩니다.=====  
필요없는 파일 이름: C:\myupload\2021\06\28\05fe094e-e0a1-4e36-9d46-19d13046397f_샘플이미지img105.jpg  
=====다음의 파일들이 삭제됩니다.=====  
필요없는 파일 이름: C:\myupload\2021\06\28\564cd745-1b1d-46c0-a4dc-43e0c71854d2_샘플PDF.pdf  
=====다음의 파일들이 삭제됩니다.=====  
필요없는 파일 이름: C:\myupload\2021\06\28\6bbdd174-2fd9-4726-aa9d-7cc22411e20a_샘플이미지img105.jpg  
=====다음의 파일들이 삭제됩니다.=====  
필요없는 파일 이름: C:\myupload\2021\06\28\7f562717-ae76-4f1f-8f0e-59000324f8d4_샘플PDF.pdf  
=====다음의 파일들이 삭제됩니다.=====  
필요없는 파일 이름: C:\myupload\2021\06\28\df402715-f458-48cd-ac3e-b49706f12740_샘플PDF.pdf  
=====다음의 파일들이 삭제됩니다.=====  
필요없는 파일 이름: C:\myupload\2021\06\28\img100.jpg
```

```
=====다음의 파일들이 삭제됩니다.=====
필요없는 파일 이름: C:\myupload\2021\06\28\img101.png
=====다음의 파일들이 삭제됩니다.=====
필요없는 파일 이름: C:\myupload\2021\06\28\img102.jpg
=====다음의 파일들이 삭제됩니다.=====
필요없는 파일 이름: C:\myupload\2021\06\28\img103.png
=====다음의 파일들이 삭제됩니다.=====
필요없는 파일 이름: C:\myupload\2021\06\28\img104.jpg
=====다음의 파일들이 삭제됩니다.=====
필요없는 파일 이름: C:\myupload\2021\06\28\img105.jpg
=====다음의 파일들이 삭제됩니다.=====
필요없는 파일 이름: C:\myupload\2021\06\28\s_05fe094e-e0a1-4e36-9d46-19d13046397f_샘플이미지img105.jpg
=====다음의 파일들이 삭제됩니다.=====
필요없는 파일 이름: C:\myupload\2021\06\28\s_6bbdd174-2fd9-4726-aa9d-7cc22411e20a_샘플이미지img105.jpg
```

→ 실습 완료 후에, `ClearMyUploadFileRepo` 클래스의 `@Scheduled` 어노테이션을 주석처리합니다.

주석처리를 않하면, 이 후 실습에서 필요없이 계속 파일 삭제가 시도될 수 있습니다.

## [12] 사용자 접속 및 액세스 제어: 스프링 시큐리티

- 프로젝트에 스프링 시큐리티 적용을 위해 다음의 작업들이 진행됩니다.

### 1. 영속계층 구현: 인증 및 승인 시에 필요한 데이터 유지 및 데이터베이스 액세스 구성

- 데이터베이스 저장구조 구성
- VO 클래스 생성 및 Mybatis 매퍼 구성

### 2. 스프링 시큐리티 기본구성:

- 스프링 시큐리티 라이브러리 설치
- 승인/인증을 위해 필요한 데이터 사용 구성: 데이터 사용을 위해 필요한 스프링 시큐리티 빈 클래스 생성
- 스프링 시큐리티 컨텍스트 설정파일 생성 및 설정 구성
- 사용자 암호에 대한 스프링 시큐리티 암호화 구성
- 스프링 시큐리티 어노테이션 사용 활성화 설정.

### 3. 스프링 시큐리티 활용

- 로그인/로그아웃 처리
  - 커스텀 로그인/로그아웃 JSP 페이지 사용 구성
  - 로그인/로그아웃 후, 특정 URL 페이지로 이동 구성
- 권한 부족으로 인한 접근 금지 오류 시의 처리 로직 구현
  - AccessDeniedHandler 구현 클래스 이용
- 로그인 성공 후, 수행될 내용 구성: 관련 빈 클래스 생성 및 설정
- 스프링 시큐리티 어노테이션과 스프링 시큐리티 표현식을 이용한 액세스 제어 구현

### 12-1 스프링 시큐리티 개요

☞ 거의 대부분의 웹 애플리케이션 프로젝트에서는, 사용자의 권한이나 등급에 기반을 두는 로그인 처리를, 웹에서 기본적으로 제공하는 쿠키나 세션으로 구현합니다. 스프링에서도 예전에는 스프링 인터셉터(Interceptor)를 이용하여 동일한 기능을 구현하였습니다.

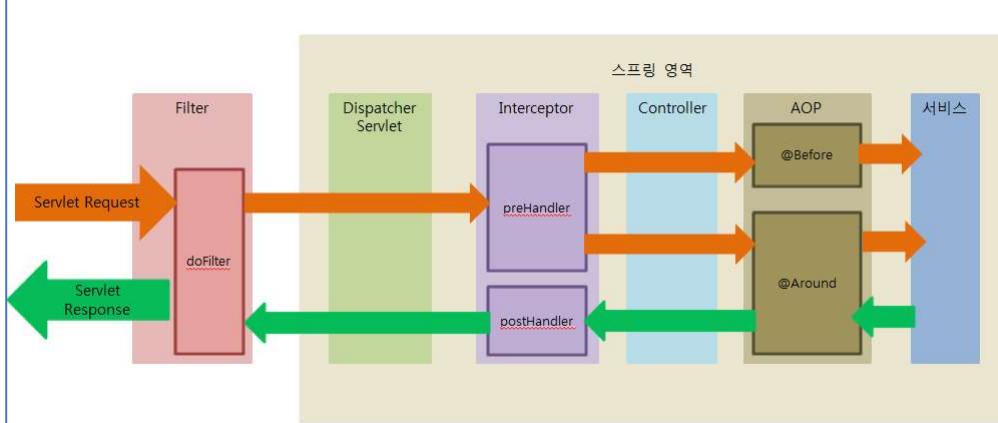
☞ 스프링 시큐리티를 이용하면, 스프링 인터셉터를 직접 이용하는 것보다, 사용자의 로그인 처리 시에 필요한 인증이나 승인 로직을 보다 수월하게 구현할 수 있습니다.

☞ 스프링 웹 시큐리티는, 사용자ID(username), 사용자 암호(password), 사용자 권한(authority) 같은 사용자의 정보 데이터를 이용하여, 사용자의 로그인(인증, Authentication)/로그아웃, 웹페이지에 대한 액세스 제어(Authorization) 등을 구현할 수 있도록 제공되는 라이브러리이며, 다음의 작업을 쉽게 구현할 수 있습니다.

- 로그인 처리와 CSRF 토큰 처리
- 암호화 처리
- 자동로그인
- JSP에서의 로그인 처리

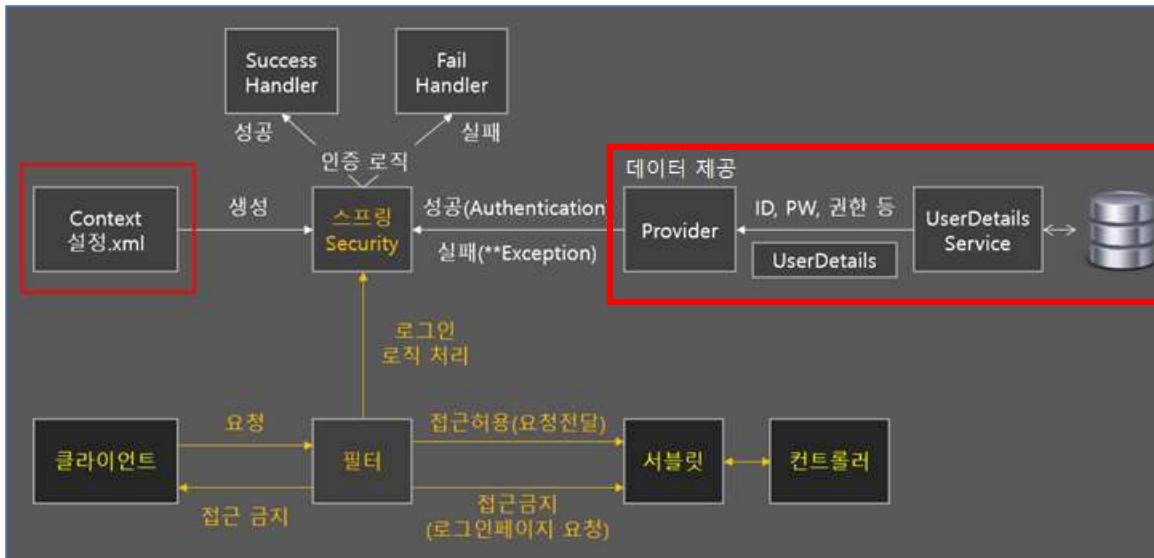
☞ 스프링 시큐리티는 내부적으로 서블릿의 필터(Filter)들과 스프링 인터셉터(Interceptor)를 이용하여 사용자에 대한 인증과 승인 로직을 자동으로 처리하며, 기본적으로 현재 동작하는 스프링 컨텍스트 내에서 동작하므로 이미 컨텍스트에 포함된 여러 빈들을 이용해서, 인증처리를 다양한 방식으로 구현할 수 있습니다.

## ◆ 필터와 스프링 인터셉터란?



- ☞ 필터는 서블릿의 필터를 의미하고, 인터셉터는 스프링에서 필터와 유사한 역할을 하는 객체입니다.
- ☞ 필터와 인터셉터는 특정한 서블릿이나 컨트롤러의 접근에 관여한다는 점에서는 유사하지만, 필터는 스프링과 무관한 서블릿(톰캣)의 자원이고, 인터셉터는 스프링의 빈으로 관리되는 스프링 컨텍스트 내의 자원이라는 점에서 차이가 납니다.
- ☞ 앞의 그림에서, 사용자 요청은 일단 톰캣에 전달되어 서블릿 컨텍스트의 자원인 필터에 의해 처리된 후, 스프링 컨텍스트 영역에 있는 Dispatcher-Servlet에게 전달됩니다. 사용자의 요청은 Dispatcher-Servlet에 의해 인터셉터를 거쳐 컨트롤러에게 전달됩니다.

## ◆ 다음 그림은 스프링 시큐리티를 구성하는 요소들(필터와 인터셉터 포함)들을 간단하게 보여줍니다.



[그림] 스프링 시큐리티 구성 요소 및 처리 과정 개요

- ◆ 스프링 시큐리티에 의해 사용자에 대한 인증과 승인이 처리되기 위해서 다음의 요소들이 필요합니다.

1. 스프링 시큐리티에서 사용되는 DelegatingFilterProxy 빈을 생성하는 필터 ← **web.xml에 설정으로 자동 구성됨**
2. 승인/인증을 위하여 필요한 사용자 정보를 스프링 시큐리티로 전달하는 클래스(2개의 클래스)
  - UserDetailsService 구현 객체 클래스와 User 클래스(UserDetails의 구현 객체)를 상속받은 클래스
  - 스프링 시큐리티는 내부적으로 Provider를 통해, 위의 클래스들을 사용
3. **로그인 성공/실패 (인증 성공/실패) 후, 추가적인 로직 처리 클래스**(Handler 대신 특정 URL로 리다이렉트 시키는 설정을 대신 사용할 수도 있음)

- 1) 인증 성공 시, 사용자에 대한 Authentication 유형 객체가 스프링 시큐리티에게 유지됨.  
→ Authentication 유형 객체를 전달받아 SuccessHandler의 구현 객체에서 추가적인 로직을 구현
- 2) 인증 실패 시, \*Exception 객체가 스프링 시큐리티에게 전달됨  
→ \*Exception 객체를 전달받아 FailHandler의 구현 객체에서 처리될 추가적인 로직을 구현

☞ 문서에서는 로그인 성공 시, SuccessHandler의 구현 객체 클래스만 생성하여 실습 후, 사용구성을 해제합니다.  
로그인 실패 시는 설정으로 대체합니다.

#### 4. 사용자의 로그인(GET)/로그아웃(GET) 페이지 호출 요청을 처리할 컨트롤러 클래스

5. 권한이 없는 URL에 접근 시, 톰캣의 접근 금지 오류 메시지 전달 대신, 처리할 로직이 구현된 AccessDeniedHandler의 구현 클래스 (URL 설정할 수도 있음)

#### 6. 로그인/로그아웃/접근금지 JSP 페이지

☞ 빨간색 요소들을 직접 작성해야 하며, 이 요소들이 빈으로 구성되도록 별도로 준비된,  
빈 구성 스프링 컨텍스트 XML 파일(security-context.xml)에 설정해야 합니다.

## 12-2. 스프링 시큐리티 사용을 위한 기본 구성

- ☞ 스프링 시큐리티는 승인과 인증을 위해 필요한 아이디(username), 암호(password), 권한(authority) 데이터를, Provider를 통해 다음의 과정을 거쳐 제공받습니다.
- Provider는, UserDetailsService 인터페이스의 구현클래스를 통해 데이터베이스로부터 가져옵니다.
  - UserDetailsService 인터페이스의 구현클래스는, 데이터베이스로부터 가져온 데이터들을 User 클래스(←UserDetails 인터페이스의 구현클래스)를 상속받은 클래스에 저장하여 Provider에게 전달합니다.
  - Provider는 전달된 데이터가 저장된 객체를 스프링 시큐리티에게 전달합니다.
  - 스프링 시큐리티는 전달된 데이터들을 이용하여 사용자의 로그인 인증이나 페이지 액세스 여부 및 기능 사용에 대한 승인 과정을 처리합니다.
- ☞ 문서에서는, 위의 동작 수행에 필요한 구성을 "스프링 시큐리티 사용을 위한 기본 구성"이라고 하겠습니다.
- ☞ 프로젝트에 스프링 시큐리티 기능을 적용하여 사용하기 위해서는, 승인, 인증 데이터를 제공하는 영속계층(Persistence-Layer)을 먼저 구성합니다.
1. 데이터베이스 테이블 생성(승인, 인증을 위한 데이터 저장 테이블)
  2. 도메인(VO) 클래스 생성
  3. Mybatis SQL 매퍼 구성
- ☞ 영속 계층의 구현이 끝나면, 다음을 수행하여 스프링 시큐리티를 사용할 수 있는 기본 구성을 합니다.
4. 스프링 시큐리티 라이브러리 설치
  5. web.xml에 스프링 시큐리티 필터를 추가: 이들 필터들에 의해 스프링 시큐리티의 전체 기능이 동작됩니다.

6. User 클래스를 상속받은 데이터 전달 객체 생성
7. UserDetailsService의 구현 클래스 생성 및 사용자의 / 요청을 처리할 컨트롤러 구성
8. 스프링 시큐리티 빈 설정 XML 컨텍스트 설정파일 생성 및 스프링 시큐리티 기능 활성화
9. 스프링 시큐리티 어노테이션 사용을 위한 설정(servlet-context.xml 파일)

☞ 위의 1 ~ 9 단계를 모두 수행하면, 스프링 시큐리티를 사용할 기본적인 준비가 완료됩니다.

## 12-2-1. 영속계층 구현: 인증 및 승인 시에 필요한 데이터 유지 및 데이터베이스 액세스 구성

### (1) 데이터베이스 저장구조 구성

- ☞ 회원 데이터를 저장하기 위한 다음과 같은 데이터베이스 객체 생성
- book\_ex.tb1\_mymember 테이블: 사용자 정보 저장
  - book\_ex.tb1\_mymember\_authorities 테이블: 사용자 권한 정보 저장

→ SQL\*Developer를 이용하여 오라클 데이터베이스 서버에 book\_ex 계정으로 접속 후, 다음의 SQL문을 실행하여 테이블을 생성

```
-- 회원 정보 저장 테이블(인증 시에 스프링 시큐리티에게 제공될 사용자 정보)
CREATE TABLE book_ex.tb1_mymember(
    userid VARCHAR2(50) PRIMARY KEY,      --스프링 시큐리티에서, username으로 사용됨
    userpw VARCHAR2(100) NOT NULL,        --스프링 시큐리티에서, password로 사용됨
    userName VARCHAR2(100) NOT NULL,
    mregDate TIMESTAMP(0) DEFAULT SYSTIMESTAMP(0), --회원 가입일
    mmodDate TIMESTAMP(0) DEFAULT SYSTIMESTAMP(0), --암호 등 회원 정보변경일(휴면계정/탈퇴 변경날짜는 별도 컬럼 사용)
    mdropFlg CHAR(1) DEFAULT '0',         --'1' 탈퇴, '0' 유지
    enabled CHAR(1) DEFAULT '1'          -- 스프링 시큐리티에서 사용함, '0' 비활성(false), '1' 활성(true)
) TABLESPACE users;                  -- 계정 잠김으로 기준 데이터로 활용 가능

--회원 권한 저장 테이블(인증 시에 스프링 시큐리티에게 제공될 권한 정보 → 승인 시에 계속 사용됨)
CREATE TABLE book_ex.tb1_mymember_authorities (
    userid VARCHAR2(50) NOT NULL,
    authority VARCHAR2(50) NOT NULL,
    CONSTRAINT pk_mymember_authority PRIMARY KEY (userId, authority),
    CONSTRAINT fk_mymember_authority FOREIGN KEY (userId)
        REFERENCES book_ex.tb1_mymember(userId) ON DELETE CASCADE --회원 삭제 시, 회원 권한이 자동 삭제됨(ON DELETE CASCADE)
) TABLESPACE users;
```

### (2) vo 클래스 생성

→ MyAuthorityVO(권한 정보) 클래스: src/main/java/com.spring5213.mypro00.domain.MyAuthorityVO 클래스 생성

```
package com.spring5213.mypro00.domain;

import lombok.Data;
```

```

@Data
public class MyAuthorityVO {

    //book_ex.tbl_mymember_authority 테이블의 컬럼명과 동일하게 필드이름 설정을 권장.
    private String userid;
    private String authority;
}

```

→ MyMemberVO(사용자 정보) 클래스: src/main/java/com.spring5213.mypro00.domain.MyMemberVO 클래스 생성

```

package com.spring5213.mypro00.domain;

import java.sql.Timestamp;
import java.util.List;

import lombok.Data;

@Data
public class MyMemberVO {

    //book_ex.tbl_mymember 테이블의 컬럼명과 동일하게 필드이름 설정을 권장.
    private String userid;
    private String userpw;
    private String userName;
    private Timestamp mregDate;
    private Timestamp mmodDate; //암호 등 계정 정보 수정 날짜시간
    private String mdropFlg;    //'0'(false) - 유지,      '1'(true) - 탈퇴요청
    private boolean enabled;   // 0 (false) - 비활성화 상태,  1 (true) - 활성화 : 휴면계정, 정지상태로 활용
                                // 컬럼 데이터유형은 CHAR(1)

    private List<MyAuthorityVO> authorityList;
}

```

### (3) Mybatis 매퍼 구성 및 사용 설정

→ MyMemberMapper 인터페이스 생성: src/main/java/com.spring5213.mypro00.mapper.MyMemberMapper 인터페이스 생성

```

package com.spring5213.mypro00.mapper;

import com.spring5213.mypro00.domain.MyAuthorityVO;
import com.spring5213.mypro00.domain.MyMemberVO;

public interface MyMemberMapper {

    //회원 조회: 회원 권한도 함께 조회됨 (스프링 시큐리티도 사용함)
    public MyMemberVO selectMyMember(String userid);

    //회원 등록: 회원 등록 시 회원 권한 추가도 같이 수행
    public Integer insertMyMember(MyMemberVO myMember);

    //회원 권한 추가
    public Integer insertMyMemAuthority(MyAuthorityVO myAuthority);

}

```

→ MyMemberMapper XML 매퍼 파일 생성: src/main/resources/com/spring5213/mypro00/mapper/MyMemberMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.spring5213.mypro00.mapper.MyMemberMapper"> <!-- 패키지명이 포함된 인터페이스 이름을 설정. -->

    <resultMap id="authorityMap" type="com.spring5213.mypro00.domain.MyAuthorityVO">
        <result property="userid" column="userid" />
        <result property="authority" column="authority" />
    </resultMap>

    <resultMap id="memberMap" type="com.spring5213.mypro00.domain.MyMemberVO">
        <result property="userid" column="userid" />
        <result property="userpw" column="userpw" />
        <result property="userName" column="userName"/>
        <result property="enabled" column="enabled" />
        <result property="mregDate" column="mregDate" />
        <result property="mmodDate" column="mmodDate" />
        <result property="mdropFlg" column="mdropFlg" />

        <collection property="authorityList" resultMap="authorityMap" /><!-- 권한 정보가 저장됨 -->
    </resultMap>

    <!-- 특정 회원 조회(스프링 시큐리티도 사용함) - OUTER JOIN을 사용할 이유가 없습니다. -->
    <select id="selectMyMember" resultMap="memberMap">
        <![CDATA[
            SELECT mem.userid, mem.userpw, mem.username, mem.enabled,
                   mem.mregdate, mem.mmoddate, mem.mdropflg, auth.authority
            FROM book_ex.tbl_mymember mem INNER JOIN book_ex.tbl_mymember_authorities auth ON mem.userid = auth.userid
            WHERE mem.userid = #{userid}
        ]]>
    </select>

    <!-- 회원 등록: 회원 등록 시 회원 권한 추가도 같이 수행 -->
    <insert id="insertMyMember">
        INSERT INTO book_ex.tbl_mymember
        VALUES (#{userid}, #{userpw}, #{userName}, DEFAULT, DEFAULT, DEFAULT, DEFAULT)
    </insert>

    <!-- 회원 권한 추가 -->
    <insert id="insertMyMemAuthority">
        INSERT INTO book_ex.tbl_mymember_authorities
        VALUES (#{userid}, #{authority})
    </insert>
</mapper>

```

☞ 매퍼 구성 테스트는, 사용자 암호에 대하여 스프링 시큐리티의 암호화 구성을 한 다음에 수행합니다.

☞ 한명의 사용자는 여러 개의 권한을 가진 수 있으므로 `tbl_members`의 고유한 `userid`의 행(1행)은 `tbl_member_authorities`의 한 개 이상의 행(N개 행)과 합쳐집니다. 따라서, 한 명의 회원정보가 담긴 `MyMemberVO` 객체 하나와 하나 이상의 `MyAuthorityVO` 객체들에 의해 조인-SELECT문의 결과가 처리되어야 합니다.

☞ MyBatis는 `<resultMap>` 태그에 1:N의 결과가 처리되도록 구성할 수 있는 하부 요소를 지원합니다.

즉, `memberMap`이라는 `id`가 설정된 `resultMap`은 `<result>`와 `<collection>`을 이용해서 바깥쪽 객체(`MyMemberVO`의 객체 한 개)와 안쪽 객체(`MyAuthorityVO`의 객체 N 개)를 처리할 수 있습니다.

→ `src/main/webapp/WEB-INF/spring/mybatis-context.xml` 파일의 `SqlSessionFactoryBean` 설정에,

생성한 MyMemberMapper.xml 파일을 다음처럼 등록합니다.

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
    <property name="mapperLocations">
        <list>
            <value>classpath:com/spring5213/mypro00/mapper/MyBoardMapper.xml</value>
            <value>classpath:com/spring5213/mypro00/mapper/MyReplyMapper.xml</value>
            <value>classpath:com/spring5213/mypro00/mapper/MyBoardAttachFileMapper.xml</value>
            <value>classpath:com/spring5213/mypro00/mapper/MyMemberMapper.xml</value> ← 추가
        </list>
    </property>
</bean>
```

## 12-2-2. 스프링 시큐리티 기본구성

### (1) 스프링 시큐리티 라이브러리 설치(pom.xml 설정)

☞ pom.xml 파일에 스프링 시큐리티 라이브러리 구성은 추가합니다.

- 프로젝트 탐색 창에서 mypro00 프로젝트의 제일 밑에 있는 pom.xml 을 코드 작성부에 오픈
- 다음의 코드를 <dependencies>와 </dependencies> 사이에 추가(실습에서는 이미 작성되어 있으므로 주석만 해제함) 후, 저장
- 라이브러리 설치가 진행 → 설치 후(이클립스 오른쪽 아래에 진행 상태가 녹색 바로 표시됨)
  - ☞ pom.xml에 작성 후, 저장 시에 Progress 뷰를 이용하면 진행사항을 쉽게 확인할 수 있습니다.
- 프로젝트 탐색 창에서 프로젝트 마우스 오른쪽 클릭 → Maven, Update Project 수행.

```
<!-- 새로 추가 : spring-security 2021.6.22 -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>5.3.10.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>5.3.10.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>5.3.10.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId><!-- 의존성에 의해 spring-security-acl 도 설치됨 -->
    <version>5.3.10.RELEASE</version>
</dependency>
```

☞ 추가된 4개의 라이브러리들의 버전이 동일해야 합니다

☞ 개발 시에, 특정한 버전의 스프링 시큐리티 라이브러리가 필요한 경우, <https://spring.io/projects/spring-security#learn> 페이지에서 필요한 스프링 시큐리티의 버전을 확인할 수 있습니다.

## (2) 스프링 시큐리티 필터 구성(web.xml 설정)

☞ 스프링 시큐리티 동작을 위해 필요한 스프링 필터들을 구성합니다. 이를 필터들에 의해 스프링 시큐리티의 전체 기능이 동작됩니다.

- (주의) 스프링 시큐리티 필터들이 기존 스프링 요소(Dispatcher 서블릿) 보다 먼저 정의되어야 하고,  
또한 UTF-8 인코딩 필터가 스프링 시큐리티 암호화 필터보다 먼저 구성되어야 합니다.

→ src/main/webapp/WEB-INF/web.xml 파일을 코드 작성부에 오픈

→ 다음의 빨간색 코드를 UTF-8 인코딩 필터 정의 밑에 추가합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <web-app version="4.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  https://java.sun.com/xml/ns/javaee/web-app_4_0.xsd"> --> ←2021년5월 이 후부터 기본 생성 XML 스키마 URL을 사용 시 오류

<web-app version="4.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee
  http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_4_0.xsd"> ← 2021년5월 이 후부터URL로 사용 가능

<!-- 스프링 UTF-8 인코딩 필터(한글처리) -->
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>
    org.springframework.web.filter.CharacterEncodingFilter
  </filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
<!--<servlet-name>appServlet</servlet-name> -->
</filter-mapping>
<!-- 아래는 아파치의 UTF-8 인코딩 필터입니다 -->
<!--
<filter>
  <filter-name>CharacterEncoding</filter-name>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>CharacterEncoding</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping> -->

<!-- Spring-Security Filter: DelegatingFilterProxy -->
<!-- 이 필터는 모든 요청을 가로채는 필터입니다
  - 스프링 시큐리티는 이 필터를 이용하여, 모든 요청을 가로채서 요청에 대한 사용자의 인증과 승인 처리를 합니다.
    o 사용자가 입력한 정보와 DB에 저장되어 있는 정보를 자동으로 비교합니다.
    o 단, App마다 달라지는 부분은 개발자가 알아서 프로그램을 작성하여 처리해야 합니다. -->

<filter><!-- 추가 : 스프링 시큐리티 필터. -->
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
```

```

</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/spring/root-context.xml
        /WEB-INF/spring/mybatis-context.xml
    </param-value>
</context-param>

<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- Processes application requests -->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>

    <!-- 파일 업로드 관련 추가 내용 -->
    <multipart-config>
        <location>C:\\\\myupload\\\\temp</location><!-- 임시 업로드 경로 -->
        <max-file-size>20971520</max-file-size><!-- 1MB * 20: 업로드가 허용되는 한 개 파일의 최대크기 -->
        <max-request-size>41943040</max-request-size><!-- 40MB: 한 번의 업로드 요청에서 허용되는 최대 파일 크기 -->
        <file-size-threshold>20971520</file-size-threshold><!-- 20MB --><!-- 메모리 제한 크기 -->
    </multipart-config><!-- 파일 업로드 관련 추가 내용 끝 -->
</servlet>

<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>

```

- ☞ 스프링 시큐리티의 사용자 패스워드에 대한 암호화 처리 전에 UTF-8 인코딩이 먼저 처리되어야 하며, 스프링 시큐리티의 관련 빈들이 컨텍스트 영역에 생성되기 전에 스프링 시큐리티 필터들이 먼저 구성되어야 하므로 위의 순서대로 구성요소를 정의해야 합니다.

### (3) 승인/인증을 위해 데이터베이스로부터 전달된 데이터를 사용하기 위한 스프링 시큐리티 빈 클래스 생성(중요!!!)

- ☞ 이 구성이 잘되어 있어야 다음 추가 구성이 가능함.

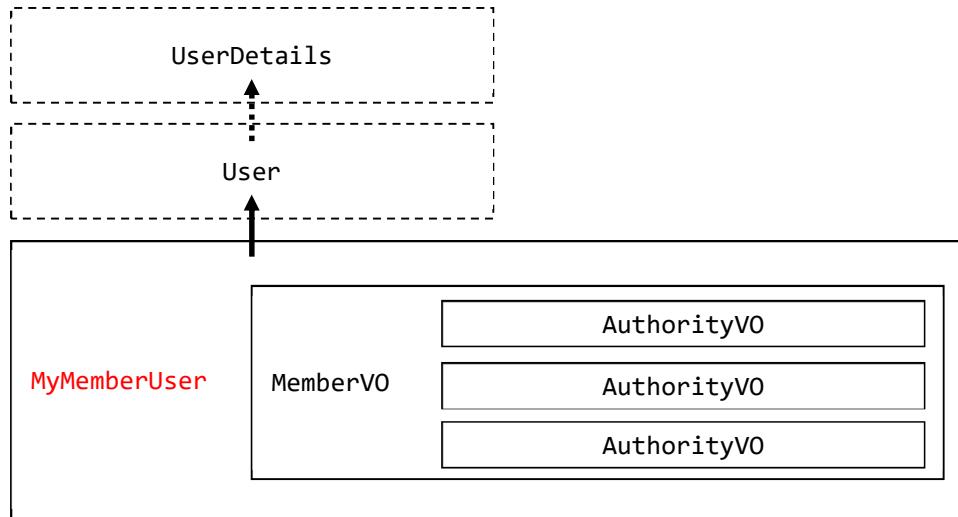
#### (3-1) MyMemberUser 클래스 생성: 스프링 시큐리티 내에서 데이터 전달 객체로 사용됨.

- ☞ 애플리케이션 실행 시에 사용자 정보가 저장된 MyMemberVO 객체는, 스프링 시큐리티가 사용할 수 있는 UserDetails 타입으로 변환되어야 합니다.

→ 이를 위해 org.springframework.security.core.userdetails.User 클래스를 상속한 MyMemberUser 클래스를 생성합니다.

☞ MyMemberUser 객체는, 생성자를 통해 부모인 User(UserDetails의 구현 클래스) 객체에 username, password, authority 값들을 설정합니다.

→ MyMemberUser 객체에 의해 값들이 설정된 UserDetails 유형의 User 객체를 스프링 시큐리티가 사용자 정보를 얻기 위해 사용합니다.



← 위의 설명을 표현한 그림

→ src/main/java/com.spring5213.mypro00.common 패키지에 security 패키지 생성

→ 생성된 security 패키지에 MyMemberUser 클래스를 생성하고, 다음 내용을 작성

```
package com.spring5213.mypro00.common.security;

import java.util.stream.Collectors;

import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import com.spring5213.mypro00.domain.MyMemberVO;

import lombok.Getter;

@Getter //myMember에 대한 getter
public class MyMemberUser extends User {

    private static final long serialVersionUID = 1L ;

    private MyMemberVO myMember ;

    //생성자를 통해 부모 클래스 객체(User 객체)에 username, password, 부여된 authority 목록을 설정.
    public MyMemberUser(MyMemberVO myMember) {

        super( myMember.getUserId(), //스프링 시큐리티의 username으로 사용됨
               myMember.getUserpw(), //스프링 시큐리티의 password로 사용됨
               myMember.getAuthorityList() //스프링 시큐리티의 authorities로 사용됨
                     .stream() //AuthorityVO 리스트를 스트림으로 변환
                     .map(auth -> new SimpleGrantedAuthority(auth.getAuthority())) //각 권한을 부여된 권한으로 변환
                     .collect(Collectors.toList()) //스트림을 리스트(부여된 권한이름만 저장)로 변환
        );

        System.out.println("MyUser 생성자에 전달된 MyMemberVO 정보:" + myMember.toString());
        System.out.println("MyUser 객체 생성을 통해 MyUser의 부모객체(User 객체) 생성됨");
        System.out.println("=====");

        this.myMember = myMember ;
    }
}
```

☞ 다음은 스프링 시큐리티의 공신 API 문서에 정의된 `UserDetails` 인터페이스에 대한 내용입니다.

- `UserDetails` 인터페이스는 구현클래스를 통해 사용자의 정보와 권한 정보 등을 담는 타입으로 사용됩니다.

`UserDetails` 인터페이스는 `getAuthorities()`, `getPassword()`, `getUserName()` 등의 여러 추상 메서드를 가지고 있어서, 개발 전에 이를 직접 구현할 것인지, 스프링 시큐리티에서 제공하는 API 구현 클래스를 사용할 것인지를 결정해야 하지만, 일반적으로, 여러 하위 클래스들 중에서 `userdetails`의 구현클래스인 `User` 클래스 API를 상속하는 형태로 구현클래스를 생성하여 사용하는 방법이 많이 사용됩니다.

org.springframework.security.core.userdetails

## Interface `UserDetails`

### All Superinterfaces:

`java.io.Serializable`

### All Known Subinterfaces:

`LdapUserDetails`

### All Known Implementing Classes:

`InetOrgPerson`, `LdapUserDetailsImpl`, `Person`, `User`

---

```
public interface UserDetails
extends java.io.Serializable
```

Provides core user information.

Implementations are not used directly by Spring Security for security purposes. They simply store user information which is later encapsulated into `Authentication` objects. This allows non-security related user information (such as email addresses, telephone numbers etc) to be stored in a convenient location.

Concrete implementations must take particular care to ensure the non-null contract detailed for each method is enforced. See `User` for a reference implementation (which you might like to extend or use in your code).

### See Also:

`UserDetailsService`, `UserCache`

## Method Summary

### All Methods

Modifier and Type	Method and Description
<code>java.util.Collection&lt;? extends GrantedAuthority&gt;</code>	<code>getAuthorities()</code> Returns the authorities granted to the user.
<code>java.lang.String</code>	<code>getPassword()</code> Returns the password used to authenticate the user.
<code>java.lang.String</code>	<code>getUsername()</code> Returns the username used to authenticate the user.
<code>boolean</code>	<code>isAccountNonExpired()</code> Indicates whether the user's account has expired.
<code>boolean</code>	<code>isAccountNonLocked()</code> Indicates whether the user is locked or unlocked.
<code>boolean</code>	<code>isCredentialsNonExpired()</code> Indicates whether the user's credentials (password) has expired.

boolean	<b>isEnabled()</b>
	Indicates whether the user is enabled or disabled.

## Method Detail

### getAuthorities

```
java.util.Collection<? extends GrantedAuthority> getAuthorities()
```

Returns the authorities granted to the user. Cannot return null.

**Returns:**

the authorities, sorted by natural key (never null)

### getPassword

```
java.lang.String getPassword()
```

Returns the password used to authenticate the user.

**Returns:**

the password

### getUsername

```
java.lang.String getUsername()
```

Returns the username used to authenticate the user. Cannot return null.

**Returns:**

the username (never null)

### isAccountNonExpired

```
boolean isAccountNonExpired()
```

Indicates whether the user's account has expired. An expired account cannot be authenticated.

**Returns:**

true if the user's account is valid (ie non-expired), false if no longer valid (ie expired)

### isAccountNonLocked

```
boolean isAccountNonLocked()
```

Indicates whether the user is locked or unlocked. A locked user cannot be authenticated.

**Returns:**

true if the user is not locked, false otherwise

### isCredentialsNonExpired

```
boolean isCredentialsNonExpired()
```

Indicates whether the user's credentials (password) has expired. Expired credentials prevent authentication.

**Returns:**

true if the user's credentials are valid (ie non-expired), false if no longer valid (ie expired)

### isEnabled

```
boolean isEnabled()
```

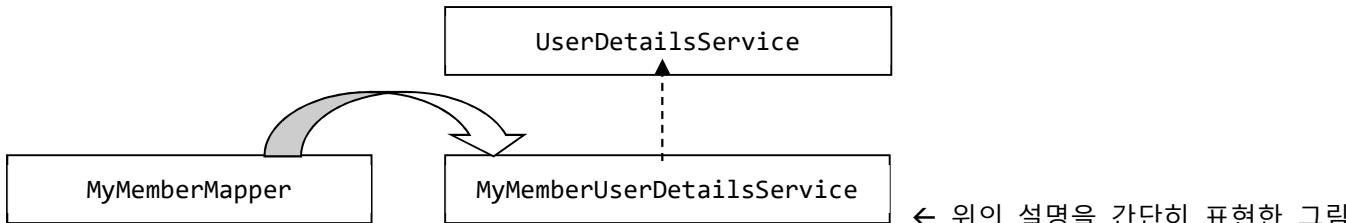
Indicates whether the user is enabled or disabled. A disabled user cannot be authenticated.

**Returns:**

true if the user is enabled, false otherwise

(3-2) MyMemberDetailsService 클래스 생성: 매퍼 인터페이스를 통해 필요한 정보를 수집하여 Provider에게 전달.

- ☞ 스프링의 UserDetailsService 인터페이스 API를 구현한 클래스(보통 커스텀 UserDetailsService라고 함)로서, 인증을 위해 필요한 데이터를 매퍼 인터페이스를 통해 가져온 후, MyMemberUser 생성자를 통해 MyMemberVO를 User 객체(UserDetails 유형)로 변환하여 Provider에게 전달함.  
→ Provider는 이를 스프링 시큐리티에게 전달하여 인증/승인 처리 시에 사용됨.
- ☞ MyMemberDetailsService 클래스에는 다음의 기능이 구현되어야 합니다.
  - Mybatis를 통해 데이터베이스로부터 사용자 정보가 저장된 MyMemberVO를 전달받음
  - MyMemberUser 생성자를 통해 User 객체(UserDetails 구현객체)에 전달된 데이터를 저장하여 UserDetails 유형으로 반환



→ src/main/java/com.spring5213.mypro00.common.security 패키지에 MyMemberDetailsService 클래스 생성 후, 내용 작성

```
package com.spring5213.mypro00.common.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

import com.spring5213.mypro00.domain.MyMemberVO;
import com.spring5213.mypro00.mapper.MyMemberMapper;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@Log4j
public class MyMemberDetailsService implements UserDetailsService {

    //setter 방식 주입이 구성되어야 security-context.xml에 정상적인 설정이 가능
    @Setter(onMethod_ = { @Autowired })
    private MyMemberMapper myMemberMapper;

    //생성자 주입방식은 security-context.xml에 설정시에 오류발생
    //private MyMemberMapper myMemberMapper;
    //public MyMemberDetailsService(MyMemberMapper myMemberMapper) {
    //    this.myMemberMapper = myMemberMapper;
    //}

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        //스프링 시큐리티가 사용하는 username을 매개변수로 사용합니다.
        log.info("Load User By UserName: " + username);

        //스프링 시큐리티가 사용하는 username을 매개변수로 사용합니다.
        MyMemberVO myMember = myMemberMapper.selectMyMember(username);
        log.warn("MyMemberMapper에 의해서 반환된 MyMemberVO: " + myMember);

        //MyMemberUser 객체 생성 -> UserDetails 유형의 User 객체로 변환되어 반환됨
        return myMember == null ? null : new MyMemberUser(myMember);
    }
}
```

- UserDetailsService 인터페이스를 구현한 클래스로서 생성된 MyMemberDetailsService 같은 클래스를, 보통 커스텀 UserDetailsService 라고하며, 이를 이용하여, 인증과 권한 승인 체크에 사용되는 데이터를 커스터마이징하여 활용할 수 있기 때문에 많이 사용됩니다.
- UserDetailsService 인터페이스는 단 하나의 추상 메서드만 정의되어 있으므로, 이를 재정의하면 커스텀 UserDetailsService 구현 클래스 생성은 완료됩니다.

다음은 스프링 시큐리티의 공식 API 문서에 정의된 UserDetailsService 인터페이스에 대한 설명입니다.

[org.springframework.security.core.userdetails](#)

## Interface UserDetailsService

### All Known Subinterfaces:

[UserDetailsManager](#)

### All Known Implementing Classes:

[CachingUserDetailsService](#), [InMemoryUserDetailsManager](#), [JdbcDaoImpl](#), [JdbcUserDetailsService](#), [LdapUserDetailsServiceManager](#), [LdapUserDetailsService](#)

---

public interface **UserDetailsService**

Core interface which loads user-specific data.

It is used throughout the framework as a user DAO and is the strategy used by the [DaoAuthenticationProvider](#).

The interface requires only one read-only method, which simplifies support for new data-access strategies.

### See Also:

[DaoAuthenticationProvider](#), [UserDetails](#)

## Method Summary

### All Methods

Modifier and Type	Method and Description
<a href="#">UserDetails</a>	<b>loadUserByUsername</b> (java.lang.String username) Locates the user based on the username.

## Method Detail

### loadUserByUsername

```
UserDetails loadUserByUsername(java.lang.String username)
throws UsernameNotFoundException
```

Locates the user based on the username. In the actual implementation, the search may possibly be case sensitive, or case insensitive depending on how the implementation instance is configured. In this case, the [UserDetails](#) object that comes back may have a username that is of a different case than what was actually requested..

#### Parameters:

username - the username identifying the user whose data is required.

#### Returns:

a fully populated user record (never null)

#### Throws:

[UsernameNotFoundException](#) - if the user could not be found or the user has no GrantedAuthority

#### (4) 사용자의 "/" URL 요청을 처리하기 위한 컨트롤러 구성

- Project Explorer 창에서 src/main/java/com.spring5213.mypro00 패키지에 있는 HomeController.java 를 복사
  - src/main/java/com.spring5213.mypro00.controller 패키지에 붙여넣기
- 원래 위치에 있던 src/main/java/com.spring5213.mypro00 패키지에 있는 HomeController.java 파일을 삭제합니다.

#### (5) 사용자 암호의 스프링 시큐리티 암호화 구성 ← 이 섹션은 내용을 확인만 합니다. 따로 실습할 내용은 없습니다.

☞ **스프링 시큐리티 5 부터는 데이터베이스를 UserDetailsService 를 통해 사용할 때, PasswordEncoder 인터페이스의 구현 클래스를 기본적으로 지정해야만 합니다.** 따라서, 데이터베이스에 저장되는 사용자의 암호는 스프링 시큐리티의 암호화 기능에 의해 암호화되어 저장됩니다. 스프링 시큐리티 4 버전까지는 PasswordEncoder 를 이용하고 싶지 않을 때, NoOpPasswordEncoder 를 이용해서 처리할 수 있었지만, 5 버전부터는 Deprecated 되어, 사용할 수 없습니다.

☞ 스프링 시큐리티의 PasswordEncoder 인터페이스는 여러 종류의 구현 클래스가 존재합니다.

다음은 스프링 프레임워크 API 문서에 명시된 PasswordEncoder 인터페이스의 구현 클래스들입니다.

org.springframework.security.crypto.password

#### Interface PasswordEncoder

##### All Known Implementing Classes:

AbstractPasswordEncoder, Argon2PasswordEncoder, BCryptPasswordEncoder, DelegatingPasswordEncoder, LdapShaPasswordEncoder, Md4PasswordEncoder, MessageDigestPasswordEncoder, NoOpPasswordEncoder, Pbkdf2PasswordEncoder, SCryptPasswordEncoder, StandardPasswordEncoder

public interface PasswordEncoder

Service interface for encoding passwords. The preferred implementation is BCryptPasswordEncoder.

☞ 실습에서는 스프링 시큐리티에서 제공하는 BCryptPasswordEncoder API 클래스를 이용하여 BCrypt 방식으로 사용자의 암호가 암호화 되도록 구성합니다. BCryptPasswordEncoder 클래스가 bCryptPasswordEncoder 이름의 빈으로 생성되어, <security:password-encoder> 태그에 ref 속성으로 주입되도록 설정합니다.

☞ BCrypt 는, 패스워드를 저장하는 용도로 설계된 해시함수로 특정 문자열을 암호화하고, 체크하는 쪽에서는 암호화 된 패스워드가 가능한 패스워드인지만 확인하고 다시 원문으로 되돌리지는 못합니다.

[참고] <http://spring.io/blog/2017/11/01/spring-security-5.0.0-rc1-released#password-storage-format> 문서를 참조하면, 스프링 시큐리티 5 버전에서 패스워드 앞에 '{noop}' 문자열을 추가하여 인코딩 처리없이 사용하거나 또는 포맷팅 처리를 지정해서 패스워드 인코딩 방식을 지정할 수 있습니다. 단 데이터베이스를 이용하는 경우에는 이 방법은 사용할 수 없습니다.

☞ 패드워드 암호화를 사용하고 싶지 않은 경우에는, 암호화가 없는 PasswordEncoder 를 직접 구현해서 사용합니다. 다음의 스프링 시큐리티 API 문서의 PasswordEncoder 인터페이스에 정의된 메소드의 설명을 참고하여 필요한 메소드가 재정의된 구현 클래스를 생성합니다.

☞ 다음은 스프링 시큐리티의 공식 API 문서에 정의된 PasswordEncoder 인터페이스의 메서드에 대한 설명입니다.

#### Method Summary

Modifier and Type

Method and Description

java.lang.String	<b>encode</b> (java.lang.CharSequence rawPassword)
	Encode the raw password.
boolean	<b>matches</b> (java.lang.CharSequence rawPassword, java.lang.String encodedPassword)
	Verify the encoded password obtained from storage matches the submitted raw password after it too is encoded.
default boolean	<b>upgradeEncoding</b> (java.lang.String encodedPassword)
	Returns true if the encoded password should be encoded again for better security, else false.

## Method Detail

### encode

```
java.lang.String encode(java.lang.CharSequence rawPassword)
```

Encode the raw password. Generally, a good encoding algorithm applies a SHA-1 or greater hash combined with an 8-byte or greater randomly generated salt.

### matches

```
boolean matches(java.lang.CharSequence rawPassword,
                java.lang.String encodedPassword)
```

Verify the encoded password obtained from storage matches the submitted raw password after it too is encoded. Returns true if the passwords match, false if they do not. The stored password itself is never decoded.

#### Parameters:

rawPassword - the raw password to encode and match

encodedPassword - the encoded password from storage to compare with

#### Returns:

true if the raw password, after encoding, matches the encoded password from storage

### upgradeEncoding

```
default boolean upgradeEncoding(java.lang.String encodedPassword)
```

Returns true if the encoded password should be encoded again for better security, else false. The default implementation always returns false.

#### Parameters:

encodedPassword - the encoded password to check

#### Returns:

true if the encoded password should be encoded again for better security, else false.

☞ 위의 메소드들 중, encode 와 matches 메소드만 재정의하여 구현 클래스를 생성합니다.

◆ 암호화가 없는 PasswordEncoder 구현 클래스 예제 - 문서 실습에서는 사용하지 않습니다.

```
package com.spring5213.mypro00.common.security;

import org.springframework.security.crypto.password.PasswordEncoder;

public class MyNoPwEncryptPasswordEncoder implements PasswordEncoder {

    //스프링 시큐리티에서 패스워드 암호화 기능을 사용하길 원하지 않을 경우
    //PasswordEncoder 를 구현한 클래스를 생성 후,
    //encode 메소드를 다음처럼 재정의하여 이를 사용하도록 security-context.xml에 구성

    @Override
    public String encode(java.lang.CharSequence rawPassword) {
        System.out.println("인코딩 처리 전 :" + rawPassword);
        return rawPassword.toString();
    }

    @Override
    public boolean matches(java.lang.CharSequence rawPassword, java.lang.String encodedPassword) {
```

```

        System.out.println("matches: " + rawPassword + ":" + encodedPassword);
        return rawPassword.toString().equals(encodedPassword);
    }
}

```

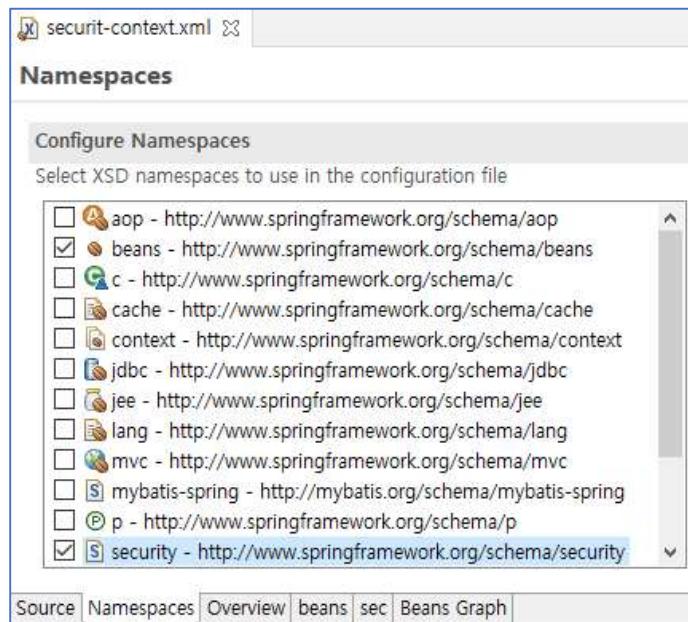
☞ 이 클래스를 빈으로 생성하여, <security:password-encoder> 태그의 ref 속성에 BCryptPasswordEncoder 클래스의 빈 대신 사용하도록 설정합니다.

## (6) 스프링 시큐리티 빈 설정 XML 컨텍스트 설정파일 생성 및 스프링 시큐리티 기본 설정의 기능 활성화 (매우 중요!!!)

- ☞ 스프링 시큐리티가 제공하는 기본값만 사용함.
- ☞ 스프링 시큐리티는 security-context.xml 파일을 생성하여 ROOT-CONTEXT 및 SERVLET-CONTEXT 와는 별도로 구성하여 사용하는 것을 권장합니다.

### (6-1) security-context.xml 스프링 빈 구성 설정 파일 생성

- Project Explorer 뷰에서 프로젝트의 src/main/webapp/WEB-INF/spring/ 폴더를 마우스 오른쪽 버튼 클릭
- 표시된 리스트에서 [New] → [other] 클릭
- Select a wizard 창에서 [Spring] 폴더를 더블클릭(확장 됨)
- 'Spring Bean Configuration File' 클릭(선택) → [Next] 클릭
- 폴더 위치를 확인 후, File name 입력란에 security-context.xml 을 입력 → [Finish] 클릭
- 코드 작성 뷰에 security-context.xml 파일이 오픈됨
- security-context.xml 코드 작성 뷰 하단의 Namespaces 탭을 클릭
- beans, security 를 선택 → 저장!!!! → 하단의 Source 탭을 클릭합니다.



→ 코드 작성뷰에 오픈된 security-context.xml 파일에, 지금까지 구성한 요소들을 사용할 수 있도록 다음의 빨간색 코드를 추가합니다.

```

<?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:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/security"

```

```

http://www.springframework.org/schema/security/spring-security-5.3.xsd
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- 사용자 password 암호화 처리 빈: BCrypt 방식 암호화 -->
<bean id="bCryptPasswordEncoder"
      class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
<!-- 사용자 password 암호화 처리 빈: 암호화 하지 않음. -->
<!--<bean id="myNoPwEncryptPasswordEncoder"
      class="com.spring5213.mypro00.common.security.MyNoPwEncryptPasswordEncoder" /> -->

<bean id="myMemberDetailsService"
      class="com.spring5213.mypro00.common.security.MyMemberDetailsService"/>

<security:http>
    <!-- csrf 기능 활성화(디폴트): <security:csrf/>, 비활성화: <security:csrf disabled="true" /> -->

    <!-- 요청 URL에 대한 권한 설정 -->
    <!-- pattern 속성에는 contextPath를 포함하지 않습니다. auto-config="true" -->
    <!-- access에 설정하는 권한을 ROLE을 통해서 설정하는 경우, 권한이름은 ROLE_로 시작해야 함 -->

    <!-- <security:intercept-url pattern="/**" access="permitAll" /> -->
    <security:intercept-url pattern="/" access="permitAll" />
    <security:intercept-url pattern="/myboard/list" access="permitAll" />
    <security:intercept-url pattern="/myboard/modify" access="hasRole('ROLE_MEMBER')" />
    <security:intercept-url pattern="/myboard/register" access="hasAnyRole('ROLE_MEMBER', 'ROLE_USER')" />
    <security:intercept-url pattern="/myboard/detail" access="hasAuthority('ROLE_ADMIN')" />

    <!-- 로그인 된 상태에서는 로그인이나 회원가입 화면에 접근 못하도록 함 -->
    <!-- <security:intercept-url pattern="/myLoginPage" access="isAnonymous()" /> -->
    <!-- <security:intercept-url pattern="/myMember/register" access="isAnonymous()" /> -->
    <!-- 관리자페이지는 관리자만 접근 허용 -->
    <!-- <security:intercept-url pattern="/admin/**" access="hasRole('ROLE_ADMIN')" /> -->
    <!-- 정적 리소스는 모두 접근 허용 -->
    <!-- <security:intercept-url pattern="/resources/**" access="permitAll" /> -->

    <security:form-login/>

    <security:logout/>

</security:http>

<security:authentication-manager>
    <security:authentication-provider user-service-ref="myMemberDetailsService">
        <security:password-encoder ref="bCryptPasswordEncoder" />
        <!-- <security:password-encoder ref="myNoPwEncryptPasswordEncoder" /> -->
    </security:authentication-provider>
</security:authentication-manager>

</beans>

```

- ☞ 스프링 시큐리티 빈 구성 파일은 크게 `<security:http>` 요소 부분과 `<security:authentication-manager>` 요소로 구성됩니다.
- ☞ 스프링 시큐리티 제어 구성(로그인 설정, 로그아웃 설정, 사용자 권한 설정 등)은 `<security:http>` 요소에 설정합니다.
- ☞ 사용자 정보를 가져와서 인증하는 구성은 `<security:authentication-manager>` 요소에 설정합니다.
- ☞ `<security:intercept-url>`를 이용하여, 특정한 URI에 접근할 때 인터셉터를 통해 접근이 제한되도록 설정합니다.
- ☞ `<security:intercept-url>`에는 pattern 및 access 속성이 지정되어야만 합니다.
- ☞ pattern 속성에는 "URI의 패턴"을 설정합니다. 이 때 컨텍스트 이름은 명시하면 됩니다.

☞ `access` 속성에는 허용되는 권한을 설정합니다. `access` 속성에는 "Spring Security 표현식" 또는 "권한명을 의미하는 문자열"을 사용할 수 있습니다.

☞ `hasRole()`, `hasAnyRole`, `permitAll`은 스프링 시큐리티에서 제공하는 표현식입니다.  
표현식에 대해서는 뒷 내용에서 보다 자세히 설명합니다.

☞ 사용자 정보를 가져오는 설정은 `<security:authentication-provider>` 요소의 `user-service-ref` 속성에 `UserDetailsService` 구현 클래스의 빈을 설정합니다.

☞ 사용자의 `password`를 암호화 하는 경우, `<security:password-encoder>` 요소의 `ref` 속성에 암호화를 수행하는 `PasswordEncoder`의 구현 클래스를 설정합니다.

☞ 로그인을 스프링 시큐리티로 처리하려는 경우, `<security:form-login/>` 요소를 명시합니다.

☞ 로그아웃을 스프링 시큐리티로 처리하려는 경우, `<security:logout/>` 요소를 명시합니다.

(6-2) `web.xml` 파일에 `security-context.xml` 스프링 빈 구성 설정 파일 등록

→ `src/main/webapp/WEB-INF/web.xml` 파일에, `security-context.xml` 파일이 사용될 수 있도록 설정합니다.

...(생략)...

```
<!-- Spring-Security Filter 설정 -->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

```
<!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/spring/root-context.xml
        /WEB-INF/spring/mybatis-context.xml
        /WEB-INF/spring/security-context.xml      ← 추가
    </param-value>
</context-param>
```

...(생략)...

(6-3) `servlet-context.xml` 파일에 `com.spring5213.mypro00.common.security` 패키지 등록

→ `src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml` 파일에 `com.spring5213.mypro00.common.security` 패키지의 클래스가 자동으로 빈으로 구성되도록 설정합니다.

...(생략)...

```
<context:component-scan base-package="com.spring5213.mypro00.service" />
<context:component-scan base-package="com.spring5213.mypro00.controller" />
<context:component-scan base-package="com.spring5213.mypro00.common.fileupload" />
```

```

<context:component-scan base-package="com.spring5213.mypro00.common.filedownload" />
<context:component-scan base-package="com.spring5213.mypro00.common.task" />
<context:component-scan base-package="com.spring5213.mypro00.common.security" /> ← 추가
...(생략)...

```

## (7) 스프링 시큐리티 어노테이션 사용 활성화 설정(servlet-context.xml에 설정해야 함)(중요!!)

- ☞ XML이나 Java 설정을 이용하는 대신 어노테이션을 이용해서 스프링 시큐리티에 필요한 설정을 추가할 수 있습니다.
  - ☞ 사용되는 어노테이션은 주로 @Secured 와 @PreAuthorize, @PostAuthorize 입니다.
  - ☞ @Secured는 스프링 시큐리티 초기부터 사용되었고, () 안에 'ROLE\_ADMIN'과 같은 문자열 혹은 문자열 배열만을 이용합니다.
  - ☞ @PreAuthorize와 @PostAuthorize는 스프링 시큐리티 3 버전부터 지원되었으며, ()안에 스프링 시큐리티 표현식을 사용할 수 있으므로 최근에는 더 많이 사용됩니다.
  - ☞ 스프링 시큐리티 어노테이션을 사용하려면, 스프링 Dispatcher Servlet의 컨텍스트 설정파일(servlet-context.xml 파일)에 어노테이션 사용을 활성해주는 설정을 추가해 주어야 합니다.
- src/main/webapp/spring/appServlet/servlet-context.xml 파일을 코드 작성 뷰에 오픈
- 코드 작성뷰 하단의 Namespaces 탭 클릭 → security 네임스페이스 추가 → 저장!!! → 하단의 Source 탭 클릭
- 다음의 스프링 시큐리티 어노테이션 사용 활성화 설정을 </beans:beans> 종료 태그 위에 추가
- ```

<!-- 어노테이션을 이용한 스프링 시큐리티 활성화, 아래 코드 추가 -->
<security:global-method-security pre-post-annotations="enabled" secured-annotations="enabled" />

</beans:beans>

```

- ☞ @Secured, @PreAuthorize와 @PostAuthorize 어노테이션을 모두 사용할 수 있도록 설정됩니다.

## (8) 구성 사항 확인을 위한 서버 기동

- 템켓 서버를 기동합니다. → 오류 없이 기동되어야 하는데, 다음의 아래의 오류가 발생될 수 있습니다.

```

6월 08, 2021 8:43:58 오전 org.apache.catalina.core.ContainerBase startInternal
심각: 자식 컨테이너를 시작 중 실패했습니다.
java.util.concurrent.ExecutionException: org.apache.catalina.LifecycleException: 구성요소 [StandardEngine[Catalina]. StandardHost[localhost]. StandardContext[/ex00]]을(를) 시작하지 못했습니다.
at java.util.concurrent.FutureTask.report(FutureTask.java:122)
at java.util.concurrent.FutureTask.get(FutureTask.java:192)
at org.apache.catalina.core.ContainerBase.startInternal(ContainerBase.java:916)
at org.apache.catalina.core.StandardHost.startInternal(StandardHost.java:843)
at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:183)

...(생략)...

at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:342)
at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:473)
Caused by: org.apache.catalina.LifecycleException: 구성요소 [StandardEngine[Catalina]. StandardHost[localhost]. StandardContext[/ex00]]을(를) 시작하지 못했습니다.
at org.apache.catalina.util.LifecycleBase.handleSubClassException(LifecycleBase.java:440)
at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:198)

...(생략)...

at java.util.concurrent.AbstractExecutorService.submit(AbstractExecutorService.java:134)

```

```

at org.apache.catalina.core.ContainerBase.startInternal(ContainerBase.java:909)
... 21 more
Caused by: java.lang.IllegalArgumentException: 이름이 [spring_web]인, 둘 이상의 fragment들이 발견되었습니다. 이는 상대적 순서배열에서 불허됩니다. 상세
정보는 서블릿 스펙 8.2.2 2c 장을 참조하십시오. 절대적 순서배열을 사용하는 것을 고려해 보십시오.
at org.apache.tomcat.util.descriptor.web.WebXml.orderWebFragments(WebXml.java:2260)
at org.apache.tomcat.util.descriptor.web.WebXml.orderWebFragments(WebXml.java:2218)
at org.apache.catalina.startup.ContextConfig.webConfig(ContextConfig.java:1294)
at org.apache.catalina.startup.ContextConfig.configureStart(ContextConfig.java:986)
at org.apache.catalina.startup.ContextConfig.lifecycleEvent(ContextConfig.java:303)
at org.apache.catalina.util.LifecycleBase.fireLifecycleEvent(LifecycleBase.java:123)
at org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5077)
at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:183)
... 27 more
6월 08, 2021 8:43:58 오전 org.apache.catalina.core.ContainerBase startInternal
심각: 자식 컨테이너를 시작 중 실패했습니다.
java.util.concurrent.ExecutionException: org.apache.catalina.LifecycleException: 자식 컨테이너를 시작 중 실패했습니다.
at java.util.concurrent.FutureTask.report(FutureTask.java:122)
...(생략)...

```

## [조치]

→ src/main/webapp/WEB-INF/web.xml 파일에, 스키마 정의 밑에 <absolute-ordering> 요소 추가, 저장 → 톰캣 서버 기동

```

<?xml version="1.0" encoding="UTF-8"?>

<!-- 4.0: 오라클 사의 정의 정의 파일 사용 --&gt;
&lt;web-app version="4.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee
  http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_4_0.xsd"&gt;

&lt;absolute-ordering/&gt; ← 추가, 문서에서는 오류가 발생되지 않아 추가하지 않음

<!-- 스프링 UTF-8 인코딩 필터(한글처리) --&gt;
&lt;filter&gt;
  &lt;filter-name&gt;encodingFilter&lt;/filter-name&gt;
  &lt;filter-class&gt;
    org.springframework.web.filter.CharacterEncodingFilter
  &lt;/filter-class&gt;
  &lt;init-param&gt;
    &lt;param-name&gt;encoding&lt;/param-name&gt;
    &lt;param-value&gt;UTF-8&lt;/param-value&gt;
  &lt;/init-param&gt;
  &lt;init-param&gt;
    &lt;param-name&gt;forceEncoding&lt;/param-name&gt;
    &lt;param-value&gt;true&lt;/param-value&gt;
  &lt;/init-param&gt;
&lt;/filter&gt;
&lt;filter-mapping&gt;
  &lt;filter-name&gt;encodingFilter&lt;/filter-name&gt;
  &lt;url-pattern&gt;/*&lt;/url-pattern&gt;
  &lt;servlet-name&gt;appServlet&lt;/servlet-name&gt;
&lt;/filter-mapping&gt;

<!-- 스프링 시큐리티 필터 --&gt;
&lt;filter&gt;
  &lt;filter-name&gt;springSecurityFilterChain&lt;/filter-name&gt;
  &lt;filter-class&gt;org.springframework.web.filter.DelegatingFilterProxy&lt;/filter-class&gt;
&lt;/filter&gt;
&lt;filter-mapping&gt;
  &lt;filter-name&gt;springSecurityFilterChain&lt;/filter-name&gt;
  &lt;url-pattern&gt;/*&lt;/url-pattern&gt;
&lt;/filter-mapping&gt;

...(생략)...
</pre>

```

→ 톰캣 서버 기동 → 정상 기동 확인 및 로그 확인 → 톰캣 서버를 중지합니다

☞ 다음은 톰캣 서버가 정상 가동 시에 콘솔에 표시된 로그입니다(스프링 시큐리티 5.3.8 버전에서 암호화 사용 시).

```
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 서버 버전 이름: Apache Tomcat/9.0.45
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: Server 빌드 시간: Mar 30 2021 10:29:04 UTC
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: Server 버전 번호: 9.0.45.0
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 운영체제 이름: Windows 10
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 운영체제 버전: 10.0
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 아키텍처: amd64
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 자바 홈: C:\myJavaDev\jdk1.8.0_291\jre
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: JVM 버전: 1.8.0_291-b10
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: JVM 벤더: Oracle Corporation
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: CATALINA_BASE: C:\myJavaDev\mySpring5Workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: CATALINA_HOME: C:\myJavaDev\tomcat9
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 명령 행 아규먼트: -Dcatalina.base=C:\myJavaDev\mySpring5Workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 명령 행 아규먼트: -Dcatalina.home=C:\myJavaDev\tomcat9
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 명령 행 아규먼트: -Dwtp.deploy=C:\myJavaDev\mySpring5Workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 명령 행 아규먼트: -Djava.endorsed.dirs=C:\myJavaDev\tomcat9\endorsed
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.VersionLoggerListener log
정보: 명령 행 아규먼트: -Dfile.encoding=UTF-8
6월 08, 2021 10:55:31 오전 org.apache.catalina.core.AprLifecycleListener lifecycleEvent
정보: APR 버전 [1.7.0]을(를) 사용한, APR 기반 Apache Tomcat Native 라이브러리 [1.2.27]을(를) 로드했습니다.
6월 08, 2021 10:55:31 오전 org.apache.catalina.core.AprLifecycleListener lifecycleEvent
정보: APR 용량정보들: IPv6 [true], sendfile [true], accept filters [false], random [true].
6월 08, 2021 10:55:31 오전 org.apache.catalina.core.AprLifecycleListener lifecycleEvent
정보: APR/OpenSSL 설정: useAprConnector [false], useOpenSSL [true]
6월 08, 2021 10:55:31 오전 org.apache.catalina.core.AprLifecycleListener initializeSSL
정보: OpenSSL이 성공적으로 초기화되었습니다: [OpenSSL 1.1.1k 25 Mar 2021]
6월 08, 2021 10:55:31 오전 org.apache.coyote.AbstractProtocol init
정보: 프로토콜 핸들러 ["http-nio-8080"]을(를) 초기화합니다.
6월 08, 2021 10:55:31 오전 org.apache.catalina.startup.Catalina load
정보: [1190] 밀리초 내에 서버가 초기화되었습니다.
6월 08, 2021 10:55:31 오전 org.apache.catalina.core.StandardService startInternal
정보: 서비스 [Catalina]을(를) 시작합니다.
6월 08, 2021 10:55:31 오전 org.apache.catalina.core.StandardEngine startInternal
정보: 서버 앤진을 시작합니다: [Apache Tomcat/9.0.45]
6월 08, 2021 10:55:33 오전 org.apache.jasper.servlet.TldScanner scanJars
정보: 적어도 하나의 JAR가 TLD들을 찾기 위해 스캔되었으나 아무 것도 찾지 못했습니다. 스캔했으나 TLD가 없는 JAR들의 전체 목록을 보시려면, 로그 레벨을 디버그 레벨로 설정하십시오. 스캔 과정에서 불필요한 JAR들을 건너뛰면, 시스템 시작 시간과 JSP 컴파일 시간을 단축시킬 수 있습니다.
6월 08, 2021 10:55:33 오전 org.apache.catalina.core.ApplicationContext log
정보: Initializing Spring root WebApplicationContext
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext: initialization started
INFO : org.springframework.security.core.SpringSecurityCoreVersion - You are running with Spring Security Core 5.3.8.RELEASE
INFO : org.springframework.security.config.SecurityNamespaceHandler - Spring Security 'config' module version is 5.3.8.RELEASE
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute 'permitAll' for /
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute 'permitAll' for /myboard/list
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute 'hasRole('ROLE_MEMBER')' for /myboard/modify
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute 'hasAnyRole('ROLE_MEMBER', 'ROLE_USER')' for /myboard/register
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute 'hasRole('ROLE_ADMIN')' for /myboard/detail
INFO : org.springframework.security.config.http.AuthenticationConfigBuilder - No login page configured. The default internal one will be used. Use the 'login-page' attribute to set the URL of the login page.
INFO : org.springframework.security.config.http.HttpSecurityBeanDefinitionParser - Checking sorted filter chain: [Root bean: class [org.springframework.security.web.context.SecurityContextPersistenceFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 200, Root bean: class [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 400, Root bean: class [org.springframework.security.web.header.HeaderWriterFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 500, Root bean: class [org.springframework.security.web.csrf.CsrfFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 700, <org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter#0>, order = 1400, Root bean: class [org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 1600, Root bean: class [org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 2100, Root bean: class [org.springframework.security.web.savedrequest.RequestCacheAwareFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 2100, Root bean: class [null]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false;
```

```

factoryBeanName=org.springframework.security.config.http.HttpConfigurationBuilder$SecurityContextHolderAwareRequestFilterBeanFactory#0;
factoryMethodName=getBean; initMethodName=null; destroyMethodName=null, order = 2200, Root bean: class
[org.springframework.security.web.authentication.AnonymousAuthenticationFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0;
dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null;
destroyMethodName=null, order = 2500, Root bean: class [org.springframework.security.web.session.SessionManagementFilter]; scope=; abstract=false;
lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null;
initMethodName=null; destroyMethodName=null, order = 2700, Root bean: class [org.springframework.security.web.access.ExceptionTranslationFilter];
scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null;
factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 2800,
<org.springframework.security.web.access.intercept.FilterSecurityInterceptor#0>, order = 2900]
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
INFO : org.springframework.security.web.DefaultSecurityFilterChain - Creating filter chain: any request,
[org.springframework.security.web.context.SecurityContextPersistenceFilter@2f4919b0,
org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@a8a8b75,
org.springframework.security.web.header.HeaderWriterFilter@75b21c3b, org.springframework.security.web.csrf.CsrfFilter@72be135f,
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@155d1021,
org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter@4bd2f0dc,
org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter@2e647e59,
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@2c42b421,
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@51e37590,
org.springframework.security.web.authentication.AnonymousAuthenticationFilter@deb3b60,
org.springframework.security.web.session.SessionManagementFilter@701a32,
org.springframework.security.web.access.ExceptionTranslationFilter@39aa45a1,
org.springframework.security.web.access.intercept.FilterSecurityInterceptor@73aff8f1]
INFO : org.springframework.security.config.http.DefaultFilterChainValidator - Checking whether login URL '/login' is accessible with your
configuration
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext initialized in 4833 ms
6월 08, 2021 10:55:38 오전 org.apache.catalina.core.ApplicationContext log
정보: Initializing Spring DispatcherServlet 'appServlet'
INFO : org.springframework.web.servlet.DispatcherServlet - Initializing Servlet 'appServlet'
INFO : org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler - Initializing ExecutorService 'clearUploadRepoJobScheduler'
INFO : org.springframework.context.support.PostProcessorRegistrationDelegate$BeanPostProcessorChecker - Bean 'clearUploadRepoJobScheduler' of type
[org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler] is not eligible for getting processed by all BeanPostProcessors (for example:
not eligible for auto-proxying)
INFO : org.springframework.web.servlet.DispatcherServlet - Completed initialization in 3659 ms
6월 08, 2021 10:55:42 오전 org.apache.coyote.AbstractProtocol start
정보: 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작합니다.
6월 08, 2021 10:55:42 오전 org.apache.catalina.startup.Catalina start
정보: 서버가 [10169] 밀리초 내에 시작되었습니다.

```

☞ 다음은 톰캣 서버가 정상 가동 시에 콘솔에 표시된 로그입니다(스프링 시큐리티 5.3.10 버전에서 암호화 사용 시, spring 로그만 포함).

```

INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext: initialization started
INFO : org.springframework.security.core.SpringSecurityCoreVersion - You are running with Spring Security Core 5.3.10.RELEASE
INFO : org.springframework.security.config.SecurityNamespaceHandler - Spring Security 'config' module version is 5.3.10.RELEASE
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute
'permitAll' for /
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute
'permitAll' for /myboard/list
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute
'hasRole('ROLE_MEMBER')' for /myboard/modify
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute
'hasAnyRole('ROLE_MEMBER','ROLE_USER' )' for /myboard/register
INFO : org.springframework.security.config.http.FilterInvocationSecurityMetadataSourceParser - Creating access control expression attribute
'hasAuthority('ROLE_ADMIN')' for /myboard/detail
INFO : org.springframework.security.config.http.AuthenticationConfigBuilder - No login page configured. The default internal one will be used. Use
the 'login-page' attribute to set the URL of the login page.
INFO : org.springframework.security.config.http.HttpSecurityBeanDefinitionParser - Checking sorted filter chain: [Root bean: class
[org.springframework.security.web.context.SecurityContextPersistenceFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0;
dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null;
destroyMethodName=null, order = 200, Root bean: class [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter];
scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null;
factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 400, Root bean: class
[org.springframework.security.web.header.HeaderWriterFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0;
autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 500,
Root bean: class [org.springframework.security.web.csrf.CsrfFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0;
autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 700,
Root bean: class [org.springframework.security.web.authentication.logout.LogoutFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0;
dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null;
destroyMethodName=null, order = 800, <org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter#0>, order = 1400, Root
bean: class [org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter]; scope=; abstract=false; lazyInit=null;
autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null;
destroyMethodName=null, order = 1600, Root bean: class [org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter];
scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null;
factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 1700, Root bean: class
[org.springframework.security.web.savedrequest.RequestCacheAwareFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0;
autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 2100,
Root bean: class [null]; scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false;
factoryBeanName=org.springframework.security.config.http.HttpConfigurationBuilder$SecurityContextHolderAwareRequestFilterBeanFactory#0;
factoryMethodName=getBean; initMethodName=null; destroyMethodName=null, order = 2200, Root bean: class
[org.springframework.security.web.authentication.AnonymousAuthenticationFilter]; scope=; abstract=false; lazyInit=null; autowireMode=0;
dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null; initMethodName=null;
destroyMethodName=null, order = 2500, Root bean: class [org.springframework.security.web.session.SessionManagementFilter]; scope=; abstract=false;
lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null; factoryMethodName=null;
initMethodName=null; destroyMethodName=null, order = 2700, Root bean: class [org.springframework.security.web.access.ExceptionTranslationFilter];
scope=; abstract=false; lazyInit=null; autowireMode=0; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=null;
factoryMethodName=null; initMethodName=null; destroyMethodName=null, order = 2800,
<org.springframework.security.web.access.intercept.FilterSecurityInterceptor#0>, order = 2900]
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.

```

```

INFO : org.springframework.security.web.DefaultSecurityFilterChain - Creating filter chain: any request,
[org.springframework.security.web.context.SecurityContextPersistenceFilter@2408ca4c,
org.springframework.security.web.context.async.WebAsyncManagerIntegrationFilter@2f7e2481,
org.springframework.security.web.header.HeaderWriterFilter@5f745970, org.springframework.security.web.csrf.CsrfFilter@57cd77e1,
org.springframework.security.web.authentication.logout.LogoutFilter@35f59dfe,
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@6b71fded,
org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter@1633962a,
org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter@2afd8972,
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@33fefbbe,
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@5784f6b9,
org.springframework.security.web.authentication.AnonymousAuthenticationFilter@5d0ca8a5,
org.springframework.security.web.session.SessionManagementFilter@1b7c1c6f,
org.springframework.security.web.access.ExceptionTranslationFilter@5f9b37b5,
org.springframework.security.web.access.intercept.FilterSecurityInterceptor@2f5a23c1]
INFO : org.springframework.security.config.http.DefaultFilterChainValidator - Checking whether login URL '/login' is accessible with your
configuration
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext initialized in 4776 ms
6월 29, 2021 11:06:30 오전 org.apache.catalina.core.ApplicationContext log
정보: Initializing Spring DispatcherServlet 'appServlet'
INFO : org.springframework.web.servlet.DispatcherServlet - Initializing Servlet 'appServlet'
INFO : org.springframework.security.config.SecurityNamespaceHandler - Spring Security 'config' module version is 5.3.10.RELEASE
INFO : org.springframework.security.config.method.GlobalMethodSecurityBeanDefinitionParser - Expressions were enabled for method security but no
SecurityExpressionHandler was configured. All hasPermission() expressions will evaluate to false.
INFO : org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler - Initializing ExecutorService 'clearUploadRepoJobScheduler'
INFO : org.springframework.context.support.PostProcessorRegistrationDelegate$BeanPostProcessorChecker - Bean 'clearUploadRepoJobScheduler' of type
[org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler] is not eligible for getting processed by all BeanPostProcessors (for example:
not eligible for auto-proxying)
INFO : org.springframework.web.servlet.DispatcherServlet - Completed initialization in 2877 ms
6월 29, 2021 11:06:33 오전 org.apache.coyote.AbstractProtocol start
정보: 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작합니다.
6월 29, 2021 11:06:33 오전 org.apache.catalina.startup.Catalina start
정보: 서버가 [14997] 밀리초 내에 시작되었습니다.

```

[참고] 서버 기동 중에 다음의 경고 메시지가 표시될 수 있습니다

```

WARN : org.springframework.security.core.SpringSecurityCoreVersion - **** You are advised to use Spring 5.2.8.RELEASE
or later with this version. You are running: 5.2.7.RELEASE

```

→ 해당 버전의 스프링 시큐리티를 사용하기 위한 스프링 프레임워크 버전이 낮은 경우, 아래의 경고가 표시됩니다. 경고를 없애기 위해서는 스프링 프레임워크를 메시지에 표시된 버전 이상으로 업그레이드하거나, 스프링 시큐리티 라이브러리 버전을 낮추어야 합니다.

## (9) 테스트 - 구성된 요소 작동 확인

### (9-1) 매퍼 구성 테스트: 테스트 클래스 생성 및 테스트

☞ 이 실습을 수행해야만 앞으로의 실습에서 사용되는 계정과 권한들이 구성됩니다.

→ src/test/java/com.spring5213.mypro00.mapper.MyMemberMapperTests 클래스 생성 후, 다음 코드 작성

→ 3개의 테스트 메서드를 모두 주석 처리 후, 테스트 하는 하나의 메서드만 주석을 해제 하여 테스트 수행

```

package com.spring5213.mypro00.mapper;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;

import com.spring5213.mypro00.domain.MyAuthorityVO;

```

```

import com.spring5213.mypro00.domain.MyMemberVO;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration({ "file:src/main/webapp/WEB-INF/spring/mybatis-context.xml",
                      "file:src/main/webapp/WEB-INF/spring/security-context.xml",
                      "file:src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml" })
@Log4j
public class MyMemberMapperTests {

    //사용자 패스워드 암호화
    @Setter(onMethod_ = @Autowired)
    private PasswordEncoder pwencoder;

    @Setter(onMethod_ = @Autowired)
    private MyMemberMapper myMemberMapper;

    //회원 등록 테스트: 테스트(1)
    @Test
    public void testInsertMyMember() {

        MyMemberVO myMember = new MyMemberVO();

        for(int i = 0; i < 100; i++) {

            myMember.setUserpw(pwencoder.encode("pw" + i));

            if(i < 80) {
                myMember.setUserId("user" + i);
                myMember.setName("일반사용자" + i);

            } else if (i < 90) {
                myMember.setUserId("manager" + i);
                myMember.setName("운영자" + i);

            } else {
                myMember.setUserId("admin" + i);
                myMember.setName("관리자" + i);
            }

            log.info(myMember);
            myMemberMapper.insertMyMember(myMember);
        } //for-End
    }

    //회원 권한 등록 테스트: 테스트(2)
    @Test
    public void testInsertMyMemAuthority() {

        MyAuthorityVO myAuthority = new MyAuthorityVO();

        for(int i = 0; i < 100; i++) {

            if(i < 80) {
                myAuthority.setUserId("user" + i);
                myAuthority.setAuthority("ROLE_USER");

            } else if (i < 90) {
                myAuthority.setUserId("manager" + i);
                myAuthority.setAuthority("ROLE_MEMBER");

            } else {
                myAuthority.setUserId("admin" + i);
                myAuthority.setAuthority("ROLE_ADMIN");
            }
        }
    }
}

```

```

    }
    log.info(myAuthority);
    myMemberMapper.insertMyMemAuthority(myAuthority);
} //for-End

myAuthority.setUserId("admin99");
myAuthority.setAuthority("ROLE_MEMBER");
myMemberMapper.insertMyMemAuthority(myAuthority);

myAuthority.setUserId("admin99");
myAuthority.setAuthority("ROLE_USER");
myMemberMapper.insertMyMemAuthority(myAuthority);

myAuthority.setUserId("admin91");
myAuthority.setAuthority("ROLE_MEMBER");
myMemberMapper.insertMyMemAuthority(myAuthority);
}

//회원 정보 조회 테스트: 테스트(3)
@Test
public void testRead() {
    MyMemberVO myMember = myMemberMapper.selectMyMember("admin99");
    log.info(myMember);

    myMember.getAuthorityList().forEach(authorityVO -> log.info(authorityVO));
}
}

```

### [테스트 결과]

```

//회원 등록 테스트
INFO : org.springframework.security.web.DefaultSecurityFilterChain - Creating filter chain: any request,
[org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@20765ed5,
org.springframework.security.web.context.SecurityContextPersistenceFilter@58cd06cb,
org.springframework.security.web.header.HeaderWriterFilter@df5f5c0, org.springframework.security.web.csrf.CsrfFilter@61a5b4ae,
org.springframework.security.web.authentication.logout.LogoutFilter@6b5f8707,
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@44ea608c,
org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter@10567255,
org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter@3b582111,
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@64b31700,
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@5fd9b663,
org.springframework.security.web.authentication.AnonymousAuthenticationFilter@2899a8db,
org.springframework.security.web.session.SessionManagementFilter@66b72664,
org.springframework.security.web.access.ExceptionTranslationFilter@1bb9aa43,
org.springframework.security.web.access.intercept.FilterSecurityInterceptor@5a6d5a8f]
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyMemberVO(userid=user0,
userpw=$2a$10$3wCRHBjL3V.3RrA5r4ajGuUpPzq8ugS.LoYaGUISCX0.ibJLBP4qm, userName=일반사용자0, mregDate=null, mmodDate=null, mdropFlg=null,
enabled=false, authorityList=null)
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember VALUES ('user0', '$2a$10$3wCRHBjL3V.3RrA5r4ajGuUpPzq8ugS.LoYaGUISCX0.ibJLBP4qm',
'일반사용자0', DEFAULT, DEFAULT, DEFAULT, DEFAULT)
INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember VALUES ('user0', '$2a$10$3wCRHBjL3V.3RrA5r4ajGuUpPzq8ugS.LoYaGUISCX0.ibJLBP4qm',
'일반사용자0', DEFAULT, DEFAULT, DEFAULT, DEFAULT)
{executed in 85 msec}

...(생략)...

INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyMemberVO(userid=manager80,
userpw=$2a$10$dDv1QnprYpGuLph5xauGnenD05Uuhk28nWlYoB/7w.nE7NXp/JrEO, userName=운영자80, mregDate=null, mmodDate=null, mdropFlg=null, enabled=false,
authorityList=null)
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember VALUES ('manager80', '$2a$10$dDv1QnprYpGuLph5xauGnenD05Uuhk28nWlYoB/7w.nE7NXp/JrEO',
'운영자80', DEFAULT, DEFAULT, DEFAULT, DEFAULT)
INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember VALUES ('manager80', '$2a$10$dDv1QnprYpGuLph5xauGnenD05Uuhk28nWlYoB/7w.nE7NXp/JrEO',
'운영자80', DEFAULT, DEFAULT, DEFAULT, DEFAULT)
{executed in 2 msec}

...(생략)...

INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyMemberVO(userid=admin90,
userpw=$2a$10$0dYmSci10wQgHGP2FF2nWuQfx4n/GCw2btE9y9K0WnxdxreV5Md8i, userName=관리자90, mregDate=null, mmodDate=null, mdropFlg=null, enabled=false,
authorityList=null)
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember VALUES ('admin90', '$2a$10$0dYmSci10wQgHGP2FF2nWuQfx4n/GCw2btE9y9K0WnxdxreV5Md8i',
'관리자90', DEFAULT, DEFAULT, DEFAULT, DEFAULT)
INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember VALUES ('admin90', '$2a$10$0dYmSci10wQgHGP2FF2nWuQfx4n/GCw2btE9y9K0WnxdxreV5Md8i',
'관리자90', DEFAULT, DEFAULT, DEFAULT, DEFAULT)

```

```
'관리자90', DEFAULT, DEFAULT, DEFAULT)
{executed in 1 msec}

...(생략)...

//회원 권한 등록 테스트
...(생략)...
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=user0, authority=ROLE_USER)
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('user0', 'ROLE_USER')

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('user0', 'ROLE_USER')
{executed in 16 msec}
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=user1, authority=ROLE_USER)
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('user1', 'ROLE_USER')

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('user1', 'ROLE_USER')
{executed in 2 msec}

...(생략)...

INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=manager80, authority=ROLE_MEMBER)
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('manager80', 'ROLE_MEMBER')

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('manager80', 'ROLE_MEMBER')
{executed in 2 msec}

...(생략)...

INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=admin99, authority=ROLE_ADMIN)
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('admin99', 'ROLE_ADMIN')

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('admin99', 'ROLE_ADMIN')
{executed in 1 msec}
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('admin99', 'ROLE_MEMBER')

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('admin99', 'ROLE_MEMBER')
{executed in 1 msec}
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('admin99', 'ROLE_USER')

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('admin99', 'ROLE_USER')
{executed in 1 msec}
INFO : jdbc.sqlonly - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('admin91', 'ROLE_MEMBER')

INFO : jdbc.sqltiming - INSERT INTO book_ex.tbl_mymember_authorities VALUES ('admin91', 'ROLE_MEMBER')
{executed in 1 msec}

//회원 정보 조회 테스트(password 암호화: BCrypt 암호화 사용)

...(생략)...

INFO : org.springframework.security.web.DefaultSecurityFilterChain - Creating filter chain: any request,
[org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@4c432866,
org.springframework.security.web.context.SecurityContextPersistenceFilter@1c05a54d,
org.springframework.security.web.header.HeaderWriterFilter@3b65e559, org.springframework.security.web.csrf.CsrfFilter@7e8e8651,
org.springframework.security.web.authentication.logout.LogoutFilter@51bde877,
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@20765ed5,
org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter@54336c81,
org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter@12365c88,
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@5fd9b663,
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@6548bb7d,
org.springframework.security.web.authentication.AnonymousAuthenticationFilter@6105f8a3,
org.springframework.security.web.session.SessionManagementFilter@74a9c4b0,
org.springframework.security.web.access.ExceptionTranslationFilter@3be8821f,
org.springframework.security.web.access.intercept.FilterSecurityInterceptor@6650813a]
INFO : jdbc.sqlonly - SELECT mem.userid, mem.userpw, mem.username, mem.enabled, mem.mregdate, mem.mmoddate, mem.mdropflg,
auth.authority FROM book_ex.tbl_mymember mem INNER JOIN book_ex.tbl_mymember_authorities auth
ON mem.userid = auth.userid WHERE mem.userid = 'admin99'

INFO : jdbc.sqltiming - SELECT mem.userid, mem.userpw, mem.username, mem.enabled, mem.mregdate, mem.mmoddate, mem.mdropflg,
auth.authority FROM book_ex.tbl_mymember mem INNER JOIN book_ex.tbl_mymember_authorities auth
ON mem.userid = auth.userid WHERE mem.userid = 'admin99'
{executed in 120 msec}
INFO : jdbc.resultsettable -
+-----+-----+-----+-----+-----+-----+
|-----|-----|-----|-----|-----|-----|
|userid |userpw |username |enabled |mregdate |mmoddate |mdropflg |
|-----|-----|-----|-----|-----|-----|-----|
|admin99 |$2a$10$4PYEq1ZB5kavLCOY17guRupcLmD0Zxu5M1Soh7xPzRoxuGt0Xt02i |관리자99 |true |2021-03-22 19:44:02.0 |2021-03-22 19:44:02.0 |0
|ROLE_ADMIN |
|admin99 |$2a$10$4PYEq1ZB5kavLCOY17guRupcLmD0Zxu5M1Soh7xPzRoxuGt0Xt02i |관리자99 |true |2021-03-22 19:44:02.0 |2021-03-22 19:44:02.0 |0
|ROLE_MEMBER |
|admin99 |$2a$10$4PYEq1ZB5kavLCOY17guRupcLmD0Zxu5M1Soh7xPzRoxuGt0Xt02i |관리자99 |true |2021-03-22 19:44:02.0 |2021-03-22 19:44:02.0 |0
|ROLE_USER |
|-----|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----|-----|
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyMemberVO(userid=admin99,
userpw=$2a$10$4PYEq1ZB5kavLCOY17guRupcLmD0Zxu5M1Soh7xPzRoxuGt0Xt02i, userName=관리자99, mregDate=2021-03-22 19:44:02.0, mmodDate=2021-03-22
19:44:02.0, mdropFlg=0, enabled=true, authorityList=[MyAuthorityVO(userid=admin99, authority=ROLE_ADMIN), MyAuthorityVO(userid=admin99,
authority=ROLE_MEMBER), MyAuthorityVO(userid=admin99, authority=ROLE_USER)])
```

```
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=admin99, authority=ROLE_ADMIN)
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=admin99, authority=ROLE_MEMBER)
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=admin99, authority=ROLE_USER)
```

//[참고] 아래 로그는 암호화를 사용하지 않는 구성 한 후에, 테스트를 다시 수행했을 때, 회원 정보 조회 테스트(password 암호화: 암호화 사용 않함)  
//데이터베이스에 저장된 암호가 전혀 암호화 되지 않았습니다.

...(생략)...

```
INFO : jdbc.resultsettable -
```

| userid  | userpw | username | enabled | mregdate              | mmoddate              | mdropflg | authority   |
|---------|--------|----------|---------|-----------------------|-----------------------|----------|-------------|
| admin99 | pw99   | 관리자99    | true    | 2021-06-08 16:15:05.0 | 2021-06-08 16:15:05.0 | 0        | ROLE_ADMIN  |
| admin99 | pw99   | 관리자99    | true    | 2021-06-08 16:15:05.0 | 2021-06-08 16:15:05.0 | 0        | ROLE_MEMBER |
| admin99 | pw99   | 관리자99    | true    | 2021-06-08 16:15:05.0 | 2021-06-08 16:15:05.0 | 0        | ROLE_USER   |

```
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyMemberVO(userid=admin99, userpw=pw99, userName=관리자99, mregDate=2021-06-08 16:15:05.0, mmodDate=2021-06-08 16:15:05.0, mdropFlg=0, enabled=true, authorityList=[MyAuthorityVO(userid=admin99, authority=ROLE_ADMIN), MyAuthorityVO(userid=admin99, authority=ROLE_MEMBER), MyAuthorityVO(userid=admin99, authority=ROLE_USER)])
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=admin99, authority=ROLE_ADMIN)
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=admin99, authority=ROLE_MEMBER)
INFO : com.spring5213.mypro00.mapper.MyMemberMapperTests - MyAuthorityVO(userid=admin99, authority=ROLE_USER)
```

...(생략)...

☞ 회원 정보 조회 테스트 시에 JOIN 문장이 실행되어 결과가 잘 전달된 것이 확인됩니다.

☞ 문서 실습에서는 password를 Bcrypt로 암호화 시키는 환경을 사용합니다.

## (9-2) 스프링 시큐리티 기본 접근 제한 구성 확인 테스트

☞ 현재 security-context.xml에 설정된 구성을 기반한 스프링 시큐리티의 작동 방식을 확인합니다.

→ 톰캣 서버가 중지 중이면, 톰캣 서버 기동

→ 웹브라우저에서 <http://localhost:8080/mypro00/> URL로 접근

→ home.jsp 페이지 표시됨 (permitAll 접근 설정 때문)

→ 웹브라우저에서 <http://localhost:8080/mypro00/myboard/list> URL로 접근

→ 게시물 목록 페이지 표시됨 (permitAll 접근 설정 때문)

→ 게시물 목록 페이지에서 새글 등록 버튼 또는 게시물 제목 하나 클릭

→ 스프링 시큐리티가 제공하는 로그인 페이지 표시됨 → 권한 확인을 하기 위해 로그인 요청

→ username에 admin99, password 입력란에 pw99를 각각 입력하고, Sing in 클릭 → 게시물 등록/게시물 상세 페이지 표시됨.

→ 웹브라우저에서 <http://localhost:8080/mypro00/login> URL로 접근

→ 로그인 페이지 표시됨 → 입력을 하지 않고 Sing in 클릭하면, username과 password 입력 요구함.

→ 로그인 페이지에서 username/password 입력란에 각각 any를 입력 → Sing in 클릭

→ [UserDetailsService returned null, which is an interface contract violation] 메시지가 표시되며, 다시 로그인 요구

→ 로그인 페이지에서 username에 admin99, 암호입력란에 11111 입력 → Sing in 클릭

→ [Bad credentials] 메시지가 표시되며, 다시 로그인 요구

☞ 이러한 아이디와 암호 검증을 위하여, 예전에는 사용자 아이디와 암호 확인 프로그램을 직접 작성했거나 스프링 인터셉터를 하도록

프로그램을 작성했어야 했습니다. 스프링 시큐리티를 이용하도록 구성 및 설정한 것 만으로도 다른 프로그램 작성 없이 아이디와 암호에 대한 검증이 수행됩니다.

- ☞ 스프링 시큐리티가, 사용자가 입력한 `username/password`를 확인하여, 존재하지 않는 계정이거나 계정이 존재하더라도 암호가 틀리면, 값이 없는 `error` 매개변수(승인 오류)가 포함된 로그인 페이지를 사용자 웹 브라우저에게 Get 방식으로 전송  
→ 관련 오류 메시지가 표시됨.  
→ 이 때, 브라우저의 url이 `http://localhost:8080/mypro00/login?error`로 바뀐 것을 확인할 수 있습니다.

[로그인 아이디 또는 암호를 입력하지 않은 경우]

Please sign in

Username

! 이 입력란을 작성하세요.

Sign in

[사용자 아이디와 암호 입력, 아이디 오류]

Please sign in

any

...

Sign in

## [사용자 아이디가 잘못된 경우 - 로그인 시도]

Please sign in

localhost:8080/mypro00/login?error

# Please sign in

UserDetailsService returned null,  
which is an interface contract  
violation

Username

Password

Sign in

[사용자 아이디는 올바르지만 암호가 잘못 입력된 경우 - 로그인 시도]

Please sign in

localhost:8080/mypro00/login?error

Please sign in

Bad credentials

Username

Password

Sign in

☞ 다음은 잘못된 아이디 입력 시, 이클립스 콘솔에 표시된 로그 내용입니다.

```
INFO : com.spring5213.mypro00.common.security.MyMemberDetailsService - Load User By UserName: admin ←admin99가 아님
INFO : jdbc.sqlonly - SELECT mem.userid, mem.userpw, mem.username, mem.enabled, mem.mregdate, mem.mmoddate, mem.mdropflg,
auth.authority FROM book_ex.tbl_myMember mem INNER JOIN book_ex.tbl_myMemberAuthorities auth
ON mem.userid = auth.userid WHERE mem.userid = 'admin'
```

```
INFO : jdbc.sqltiming - SELECT mem.userid, mem.userpw, mem.username, mem.enabled, mem.mregdate, mem.mmoddate, mem.mdropflg, auth.authority FROM book_ex.tbl_myMember mem INNER JOIN book_ex.tbl_myMemberAuthorities auth ON mem.userid = auth.userid WHERE mem.userid = 'admin'  
{executed in 0 msec}  
INFO : jdbc.sqltiming -
```

```
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|-----|
|userid|userpw |username|enabled |mregdate |mmoddate |mdropflg |authority
|-----|-----|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----|-----|-----|
```

WARN : com.spring5213.mypro00.common.security.MyMemberDetailsService - MyMemberMapper에 의해서 반환된 MyMemberVO: null

ERROR: org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter - An internal error occurred while trying to authenticate the user.

`org.springframework.security.authentication.InternalAuthenticationServiceException: UserDetailsService returned null, which is an interface`

```

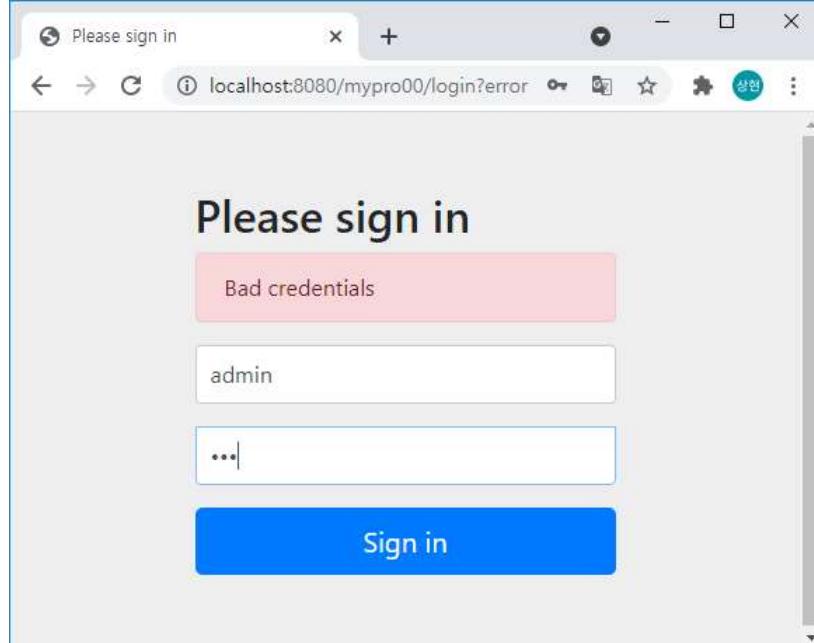
contract violation
at org.springframework.security.authentication.dao.DaoAuthenticationProvider.retrieveUser(DaoAuthenticationProvider.java:110)
at org.springframework.security.authentication.dao.AbstractUserDetailsAuthenticationProvider.authenticate(AbstractUserDetailsAuthenticationProvider.java:144)
at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:199)
at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:219)
at org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter.attemptAuthentication(UsernamePasswordAuthenticationFilter.java:95)
at org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter.doFilter(AbstractAuthenticationProcessingFilter.java:212)
at org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:334)
at org.springframework.security.web.authentication.logout.LogoutFilter.doFilter(LogoutFilter.java:116)
at org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:334)
at org.springframework.security.web.csrf.CsrfFilter.doFilterInternal(CsrfFilter.java:132)
at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:119)
at org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:334)
at org.springframework.security.web.header.HeaderWriterFilter.doHeadersAfter(HeaderWriterFilter.java:92)
at org.springframework.security.web.header.HeaderWriterFilter.doFilterInternal(HeaderWriterFilter.java:77)
at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:119)
at org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:334)
at org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter.doFilterInternal(WebAsyncManagerIntegrationFilter.java:56)

```

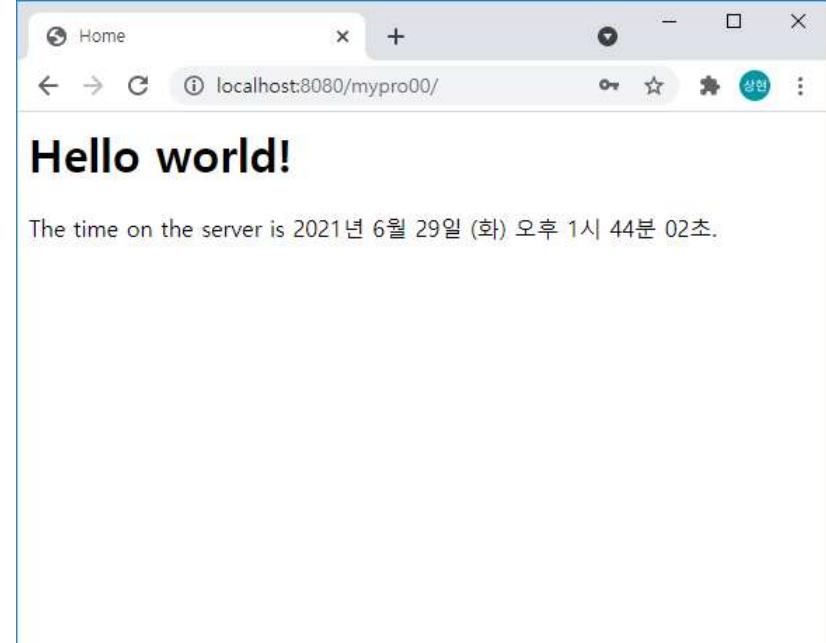
...(생략)...

→ username/password 입력란에 각각 admin99 와 pw99 를 입력 → Sing in 클릭 → home.jsp 내용이 표시됨

[올바른 아이디와 암호 입력 후 Sign in 클릭]



[정상적인 로그인 처리 후]



☞ 스프링 시큐리티는 인증을 위한 로그인이 정상적으로 처리되면, 기본적으로 "\${contextPath}://" URL로 처리된 HTML 내용을 사용자 브라우저에 표시합니다.

☞ 다음은 정상적인 로그인 시에 이클립스 콘솔에 표시된 로그 내용입니다.

```

INFO : com.spring5213.mypro00.common.security.MyMemberDetailsService - Load User By UserName: admin99
INFO : jdbc.sqlonly - SELECT mem.userid, mem.userpw, mem.username, mem.enabled, mem.mregdate, mem.mmoddate, mem.mdropflg,
auth.authority FROM book_ex.tbl_mymember mem INNER JOIN book_ex.tbl_mymemberAuthorities auth
ON mem.userid = auth.userid WHERE mem.userid = 'admin99'

INFO : jdbc.sqltiming - SELECT mem.userid, mem.userpw, mem.username, mem.enabled, mem.mregdate, mem.mmoddate, mem.mdropflg,
auth.authority FROM book_ex.tbl_mymember mem INNER JOIN book_ex.tbl_mymemberAuthorities auth
ON mem.userid = auth.userid WHERE mem.userid = 'admin99'
{executed in 296 msec}
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----|-----|
|userid |userpw          |username |enabled |mregdate        |mmoddate      |mdropflg   |
|-----|-----|-----|-----|-----|-----|-----|
|admin99 |$2a$10$nYbEYDPBZIxNS499qdm1nuK75MHxubBWq7CsrhVrOeoVmp9Zhk8wq |관리자99    |true     |2021-06-08 12:09:58.0 |2021-06-08 12:09:58.0 |0
|ROLE_ADMIN |
|admin99 |$2a$10$nYbEYDPBZIxNS499qdm1nuK75MHxubBWq7CsrhVrOeoVmp9Zhk8wq |관리자99    |true     |2021-06-08 12:09:58.0 |2021-06-08 12:09:58.0 |0
|ROLE_MEMBER |
|admin99 |$2a$10$nYbEYDPBZIxNS499qdm1nuK75MHxubBWq7CsrhVrOeoVmp9Zhk8wq |관리자99    |true     |2021-06-08 12:09:58.0 |2021-06-08 12:09:58.0 |0
|ROLE_USER  |
|-----|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----|-----|
WARN : com.spring5213.mypro00.common.security.MyMemberDetailsService - MyMemberMapper에 의해서 반환된 MyMemberVO: MyMemberVO(userid=admin99,

```

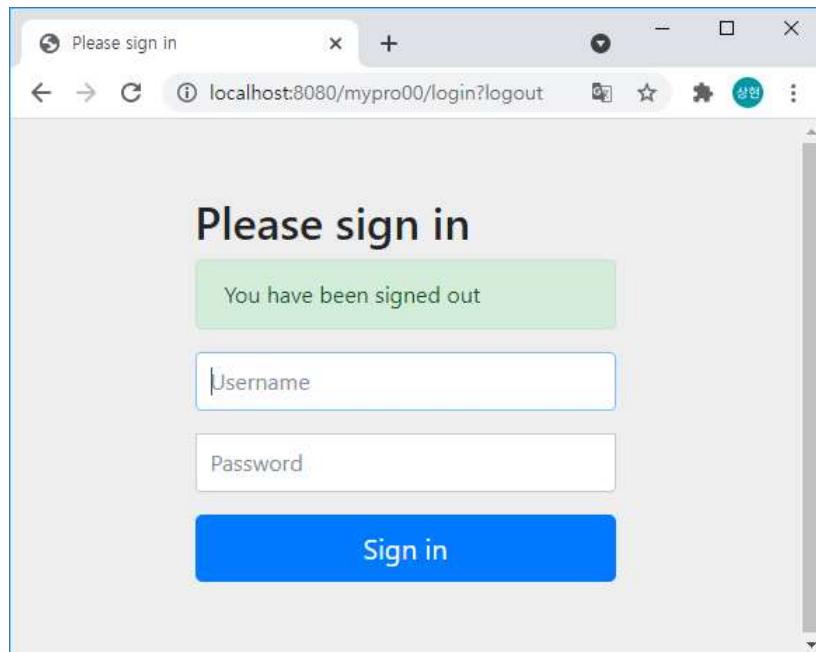
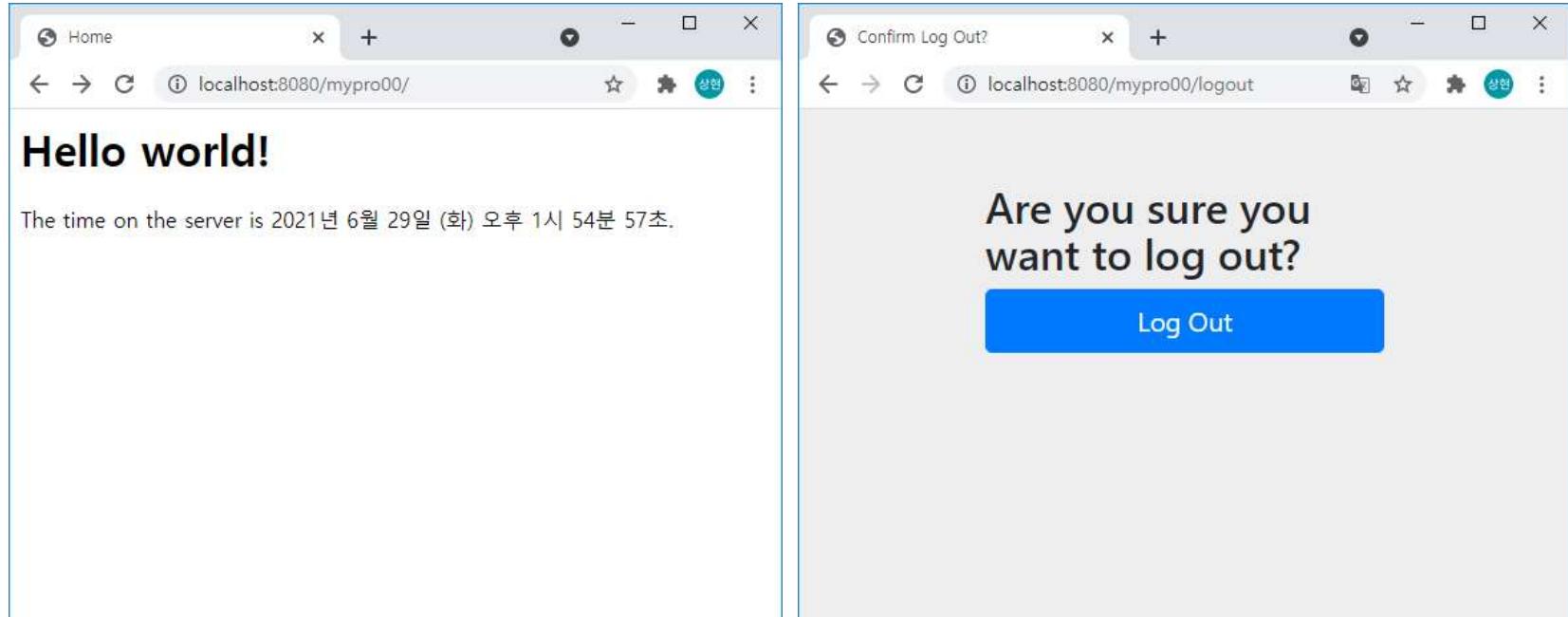
```

userpw=$2a$10$nYbEYDPBZIxNS499qdm1nuK75MHxubBWq7CsrhVrOeoVmp9Zhk8wq, userName=관리자99, mregDate=2021-06-08 12:09:58.0, mmodDate=2021-06-08 12:09:58.0, mdropFlg=0, enabled=true, authorityList=[MyAuthorityVO(userid=admin99, authority=ROLE_ADMIN), MyAuthorityVO(userid=admin99, authority=ROLE_MEMBER), MyAuthorityVO(userid=admin99, authority=ROLE_USER)])
MyUser 생성자에 전달된 MyMemberVO 정보: MyMemberVO(userid=admin99, userpw=$2a$10$nYbEYDPBZIxNS499qdm1nuK75MHxubBWq7CsrhVrOeoVmp9Zhk8wq, userName=관리자99, mregDate=2021-06-08 12:09:58.0, mmodDate=2021-06-08 12:09:58.0, mdropFlg=0, enabled=true, authorityList=[MyAuthorityVO(userid=admin99, authority=ROLE_ADMIN), MyAuthorityVO(userid=admin99, authority=ROLE_MEMBER), MyAuthorityVO(userid=admin99, authority=ROLE_USER)])
MyUser 객체 생성을 통해 MyUser의 부모객체(User객체) 생성됨
=====
INFO : com.spring5213.mypro00.controller.HomeController - Welcome home! The client locale is ko_KR.

```

☞ 로그인 시에 사용자 정보와 권한들의 데이터가 스프링 시큐리티로 전달되는 것을 확인할 수 있습니다.

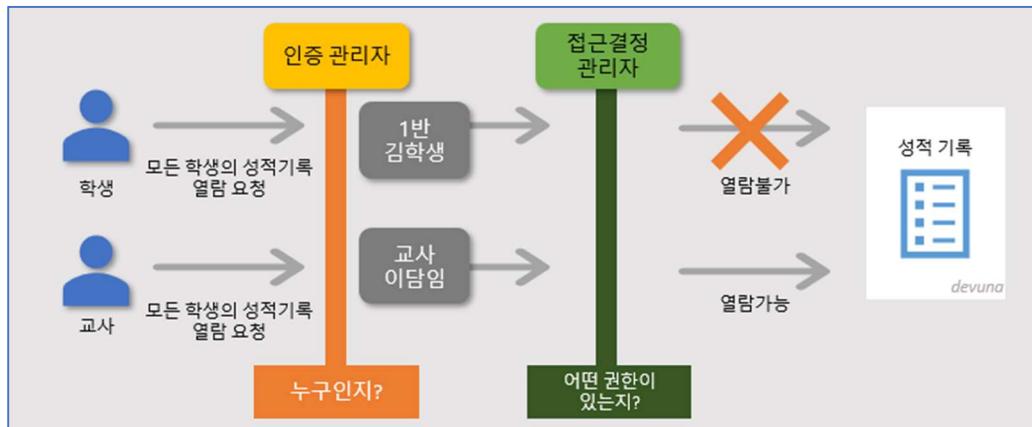
- 로그인 된 웹브라우저에서 <http://localhost:8080/mypro00/logout> 으로 접근 → 로그아웃 페이지가 표시됨 → Log Out 클릭
- 스프링 시큐리티가 \${contextPath}/logout 요청을 인터셉트
  - 값이 없는 logout 매개변수(로그 아웃)가 포함된 로그인 페이지를 사용자 웹 브라우저에게 Get 방식으로 전송
  - 로그인 페이지의 코드에 의해 You have been signed out 로 표시됨.
  - 이 때, 브라우저의 url 입력란에는 <http://localhost:8080/mypro00/login?logout> 으로 표시됩니다.



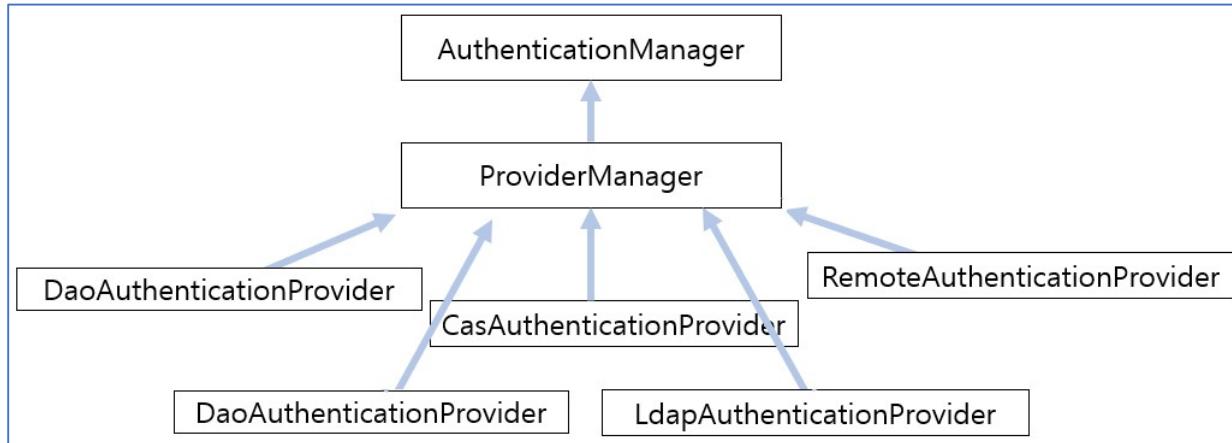
- ☞ 권한 확인은, 로그인을 통해 인증(Authentication)을 거친 후, 계정에 설정된 권한을 확인
- 사용자 접근 URL에 설정된 권한과 비교 → 접근 유무 결정됨.

## ◆ 인증(Authentication)과 승인(Authorization)

- ☞ 스프링 시큐리티의 동작을 이해하기 위해서는 "인증"과 "승인"에 대한 정확한 이해가 필요합니다.
- ☞ '인증(Authentication, AuthenticationManager)'은 자신인 누구인지를 증명하는 것입니다.  
다시 말해서 인증은, 스스로가 자신을 증명할 만한 자료를 제시하여 이를 확인받는 것입니다.
- ☞ '승인(Authorization, AccessDecisionManager)'은 무엇인가를 할 수 있는 권한을 부여받았음을 인정받는 것입니다.  
이 때 무엇을 할 수 있는 권한은, 남에 의해서 부여되어 해당 자격이 생기게 됩니다. 즉, 관리자가 설정해 주어야 합니다.
- ☞ 따라서, 인증은 사용자가 제시한 정보를 확인하여 서버에서 확인받는 행위이고(대표적으로 로그인 처리),  
승인은 사용자에 대하여 서버에 설정된 권한범위 안에서 사용자의 액세스여부를 결정하는 행위(대표적으로 게시물 수정)입니다.



- ☞ 위의 그림에서 인증을 통해 누구인지 확인됩니다. 보통 로그인 ID를 통해 누구인지 서버에 의하여 확인됩니다.  
이러한 로그인 과정 중에 해당 로그인 ID에 대한 권한도 같이 확인됩니다.
- ☞ 로그인 후, 사용자는 로그인 시 확인된 권한을 기준으로 서비스를 이용할 수 있는지 여부(즉, 승인여부)를 확인받습니다.  
승인은 로그인 주체가 학생자격인 경우 성적기록을 볼 수 없지만, 교사자격인 경우에는 성적기록을 볼 수 있도록 서버에 구성됩니다.
- ☞ 일반적으로 **승인은 인증과정을 거친 후, 앱 애플리케이션의 기능을 사용하려는 시도 중에 권한 유무를 확인하는 형태로 처리됩니다.**
- ☞ 스프링 시큐리티의 내부에도 이와 비슷한 구조를 가지고 있습니다. 스프링 시큐리티에서 가장 중요한 역할을 하는 것 중 하나는, '인증'을 담당하는 AuthenticationManager(인증매니저)입니다. AuthenticationManager는 다양한 방식의 인증을 처리할 수 있도록 아래와 같은 구조로 설계되어 있습니다.



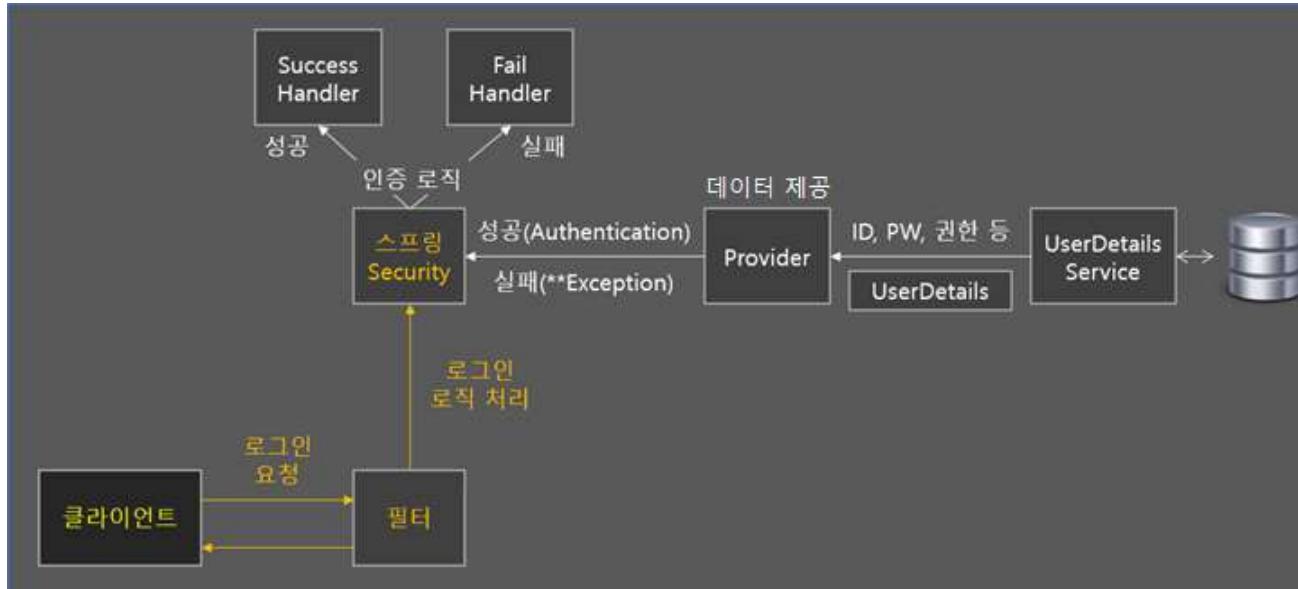
- ☞ ProviderManager는 인증에 대한 처리를 다양한 타입의 **XxxxxAuthenticationProvider** 객체를 이용해서 처리를 위임합니다.
- ☞ **XxxxxAuthenticationProvider**(인증제공자)는 실제 인증 작업을 진행합니다.
- ☞ 이때 인증된 정보에는 권한에 대한 정보도 같이 포함되며, 이 처리는 UserDetailsService 인터페이스의 구현객체를 통해 사용자의 정보와 사용자가 가진 권한의 정보를 처리해서 반환받게 됩니다.

다른 말로는 인증과 승인(권한)에 대한 실제적인 처리는 UserDetailsService의 구현 클래스를 구현하여 이루어집니다.



- ☞ 개발자가 스프링 시큐리티를 커스터마이징 하는 방식은 크게 \*AuthenticationProvider 를 직접 구현하는 방식과 실제 처리를 담당하는 UserDetailsService 를 구현하는 방식으로 나누어집니다.
- ☞ 대부분의 경우에는 UserDetailsService 를 구현하는 형태를 사용하는 것만으로도 충분하지만, 새로운 프로토콜이나 인증 구현 방식을 직접 구현하는 경우에는 AuthenticationProvider 인터페이스의 구현 객체를 직접 구현해서 사용합니다.

#### ◆ 로그인 처리 과정



- ☞ 사용자 브라우저에서 로그인 페이지 요청(Get 방식)
  - 필터가 로그인 페이지 요청을 인터셉트하여 스프링 시큐리티로 전달
  - 스프링 시큐리티에 설정된 로그인 페이지를 사용자 브라우저에 전달
  - 사용자 브라우저로부터 username과 password 가 저장된 품을 Post 방식으로 톰캣 서버에 전달
  - 요청을 필터가 인터셉트하여 스프링 시큐리티로 전달
    - UserDetailsService 객체는 데이터베이스로부터 전달된 사용자 정보를 UserDetails 유형의 객체에 담아 Provider에게 전달
    - Provider는 해당 데이터를 스프링 시큐리티에 전달
    - 스프링 시큐리티에 의해 인증이 수행되고 권한 설정이 확인됨
      - 인증처리 결과(로그인 성공/로그인 실패)에 따라 별도의 핸들러가 구성된 경우, 핸들러의 처리 결과가 전달됨
      - 핸들러가 없는 경우, 시큐리티 기본 설정에 따른 결과의 URL이 전달됨 (기본은 {contextPath}/ URL)

- ☞ 다음은 실습에서 웹브라우저에 표시된 스프링 시큐리티가 제공하는 로그인/로그아웃 페이지를 소스보기 해서 표시된 HTML 소스입니다.

#### ○ 로그인 페이지 HTML 소스

```
<!DOCTYPE html>
<html lang="en">
  <head>
```

```

<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
<meta name="description" content="">
<meta name="author" content="">
<title>Please sign in</title>
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css" rel="stylesheet"
      integrity="sha384-Y6pD6FV/Vv2HJnA6t+vslU6fwYXjCFtcEpHbNJ0lyAFsXTsjBbfaDjzALEQsN6M"
      crossorigin="anonymous">
<link href="https://getbootstrap.com/docs/4.0/examples/signin/signin.css" rel="stylesheet"
      crossorigin="anonymous"/>
</head>
<body>
  <div class="container">
    <form class="form-signin" method="post" action="/myprofile/login">
      <h2 class="form-signin-heading">Please sign in</h2>
      <p>
        <label for="username" class="sr-only">Username</label>
        <input type="text" id="username" name="username" class="form-control" placeholder="Username"
               required autofocus>
      </p>
      <p>
        <label for="password" class="sr-only">Password</label>
        <input type="password" id="password" name="password" class="form-control" placeholder="Password" required>
      </p>
      <input name="_csrf" type="hidden" value="ba0d0863-b30f-405a-b6b9-3dcb2f26217a" />
      <button class="btn btn-lg btn-primary btn-block" type="submit">Sign in</button>
    </form>
  </div>
</body></html>

```

### ○ 로그아웃 페이지 HTML 소스

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="">
    <title>Confirm Log Out?</title>
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css" rel="stylesheet"
          integrity="sha384-Y6pD6FV/Vv2HJnA6t+vslU6fwYXjCFtcEpHbNJ0lyAFsXTsjBbfaDjzALEQsN6M"
          crossorigin="anonymous">
    <link href="https://getbootstrap.com/docs/4.0/examples/signin/signin.css" rel="stylesheet"
          crossorigin="anonymous"/>
  </head>
  <body>
    <div class="container">
      <form class="form-signin" method="post" action="/myprofile/logout">
        <h2 class="form-signin-heading">Are you sure you want to log out?</h2>
      <input name="_csrf" type="hidden" value="ba0d0863-b30f-405a-b6b9-3dcb2f26217a" />
        <button class="btn btn-lg btn-primary btn-block" type="submit">Log Out</button>
      </form>
    </div>
  </body>
</html>

```

## 12-3. 스프링 시큐리티 활용

- 사용자 정의 로그인/로그아웃 페이지 사용 구성 및 로그인/로그아웃 후, 특정 URL 이동 설정
- 자동로그인 구성
- Access Denied Error 시의 처리
- 로그인 성공 후의 추가 처리
- JSP와 컨트롤러에서의 스프링 시큐리티 처리

### [1] 사용자 정의 로그인/로그아웃 페이지 사용 구성 및 로그인/로그아웃 후, 특정 URL 이동 설정

☞ 스프링 시큐리티가 제공하는 로그인 로그아웃 페이지 대신 직접 제작한 로그인 로그아웃 페이지를 사용하도록 구성합니다.

→ 사용자 브라우저로부터 전송된 로그인 페이지 요청을 스프링 시큐리티가 인터셉트 하여 로그인 처리가 진행되도록 구성합니다.

☞ 다음은 앞의 실습으로 기본 사항으로만 구성된 security-context.xml 파일의 내용입니다.

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

    <!-- 사용자 password 암호화 처리 빈: BCrypt 방식 암호화 -->
    <bean id="bCryptPasswordEncoder"
        class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
    <!-- UserDetailsService 구현클래스 빈 -->
    <bean id="myMemberDetailsService"
        class="com.spring5213.mypro00.common.security.MyMemberDetailsService"/>

    <security:http>
        <security:intercept-url pattern="/" access="permitAll" />
        <security:intercept-url pattern="/myboard/list" access="permitAll" />
        <security:intercept-url pattern="/myboard/modify" access="hasRole('ROLE_MEMBER')"/>
        <security:intercept-url pattern="/myboard/register" access="hasAnyRole('ROLE_MEMBER', 'ROLE_USER')"/>
        <security:intercept-url pattern="/myboard/detail" access="hasAuthority('ROLE_ADMIN')"/>

        <security:form-login />

        <security:logout/>

    </security:http>

    <security:authentication-manager>
        <security:authentication-provider user-service-ref="myMemberDetailsService">
            <security:password-encoder ref="bCryptPasswordEncoder" />
        </security:authentication-provider>
    </security:authentication-manager>

</beans>
```

☞ permitAll, hasRole(), hasAnyRole() 표현식은 뒤에서 스프링 시큐리티 표현식에서 설명합니다.

(1-1) GET 방식의 로그인/로그아웃 페이지 URL 요청을 처리하는 컨트롤러 클래스 생성

→ src/main/java/com.spring5213.mypro00.common.security 패키지에 MyLoginLogoutController 클래스 추가

```
package com.spring5213.mypro00.common.security;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import lombok.extern.log4j.Log4j;

@Controller
@Log4j
public class MyLoginLogoutController {

    //로그인 페이지 호출: 로그인 JSP 페이지 이름 문자열 반환: WEB-INF/views/common/myLogin.jsp
    //로그인 페이지 호출은 GET 방식으로만 제한됨
    //스프링 시큐리티가 반환하는 로그인 처리 결과에 대하여 메시지 처리를 하려면,
    //String 유형의 error, logout 매개변수가 선언되어야 함
    @GetMapping("/login") // 사용자 정의 로그인 JSP 페이지 호출 URL, 다른 URL을 사용해도 됨
    public String loginPageGET(String error, String logout, Model model) {

        if (error != null) {
            log.info("로그인 오류 시 error.hashCode(): " + error.hashCode());
            model.addAttribute("error", "로그인 오류. 계정이나 암호를 확인하세요");
        }

        return "common/myLogin";

    } else if (logout != null) {
        log.info("로그인 오류 시 logout.hashCode(): " + logout.hashCode());
        model.addAttribute("logout", "정상적으로 로그아웃 됨");
        return "common/myLogin";
    }

    //정상적인 로그인 페이지 호출
    model.addAttribute("normal", "정상적인 로그인 페이지 호출 처리..");
    log.info("정상적인 로그인 페이지 호출");
    return "common/myLogin";
}

//로그아웃 페이지 호출: 로그아웃 JSP 페이지 이름 문자열 반환: WEB-INF/views/common/myLogout.jsp
//로그아웃 페이지 호출은 GET 방식으로만 제한됨
//@GetMapping("/myLogoutPage")
@GetMapping("/logout") // 사용자 정의 로그아웃 JSP 페이지 호출 URL, 다른 URL을 사용해도 됨
public String logoutPageGET() {
    log.info("로그 아웃 페이지만 호출: 로그아웃은 처리 안됨.....");
    return "common/myLogout";
}
}
```

☞ 생성된 컨트롤러의 각 메서드에 의해 사용자 브라우저에서의 /login, /logout URL에 대하여

아래에서 생성되는 JSP 페이지들의 내용이 전달됩니다.

☞ 스프링 시큐리티는, 기본적으로 로그인/로그아웃 페이지를 호출하는 /login, /logout URL에 대하여, 개발자가 구성한 컨트롤러의 메서드가 지정된 경우, 스프링 시큐리티에 내장된 로그인/로그아웃 페이지를 사용하지 않고, 별도로 구성된 로그인/로그아웃 페이지를 사용합니다.

☞ 따라서, 컨트롤러 메서드에, 개발자가 작성한 로그인/로그아웃 JSP 페이지가 호출될 수 있는 문자열이 반환되도록 구현합니다.

## (1-2) 로그인/로그아웃 JSP 파일 생성

- src/main/webapp/WEB-INF/views 폴더에 common 폴더를 생성
- 생성된 common 폴더에, BootStrap 템플릿의 login.html 템플릿을 이용하여 로그인/로그아웃 JSP 파일을 생성합니다.
- 아래 예에서 수정되거나 추가된 코드는 빨간색으로 표시(지시는 녹색 주석)해 놓았습니다.

→ 로그인 JSP 파일 생성: src/main/webapp/WEB-INF/views/common/myLogin.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<!DOCTYPE html>
<html lang="ko"><!-- 수정 -->
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Login</title><!-- 수정 -->

    <!-- Bootstrap Core CSS -->
    <link href="${contextPath}/resources/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet"><!-- 수정 -->
    <!-- MetisMenu CSS -->
    <link href="${contextPath}/resources/vendor/metisMenu/metisMenu.min.css" rel="stylesheet"><!-- 수정 -->
    <!-- Custom CSS -->
    <link href="${contextPath}/resources/dist/css/sb-admin-2.css" rel="stylesheet"><!-- 수정 -->
    <!-- Custom Fonts -->
    <link href="${contextPath}/resources/vendor/font-awesome/css/font-awesome.min.css" rel="stylesheet"
        type="text/css"><!-- 수정 -->

</head>
<body>
    <div class="container">
        <div class="row">
            <div class="col-md-4 col-md-offset-4">
                <div class="login-panel panel panel-default" style="margin-top: 10%"><!-- 추가 -->
                    <div class="panel-heading">
                        <h2 class="panel-title">Please Sign In</h2><!-- 추가 -->
                    </div>
                    <div class="panel-body">
                        <div><!-- 오류에 의한 호출 시 오류 메시지 표시 div 추가 -->
                            <h5 class="text-muted"><c:out value="${error}" /></h5>
                        </div>
                        <form role="form" method='post' action="${contextPath}/login"><!-- method, action 속성 추가 -->
                            <fieldset>
                                <h6 class="text-muted"><c:out value="${error}" /></h6>
                                <h6 class="text-muted"><c:out value="${logout}" /></h6>
                            </fieldset>
                            <fieldset>
                                <div class="form-group">
                                    <input class="form-control" name="username" id="username" type="text"
  placeholder="Username 입력" required autofocus><!-- required 속성 추가 -->
                                </div>
                                <div class="form-group">
                                    <!-- input의 name 속성값이 password 인지 확인, value 속성 삭제, required 속성 추가 -->
                                    <input class="form-control" name="password" id="password" type="password"
  placeholder="Password 입력" required > ← value 속성 삭제,
                                </div>
                                <div class="checkbox">
                                    <label><!-- 아래 input의 name 속성값이 remember-me 인지 확인, value 속성 삭제 -->
  <input name="remember-me" type="checkbox">자동로그인(Remember Me)
                                    </label>
                                </div>
                            </fieldset>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>
```

```

        </div>
        <div><!-- csrf 토큰값 처리를 위한 div 추가 -->
            <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
        </div>
        <!-- Change this to a button or input when using this as a form -->
        <!-- a 요소를 주석처리 혹은 삭제 -->
        <!-- <a href="index.html" class="btn btn-lg btn-success btn-block">Login</a> -->
        <div><!-- button div 추가 -->
            <button type='submit' class="btn btn-lg btn-success btn-block">Sign in</button>
        </div>
    </fieldset>
    <fieldset><!-- fieldset 내용 추가 -->
        <hr>
        <h6 class="text-muted text-center"><c:out value="${normal}"/></h6>
    </fieldset>
</form>
</div>
</div>
</div>
</div>
</div>
</div>
<!-- jQuery --><!-- jQuery 라이브러리 설정, myheader.jsp에서 복사 붙여넣기 -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<!-- Bootstrap Core JavaScript -->
<script src="${contextPath}/resources/vendor/bootstrap/js/bootstrap.min.js"></script><!-- 수정 -->
<!-- Metis Menu Plugin JavaScript -->
<script src="${contextPath}/resources/vendor/metisMenu/metisMenu.min.js"></script><!-- 수정 -->
<!-- Custom Theme JavaScript -->
<script src="${contextPath}/resources/dist/js/sb-admin-2.js"></script><!-- 수정 -->
</body>
</html>

```

☞ form의 method 속성에는 반드시 post로 설정되어야 합니다.

☞ 위의 코드에서 <input type="hidden" name="\${\_csrf.parameterName}" value="\${\_csrf.token}"> 는, 실습 시에 웹브라우저에서 소스보기를 확인하면, "<input type="hidden" name="\_csrf" value="4898b455-595c-45e5-a959-ad1ddd6cf40" />" 로 변경된 내용이 표시됩니다. value 속성에는 임의의 값이 할당되어 표시됩니다. 뒤에서 CSRF 소개 시에 설명합니다.

☞ post 방식으로 전송하는 경우, 위의 hidden 유형의 input 을 이용하여 csrf 토큰값이 같이 전달되도록 form 에 구성해야 되는 것만 기억합니다(중요)!!

→ 로그아웃 JSP파일 생성: src/main/webapp/WEB-INF/views/common/myLogout.jsp 파일 생성

→ myLogin.jsp 파일의 내용을 복사해서 붙여넣고, 아래처럼 필요없는 것을 삭제하고, 수정합니다.

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Logout</title><!-- 수정 -->

```

```

<!-- Bootstrap Core CSS -->
<link href="${contextPath}/resources/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
<!-- MetisMenu CSS -->
<link href="${contextPath}/resources/vendor/metisMenu/metisMenu.min.css" rel="stylesheet">
<!-- Custom CSS -->
<link href="${contextPath}/resources/dist/css/sb-admin-2.css" rel="stylesheet">
<!-- Custom Fonts -->
<link href="${contextPath}/resources/vendor/font-awesome/css/font-awesome.min.css" rel="stylesheet"
type="text/css">

</head>
<body>
    <div class="container">
        <div class="row">
            <div class="col-md-4 col-md-offset-4">
                <div class="login-panel panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Really Logout ?</h2><!-- 수정 -->
                    </div>
                    <div class="panel-body">
                        <form role="form" action="${contextPath}/logout" method='post'><!-- 수정 -->
                            <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
                            <fieldset>
                                <div>   <!-- 아래 버튼이름 설정 -->
                                    <button type='submit' class="btn btn-lg btn-success btn-block">Log Out</button>
                                </div>
                            </fieldset>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>
</html>

```

### (1-3) 커스텀 로그인/로그아웃 페이지 사용 설정

→ security-context.xml 파일에 생성된 myLogin.jsp 파일과 myLogout.jsp 파일이 사용되도록, 다음처럼 설정합니다.

```

...(생략)...

<security:http>

    <security:intercept-url pattern="/" access="permitAll" />
    <security:intercept-url pattern="/myboard/list" access="permitAll" />
    <security:intercept-url pattern="/myboard/modify" access="hasRole('ROLE_MEMBER')"/>
    <security:intercept-url pattern="/myboard/register" access="hasAnyRole('ROLE_MEMBER', 'ROLE_USER')"/>
    <security:intercept-url pattern="/myboard/detail" access="hasAuthority('ADMIN')"/>

    <security:form-login login-page="/login" login-processing-url="/login"/><!-- 수정 -->

```

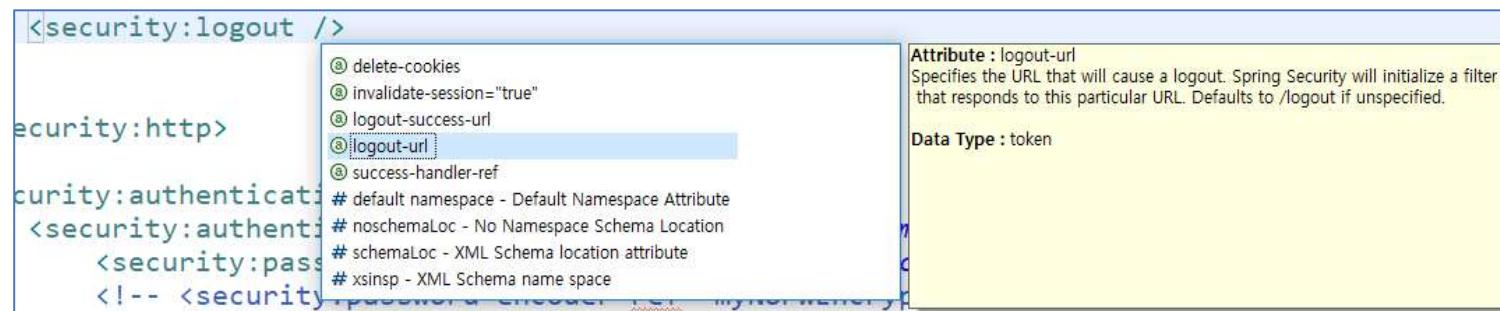
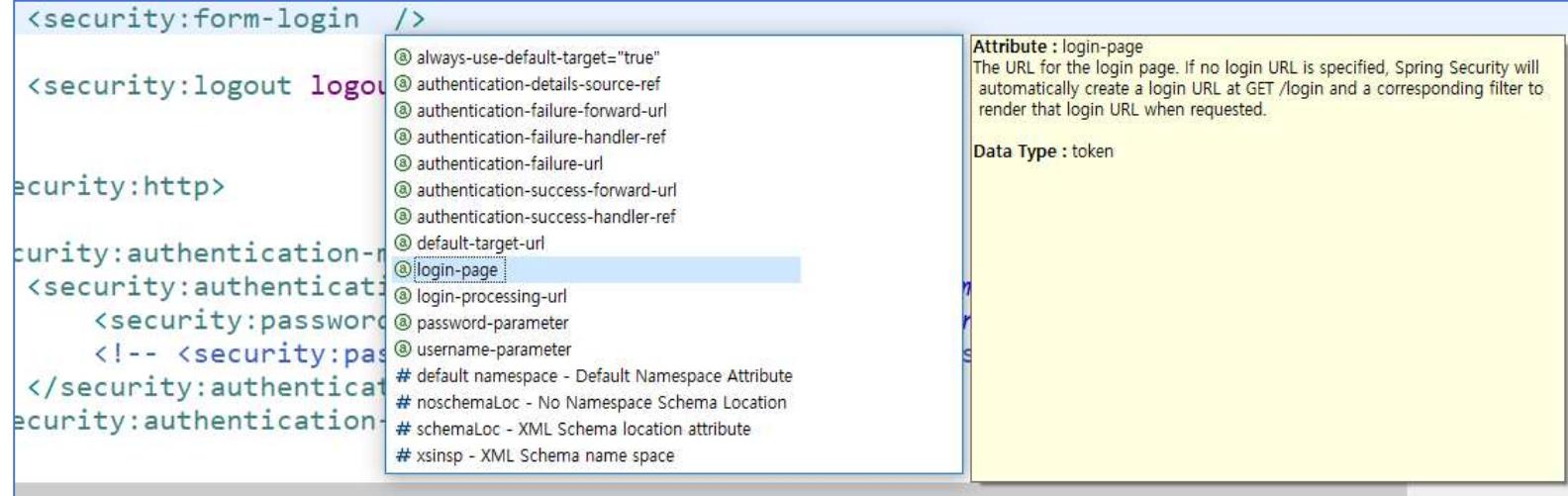
```
<security:logout logout-url="/logout"/><!-- 수정 -->
```

```
</security:http>
```

```
...(생략)...
```

☞ 위의 코드 추가 시에 , <security:form-login /> 요소의 /> 앞에 커서를 위치시키고,

ctrl + space-bar 키를 같이 누르면, 아래처럼 자동입력 가능한 리스트 목록과 각 속성의 설명을 확인할 수 있습니다.



☞ 다음은 <security:form-login> 요소의 대표적인 속성과 그 기능을 정리한 내용입니다.

속성	기능
login-page	로그인 JSP 페이지 호출 URL을 설정 (기본값: /login) ← 별도로 생성한 컨트롤러 메서드가 처리
login-processing-url	로그인 처리 요청 URL, 로그인 JSP 페이지에 있는 <form>의 action 속성값을 설정 (기본값: /login) ← 스프링 시큐리티가 URL 요청을 처리
default-target-url	로그인에 성공하면 보내줄 JSP 페이지 호출 URL을 설정 (기본값: 애플리케이션 루트 URL)
authentication-failure-url	로그인에 실패하면 보내줄 JSP 페이지 호출 URL을 설정 (기본값: /login?error)
username-parameter	로그인 form의, 사용자 아이디를 입력하는 input 요소의 name 속성값을 설정 (기본값: username)
password-parameter	로그인 form의, 사용자 암호를 입력하는 input 요소의 name 속성값을 설정 (기본값: password)
authentication-success-handler-ref	로그인에 성공하면 실행될 로직이 정의된 빈 설정

☞ 다음은 <security:form-login> 요소의 default-target-url 속성에 대한 이클립스 도움말의 일부 내용입니다.

Attribute : default-target-url

The URL that will be redirected to after successful authentication, if the user's previous action could not be resumed. This generally happens if the user visits a login page without having first requested a secured operation that triggers authentication. If unspecified, defaults to the root of the application.

☞ 다음은 <security:logout> 요소의 대표적인 속성과 그 기능을 정리한 내용입니다.

속성	기능
logout-url	로그아웃 처리 요청 URL, 로그아웃 JSP 페이지에 있는 <form>의 action 속성값을 설정(기본값: /logout) ↳ 스프링 시큐리티가 URL 요청을 처리
logout-success-url	로그아웃 성공하면 보내줄 JSP 페이지 호출 URL을 설정 (기본값: /login?logout)
invalidate-session	로그아웃 시, 세션 설정을 모두 무효화(HttpSession 무효화) 여부 (기본값: "true")
delete-cookies="remember-me, JSESSION_ID"	로그아웃 시, 삭제할 쿠키의 이름들을 콤마로 구분해서 설정

(1-4) com.spring5213.mypro00.common.security 패키지의 자동 빈 생성 설정

→ src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml 파일에 com.spring5213.mypro00.common.security 패키지를 스캔하여 생성된 컨트롤러 클래스가 빈으로 생성되도록 설정합니다(이미 되어 있음)

```
...(생략)...
<context:component-scan base-package="com.spring5213.mypro00.common.task" />
<context:component-scan base-package="com.spring5213.mypro00.common.security" /><!-- 추가 -->
...(생략)...
```

[테스트]

→ 톰캣 서버를 다시 기동합니다.

☞ 다음은 톰캣이 기동될 때 표시된 로그의 일부 내용입니다.

```
...(생략)...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
INFO : org.springframework.security.web.DefaultSecurityFilterChain - Creating filter chain: any request,
[org.springframework.security.web.context.SecurityContextPersistenceFilter@3bc735b3,
org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@577f9109,
org.springframework.security.web.header.HeaderWriterFilter@4303b7f0, org.springframework.security.web.csrf.CsrfFilter@757529a4,
org.springframework.security.web.authentication.logout.LogoutFilter@779de014,
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@5c41d037,
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@2234078,
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@5ec77191,
org.springframework.security.web.authentication.AnonymousAuthenticationFilter@4642b71d,
org.springframework.security.web.session.SessionManagementFilter@1450078a,
org.springframework.security.web.access.ExceptionTranslationFilter@c68a5f8,
org.springframework.security.web.access.intercept.FilterSecurityInterceptor@69c6161d]
INFO : org.springframework.security.config.http.DefaultFilterChainValidator - Checking whether login URL '/login' is accessible with your
configuration
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext initialized in 4248 ms
6월 08, 2021 5:48:07 오후 org.apache.catalina.core.ApplicationContext log
정보: Initializing Spring DispatcherServlet 'appServlet'
...(생략)...
```

☞ 사용자 정의 로그인 페이지 구성이 되어 있는 경우, 기동 시에 다음의 메시지가 표시되지 않습니다.

```
INFO : org.springframework.security.config.http.AuthenticationConfigBuilder - No login page configured. The default internal one will be used. Use
the 'login-page' attribute to set the URL of the login page.
```

→ 템켓 서버 기동 후,

웹브라우저에서 `http://localhost:8080/mypro00/login` URL로 접근

→ `MyLoginLogoutController` 컨트롤러에 의해서 사용자 정의 로그인 페이지 표시됨

→ 로그인 페이지에서 `username/password` 입력란에 각각 `admin99` 와 `pw99` 를 입력 → `Sing in` 클릭 → 스프링시큐리티 작동

→ `home.jsp` 내용이 표시됨

☞ 스프링 시큐리티는 인증을 위한 로그인이 정상적으로 처리되면, 기본적으로 `"${contextPath}/"` URL의 처리 HTML 을 사용자 브라우저에 표시

The left screenshot shows a 'Login' page titled 'Please Sign In'. It has two input fields: 'Username' containing 'admin99' and 'Password' containing '....'. There is a 'Remember Me' checkbox and a green 'Sign in' button. Below the form is a note: '정상적인 로그인 페이지 호출 처리..'. The URL in the address bar is `localhost:8080/mypro00/login`.  
The right screenshot shows a 'Home' page with the title 'Hello world!'. Below it is the text 'The time on the server is 2021년 6월 8일 (화) 오후 9시 44분 34초.' The URL in the address bar is `localhost:8080/mypro00/`.

→ 웹브라우저에서 `http://localhost:8080/mypro00/logout` URL로 접근

→ `MyLoginLogoutController` 컨트롤러에 의해서 사용자 정의 로그아웃 페이지 표시됨

→ 로그아웃 페이지에서 `Log Out` 클릭 → 스프링 시큐리티가 `"/logout"` 요청을 인터셉트

→ 값이 없는 `logout` 매개변수(로그 아웃)가 포함된 로그인 페이지를 사용자 웹 브라우저에게 Get 방식으로 전송

→ 사용자 브라우저에 `http://localhost:8080/mypro00/login?logout` 이 표시됨.

The left screenshot shows a 'Logout' page with the question 'Really Logout ?' and a large green 'Log Out' button.  
The right screenshot shows a 'Login' page with the message '정상적으로 로그아웃 됨'. It has two input fields: 'Username 입력' and 'Password 입력'. There is a 'Remember Me' checkbox and a green 'Sign in' button. The URL in the address bar is `localhost:8080/mypro00/login?logout`.

→ 로그인 페이지에서 계정이름이 잘못되거나, 암호가 틀린 경우, 오류 메시지가 표시되는지 확인

→ 브라우저의 URL 이 `http://localhost:8080/mypro00j/login?error` 로 표시되는지 확인

→ 없는 계정의 경우

Please Sign In

any

...

자동로그인(Remember Me)

**Sign in**

정상적인 로그인 페이지 호출 처리..

Please Sign In

로그인 오류. 계정이나 암호를 확인하세요

로그인 오류. 계정이나 암호를 확인하세요

Username 입력

Password 입력

자동로그인(Remember Me)

**Sign in**

→ 암호가 틀린 경우

Please Sign In

로그인 오류. 계정이나 암호를 확인하세요

로그인 오류. 계정이나 암호를 확인하세요

admin99

....

자동로그인(Remember Me)

**Sign in**

Please Sign In

로그인 오류. 계정이나 암호를 확인하세요

로그인 오류. 계정이나 암호를 확인하세요

Username 입력

Password 입력

자동로그인(Remember Me)

**Sign in**

→ 추가적으로 로그인 JSP 페이지에 있는 form의 action 속성값을 \${contextPath}/myLogin 으로 변경하고,

→ MyLoginLogoutController 의 loginPageGet() 메서드의 매팅 URL 과 security-context.xml 파일의 <security:form-login>

요소의 login-processing-url 속성을 모두 "/myLogin"으로 설정한 후

→ 동일한 로그인 처리 과정이 진행되는지 확인합니다.

→ 추가 실습 후, 추가적으로 로그인 JSP 페이지에 있는 form의 action 속성값은 \${contextPath}/login 으로 변경하고,

→ MyLoginLogoutController 의 loginPageGet() 메서드의 매팅 URL 과 security-context.xml 파일의 <security:form-login>

요소의 login-processing-url 속성에는 을 "/login"으로 원래의 설정 구성으로 변경해 놓습니다.

### (1-5) 로그인/로그아웃 후 이동 URL 변경

→ security-context.xml 파일에 다음의 코드 추가

...(생략)...

```
<security:http>
    <security:intercept-url pattern="/" access="permitAll" />
    <security:intercept-url pattern="/myboard/list" access="permitAll" />

    ... (생략) ...

    <security:form-login login-page="/login" login-processing-url="/login"
        default-target-url="/myboard/list" />

    <security:logout logout-url="/logout" logout-success-url="/" />
    <!-- logout-success-url 속성에 지정되는 URL에는 컨텍스트패스가 포함되면 않되고, -->
    <!-- 해당 URL에 대하여 intercept-url 요소의 access 속성으로 permitAll로 설정되어야 함 -->

</security:http>
```

...(생략)...

→ 톰캣 서버 재기동

→ 웹브라우저에서 <http://localhost:8080/mypro00/login> URL로 접근 → 사용자 정의 로그인 페이지 표시됨

→ username/password 입력란에 각각 admin99 와 pw99 를 입력 → Sign in 클릭

→ 스프링시큐리티 작동 → 게시물 목록 페이지 표시됨

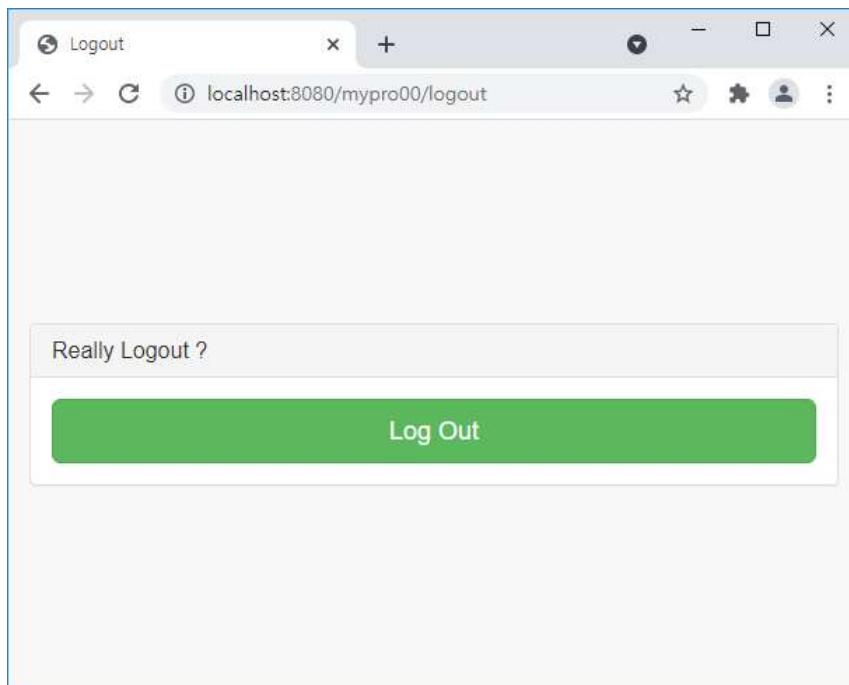
The left screenshot shows a 'Login' page with fields for 'username' (admin99) and 'password' (pw99), a 'Remember Me' checkbox, and a green 'Sign in' button. Below the form is a note: '정상적인 로그인 페이지 호출 처리..'. The right screenshot shows a 'My Admin Board' page with a table of posts. The table has columns: 번호 (Number), 제목 (Title), 작성자 (Writer), 작성일 (Created Date), 수정일 (Modified Date), and 조회수 (View Count). The posts listed are:

번호	제목	작성자	작성일	수정일	조회수
229387	첨부파일 게시물 등록 1 [댓글수: 0]	test	2021/06/06	2021/06/06 22:13:12	8
229385	첨부파일 게시물 등록 1 [댓글수: 0]	test	2021/06/06	2021/06/06 14:14:25	16
229384	화면 새글등록 테스트 1 [댓글수: 0]	test	2021/06/06	2021/06/06 14:04:40	2
229383	화면 새글등록 테스트 2 [댓글수: 0]	test	2021/06/06	2021/06/06 13:24:21	4
229382	화면 새글등록 테스트 1 [댓글수: 0]	test	2021/06/06	2021/06/06 13:22:56	0
229381	작성자에 의하여 삭제된 게시글입니다.				
229380	첨부파일 게시물 등록 1 [댓글수: 0]	test	2021/06/06	2021/06/06	1

→ 웹브라우저에서 <http://localhost:8080/mypro00/logout> URL로 접근 → 사용자 정의 로그아웃 페이지 표시됨

→ Log Out 클릭 → home.jsp 페이지의 내용이 표시됨

(주의) 로그 아웃 후 이동되는 URL은 access 가 permitAll로 설정되어야 합니다.



## [2] 데이터베이스를 이용한 자동 로그인 구성

☞ 웹 애플리케이션의 로그인 페이지에서 대부분 제공되는 자동로그인 기능은, 한 번 로그인하면 일정시간 동안 다시 로그인을 하지 않아도 되는 기능입니다. 자동로그인 기능은 'remember-me'라고도 표현되며, 대부분의 경우, 쿠키(Cookie)를 이용해서 구현됩니다. 많은 국내의 사이트에서도 이 기능을 제공합니다.

☞ 스프링 시큐리티를 이용하여 자동 로그인 기능을 구현하는 방법은, (1) 세션 쿠키를 이용하여 구성하거나, (2) 데이터베이스에 자동로그인 관련 정보를 저장한 후, 이를 이용하도록 구성하는 2 가지 방법으로 구현할 수 있습니다. 이 중, 데이터베이스를 이용하는 방법이 가장 많이 사용됩니다.

사용자가 로그인 했었던 정보를 저장해두고 사용자가 재방문 했을 때, 세션에 정보가 없으면 데이터베이스를 조회해서 필요한 정보를 확인하여 사용합니다. 데이터베이스에 자동 로그인 관련 정보가 저장되므로, 서버의 메모리 상에만 데이터가 유지되는 방식보다 좀 더 안정적으로 운영이 가능합니다.

☞ 자동로그인 기능을 스프링 시큐리티를 이용하면, 쿠키를 생성하여 사용하도록 구성하는 프로그램 작성 없이도 설정만으로 간단하게 구성할 수 있습니다.

### (2-1) 데이터베이스 테이블 생성: 스프링 시큐리티가 자동로그인 관련 데이터가 저장됨

→ SQL\*Developer를 이용하여 오라클 데이터베이스 서버에 book\_ex 계정으로 접속 후, 다음의 SQL문을 실행합니다.

```
--자동 로그인 구성 관련 데이터 저장 테이블  
--스프링 시큐리티에 의해 자동으로 사용되므로, 데이터타입의 길이를 늘리는 것 외에, 테이블이름과 컬럼이름을 변경하면 안됨  
CREATE TABLE book_ex.persistent_logins (  
username VARCHAR2(64) NOT NULL,  
series VARCHAR2(64) PRIMARY KEY,  
token VARCHAR2(64) NOT NULL,  
last_used TIMESTAMP(0) NOT NULL  
) TABLESPACE users;
```

☞ <https://docs.spring.io/spring-security/site/docs/5.3.8.RELEASE/reference/html5/#appendix-schema> 스프링 공식 문서에 위의 테이블 생성 구문이 제공됩니다.

### (2-2) 스프링 시큐리티 빈 구성 XML 설정파일 구성

→ src/main/webapp/WEB-INF/spring/security-context.xml 파일에 다음의 설정(빨간색 코드) 추가

```
...  
<security:http>  
...  
  
<security:form-login login-page="/login" login-processing-url="/login" default-target-url="/myboard/list"/>  
<!-- 스프링 시큐리티 자동로그인: 1주일 유지 --&gt;<br/><!-- Referenced bean 'dataSource' not found 경고는 무시(mybatis-context.xml에 정의되어 있음) --&gt;<br/><security:remember-me data-source-ref="dataSource"  
    remember-me-parameter="remember-me"  
    tokenValiditySeconds = "604800" />
```

```

<security:logout logout-url="/logout" logout-success-url="/" />

</security:http>

...(생략)...

```

☞ 자동로그인은 <security:remember-me> 요소를 이용해서 설정합니다. 다음은 <security:remember-me> 요소의 대표적인 속성과 기능의 내용입니다.

속성	설명
key	쿠키에 사용되는 값을 암호화하기 위한 키(key)값
data-source-ref	데이터베이스에 접속하기 위해 사용되는 빈의 id를 설정
remember-me-cookie	브라우저에서 보관되는 자동로그인 쿠키의 이름 설정(기본값: 'remember-me')
remember-me-parameter	자동로그인을 설정하는 체크박스 input 요소의 name 속성값을 설정(기본값: 'remember-me')
tokenValiditySeconds	자동로그인 쿠키의 유효시간을 초단위로 설정.

### (2-3) 로그인 JSP 페이지에서의 자동로그인 처리

→ src/main/webapp/WEB-INF/views/common/myLogin.jsp 파일에 다음의 빨간색 코드 추가(이미 작성되어 있음)

...(생략)...

```

<form role="form" method='post' action="${contextPath}/login">
    <fieldset>
        <h6 class="text-muted"><c:out value="${error}" /></h6>
        <h6 class="text-muted"><c:out value="${logout}" /></h6>
    </fieldset>
    <fieldset>
        <div class="form-group">
            <input class="form-control" name="username" id="username" type="text" placeholder="Username 입력"
                   required autofocus>
        </div>
        <div class="form-group">
            <input class="form-control" name="password" id="password" type="password" placeholder="Password 입력"
                   required>
        </div>
        <div class="checkbox">
            <label>
                <!-- name 속성의 설정값이 security-context.xml의 remember-me-parameter 속성의 설정값과 동일해야 함 -->
                <!-- value 속성에 값이 지정되면 됩니다. -->
                <input name="remember-me" type="checkbox">자동로그인(Remember Me) ← value 속성을 삭제합니다.
            </label>
        </div>
        <div>
            <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
        </div>
        <div>
            <button type='submit' class="btn btn-lg btn-success btn-block">Sign in</button>
        </div>
    </fieldset>
    <fieldset>
        <hr>
        <h6 class="text-muted text-center"><c:out value="${normal}" /></h6>
    </fieldset>
</form>

```

...(생략)...

## [테스트]

→ 톰캣 서버 기동

☞ 다음은 톰캣 서버 기동 시에 기록된 로그의 일부입니다. RememberMeAuthenticationFilter 빈이 생성된 것을 확인할 수 있습니다.

```
...(생략)...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
INFO : org.springframework.security.web.DefaultSecurityFilterChain - Creating filter chain: any request,
[org.springframework.security.web.context.SecurityContextPersistenceFilter@6ad5923a,
org.springframework.security.web.context.async.WebAsyncManagerIntegrationFilter@4463d9d3,
org.springframework.security.web.header.HeaderWriterFilter@43b0ade, org.springframework.security.web.csrf.CsrfFilter@5395ea39,
org.springframework.security.web.authentication.logout.LogoutFilter@1517f633,
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@4fe01803,
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@13d186db,
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@6f6962ba,
org.springframework.security.web.authentication.rememberme.RememberMeAuthenticationFilter@4565a70a,
org.springframework.security.web.authentication.AnonymousAuthenticationFilter@a0a9fa5,
org.springframework.security.web.session.SessionManagementFilter@15723761,
org.springframework.security.web.access.ExceptionTranslationFilter@312afbc7,
org.springframework.security.web.access.intercept.FilterSecurityInterceptor@599f571f]
INFO : org.springframework.security.config.http.DefaultFilterChainValidator - Checking whether login URL '/login' is accessible with your
configuration
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext initialized in 4372 ms
6월 09, 2021 4:24:20 오후 org.apache.catalina.core.ApplicationContext log
정보: Initializing Spring DispatcherServlet 'appServlet'
...(생략)...
```

→ 웹브라우저에서 <http://localhost:8080/mypro00/login> URL로 접근 → 사용자 정의 로그인 페이지 표시됨

→ username/password 입력란에 각각 admin99 와 pw99 를 입력 → 자동로그인 체크박스 체크 → Sing in 클릭

→ 스프링시큐리티 작동 → 게시물 목록 페이지 표시됨

The left screenshot shows a 'Login' page at [localhost:8080/mypro00/login](http://localhost:8080/mypro00/login). It has two input fields for 'username' and 'password', both containing 'admin99'. A checked checkbox labeled '자동로그인(Remember Me)' is present. A large green 'Sign in' button is at the bottom. Below the form, a note says '정상적인 로그인 페이지 호출 처리..'

The right screenshot shows a 'My Admin Board' page at [localhost:8080/mypro00/myboard/list](http://localhost:8080/mypro00/myboard/list). The title is '게시글 목록'. It includes search filters for '10개' (10 items), '검색범위' (Search range), and a search input field with placeholder '검색어를 입력하세요' (Enter search term). A blue '검색' (Search) button is highlighted. Below the search area is a table with columns: 번호 (Number), 제목 (Title), 작성자 (Author), 작성일 (Created Date), 수정일 (Modified Date), and 조회수 (View Count). One row is visible: 번호 229387, 제목 '첨부파일 게시물 등록', 작성자 'test', 작성일 '2021/06/06', 수정일 '2021/06/06', and 조회수 '8'.

→ SQL\*Developer에서 book\_ex 계정으로 오라클 데이터베이스 서버 접속 → persistent\_logins 테이블에 조회\

→ 테이블에 자동로그인 관련 데이터가 저장된 것을 확인할 수 있습니다.

The screenshot shows the Oracle SQL Developer interface with a connection named 'CONN\_BOOK\_EX'. In the bottom-left panel, a SQL script named 'CONN\_BOOK\_EX.sql' is open, containing the query: 'SELECT \* FROM book\_ex.persistent\_logins ;'. The results pane shows one row of data:

USERNAME	SERIES	TOKEN	LAST_USED
admin99	QrzS6pDCn/ORzQpkem17DQ==	j1UEjQoJ365NGUe/pYHGag==	21/06/09 16:33:25

At the bottom of the interface, status bars show '313 행 20 열 | 삽입 | 수정됨 | Windows: CR'.

→ 로그인 된 크롬 브라우저에서 개발자 도구를 표시 → Application 탭에 있는 Cookies 항목 클릭

→ remember-me 쿠키가 설정된 것을 확인할 수 있습니다.

The screenshot shows the Chrome DevTools interface with the Application tab selected. On the left, there's a sidebar with sections for Manifest, Service Workers, Storage, and Cookies. Under Cookies, there are two entries: 'remember-me' and 'JSESSIONID'. The 'remember-me' cookie is highlighted. Below the table, there's a 'Cookie Value' section showing the raw cookie value: UXJ6UzZwRENuJTJG... .

Name	Value	Domain	Path	Expires / Max-Age	S...	H...	S...	S...	P...
remember-me	UXJ6UzZwRENuJTJG...	localhost	/mypro00	2021-06-16T07:33:24.746Z	93	✓			M...
JSESSIONID	94717C1B45E459FF...	localhost	/mypro00	Session	42	✓			M...

→ 기존 브라우저를 종료

→ 새로운 크롬 브라우저에서 <http://localhost:8080/mypro00/myboard/register> URL로 접근

→ 로그인 없이 접근됩니다.

The screenshot shows a browser window titled 'My Admin Board' with the URL 'localhost:8080/mypro00/myboard/register'. The page title is 'Board - Register'. The form has four fields: '제목' (Title), '내용' (Content), '작성자' (Writer), and '게시글 등록' (Post Article). The 'remember-me' cookie is still present in the browser's cookie store.

→ 브라우저에서 개발자 도구를 표시 → Application 탭에 있는 Cookies 항목 클릭

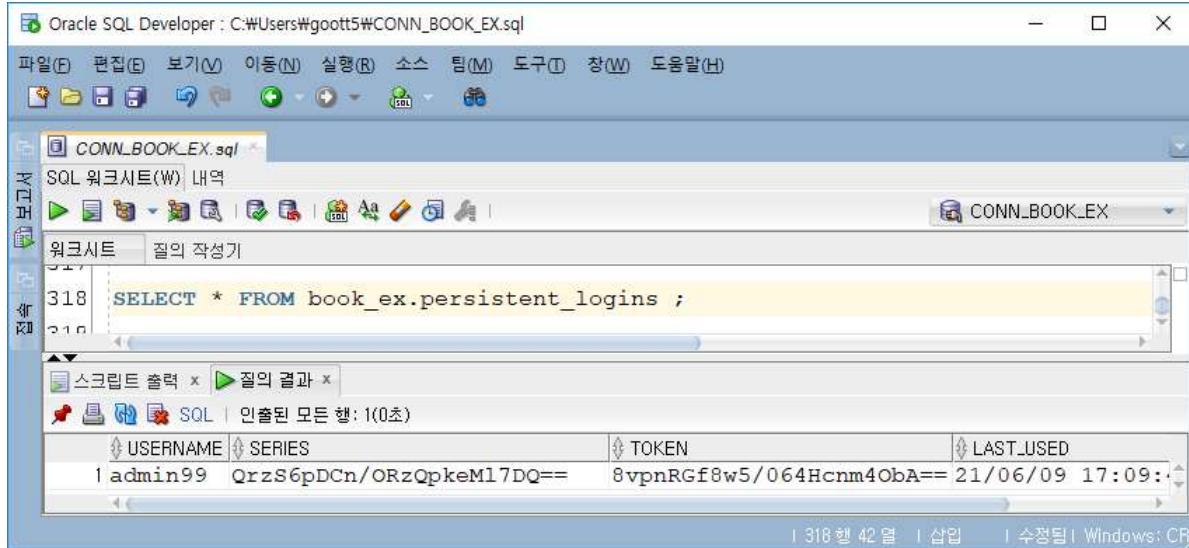
→ 쿠키가 동일한 것을 확인할 수 있습니다.

The screenshot shows the Chrome DevTools interface with the Application tab selected. The sidebar shows the same cookie entries as before. The 'remember-me' cookie is still highlighted. The 'Cookie Value' section below the table shows the raw cookie value again: UXJ6UzZwRENuJTJG... .

Name	Value	Domain	Path	Expires / Max-Age	S...	H...	S...	S...	P...
remember-me	UXJ6UzZwRENuJTJG...	localhost	/mypro00	2021-06-16T07:33:24.746Z	93	✓			M...
JSESSIONID	94717C1B45E459FF...	localhost	/mypro00	Session	42	✓			M...

→ SQL\*Developer에서 persistent\_logins 테이블에 조회

→ LAST\_USED 컬럼 값이 바뀐 것을 알 수 있습니다.



The screenshot shows the Oracle SQL Developer interface. A query window titled 'CONN\_BOOK\_EX.sql' displays the following SQL statement:

```
318 | SELECT * FROM book_ex.persistent_logins ;
```

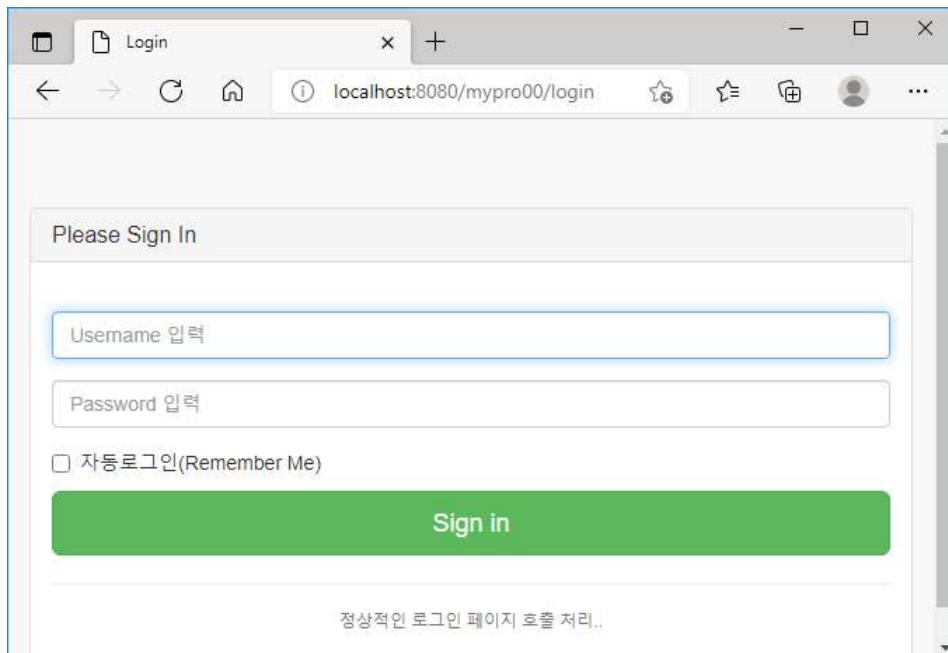
The results pane shows one row from the persistent\_logins table:

USERNAME	SERIES	TOKEN	LAST_USED
admin99	QrzS6pDCn/ORzQpkeM17DQ==	8vpnRGf8w5/064Hcnm4ObA==	21/06/09 17:09:11

At the bottom of the results pane, it says '1 행 42 열 | 삽입 | 수정됨 | Windows: CR'.

→ 엣지 브라우저에서 <http://localhost:8080/mypro00/myboard/register> URL로 접근

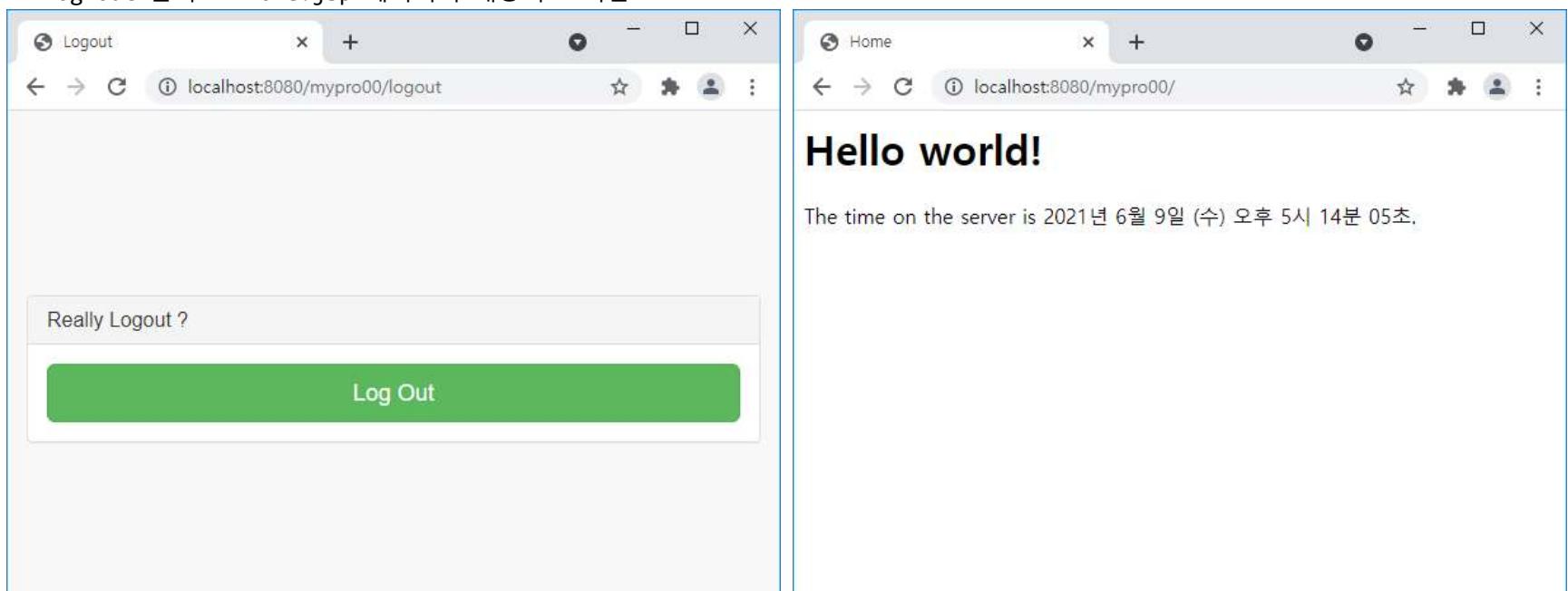
→ 로그인 페이지가 표시됩니다.



→ 크롬 브라우저에서 <http://localhost:8080/mypro00/logout> URL로 접근

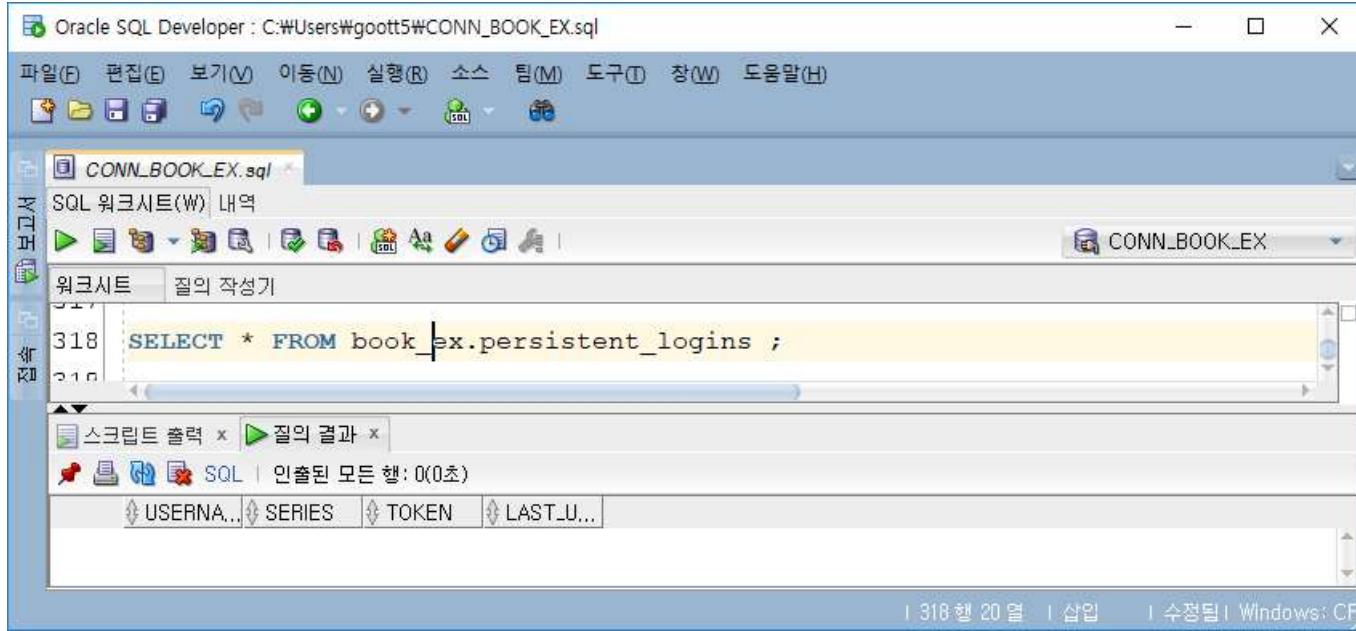
→ 사용자 정의 로그아웃 페이지 표시됨

→ Log Out 클릭 → home.jsp 페이지의 내용이 표시됨



The screenshot shows two adjacent Chrome browser windows. The left window is titled 'Logout' and shows a confirmation message 'Really Logout ?' with a green 'Log Out' button. The right window is titled 'Home' and shows the 'Hello world!' welcome message and the server time 'The time on the server is 2021년 6월 9일 (수) 오후 5시 14분 05초.'

- SQL\*Developer에서 persistent\_logins 테이블에 조회
- 자동 로그인 관련 데이터가 자동으로 삭제된 것을 확인할 수 있습니다.



The screenshot shows the Oracle SQL Developer interface. The title bar says "Oracle SQL Developer : C:\Users\goott5\CONN\_BOOK\_EX.sql". The menu bar includes "파일(F)", "편집(E)", "보기(V)", "이동(N)", "실행(R)", "소스(S)", "팀(M)", "도구(I)", "창(W)", and "도움말(H)". Below the menu is a toolbar with various icons. The main workspace has a tab labeled "CONN\_BOOK\_EX.sql" and a sub-tab "SQL 워크시트(W) 내역". A toolbar below the tabs includes icons for running, saving, and zooming. The SQL worksheet pane contains the following code:

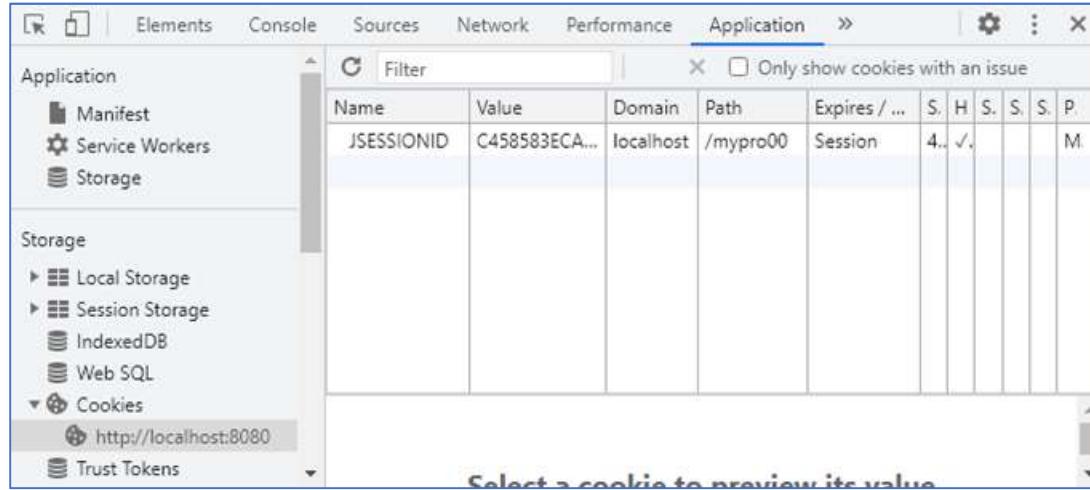
```
318: SELECT * FROM book_ex.persistent_logins ;
```

Below the code, there are two tabs: "스크립트 출력(x)" and "질의 결과(x)". The "질의 결과(x)" tab shows the results of the query:

USERNA...	SERIES	TOKEN	LAST_U...

At the bottom of the window, it says "318 행 20 열 | 삽입 | 수정됨 | Windows: CR".

- 브라우저의 쿠키 정보를 확인하면, remember-me 쿠키가 없는 것이 확인됩니다.



The screenshot shows the Chrome DevTools Application tab. On the left, there's a sidebar with sections for "Application", "Manifest", "Service Workers", and "Storage". Under "Storage", there are sections for "Local Storage", "Session Storage", "IndexedDB", "Web SQL", and "Cookies". The "Cookies" section is expanded, showing a table of cookies for the domain "http://localhost:8080". One cookie is listed:

Name	Value	Domain	Path	Expires / ...	S.	H.	S.	S.	P.
JSESSIONID	C458583ECA...	localhost	/mypro00	Session	4.. ✓.				M.

At the bottom of the table, it says "Select a cookie to preview its value".

- ☞ 기본적으로 로그아웃을 하면, 브라우저에 쿠키로 설정된 항목(자동로그인 쿠키 포함)들이 자동으로 모두 삭제되고, 세션에 설정된 것들이 모두 해제됩니다.

#### [참고] 'JSESSIONID' 쿠키

- ☞ JSESSIONID는 톰캣에서 발생된 세션 쿠키의 이름이며, 동일한 방식으로 WAS 마다 접속한 웹 브라우저에게 고유한 이름의 세션 쿠키를 설정합니다.
- ☞ 세션쿠키를 통해 설정된 세션아이디는, (1) 접속한 컴퓨터, (2) 브라우저 종류 및 (3) 시점에 따라 각각 고유한 값이 할당되어 사용되며, 이러한 세션 쿠키는 브라우저가 종료되기 전까지 유지되다가 브라우저가 종료되면 사라집니다.
- ☞ 스프링 시큐리티는, 웹브라우저에 발행된 세션쿠키를 통해, 현재 세션의 상태가, 로그인 인증과정을 거쳐 특정 URL에 대한 액세스가 승인되었다고 확인하게 됩니다.

[참고] 접속 세션의 최대 허용 시간 설정

프로젝트 탐색기 → Servers 프로젝트 확장 → 사용 중인 서버 이름 확장 → web.xml 파일을 코드 작성 뷰에 오픈

→ session-config 를 검색 →

```
<!-- ===== Default Session Configuration ===== -->
<!-- You can set the default session timeout (in minutes) for all newly    -->
<!-- created sessions by modifying the value below.                      -->

<session-config>
    <session-timeout>60</session-timeout>   ← 설정값을 변경하여 접속 세션의 최대 허용시간(분단위)을 설정할 수 있습니다.
</session-config>
```

→ 스프링 시큐리티도 위 설정의 제어를 받습니다.

### [3] 사용자 정의 오류 페이지 표시 구성

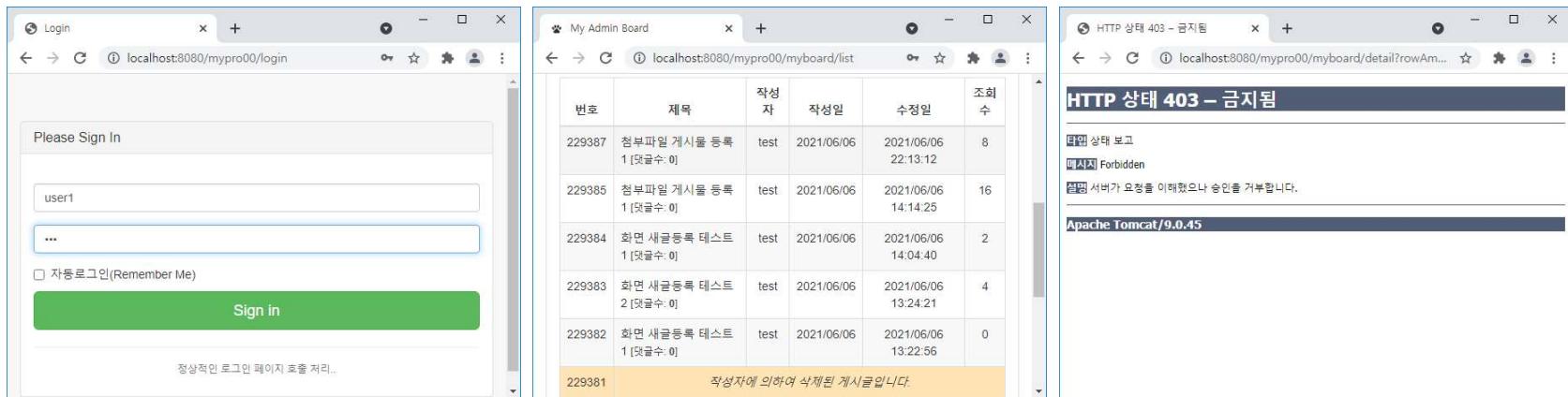
☞ 스프링 시큐리티가 작동된 후에는, 로그인 되지 않은 상태에서 권한 확인이 필요한 URL에 접근 시에 로그인 페이지가 표시되는 것을 앞의 실습으로 확인했습니다.

☞ 로그인 상태에서는, 권한이 없거나 부족한 URL에 접근을 시도하게 되면, 사용자의 웹 브라우저에는 톰캣으로부터 전송된 "HTTP 상태 403 금지됨"이라는 오류 페이지가 표시됩니다.

→ 웹 브라우저에서 `http://localhost:8080/mypro00/login` URL로 접근 → 로그인 페이지

→ `username/password` 입력란에 각각 `user1`과 `pw1`를 입력 → `Sign in` 클릭 → 게시물 목록 정보 페이지

→ 첫 번째 게시물의 제목 클릭 → "HTTP 상태 403 금지됨" 오류 페이지가 표시됨.



☞ `user1`은 `ROLE_USER` 권한이 설정되어 있고, 게시물 상세 페이지는 `ROLE_ADMIN` 를 가진 사용자만 접근이 가능하므로, 오류 페이지가 표시됩니다.

☞ 이러한 오류 페이지는 웹 애플리케이션 서비스를 제공받는 입장에서는 승인이 거부당했다는 메시지가 당황스러울 수도 있습니다. 따라서, 오류 페이지보다는 서비스와 일관성이 있는 적절한 메시지가 포함된 형태의 내용으로 된 내용을 표시해주는 것이 효과적일 것입니다.

☞ 스프링 시큐리티에 의한 접근 제한 오류 발생 시, 톰캣 오류페이지 대신 앞에서 준비한 `myAccessForbidden.jsp`의 HTML 내용이 사용자 웹브라우저로 전달되도록 구성하는 것은 2 가지 방법을 사용할 수 있습니다.

1. `<security:access-denied-handler>` 요소의 `error-page` 속성을 이용하는 방법

2. `<security:access-denied-handler>` 요소의 `ref` 속성을 이용하는 방법

(1) 2 가지 방법에서 공통적으로 사용되는 구성 요소 준비: 컨트롤러 클래스와 JSP 페이지

(1-1) `myAccessForbiddenMsg.jsp` 페이지 생성: 사용자 브라우저로 전달되는 접근 금지 메시지 HTML을 생성합니다.

→ `src/main/webapp/WEB-INF/views/common` 폴더에 `err_msg` 폴더를 생성한 후,

→ 생성된 `err_msg` 폴더에 `myAccessForbiddenMsg.jsp` 파일을 생성하고 다음의 내용을 작성합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
```

```

<html>
<head>
    <meta charset="UTF-8">
    <title>접근제한</title>
</head>
<body>
    <h2>해당 페이지에 접근이 제한됨</h2>
    <fieldset>
        <h4>회원님은 해당 페이지를 이용하실 수 없습니다.</h4>
    </fieldset>
    <br>
    <fieldset>
        <h4><c:out value="\${SPRING_SECURITY_403_EXCEPTION.getMessage()}" /></h4>
        <hr>
        <h4>msg: <c:out value="\${msg}" /></h4>
    </fieldset>
</body>
</html>

```

- ☞ 접근하려는 URL에 대하여 승인되지 못하면, HTTP 상태 - 403 오류가 발생되며, 설정에 명시된 URL을 통해 호출된 JSP에는, HttpServletRequest Request에 'SPRING\_SECURITY\_403\_EXCEPTION'이라는 이름으로 AccessDeniedException 객체가 전달됩니다.
- ☞ 구성 확인을 목적으로 하므로, 디자인과는 상관없이 표시된 내용만 확인합니다.

(1-2) MyErrMsgController 생성: 접근 금지 URL 요청을 처리

→ src/main/java/com.spring5213.mypro00.common.security 패키지에 MyErrMsgController 클래스 생성

```

package com.spring5213.mypro00.common.security;

import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import lombok.extern.log4j.Log4j;

//접근금지(403) 오류 발생 시, 스프링 시큐리티의 요청을 처리할 컨트롤러
@Controller
@Log4j
public class MyErrMsgController {

    //접근 금지(403) 오류에 대한 메시지 페이지 전달
    @GetMapping("/accessForbiddenError")
    public String callAccessForbiddenPage(Authentication authentication, Model model) {
        System.out.println("Forbidden 오류에 대한 메시지 전송 메소드");
        log.info("액세스 거부 시 전달된 Authentication 객체: " + authentication);
        model.addAttribute("msg", "접근이 금지됨.");
        
        return "common/err_msg/myAccessForbiddenMsg";
    }
}

```

(2) 접근 금지(403) 오류에 대한 처리 1: <security:access-denied-handler> 요소의 error-page 속성 이용

→ src/main/webapp/WEB-INF/spring/security-context.xml 파일의 <security:http> 요소에 다음의 설정을 추가합니다.

...(생략)...

```
<security:http>  
...  
<security:access-denied-handler error-page="/accessForbiddenError" />  
  
<security:form-login login-page="/login" login-processing-url="/login"  
default-target-url="/myboard/list"/>  
...
```

#### [테스트 - 접근 금지에 대한 사용자 정의 메시지 페이지 표시]

→ 템켓 재기동

→ 웹 브라우저에서 http://localhost:8080/mypro00/login URL로 접근 → 로그인 페이지

→ username/password 입력란에 각각 user1 과 pw1를 입력 → Sing in 클릭

→ 게시물 목록 정보 페이지 → 첫 번째 게시물의 제목 클릭 → 실습에서 준비된 사용자 정의 접근 금지 메시지 페이지가 표시됩니다.

The figure consists of three screenshots. The first screenshot shows a browser window titled 'Login' with the URL 'localhost:8080/mypro00/login'. It displays a 'Please Sign In' form with two input fields ('user1' and 'pw1'), a 'Remember Me' checkbox, and a green 'Sign in' button. The second screenshot shows a browser window titled 'My Admin Board' with the URL 'localhost:8080/mypro00/myboard/list'. It displays a table of posts with columns: 번호 (Number), 제목 (Title), 작성자 (Author), 작성일 (Created Date), 수정일 (Modified Date), and 조회수 (View Count). The third screenshot shows a browser window titled '접근제한' with the URL 'localhost:8080/mypro00/myboard/detail?rowAm...'. It displays a single message: '회원님은 해당 페이지를 이용하실 수 없습니다.' (Member, you cannot use this page.)

#### 해당 페이지에 접근이 제한됨

회원님은 해당 페이지를 이용하실 수 없습니다.

getMessage(): Access is denied

msg: 접근이 금지됨.

☞ 메시지가 표시되는 브라우저의 URL에는, 사용자가 접근하려고 했던 URL(문서 실습에서는 229387 글번호의 게시물 상세 페이지 URL)이 표시됩니다.

→ 다음은 실습에서 Forbidden(403) 오류 발생 시, 동작한 컨트롤러의 메소드에 의해 콘솔에 표시된 내용입니다.

```
INFO : com.spring5213.mypro00.common.security.MyAccessMessageController - 액세스 거부 시 전달된 Authentication 객체:  
org.springframework.security.authentication.UsernamePasswordAuthenticationToken@bbe149ac: Principal:  
org.springframework.security.core.userdetails.User@179ec: Username: user1; Password: [PROTECTED]; Enabled: true; AccountNonExpired: true;  
credentialsNonExpired: true; AccountNonLocked: true; Granted Authorities: ROLE_USER; Credentials: [PROTECTED]; Authenticated: true; Details:  
org.springframework.security.web.authentication.WebAuthenticationDetails@ffffd3270: RemoteIpAddress: 0:0:0:0:0:0:1; SessionId:  
E146910B81E1C067F570113340D72F86; Granted Authorities: ROLE_USER
```

(3-3) 접근 금지(403) 오류에 대한 처리 2: <security:access-denied-handler> 요소의 ref 속성 이용

☞ 접근 금지(403) 오류가 발생된 경우에, 다양한 처리를 하고 싶다면, AccessDeniedHandler 인터페이스의 구현객체를 직접 구현합니다. 예를 들어 접근이 금지되었을 때, 쿠키나 세션에 특정한 작업을 하거나 또는 HttpServletResponse에 특정한 헤더 정보를 추가하는 등의 처리를 구현할 수 있습니다.

☞ AccessDeniedHandler 인터페이스의 재정의 할 추상메서드는 handle() 뿐이며, HttpServletRequest, HttpServletResponse를 파라미터로 사용하기 때문에 직접적으로 서블릿 API를 이용하는 처리가 가능합니다.

☞ <https://docs.spring.io/spring-security/site/docs/5.3.8.RELEASE/reference/html5/#servlet-exceptiontranslationfilter> 공식문서를 참고하면, 보다 자세한 내용을 확인할 수 있습니다.

☞ 이 방법은 org.springframework.security.web.access.AccessDeniedHandler 인터페이스의 구현 클래스의 빈을 설정하여 필요한 처리를 구현할 수 있습니다.

☞ 실습에서는 앞의 실습에서와 동일하게 접근 금지 오류 발생 시, 준비된 메시지 페이지를 사용자 브라우저로 전달하도록 구현합니다.

→ src/main/java/com.spring5213.mypro00.common.security 패키지에 MyAccessDeniedHandler 클래스 생성

```
package com.spring5213.mypro00.common.security;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.web.access.AccessDeniedHandler;

import lombok.extern.log4j.Log4j;

@Log4j
public class MyAccessDeniedHandler implements AccessDeniedHandler {

    @Override
    public void handle(HttpServletRequest request,
                       HttpServletResponse response,
                       AccessDeniedException accessDeniedException)
            throws IOException, ServletException {
        log.error("AccessDeniedHandler의 구현객체 - 사용자 요청 Redirect.....");
        //접근이 제한되면 리다이렉트 방식으로 명시된 URL의 컨트롤러 호출
        response.sendRedirect("/mypro00/accessFobiddenError");
    }
}
```

→ src/main/webapp/WEB-INF/spring/security-context.xml 파일에 다음의 설정을 추가합니다.

```
...(생략)...
<!-- AccessDeniedHandler 구현클래스 빈 -->
<bean id="myAccessDeniedHandler"
      class="com.spring5213.mypro00.common.security.MyAccessDeniedHandler"></bean>

<security:http >

...(생략)...

<security:access-denied-handler ref="myAccessDeniedHandler" />

<security:form-login login-page="/login" login-processing-url="/login"
                      default-target-url="/myboard/list"/>

<security:remember-me data-source-ref="dataSource"
                       remember-me-parameter="remember-me"
                       tokenValiditySeconds = "604800" />

<security:logout logout-url="/logout" logout-success-url="/" />

</security:http>
```

...(생략)...

[테스트 - 접근 금지에 대한 사용자 정의 메시지 페이지 표시(AccessDeniedHandler 구현 클래스 이용)]

→ 템켓 재기동

→ 웹 브라우저에서 http://localhost:8080/mypro00/login URL로 접근 → 로그인 페이지

→ username/password 입력란에 각각 user1 과 pw1 를 입력 → Sing in 클릭

→ 게시물 목록 정보 페이지 → 첫 번째 게시물의 제목 클릭 → 사용자 정의 접근 금지 메시지가 표시됩니다.

☞ 실습 결과는 access-error 속성 사용 시와 동일합니다.

## [4] 로그인 성공 시, 단순 URL 이동을 포함한 추가적인 로직 처리 클래스

- ☞ 로그인 성공 이후에, 조건에 따라 특정한 동작이 수행되도록 제어하고 싶은 경우가 있습니다. 예를 들어, 관리자(실습에서 admin99 계정)이 로그인 했다면, 관리자가 어떤 경로의 로그인 페이지를 통해 들어오든지 무조건 '/mypro00/admin' URL로 이동하도록 하거나, 로그인 계정에 대하여 별도의 쿠키 등을 생성해서 로그인을 처리하고 싶은 경우입니다.
- ☞ 스프링 시큐리티에서는 `AuthenticationSuccessHandler`라는 인터페이스의 구현객체를 생성하여 이러한 요구사항을 구현할 수 있도록 지원합니다.
- ☞ 로그인 성공 후에 처리할 로직을 구현해야 하는 경우, `AuthenticationSuccessHandler` 인터페이스의 구현 클래스를 생성하여 시큐리티 빈 구성 설정파일에 빈으로 설정 한 후, `<security:form-login>` 요소의 `authentication-success-handler-ref` 속성에 빈으로 등록합니다.

☞ 다음은 `AuthenticationSuccessHandler` 인터페이스에 대한 스프링 5.3.8 버전의 공식 문서 API의 내용입니다.

[org.springframework.security.web.authentication](#)

### Interface AuthenticationSuccessHandler

#### All Known Implementing Classes:

[ForwardAuthenticationSuccessHandler](#), [SavedRequestAwareAuthenticationSuccessHandler](#), [SimpleUrlAuthenticationSuccessHandler](#)

public interface **AuthenticationSuccessHandler**

Strategy used to handle a successful user authentication.

Implementations can do whatever they want but typical behaviour would be to control the navigation to the subsequent destination (using a redirect or a forward). For example, after a user has logged in by submitting a login form, the application needs to decide where they should be redirected to afterwards (see `AbstractAuthenticationProcessingFilter` and subclasses). Other logic may also be included if required.

#### Since:

3.0

### Method Summary

Modifier and Type	Method and Description
void	<code>onAuthenticationSuccess(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response, <b>Authentication</b> authentication)</code> Called when a user has been successfully authenticated.
default void	<code>onAuthenticationSuccess(javax.servlet.http.HttpServletRequest request, javax.servlet.FilterChain chain, <b>Authentication</b> authentication)</code> Called when a user has been successfully authenticated.

### Method Detail

#### onAuthenticationSuccess

```
default void onAuthenticationSuccess(javax.servlet.http.HttpServletRequest request,
                                    javax.servlet.http.HttpServletResponse response,
                                    javax.servlet.FilterChain chain,
                                    Authentication authentication)
                                    throws java.io.IOException, javax.servlet.ServletException
```

Called when a user has been successfully authenticated.

**Parameters:**

request - the request which caused the successful authentication  
 response - the response  
 chain - the FilterChain which can be used to proceed other filters in the chain  
 authentication - the Authentication object which was created during the authentication process.

**Throws:**

java.io.IOException  
 javax.servlet.ServletException

**Since:**

5.2.0

**onAuthenticationSuccess**

```
void onAuthenticationSuccess(javax.servlet.http.HttpServletRequest request,
                            javax.servlet.http.HttpServletResponse response,
                            Authentication authentication)
                            throws java.io.IOException, javax.servlet.ServletException
```

Called when a user has been successfully authenticated.

**Parameters:**

request - the request which caused the successful authentication  
 response - the response  
 authentication - the Authentication object which was created during the authentication process.

**Throws:**

java.io.IOException  
 javax.servlet.ServletException

☞ 실습에서는 로그인 성공 후에, 계정에 대한 권한 유무에 따라 이동되는 페이지가 다르도록 구현합니다.

- ROLE\_ADMIN 권한이 설정된 경우 → bno 1 의 게시물 상세 페이지로 무조건 이동
- ROLE\_ADMIN 권한이 없는 경우 → / URL로 이동

☞ 권한 이름은, 로그인 성공 후에 스프링 시큐리티에 유지되는 Authentication 객체를 전달받아 권한이름을 추출하도록 구현합니다.

(1) **MyLoginSuccessHandler** 구현 클래스 생성 ← **AuthenticationSuccessHandler** 인터페이스의 구현 클래스이어야 함

→ src/main/java/com.spring5213.mypro00.common.security 패키지에 **MyLoginSuccessHandler** 클래스 추가

```
package com.spring5213.mypro00.common.security;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;

import lombok.extern.log4j.Log4j;
```

//로그인이 성공된 후에 처리할 내용이 구현된 클래스  
//AuthenticationSuccessHandler 인터페이스의 구현객체로 생성.

```
@Log4j
public class MyLoginSuccessHandler implements AuthenticationSuccessHandler {
```

```

@Override
public void onAuthenticationSuccess(HttpServletRequest request,
                                    HttpServletResponse response,
                                    Authentication authentication) //인증객체
                                    throws IOException, ServletException {
    log.info("로그인 성공 후: MyLoginSuccessHandler.onAuthenticationSuccess() 시작.....");
    log.info("전달된 Authentication 객체 정보: " + authentication);

    List<String> roleNames = new ArrayList<>(); //룰 목록을 저장
    //권한이름 추출
    authentication.getAuthorities() //인증객체에 저장되는 권한들(authorities)을 가져옴
        .forEach( //각 권한(authority)의 권한이름(롤이름)을 리스트 객체에 저장
            authority -> {
                roleNames.add(authority.getAuthority());
            }
        ); //End of forEach
    log.info("ROLE NAMES: " + roleNames); //권한(룰)이름 리스트를 콘솔 표시

    if (roleNames.contains("ROLE_ADMIN")) { //권한(룰)목록에 ROLE_ADMIN이 포함되어 있으면
        //응답객체를 통해, 게시물 목록 페이지로 리다이렉트 시킴
        response.sendRedirect("/mypro00/myboard/detail?bno=1");
    } else {
        //권한 목록에 ROLE_ADMIN 이 없는 경우, home.jsp() 로 리다이렉트 시킴.
        response.sendRedirect("/mypro00/");
    }
}
}

```

(2) security-context.xml 스프링 빈 구성 설정파일에 설정 추가(빨간색 코드).

```

...(생략)...
<!-- AuthenticationSuccessHandler 구현클래스 빈 추가 -->
<bean id="myLoginSuccessHandler"
      class="com.spring5213.mypro00.common.security.MyLoginSuccessHandler"></bean>

<security:http >

    <security:intercept-url pattern="/" access="permitAll" />
    <security:intercept-url pattern="/myboard/list" access="permitAll" />
    <security:intercept-url pattern="/myboard/modify" access="hasRole('ROLE_MEMBER')"/>
    <security:intercept-url pattern="/myboard/register" access="hasAnyRole('ROLE_MEMBER', 'ROLE_USER')"/>
    <security:intercept-url pattern="/myboard/detail" access="hasRole('ADMIN')"/>

    <security:access-denied-handler ref="myAccessDeniedHandler" />

    <security:form-login login-page="/login"
                          login-processing-url="/login"
                          default-target-url="/myboard/list"
                          authentication-success-handler-ref="myLoginSuccessHandler" /><!-- 추가 -->
...(생략)...

```

[테스트 - 로그인 성공 시 AuthenticationSuccessHandler 인터페이스의 구현 클래스를 이용한 처리 구현]

→ 톰캣 서버 재기동

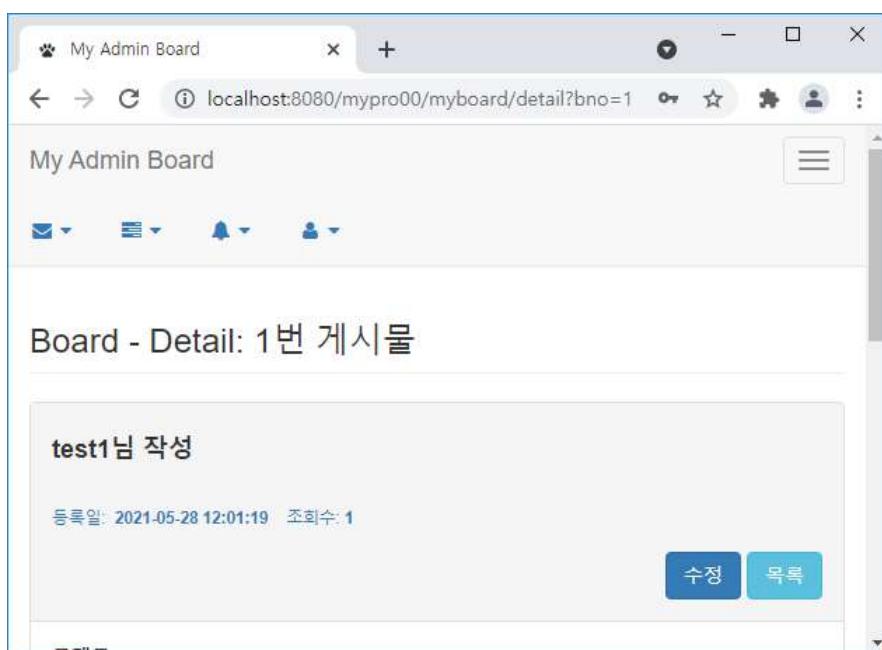
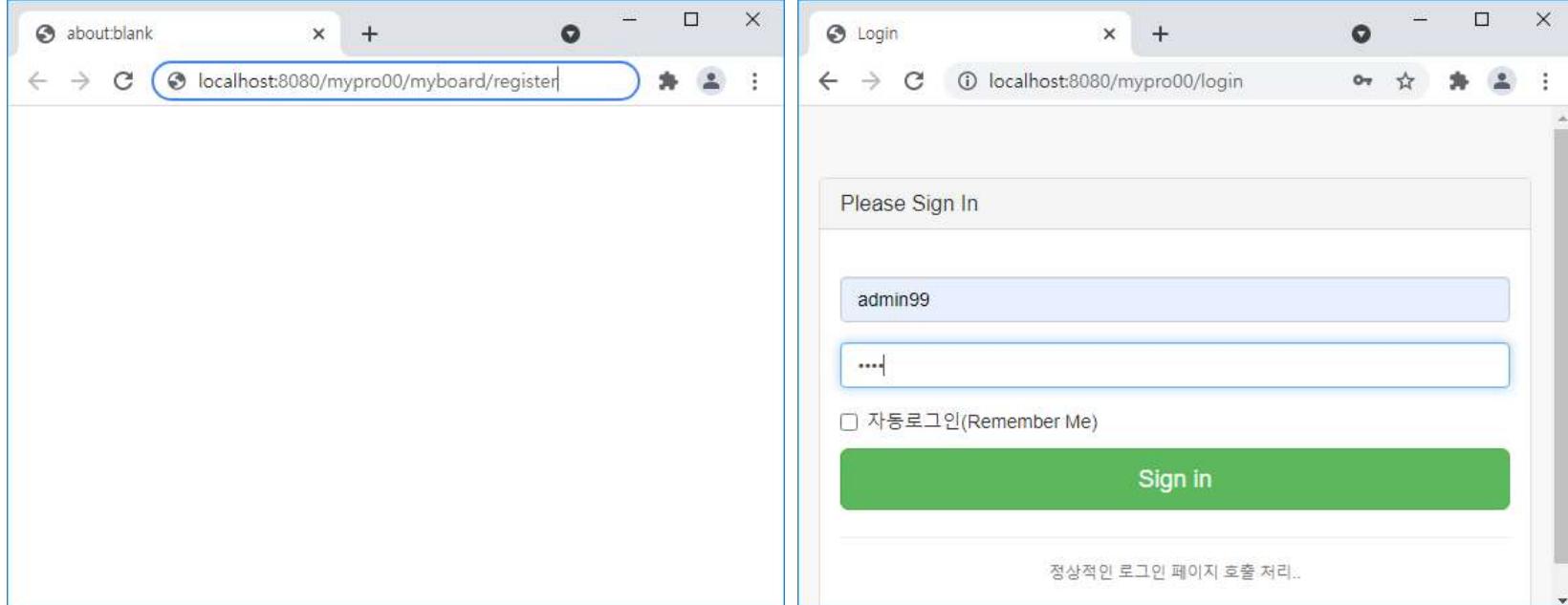
→ 웹브라우저에서 <http://localhost:8080/mypro00/myboard/register> URL 요청

→ 로그인 페이지 표시 됨, username 은 admin99, password 는 pw99 입력 후, Sing in 클릭

(ROLE\_ADMIN 룰이 설정된 admin99 계정은 해당 URL 접근 권한 없음)

→ 1 번 게시물의 상세 페이지로 이동됨

→ 처음 접근하려던 게시물 등록 URL 이 아닌 MyLoginSuccessHandler 의 설정대로 처리됨 → 계획한 바임.

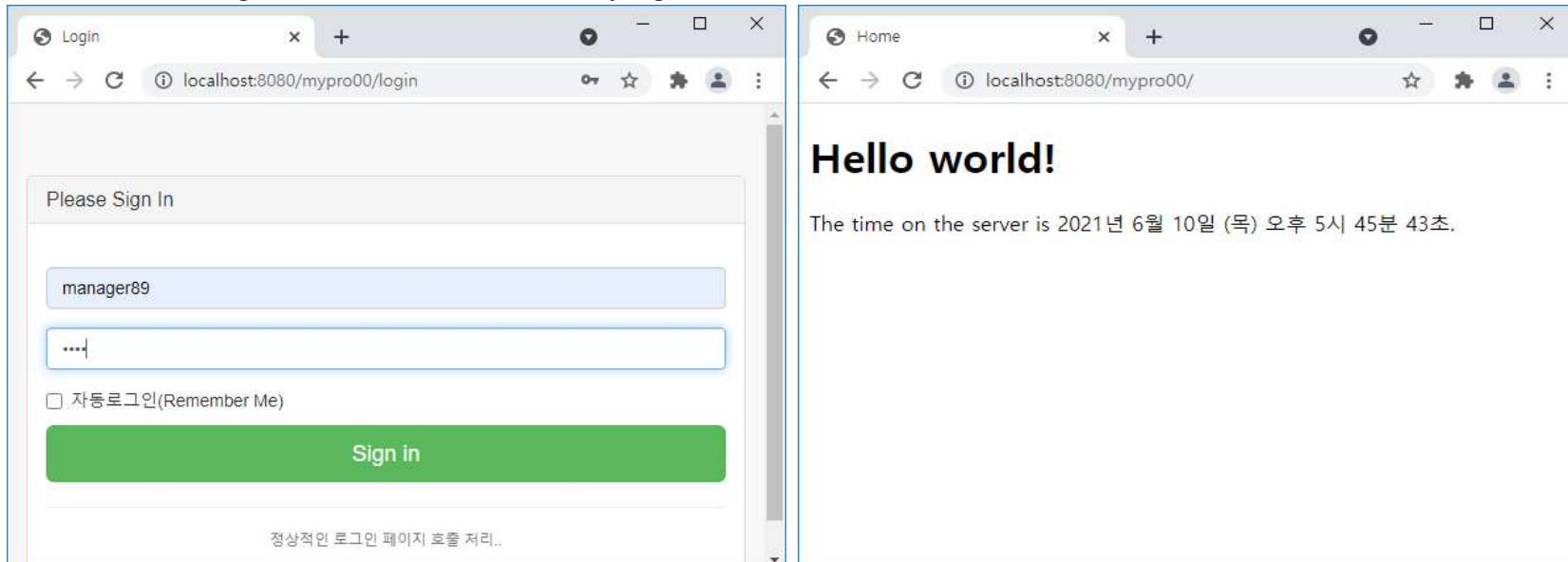


→ 웹 브라우저를 종료하고, 새로운 웹브라우저 실행 → 웹브라우저에서 <http://localhost:8080/mypro00/login> URL 로 접근

→ 로그인 페이지 표시 됨, username 은 manager89, password 는 pw89 입력 후, Sing in 클릭

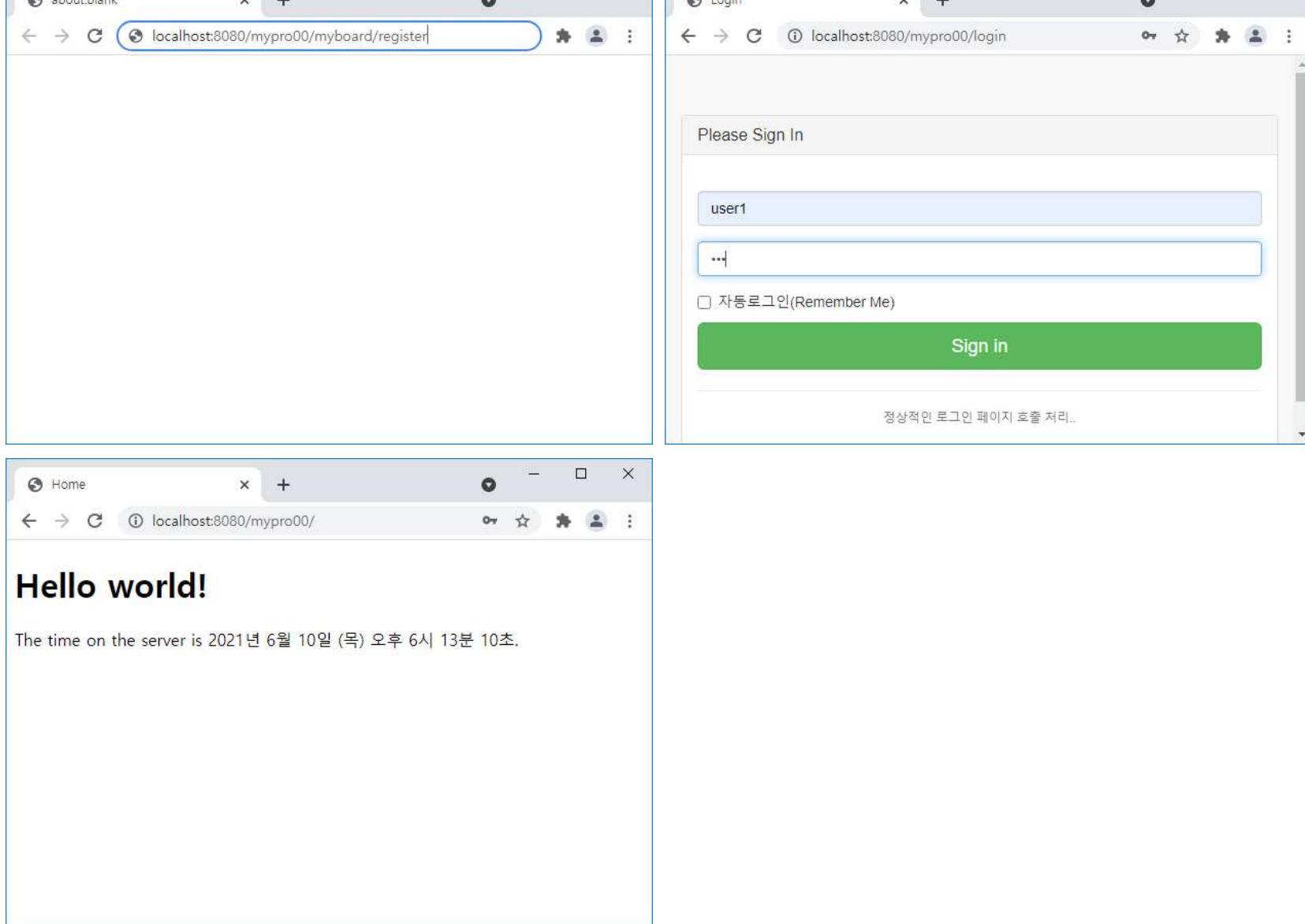
→ / 로 이동되어 home.jsp 내용이 표시됨

→ default-target-url 의 설정이 무시되고, MyLoginSuccessHandler 의 설정대로 처리됨. → 의도한 것임



- 웹 브라우저를 종료하고, 새로운 웹브라우저 실행
- 웹브라우저에서 `http://localhost:8080/mypro00/myboard/register` URL 요청 :
- 로그인 페이지 표시 됨, username 은 user1, password 는 pw1 입력 후, Sing in 클릭  
(ROLE\_ADMIN 룰이 설정되지 않은 user1 계정은 해당 URL 접근 권한 있음)
- / 로 이동되어 home.jsp 내용이 표시됨

→ 처음 접근하려던 게시물 등록 URL 이 아닌 MyLoginSuccessHandler 의 설정대로 처리됨 → 의도하지 않은 것임.



- ☞ 마지막 테스트에서, 로그인 후에는 사용자가 처음에 접근하려던 URL로 이동되는 것이 자연스럽습니다.  
그렇지만 AuthenticationSuccessHandler 인터페이스의 구현 클래스 구성에 의하여, 홈페이지로 이동되어졌습니다.
- ☞ 기본적으로 AuthenticationSuccessHandler 인터페이스를 이용하는 경우, 이전의 접근 URL을 유지하는 기능이 없기 때문입니다.
- ☞ 다음은 로그인 요청을 요구한 이전 URL을 저장하여, 로그인 후에 처음 접근하려던 이전 URL로 이동되도록 구현한 예제입니다.

### [수정 1]

→ MyLoginSuccessHandler 클래스의 onAuthenticationSuccess() 메서드를 다음처럼 수정합니다.

```
package com.spring5213.mypro00.common.security;

import java.io.IOException;
import java.util.Set;
```

```

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.web.DefaultRedirectStrategy;
import org.springframework.security.web.RedirectStrategy;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.security.web.savedrequest.HttpSessionRequestCache;
import org.springframework.security.web.savedrequest.RequestCache;
import org.springframework.security.web.savedrequest.SavedRequest;

import lombok.extern.log4j.Log4j;

@Log4j
public class MyLoginSuccessHandler implements AuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
                                       HttpServletResponse response,
                                       Authentication authentication) throws ServletException, IOException {

        log.info("로그인 성공 후: MyLoginSuccessHandler.onAuthenticationSuccess() 시작.....");
        log.info("전달된 Authentication 객체 정보: " + authentication);

        // List<String> roleNames = new ArrayList<>();//를 목록을 저장
        // 권한이름 추출
        // authentication.getAuthorities() //인증객체에 저장되는 권한들(authorities)을 가져옴
        // .forEach( //각 권한(authority)의 권한이름(롤이름)을 리스트 객체에 저장
        //     authority -> {
        //         roleNames.add(authority.getAuthority());
        //     }
        // );//End of forEach

        // 위의 주석처리된 코드(권한이름을 List 객체에 저장하는 코드)를 아래의 한 줄 코드로 대체함.
        Set<String> roleNames = AuthorityUtils.authorityListToSet(authentication.getAuthorities());
        log.info("ROLE NAMES: " + roleNames); //권한(롤)이름 리스트를 콘솔 표시

        //로그인을 요구한 요청 URL 정보를 가져 가져오는 코드 시작
        //RequestCache에 대하여 https://www.inflearn.com/questions/35556 페이지 설명 참고
        RequestCache requestCache = new HttpSessionRequestCache();

        //저장된 요청을 가져옴
        SavedRequest savedRequest = requestCache.getRequest(request, response);
        log.info("이전 요청(savedRequest): " + savedRequest);
        // [표시결과] 이전 요청(savedRequest): DefaultSavedRequest[http://localhost:8080/mypro00/myboard/register]

        //https://zgundam.tistory.com/52 참고(해당 페이지에서 RedirectStrategy 검색해야 함)
        //sendRedirect() 메서드를 response 객체가 아닌 RedirectStrategy 객체의 메소드로 사용(선택)
        RedirectStrategy redirectStrategy = new DefaultRedirectStrategy();

        if (roleNames.contains("ROLE_ADMIN")) {

            response.sendRedirect("/mypro00/myboard/detail?bno=1");
            //redirectStrategy.sendRedirect(request, response, "/myboard/detail?bno=1");

        } else {

            if (savedRequest == null) {
                //response.sendRedirect("/mypro00/");
                redirectStrategy.sendRedirect(request, response, "/"); //컨텍스트 패스가 포함되면 않됨

            } else {
                String strSavedRequest = savedRequest.getRedirectUrl();
                log.info("strSavedRequest: " + strSavedRequest);
                // [표시결과] strSavedRequest: http://localhost:8080/mypro00/myboard/register
            }
        }
    }
}

```

```

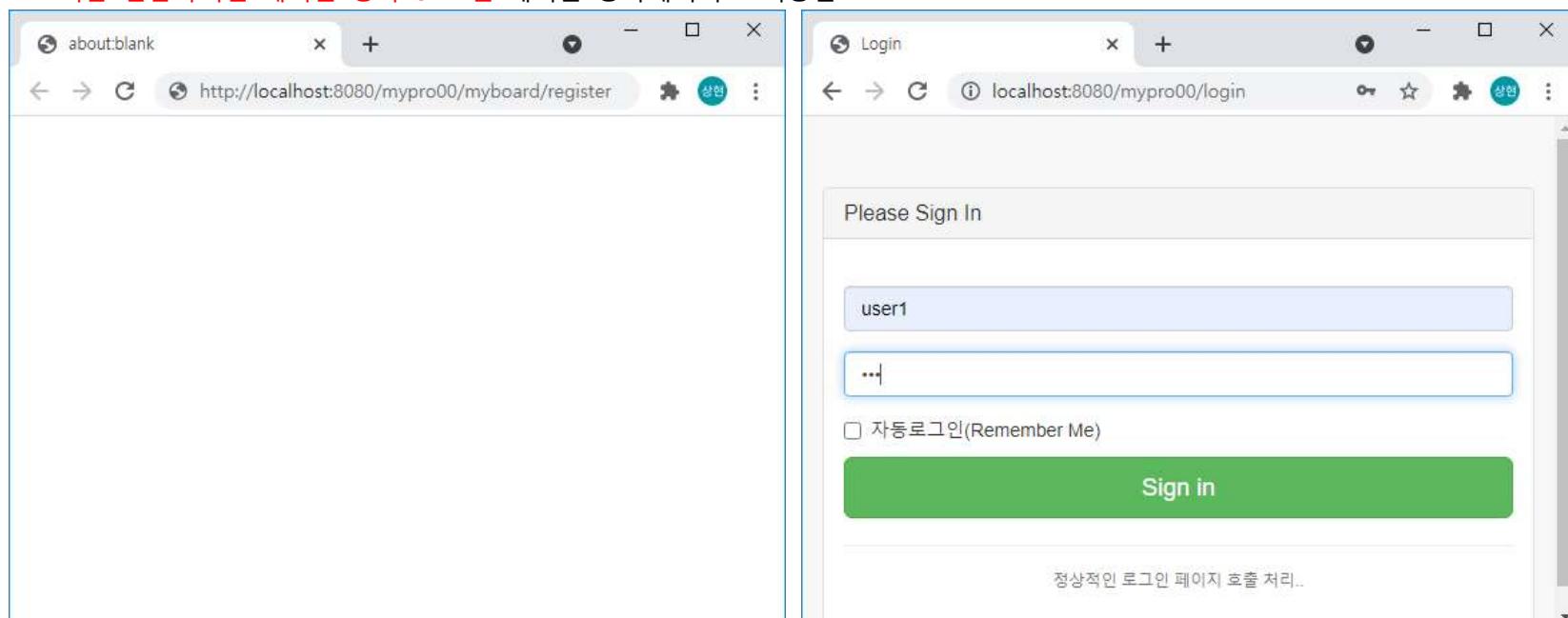
        response.sendRedirect(strSavedRequest);
        //redirectStrategy.sendRedirect(request, response, strSavedRequest);
    }
}
}

```

### [테스트 - 수정코드(1) 동작 테스트]

- 웹 브라우저를 종료하고, 새로운 웹브라우저 실행
- 웹브라우저에서 <http://localhost:8080/mypro00/myboard/register> URL 요청 :
- 로그인 페이지 표시 됨, username 은 user1, password 는 pw1 입력 후, Sing in 클릭  
(ROLE\_ADMIN 룰이 설정되지 않은 user1 계정은 해당 URL 접근 권한 있음)

→ 처음 접근하려던 게시물 등록 URL 인 게시물 등록페이지로 이동됨



[수정 2] AuthenticationSuccessHandler 구현 클래스 대신 SavedRequestAwareAuthenticationSuccessHandler 클래스를 상속받은 클래스로 코드 수정

→ MyLoginSuccessHandler 클래스를 다음처럼 수정합니다.

```
package com.spring5213.mypro00.common.security;

import java.io.IOException;
import java.util.Set;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.web.WebAttributes;
import org.springframework.security.web.authentication.SavedRequestAwareAuthenticationSuccessHandler;

import lombok.extern.log4j.Log4j;

//수정코드 - 2
@Log4j
public class MyLoginSuccessHandler extends SavedRequestAwareAuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
                                       HttpServletResponse response,
                                       Authentication authentication) throws ServletException, IOException {
        //이전 인증 오류 삭제(아래 설명 참고)
        HttpSession session = request.getSession(false);
        if (session != null) {
            session.removeAttribute(WebAttributes.AUTHENTICATION_EXCEPTION);
        }

        log.info("로그인 성공 후: MyLoginSuccessHandler.onAuthenticationSuccess() 시작.....");
        log.info("전달된 Authentication 객체 정보: " + authentication);

        //권한이름을 Set 객체에 저장
        Set<String> roleNames = AuthorityUtils.authorityListToSet(authentication.getAuthorities());
        log.info("ROLE NAMES: " + roleNames); //권한(롤)이름 리스트를 콘솔 표시

        if (roleNames.contains("ROLE_ADMIN")) {
            response.sendRedirect("/mypro00/myboard/detail?bno=1");
            //redirectStrategy.sendRedirect(request, response, "/myboard/detail?bno=1");
            return ;
        }

        super.onAuthenticationSuccess(request, response, authentication);
    }
}
```

☞ 로그인 할 때 한 번 이상 실패를 한 후에 로그인을 성공한 경우, 로그인은 성공했지만, 스프링 시큐리티에 의해 로그인 시의 오류가 세션에 저장되어 남아있게 됩니다. → 따라서 로그인 성공 핸들러에서 에러 관련 데이터를 삭제해주는 작업을 처리해주는 것이 좋습니다(아래 코드). 위의 예제에 추가되어 있습니다.

```
//이전 인증 오류 삭제
HttpSession session = request.getSession(false);
if (session != null) {
    session.removeAttribute(WebAttributes.AUTHENTICATION_EXCEPTION);
}
```

## [테스트 - 수정코드(2) 동작 테스트]

- 웹 브라우저를 종료하고, 새로운 웹브라우저 실행
- 웹브라우저에서 `http://localhost:8080/mypro00/myboard/register` URL 요청 :
- 로그인 페이지 표시 됨, username 은 user1, password 는 pw1 입력 후, Sing in 클릭  
(ROLE\_ADMIN 률이 설정되지 않은 user1 계정은 해당 URL 접근 권한 있음)
- 처음 접근하려던 게시글 등록 URL 인 게시글 등록페이지로 이동됨

The image displays three browser windows illustrating the user flow:

- Top Left Window:** Shows the registration page at `http://localhost:8080/mypro00/myboard/register`. The page has fields for '제목' (Title) and '내용' (Content).
- Top Right Window:** Shows the login page at `http://localhost:8080/mypro00/login`. It has fields for 'username' (containing 'user1') and 'password' (containing 'pw1'), a 'Remember Me' checkbox, and a green 'Sign in' button.
- Bottom Window:** Shows the 'Board - Register' page, which is the intended destination after logging in. It has fields for '제목' (Title) and '내용' (Content), with a toolbar above it.

☞ 다음은 `SavedRequestAwareAuthenticationSuccessHandler`에 대한 스프링 API 문서의 내용입니다.

`org.springframework.security.web.authentication`

`Class SavedRequestAwareAuthenticationSuccessHandler`

`java.lang.Object`

`org.springframework.security.web.authentication.AbstractAuthenticationTargetUrlRequestHandler`  
`org.springframework.security.web.authentication.SimpleUrlAuthenticationSuccessHandler`  
`org.springframework.security.web.authentication.SavedRequestAwareAuthenticationSuccessHandler`

`All Implemented Interfaces:`

`AuthenticationSuccessHandler`

```
public class SavedRequestAwareAuthenticationSuccessHandler  
extends SimpleUrlAuthenticationSuccessHandler
```

An authentication success strategy which can make use of the `DefaultSavedRequest` which may have been stored in the session by the `ExceptionTranslationFilter`. When such a request is intercepted and requires authentication, the request data is stored to record the original destination before the authentication process commenced, and to allow the request to be reconstructed when a redirect to the same URL occurs. This class is responsible for performing the redirect to the original URL if appropriate.

Following a successful authentication, it decides on the redirect destination, based on the following scenarios:

- If the `alwaysUseDefaultTargetUrl` property is set to true, the `defaultTargetUrl` will be used for the destination. Any `DefaultSavedRequest` stored in the session will be removed.
- If the `targetUrlParameter` has been set on the request, the value will be used as the destination. Any `DefaultSavedRequest` will again be removed.
- If a `SavedRequest` is found in the `RequestCache` (as set by the `ExceptionTranslationFilter` to record the original destination before the authentication process commenced), a redirect will be performed to the URL of that original destination. The `SavedRequest` object will remain cached and be picked up when the redirected request is received (See [SavedRequestAwareWrapper](#)).
- If no `SavedRequest` is found, it will delegate to the base class.

Since: 3.0

## Field Summary

Modifier and Type	Field and Description
protected org.apache.commons.logging.Log	<a href="#">logger</a>

## Constructor Summary

Constructor and Description
<a href="#">SavedRequestAwareAuthenticationSuccessHandler()</a>

## Method Summary

Modifier and Type	Method and Description
void	<a href="#">onAuthenticationSuccess</a> (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response, <a href="#">Authentication</a> authentication)
	Calls the parent class <code>handle()</code> method to forward or redirect to the target URL, and then calls <code>clearAuthenticationAttributes()</code> to remove any leftover session data.

void	<a href="#">setRequestCache</a> (RequestCache requestCache)
------	-------------------------------------------------------------

Methods inherited from class `org.springframework.security.web.authentication.SimpleUrlAuthenticationSuccessHandler`

[clearAuthenticationAttributes\(\)](#)

Methods inherited from

class `org.springframework.security.web.authentication.AbstractAuthenticationTargetUrlRequestHandler`

[determineTargetUrl](#), [determineTargetUrl](#), [getDefaultValue](#), [getRedirectStrategy](#), [getTargetUrlParameter](#), [handle](#), [isAlwaysUseDefaultTargetUrl](#), [setAlwaysUseDefaultTargetUrl](#), [setDefaultTargetUrl](#), [setRedirectStrategy](#), [setTargetUrlParameter](#), [setUseReferer](#)

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Methods inherited from interface `org.springframework.security.web.authentication.AuthenticationSuccessHandler`

[onAuthenticationSuccess](#)

## Field Detail

logger
protected final org.apache.commons.logging.Log logger

## Constructor Detail

`SavedRequestAwareAuthenticationSuccessHandler`

```
public SavedRequestAwareAuthenticationSuccessHandler()
```

#### Method Detail

##### onAuthenticationSuccess

```
public void onAuthenticationSuccess(javax.servlet.http.HttpServletRequest request,
                                    javax.servlet.http.HttpServletResponse response,
                                    Authentication authentication)
                                    throws javax.servlet.ServletException,
   java.io.IOException
```

Description copied from class: [SimpleUrlAuthenticationSuccessHandler](#)

Calls the parent class handle() method to forward or redirect to the target URL, and then calls clearAuthenticationAttributes() to remove any leftover session data.

Specified by:

[onAuthenticationSuccess](#) in interface [AuthenticationSuccessHandler](#)

Overrides:

[onAuthenticationSuccess](#) in class [SimpleUrlAuthenticationSuccessHandler](#)

Parameters:

request - the request which caused the successful authentication

response - the response

authentication - the Authentication object which was created during the authentication process.

Throws:

[javax.servlet.ServletException](#)

[java.io.IOException](#)

##### setRequestCache

```
public void setRequestCache(RequestCache requestCache)
```

☞ 앞의 실습과 유사한 방법으로, 로그인 실패 후와 로그아웃 성공 후에 추가적으로 수행해야 될 처리 과정이 있는 경우에도 스프링 시큐리티에서 각각의 경우에 제공하는 "구현 API 클래스를 상속한 클래스" 또는 "인터페이스(아래 표)의 구현 클래스"를 생성하여, 시큐리티 빈 구성 설정파일(문서에서의 `security-context.xml` 파일)에 빈으로 추가한 후, 이를 `<security:form-login>` 요소와 `<security:logout>` 요소에 등록시켜 사용되도록 구성할 수 있습니다.

☞ 다음은 로그인/로그아웃 성공 실패 시에 사용될 수 있는 스프링 시큐리티 인터페이스들과 설정방법을 요약한 내용입니다.

	구현할 수 있는 대표적인 인터페이스 이름	설정 방법
로그인 성공	<code>AuthenticationSuccessHandler</code>	<code>&lt;security:form-login&gt;</code> 요소의 <code>authentication-success-handler-ref</code> 속성에 빈 등록
로그인 실패	<code>AuthenticationFailureHandler</code>	<code>&lt;security:form-login&gt;</code> 요소의 <code>authentication-failure-handler-ref</code> 속성에 빈 등록
로그아웃 성공	<code>LogoutSuccessHandler</code>	<code>&lt;security:logout&gt;</code> 요소의 <code>success-handler-ref</code> 속성에 빈 등록
로그아웃 실패	없음	없음

☞ 스프링 인터페이스 또는 구현 API 클래스들에 대한 다양한 정보는 스프링 API 문서를 참고하세요.

## [5] 다음 섹션을 위한 준비(CSRF 토큰에 대한 이해)

- myheader.jsp 파일과 home.jsp 파일 수정.
- 로그인 성공 시 작동하는 MyLoginSuccessHandler 구성 해제
- CSRF 토큰 생성 활성화
- 스프링 시큐리티 어노테이션 사용 활성화 설정 확인 (중요!!)

(5-1) src/main/webapp/WEB-INF/views/myinclude/myheader.jsp 를 아래처럼 (빨간색 코드) 수정.

☞ 현재 프로젝트의 화면에 표시된 왼쪽 사이드 바와, 상단에 표시된 사용자 아이콘을 이용하여 로그인/로그아웃 페이지 호출이 손쉽게 처리되도록 수정합니다.

→ 웹 브라우저에서 http://localhost:8080/mypro00/myboard/list URL로 접근 후, 표시된 내용

→ 마우스 우클릭 → 목록에서 [검사] 실행 → 개발자 도구 표시됨.

→ 웹 브라우저에 표시된 게시물 목록 페이지에서 개발자 도구를 이용하여 필요없는 항목을 찾아 삭제하고, 코드를 다음처럼 수정(빨간색 코드).

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<!DOCTYPE html>
<html lang="ko"><!-- 수정 -->

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>My Admin Board</title>

    <!-- Bootstrap Core CSS -->
    <link href="${contextPath}/resources/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">

    <!-- MetisMenu CSS -->
    <link href="${contextPath}/resources/vendor/metisMenu/metisMenu.min.css" rel="stylesheet">

    <!-- DataTables CSS -->
    <%-- <link href="${contextPath}/resources/vendor/datatables-plugins/dataTables.bootstrap.css" rel="stylesheet">--%>

    <!-- DataTables Responsive CSS -->
    <%-- <link href="${contextPath}/resources/vendor/datatables-responsive/dataTables.responsive.css"
        rel="stylesheet"> --%>

    <!-- Custom CSS -->
    <link href="${contextPath}/resources/dist/css/sb-admin-2.css" rel="stylesheet">

    <!-- Custom Fonts -->
    <link href="${contextPath}/resources/vendor/font-awesome/css/font-awesome.min.css" rel="stylesheet"
        type="text/css">

    <!-- jQuery -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

    <!-- Bootstrap Core JavaScript -->
    <script src="${contextPath}/resources/vendor/bootstrap/js/bootstrap.min.js"></script>

    <!-- Metis Menu Plugin JavaScript -->
```

```

<script src="${contextPath}/resources/vendor/metisMenu/metisMenu.min.js"></script>

<!-- DataTables JavaScript --><%-->
<script src="${contextPath}/resources/vendor/datatables/js/jquery.dataTables.min.js"></script>
<script src="${contextPath}/resources/vendor/datatables-plugins/dataTables.bootstrap.min.js"></script>
<script src="${contextPath}/resources/vendor/datatables-responsive/dataTables.responsive.js"></script><%-->

<!-- Custom Theme JavaScript -->
<script src="${contextPath}/resources/dist/js/sb-admin-2.js"></script>

<!-- favicon.ico 404 Error 해결 -->
<!-- favicon 을 사용하지 않도록 설정-->
<link rel="shortcut icon" href="data:image/x-icon;" type="image/x-icon">
<!-- 다른 형식의 이미지 파일을 favicon.ico 대신 설정(다른 경로의 이미지를 이용할 수도 있음) -->
<%-- <link rel="shortcut icon" href="${contextPath}/resources/img/myfavicon.png" type="image/png"> --%>
<%-- <link rel="shortcut icon" href="${contextPath}/resources/img/favicon.ico" type="image/x-icon"> --%>

</head>
<body id="me">
    <div id="wrapper">

        <!-- Navigation -->
        <nav class="navbar navbar-default navbar-static-top" role="navigation" style="margin-bottom: 0">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="${contextPath}">My Admin Board</a><%-- href 수정 --%>
            </div><%-- /.navbar-header -->

            <ul class="nav navbar-top-links navbar-right">
                <li class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown"><%-- href 속성 삭제 --%>
                        <i class="fa fa-user fa-fw"></i> <i class="fa fa-caret-down"></i>
                    </a>
                    <ul class="dropdown-menu"><%-- 클래스 이름 중 dropdown-user 삭제 --%>
                        <li><a id="myLogin" href="${contextPath}/login">
                            <i class="fa fa-sign-in fa-fw"></i> Sign in</a><%-- a 링크 수정, Sign in 으로 수정 --%>
                        </li>
                        <li class="divider"></li>
                        <li><a id="myLogout" href="${contextPath}/logout">
                            <i class="fa fa-sign-out fa-fw"></i> Sign out</a><%-- a 링크 수정, Sign out 으로 수정 --%>
                        </li>
                    </ul><%-- /.dropdown-user -->
                </li><%-- /.dropdown -->
            </ul><%-- /.navbar-top-links -->

            <div class="navbar-default sidebar" role="navigation">
                <div class="sidebar-nav navbar-collapse">
                    <ul class="nav" id="side-menu">
                        <li>
                            <a href="${contextPath}"><i class="fa fa-home fa-fw"></i> Main Page</a>
                        </li>
                        <li>
                            <a href="${contextPath}/myboard/list"><i class="fa fa-table fa-fw"></i> User Board</a>
                        </li>
                        <li>
                            <a href="${contextPath}/myboard/register">
                                <i class="fa fa-edit fa-fw"></i> New Board Register</a>
                            </li>
                        <li>
                            <a href="${contextPath}/login"><i class="fa fa-sign-in fa-fw"></i> Sign in</a>
                        </li>
                        <li>
                            <a href="${contextPath}/logout"><i class="fa fa-sign-out fa-fw"></i> Sign out</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </div>

```

```

        </li>
    </ul>
</div><!-- /.sidebar-collapse -->
</div><!-- /.navbar-static-side -->
</nav>

```

(5-2) mypro00j/src/main/webapp/WEB-INF/views/home.jsp 수정

→ 기존 내용을 모두 주석처리 후, 아래의 내용으로 전체 교체

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<%@ include file=".myinclude/myheader.jsp" %>

<div id="page-wrapper">

    <div class="row">
        <div class="col-lg-12">
            <h3 class="page-header">Welcome, My Home World!!!</h3>
        </div>
    </div>

    <div class="row">
        <div class="col-lg-12">
            <h3 class="page-header">
                <!-- 로그인 하지 않은 경우 -principal: anonymousUser String 객체-->
                <!-- 로그인 한 경우      -principal: 로그인 사용자의 Authentication 객체 -->
            <sec:authentication property="principal" var="principal"/>

            <c:choose>
                <c:when test="\${principal eq 'anonymousUser'}">
                    <span>반갑습니다.</span>
                </c:when>

                <c:otherwise>
                    <span>\${principal.username}님, 반갑습니다.</span>
                </c:otherwise>
            </c:choose>
            </h3>
        </div>
    </div>
    <p>현재 시간은, <strong>\${serverTime}</strong>입니다 </p>

</div><%-- /.page-wrapper --%>

<%@ include file=".myinclude/myfooter.jsp" %>

<%-- 기존내용 --%><%--
<%@ page session="false" %>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html><head>
<meta charset="utf-8">

```

```

<title>Home</title>
</head>
<body>
<h1>Hello world!</h1>
<P>The time on the server is ${serverTime}.</P>
</body></html> --%

```

[테스트 - myheader.jsp 변경 사항 확인 - 브라우저 사이즈가 작을 때를 기준으로 설명함]

→ 톰캣 기동

→ 웹브라우저에서 <http://localhost:8080/mypro00> 로 접속 → home.jsp 내용 표시됨.

→ 상단 오른쪽 3 선 버튼 클릭 → 사이드 메뉴 펼쳐짐 → Sing in 클릭 → 로그인 페이지 표시

→ admin99/pw99 로 접속 → 게시물 1 번의 상세 페이지 표시(로그인 성공 시의 처리 구현 때문)

→ 상단 오른쪽 3 선 버튼 클릭 → 사이드 메뉴 펼쳐짐 → Main Page 클릭 → home.jsp 표시됨

→ 로그아웃 아이콘 클릭 → 로그아웃 페이지 표시 → Log Out 버튼 클릭 → home.jsp 표시됨

(5-3) 로그인 성공 시 작동하는 myLoginSuccessHandler 구성 해제

현재까지 구성에서의 문제점: 실습으로 확인

→ 톰캣 기동

→ 웹브라우저에서 <http://localhost:8080/mypro00> 로 접속

→ home.jsp 내용 표시됨.

→ 상단 오른쪽 3 선 버튼 클릭 → 사이드 메뉴 펼쳐짐 → New Board Register 클릭 → 로그인 페이지 표시

→ admin99/pw99 로 접속

→ 게시물 목록 화면 표시됨 (로그인 성공 시의 처리 구현 때문)

☞ 이 상황에서는 로그인 후, 게시물 등록 화면이 표시되는 것이 바람직하지만, 로그인 성공 후에 MyLoginSuccessHandler 를 이용해서 사용자의 권한에 따라서 특정 URL 로 이동하도록 구성되어 있으므로 게시물 목록 화면이 표시된 것임.

☞ 스프링 시큐리티는 기본적으로 **SavedRequestAwareAuthenticationSuccessHandler** 클래스를 이용하여 로그인 후의 표시할 URL 처리를 수행합니다. 이 클래스는 사용자가 원래 보려고 했던 페이지의 정보를 유지해서 로그인 후에 해당 원했던 페이지의 URL로 이동시켜 줍니다.

☞ 따라서, MyLoginSuccessHandler 를 이용하는 구성을 삭제 또는 주석처리 하여, SavedRequestAwareAuthenticationSuccessHandler 클래스가 사용자의 요청 URL 을 처리하도록 구성.

☞ MyLoginSuccessHandler 를 이용한 앞의 실습에서, SavedRequestAwareAuthenticationSuccessHandler 클래스를 상속받아 구현하는 실습을 수행한 경우, 기능을 해제할 필요는 없습니다.

→ security-context.xml 파일에서 AuthenticationSuccessHandler 구현 빈 과 설정을 주석처리 또는 삭제

```

...(생략)...

<!-- AuthenticationSuccessHandler 구현클래스 빈 --><!-- 아래처럼 주석처리 -->
<!--
    <bean id="myLoginSuccessHandler"
        class="org.zerock.ex00.common.security.MyLoginSuccessHandler"></bean> -->

...(생략)...

    <security:form-login  login-page="/login" login-processing-url="/login"
        default-target-url="/myboard/list" /><!--authentication-success-handler-ref 설정 삭제 -->

...(생략)...

```

#### [변경 확인]

- 톰캣 서버 재기동
- 웹브라우저에서 <http://localhost:8080/mypro00> 로 접속 → 로그아웃 → home.jsp 내용 표시됨.
- 상단 오른쪽 3 선 버튼 클릭 → 사이드 메뉴 펼쳐짐 → New Board Register 클릭
- 로그인 페이지 표시 → admin99/pw99 로 접속 → 게시물 등록 화면 표시되면, 구성해제 완료

#### (5-4) CSRF 토큰 설정(실습은 없습니다, CSRF에 대하여 이해합니다)

- ☞ 스프링 시큐리티에서 POST 방식으로 요청 처리 시에, 기본적으로 CSRF 토큰이라는 것을 이용합니다. 별도의 설정이 없다면 스프링 시큐리티가 적용된 사이트에서 모든 POST 방식을 처리시에 CSRF 토큰이 사용되는데, 이 CSRF-토큰은 'Cross-Site-Request Forgery, 사이트간 요청 위조) 공격을 방지'하기 위한 목적으로 사용됩니다.
- ☞ 스프링 시큐리티에서 POST(DELETE/PUT/PATCH) 방식의 요청 처리 시에 기본적으로 CSRF 토큰을 이용합니다.
- ☞ CSRF 토큰은 사용자에게 전해졌던 임의의 특정한 토큰값을 다시 서버로 보내어 서버에서 체크받는 방식입니다. 서버에서는 브라우저에 데이터를 전송할 때 CSRF 토큰값을 같이 전송합니다. 사용자가 POST 방식 등으로 특정한 작업을 다시 서버로 요청할 때는 브라우저로 전송된 토큰값을 다시 서버로 보내어 서버가 보관하고 있는 CSRF 토큰값과 비교됩니다. 만일 서버에서 확인된 CSRF 토큰값이 다르면 작업이 처리되지 않습니다.
- ☞ 서버에서 생성하는 토큰값은 일반적으로 난수를 생성해서 공격자가 패턴을 찾을 수 없도록 합니다.

- ☞ 다음은 크롬 웹브라우저에서 개발자도구의 Network 탭을 표시한 상태에서 `http://localhost:8080/mypro00/login` URL을 호출했을 때, 전달된 CSRF 토큰값을 Network 탭의 login 파일에 대한 Response 페이지에서 확인한 것입니다(위). Application 탭의 Cookies 항목의 JSESSIONID를 표시(아래)한 웹 브라우저의 모습 일부입니다.

The screenshot shows two instances of the Chrome DevTools interface. The top instance has the Network tab selected, displaying a timeline with several requests. The bottom instance has the Application tab selected, showing the Storage section with the Cookies table. In the Cookies table, there is one entry for 'JSESSIONID' with the value '8EA6F19A22874748C2421D3C...', domain 'localhost', path '/mypro00', and session expiration.

Name	Value	Domain	Path	Expires / ...	S..	H..	S..	S..	P..
JSESSIONID	8EA6F19A22874748C2421D3C...	localhost	/mypro00	Session	4...	✓			M..

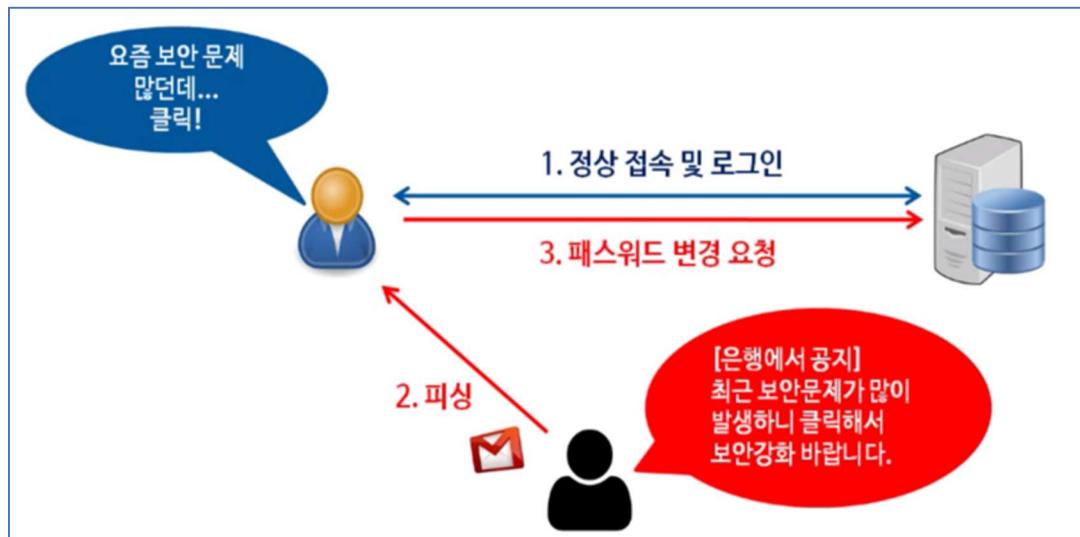
- ☞ 다음은 동일 브라우저에서 JSESSIONID 쿠키를 삭제 한 후, 다시 `http://localhost:8080/mypro00/login` URL을 호출했을 때, JSESSIONID 쿠키(위)와 Network 탭의 login 파일에 대한 Response 페이지에서 CSRF 토큰값을 확인한 것입니다(아래).

The screenshot shows two instances of the Chrome DevTools interface. The top instance has the Application tab selected, showing the Storage section with the Cookies table. The 'JSESSIONID' cookie is listed with the value '74B497973526F0B24B4A343E...'. The bottom instance has the Network tab selected, displaying a timeline with requests. The bottom portion of the screenshot shows the Network tab's response pane for the 'login' file, where the CSRF token value has changed to 'ad83df97-1037-4455-a4e1-8bf0d897b946'.

- ☞ 동일 브라우저지만 쿠키의 세션아이디로 바뀌고, CSRF-토큰값도 다른 것이 확인됩니다.
- ☞ 일반적으로 CSRF 토큰은 세션을 통해서 브라우저에 전송된 값을 서버에 보관하고, 다시 브라우저에서 전송된 CSRF 토큰값을 다시 검사하는 방식으로 처리됩니다.
- ☞ 따라서 공격자의 입장에서는 CSRF 공격을 하기 위해 필요한 변경되는 CSRF 토큰의 값을 알아야 하는데, 접속할 때 부여되는 CSRF 토큰값을 공격자가 알아낼 수 없으므로 공격자가 작성한 <form> 태그나 <img> 태그가 포함된 페이지(CSRF 토큰값이 없거나 다른 값)를 이용해서 원하는 요청을 할 수 없게 됩니다.

#### [참고] CSRF 공격이란(참고사이트: <https://oggwa.tistory.com/20>)

CSRF는 Cross Site Request Forgery의 약자로 "사이트 간 요청 위조"라고 번역됩니다. 서버에서 받아들이는 정보가 특별히 사전 조건을 검증하지 않는다는 단점을 이용하는 공격방법으로, 단순히 게시물의 조회수를 늘리는 등의 조작부터 피해자의 계정을 이용하는 다양한 공격이 가능합니다. 이해를 돋기 위해 가장 간단한 CSRF 공격을 요약하면, 해커들이 사용자들로 하여금 어떤 링크를 누르게 한 후, 링크를 누르면 사용자 모르게 어떤 특정 기능이 실행 되도록 하는 것입니다. 주로 피싱을 활용해 사용자 패스워드를 변경하는 데에 사용됩니다. 이 공격은 2008년에 1080만명의 개인정보가 유출되었던 옥션 해킹 사건에서 사용된 기법중 하나입니다.



- CSRF 공격은 다음과 같은 과정과 유사하게 진행됩니다.
1. 사용자가 정상적인 사이트 A에 접속하여 로그인 합니다. 사용자가 A 사이트에 대하여 자동로그인이 된 상태라면 로그인 상태는 항상 유지됩니다.
  2. 사용자가 사이트 A에 접속해 있는 동안, 해커가 이메일을 보내서 어떤 링크를 클릭하도록 피싱합니다. 이 그림에선 해커가 은행 직원으로 가장한 메일이나 메시지를 보낸 것입니다.
  3. 사용자가 무심코 해커가 요구한 링크를 클릭하면, 클릭한 시점에 이미 로그인 되어 있던 사이트의 패스워드를 해커가 지정한 패스워드로 변경하는 사이트 A의 요청(피싱이메일 또는 메시지에 숨겨져 있음)이 자동으로 사이트에 전송됩니다.
- 이런 식으로 사용자는 자기도 모르는 사이에 웹사이트의 패스워드가 변경되고, 이후 해커는 변경된 패스워드를 이용하여 해당 사용자의 계정으로 로그인(권한 도용) 할 수 있게됩니다.

다른 예를 하나 더 들어보겠습니다. 인터넷에 A라는 사이트가 존재한다고 가정합니다. A 사이트에는 특정 사용자의 등급을 변경하는 URI가 존재하는 것을 공격자(attacker)가 알았고, 해당 URI에는 약간의 파라미터가 필요하다는 것을 알았다고 가정합니다.

```
www.aaa.xxx/update?grade=admin&account=123
```

공격자는 A 사이트의 관리자(피해자)가 자주 방문하는 B 사이트에 <img> 태그나 <form> 태그를 이용해서 위의 URI를 추가한 게시물을 작성합니다(2008년의 경우 이메일의 링크를 통해서 A 사이트에 대한 공격이 이루어졌습니다).

```
<form action='www.aaa.xxx/update?grade=admin&account=123'>
<input type='submit' value='축 이벤트 당첨'>
</form>
혹은
<img src='www.aaa.xxx/update?grade=admin&account=123'>
```

A 사이트의 관리자(피해자)는 자신이 평상시에 방문하던 B 사이트를 방문하게 되고 공격자가 작성한 게시물을 보게됩니다. 이 때, <img> 태그 등에 사용된 URI가 호출되고 서버에서는 로그인한 관리자의 요청에 의해서 공격자는 admin 등급의 사용자로 변경됩니다.

A 사이트의 관리자(피해자)는 자신이 관리하던 A 사이트에 로그인이 되어 있는 상태라면, A 사이트의 서버 입장에서는 로그인한 사용자의 정상적인 요청으로 해석됩니다. CSRF 공격은 서버에서 받아들이는 요청을 해석하고 처리할 때, 어떤 출처에서 호출이 진행되었는지 따지지 않기 때문에 생기는 허점을 노리는 공격방식입니다. '사이트 간 위조 요청'이라고 하지만 현실적으로는 하나의 사이트 내에서도 가능합니다. CSRF는 <img> 태그 등의 URI 등을 이용할 수 있기 때문에 (사이트에서 사용자가 업로드하는 img는 별도 처리 없이 로딩시킵니다) 손쉽게 공격할 수 있는 방법이 됩니다.

CSRF 공격을 막기 위해서는 여러 방식이 존재할 수 있습니다. CSRF 공격 자체가 사용자의 요청에 대한 출처를 검사하지 않아서 생기는 허점이기 때문에 사용자에 요청에 대한 출처를 의미하는 referer 헤더를 체크하거나 REST 방식에서 사용되는 PUT, DELETE와 같은 방식을 이용하는 등의 방법을 고려(일반적인 경우에 잘 사용되지 않음)해 볼 수 있습니다.

#### ☞ 스프링 시큐리티에서 CSRF 토큰이 사용되는 과정

사용자가 웹 브라우저에서 URL 요청

- 서버에서 요청을 하여 서버에서 브라우저에 데이터 전송 시에, CSRF 토큰(난수)을 생성하여 같이 전송  
(이 때, 서버는 세션에 대한 CSRF 토큰을 보관)
- 브라우저에서 POST 방식으로 특정 작업을 서버에 요청: 이 때 처음 전송된 CSRF-토큰을 함께 전송해야 함
  - 브라우저의 POST 요청과 함께 전송된 CSRF 토큰을, 서버는, 서버가 생성해서 세션에 보관하고 있던 CSRF 토큰과 비교
  - 서버에서 보관된 CSRF 토큰과 브라우저가 보낸 토큰이 다르면 사용자의 요청 URL 작업이 처리되지 않음.

#### ☞ 스프링 시큐리티는 기본적으로 CSRF 토큰 생성이 활성화되어 있습니다.

☞ 이 후에 Post(Put/Patch/Delete) 방식으로 서버에 전송이 발생되는 JSP 페이지에는,

CSRF 토큰을 같이 전송하는 다음의 코드를 <form> 태그 영역 내에 추가하여 전송해야 오류가 발생되지 않음.

```
<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
```

☞ JSP 페이지에 %@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>가 명시된 경우, 위의 코드 대신, 간단히 <sec:csrfInput> 으로 작성할 수 있습니다.

→ mypro00/src/main/webapp/WEB-INF/views/common/myLogin.jsp 파일과 myLogout.jsp 파일을 확인하면, 이미 위의 코드가 <form> 내에 작성되어 있음을 확인할 수 있습니다.

☞ CSRF 토큰 생성이 활성화 된 경우, 스프링 시큐리티는 POST, PUT, PATCH, DELETE 방식으로 브라우저에서 서버로 데이터를 전송하는 경우에는 반드시 'X-CSRF-TOKEN'과 같은 헤더 정보와 CSRF 토큰이 전달되어야 합니다.

### (5-5) 스프링 시큐리티 어노테이션 사용 활성화 설정 확인 (중요!!)

☞ XML이나 Java 설정을 이용해서 스프링 시큐리티를 설정하고 사용하는 방식도 좋지만 매번 필요한 URL에 따라서 설정을 변경하는 일은 번거로운 일입니다.

스프링 시큐리티 역시 다른 기능들처럼 어노테이션을 이용해서 필요한 설정을 추가할 수 있습니다.

☞ 사용되는 어노테이션은 주로 @Secured 와 @PreAuthorize, @PostAuthorize 입니다.

☞ @Secured 는 스프링 시큐리티 초기부터 사용되었고, () 안에 'ROLE\_ADMIN'과 같은 문자열 혹은 문자열 배열을 이용합니다.

☞ @PreAuthorize 와 @PostAuthorize 는 스프링 시큐리티 3 버전부터 지원되었으며, ()안에 표현식을 사용할 수 있으므로 최근에는 더 많이 사용됩니다.

☞ 스프링 시큐리티 어노테이션을 사용하려면, 스프링 Dispatcher Servlet 의 컨텍스트 설정파일(servlet-context.xml 파일)에 어노테이션 사용을 활성해주는 설정을 추가해 주어야 합니다.

→ src/main/webapp/spring/appServlet/servlet-context.xml 파일을 열고 다음을 수행

→ security 네임스페이스 추가

→ 다음의 코드 추가

```
<!-- 어노테이션을 이용한 스프링 시큐리티 활성화 --> ← 추가
<security:global-method-security
    pre-post-annotations="enabled"
    secured-annotations="enabled" />

</beans:beans>
```

## [6] 프로젝트에 스프링 시큐리티 적용: 기존 게시물과 댓글 처리(추가/수정/삭제)에 대한 액세스 제어

본 섹션을 시작하기 전에, (5) 섹션에서 실습한, 로그인 성공 시 작동하는 `myLoginSuccessHandler` 구성 해제, CSRF 토큰 생성 활성화, 스프링 시큐리티 어노테이션 사용 활성화가 모두 적용되어 있어야 합니다.

### (6-1) JSP-페이지와 컨트롤러에 스프링 시큐리티 적용 개요

구현을 위해 다음의 스프링 요소들이 사용됩니다.

- `@Secured`, `@PreAuthorize`, `@PostAuthorize` 어노테이션
- Spring EL Expressions: 스프링 시큐리티의 표현식

다음은 스프링 시큐리티의 EL 표현식입니다.

Expression	Description
<code>hasRole('role')</code>	현재 로그인한 주체(principal)가 권한(역할)을 가지고 있으면 true 반환
<code>hasAnyRole('role1', 'role2')</code>	현재 로그인한 주체가 제공된 권한 중 하나라도 가지고 있으면 true 반환 (쉼표로 여러 개의 권한 목록 지정 가능)
<code>hasAuthority('authority')</code>	현재 로그인한 주체(principal)가 권한을 가지고 있으면 true 반환
<code>hasAnyAuthority('auth', 'auth')</code>	현재 로그인한 주체가 제공된 권한 중 하나라도 가지고 있으면 true 반환 (쉼표로 여러 개의 권한 목록 지정 가능)
<code>principal</code>	현재 로그인한 사용자를 나타내는 객체를 표현
<code>authentication</code>	Security Context에서 얻은 현재 Authentication 객체를 표현
<code>permitAll</code>	모든 사용자 접근 가능 (Always evaluates to true)
<code>denyAll</code>	모든 사용자 접근 불가능 (Always evaluates to false)
<code>isAnonymous()</code>	현재 로그인한 사용자가 익명 사용자(anonymous user)인 경우 true 반환
<code>isRememberMe()</code>	현재 로그인한 사용자가 자동로그인 사용자(remember-me user)인 경우 true 반환
<code>isAuthenticated()</code>	현재 로그인한 사용자가 익명이 아닌 경우 true 반환 (인증만 되어 있으면 접근 허용)
<code>isFullyAuthenticated()</code>	현재 로그인한 사용자가 익명 또는 기억하고 있는 사용자가 아닌 경우 true 반환

EL 사용에 관해 자세한 내용은 <https://docs.spring.io/spring-security/site/docs/5.3.8.RELEASE/reference/html5/#el-access> 공식 문서를 참고하세요.

@PreAuthorize, @PostAuthorize 어노테이션에서 스프링 시큐리티 빈 구성 설정파일에서 `<interceptor-url>` 요소의 `access` 속성에 스프링 EL 표현식을 사용할 수 있습니다.

어노테이션은 컨트롤러 메서드에서 사용되며, 기존의 스프링 시큐리티 설정(예, security-context.xml 파일의 설정들) 대신 사용됩니다.

JSP 파일에서는, 스프링 시큐리티 태그 라이브러리 참조를 디렉티브로 설정하면 스프링 EL 표현식을 사용할 수 있습니다.

웹 애플리케이션에서 사용자에 의한 동작은 모두 화면에서 발생되며, 화면 표시, 화면 이동, 화면에서의 기능 동작 등을 스프링 시큐리티를 통해 제어하고자 할 때, 제일 먼저 하는 것은 어떤 방식으로 구현할지를 결정해야 합니다.

다음은 대표적인 스프링 시큐리티를 적용하는 3 가지 방법입니다.

1. 화면에서의 요소에 대한 스프링 시큐리티 적용(JSP 파일에만 적용)
2. 컨트롤러에서의 스프링 시큐리티 적용
3. 방법1 + 방법2

- ☞ 방법1 은 화면에서의 동작 제어만으로 충분한 경우에 스프링 EL 표현식을 이용하여 구현합니다.  
실습에서는 로그인 사용자인 경우에 표시되어야 하는 버튼의 경우에 방법 1을 이용하여 구현합니다.
- ☞ 로그인 수행 여부 또는 필요한 권한이 있는지 없는지를 기준으로 제어해야 하는 경우에는 컨트롤러의 메서드에 스프링 시큐리티를 적용(방법1)하면 대부분 해결됩니다.  
실습에서는 게시물 등록과 댓글 등록을 방법 2를 이용하여 구현합니다.
- ☞ 방법 3은, 화면과 컨트롤러에 같이 스프링 시큐리티를 적용해야 하는 경우에 사용되며, 실습에서는 게시물과 댓글에 대한 수정/삭제-화면 이동 및 수정/삭제 처리, Ajax를 통한 데이터 요청 수행 여부, 답글 등록 및 삭제 등을 방법 3을 이용하여 구현합니다.
- ☞ 순서가 정해진 것은 아니지만 개인적으로, JSP-페이지에서의 구현을 먼저 수행하고, 컨트롤러에서의 구현을 하는 것이 작업이 보다 용이합니다. 특히, 권한 확인을 위해 로그인을 필요로 하는 화면에 대하여 스프링 시큐리티 구현을 먼저 수행하는 것이 작업이 편리합니다.
- ☞ 스프링 시큐리티의 구성은 JSP-페이지의 동작이나 컨트롤러의 동작에 구현하려는 경우, 작업을 용이하게 하려면, 사용자 요청 URL에 대하여 설정된 요구 권한을 해제하고, 구현을 하면서 추가하는 것이 좋음.  
참고: 실습에서 사용자 계정에게 설정된 권한: ROLE\_USER, ROLE\_MANAGER, ROLE\_ADMIN 3가지입니다.

- security-context.xml 파일의 <interceptor-url> 요소를 다음처럼 정리합니다.
- Tomcat 중지 → security-context.xml을 다음처럼 수정 → 모든 URL에 대한 승인(권한 확인) 해제됩니다.

```
<security:http>
    <security:intercept-url pattern="/**" access="permitAll" /><!-- 구성완료 시에 이 제어설정은 삭제 또는 주석처리 -->
<!--
    <security:intercept-url pattern="/" access="permitAll" />
    <security:intercept-url pattern="/myboard/list" access="permitAll" />
    <security:intercept-url pattern="/myboard/modify" access="hasRole('ROLE_MEMBER')"/>
    <security:intercept-url pattern="/myboard/register" access="hasAnyRole('ROLE_MEMBER', 'ROLE_USER')"/>
    <security:intercept-url pattern="/myboard/detail" access="hasRole('ADMIN')"/> -->
```

## (6-2) 게시물 목록 JSP에서의 스프링 시큐리티 처리.

- ☞ 로그인 상관없이(권한도 필요 없음) 모두 접근 가능하도록 구현: /myboard/list 요청 URL에 대하여 permitAll 설정

- security-context.xml 파일에 /myboard/list URL에 permitAll로 설정된 <interceptor-url> 요소를 추가합니다.

```
<security:http>
    <security:intercept-url pattern="/**" access="permitAll" />
    <security:intercept-url pattern="/myboard/list" access="permitAll" />
```

### (6-3) 게시물 등록 JSP에서의 스프링 시큐리티 처리

☞ 게시물 등록 화면 호출부터 게시물 등록은, 권한은 상관 없고, 로그인 유무만 확인합니다.

즉, 로그인 사용자만 게시물 등록이 가능, 로그인 하지 않은 경우, 로그인 페이지를 거쳐서 등록 화면이 표시되도록 구현합니다.

☞ 게시물 등록 화면이 브라우저에 표시될 때, 작성자 입력란에는 자동으로 로그인 사용자의 계정이름이  
읽기전용으로 표시되도록 구현합니다.

#### (6-3-1) 컨트롤러의 게시물 등록 페이지 호출 및 등록 처리 메서드에 로그인 유무 확인 구현

→ src/main/java/com.spring5213.mypro00.controller.MyBoardController 의

showBoardRegisterPage() 와 registerNewBoard() 메서드에 @PreAuthorize("isAuthenticated()") 어노테이션 추가

```
import org.springframework.security.access.prepost.PreAuthorize; //Ctrl + Shift + O 자동완성
```

```
//게시물 등록 - 페이지 호출
@PreAuthorize("isAuthenticated()") //추가
@GetMapping("/register")
public void showBoardRegisterPage() {
    log.info("컨트롤러 - 게시물 등록 페이지 호출");
}

//게시물 등록 - 처리
@PreAuthorize("isAuthenticated()") //추가
@PostMapping("/register")
public String registerNewBoard(MyBoardVO myBoard,
        RedirectAttributes redirectAttr) {
    log.info("컨트롤러:게시물등록(전달된vo 내용): " + myBoard.toString());

    System.out.println("===== attachFileInfo =====");
    if (myBoard.getAttachFileList() != null) {
        myBoard.getAttachFileList()
            .forEach(attachFile -> System.out.println("첨부 파일 정보:" + attachFile.toString()));
    }
    System.out.println("===== ===== ===== =====");

    long bno = myBoardService.registerBoard(myBoard);
    log.info("등록된 개시물의 bno: " + bno);

    redirectAttr.addFlashAttribute("result", bno);

    return "redirect:/myboard/list"; //리다이렉트 방식으로 URL 호출
}
```

#### (6-3-2) 게시물 등록 JSP 페이지에 스프링 시큐리티 적용

☞ 스프링 시큐리티 커스텀 태그와 표현식 사용하여 게시물 등록 화면의 작성자 입력란에 로그인 사용자가 읽기전용으로 표시합니다.

→ src/main/webapp/WEB-INF/views/myboard/register.jsp 파일을 코드 작성부에 오픈 후, 다음의 빨간색 코드를 수정합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %><%-- 시큐리티: 추가 --%>
```

```

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<%@ include file="../myinclude/myheader.jsp" %>

...(생략)...

<div id="page-wrapper">
  <div class="row">
    <div class="col-lg-12">
      <h1 class="page-header">Board - Register</h1>
    </div><!-- /.col-lg-12 -->
  </div><!-- /.row -->
  <div class="row">
    <div class="col-lg-12">
      <div class="panel panel-default">
        <div class="panel-heading"><h4>게시글 등록</h4></div><%-- /.panel-heading --%>
        <div class="panel-body">

<form role="form" action="${contextPath}/myboard/register" method="post" >

...(생략)...

<div class="form-group">
  <label>작성자</label>
  <input class="form-control" name="bwriter"
        value='<sec:authentication property="principal.username"/>' readonly="readonly"><%-- 추가 --%>
</div>

...(생략)...

<button type="submit" class="btn btn-primary">등록</button>
<button type="button" class="btn btn-warning" data-oper="list" id="toList"
        onClick="alert('게시물 등록이 취소되었습니다..');location.href='${contextPath}/myboard/list'">취소
</button><%-- 버튼에 id="toList" 속성 추가, onClick 속성에 alert() 추가 --%>
<sec:csrfInput /><%-- 추가, spring security taglib 가 설정된 경우 사용됨 --%>
</form>

...(생략)...

```

[참고] 컨트롤러를 인터페이스-구현 클래스 형태로 구성한 경우, 아래의 오류가 발생될 수 있습니다.

The mapped handler method class '**com.spring5213.mypro00.controllerImpl**' is not an instance of the actual controller bean class 'com.sun.proxy.\$Proxy45'. If the controller requires proxying (e.g. due to @Transactional), please use class-based proxying.

[해결법] **@Scope(proxyMode = ScopedProxyMode.TARGET\_CLASS)** 어노테이션을 구현 클래스에 선언

```

@Controller
@Scope(proxyMode = ScopedProxyMode.TARGET_CLASS)
@RequestMapping("/myboard")
public class MyBoardControllerImpl implements MyBoardController {
  ...

```

## [테스트 - 게시물 등록에 대한 스프링 시큐리티 적용 테스트]

☞ (주의) 앞에서 사용한 브라우저를 계속 사용하는 경우, 로그아웃을 한 후에 테스트 수행.

→ 웹브라우저에서 <http://localhost:8080/mypro00> URL로 접속 → home.jsp 내용 표시됨.

→ 상단 오른쪽 3선 버튼 클릭 → 사이드 메뉴 펼쳐짐

→ New Board Register 클릭 → 로그인 페이지 표시됨

→ username과 password 입력란에 각각 user1과 pw1을 입력 후, Sign in 클릭

→ 게시물 등록 화면으로 이동되는지, 작성자 입력란에 user1이 읽기전용으로 표시되는지 확인

The left screenshot shows the 'My Admin Board' homepage with a welcome message and a timestamp. The right screenshot shows the 'New Board Register' page, which includes a sidebar menu and a timestamp at the bottom of the page.

Please Sign In

user1  
...  
 자동로그인(Remember Me)  
**Sign in**

정상적인 로그인 페이지 호출 처리..

Board - Register

제목  
시큐리티 화면 글 등록 테스트1

내용  
작성자 이름 로그인 ID로 자동 입력 확인 완료

작성자  
user1

**등록** **취소**

→ 그림처럼 입력 후, 등록 클릭 → 게시물 등록 여부 확인

The figure consists of three screenshots of a web application titled "My Admin Board".

- Screenshot 1: Board - Register**  
A form for posting a new article. It includes fields for the title ("제목") containing "시큐리티 화면 글 등록 테스트1", the content ("내용") containing "작성자 이름 로그인 ID로 자동 입력 확인 완료", the author ("작성자") set to "user1", and two buttons at the bottom: "등록" (Register) and "취소" (Cancel). The "등록" button is highlighted.
- Screenshot 2: 처리결과**  
A confirmation message stating "게시글 229403 번이 등록되었습니다." (Article 229403 was registered successfully.) with an "확인" (Confirm) button.
- Screenshot 3: 게시글 목록**  
A list of articles with the following data:

번호	제목	작성자	작성일	수정일	조회수
229403	시큐리티 화면 글 등록 테스트1 [댓글수: 0]	user1	2021/06/13	2021/06/13 11:36:04	0
229402	시큐리티 화면 글 등록 테스트1 [댓글수: 0]	user1	2021/06/12	2021/06/12 12:53:52	3
229401	화면 새글등록 테스트1 [댓글수: 0]	user1	2021/06/12	2021/06/12 12:50:33	0
229387	첨부파일 게시물 등록1 [댓글수: 0]	test	2021/06/06	2021/06/06 22:13:12	8
229385	첨부파일 게시물 등록1 [댓글수: 0]	test	2021/06/06	2021/06/06 14:14:25	16
229384	화면 새글등록 테스트1 [댓글수: 0]	test	2021/06/06	2021/06/06 14:04:40	2
229383	화면 새글등록 테스트2 [댓글수: 0]	test	2021/06/06	2021/06/06 13:24:21	4

#### (6-4) 게시물 상세 JSP에서의 스프링 시큐리티 처리(detail.jsp)

▣ 다음의 요구사항들이 구현되어야 합니다.

- 게시물 상세 요청 URL은 권한 및 로그인에 상관없이 액세스 가능해야 함

- 화면 표시 요소들에 대한 요구사항

▶ 게시물 수정 버튼은 로그인한 작성자에게만 표시되어야 함

→ 로그인 하지 않았거나, 로그인 했지만 게시물 작성자가 아니면 수정 버튼은 표시되지 않음

▶ 댓글작성 버튼과 댓글입력란은 로그인 했을 때만 표시

▶ 답글작성 버튼은 로그인 했을 때만 표시

- ▶ 댓글/답글의 글내용은 로그인한 작성자일 때만 클릭되어야 함
  - 로그인 되지 않은 경우와 작성자가 아닌 경우의 안내 메시지가 각각 표시되어야 함
- ▶ 댓글/답글 작성 시, 로그인 사용자의 ID가 작성자 표시란에 읽기전용으로 표시되어야 함.
- 댓글/답글의 등록 및 수정/삭제 요청은 Ajax를 통해 post 방식으로 이루어지므로, CSRF 토큰값이 같이 전달되어야 함.
- 댓글 등록은 로그인 했을 때만 가능해야 함.
- 댓글의 수정/삭제는 댓글 작성자만 가능해야 함.

☞ 요구사항들은 details.jsp 파일과 MyReplyController.java 파일에 구현합니다.

#### (6-4-1) 게시물 상세화면 접근 제어

☞ 로그인 상관없이(권한도 필요 없음) 모두 접근 가능하도록 구성: /myboard/detail 요청 URL에 대하여 permitAll 설정

→ src/main/webapp/WEB-INF/spring/security-context.xml 파일에

/myboard/list URL에 permitAll로 설정된 <security:interceptor-url> 요소를 추가합니다.

```
<security:http>
  <security:intercept-url pattern="/**" access="permitAll" />
  <security:intercept-url pattern="/myboard/list" access="permitAll" />
  <security:intercept-url pattern="/myboard/detail" access="permitAll" /><!-- 추가 -->
```

#### (6-4-2) 스프링 시큐리티 표현식을 사용하기 위해 태그 라이브러리 참조 설정

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 파일에 빨간색 코드 추가

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %><!-- 추가 --%>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<%@ include file="../myinclude/myheader.jsp" %>
```

☞ detail.jsp는 POST 요청 form이 없으므로, csrf 토큰 입력 태그 처리를 할 필요가 없습니다.

단, 댓글처리와 관련되어서는 ajax를 통한 post 요청이 발생되므로, ajax를 호출하는 jQuery 코드에 csrf 토큰 관련 코드를 뒤에서 추가합니다.

(6-4-3) 게시물 [수정] 버튼 표시: 로그인 한 작성자일 경우에만 표시(JSP에서만 구현)

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 파일에 빨간색 코드 추가

```
<div class="col-md-6" style="height: 45px; padding-top:6px;">
    <div class="button-group pull-right">

        <%-- 수정 버튼을 찾아서 다음의 코드를 추가: 수정버튼은 로그인 한 작성자일 경우에만 표시 --%>
        <sec:authorize access="isAuthenticated()">
            <sec:authentication property="principal" var="principal"/>
            <c:if test="${principal.username eq board.bwriter}">
                <button type="button" id="btnToModify" data-oper="modify" class="btn btn-primary">
                    <span>수정</span>
                </button>
            </c:if>
        </sec:authorize>

        <button type="button" id="btnToList" data-oper="list" class="btn btn-info"><span>목록</span></button>
    </div>
</div>
```

☞ <sec:authentication> 태그는 <sec:authorize>와 </sec:authorize> 사이에서 사용해야 합니다.

☞ <sec:authorize access="isAuthenticated()"> 와 </sec:authorize> 사이의 코드가 로그인 된 경우에만 처리됩니다.

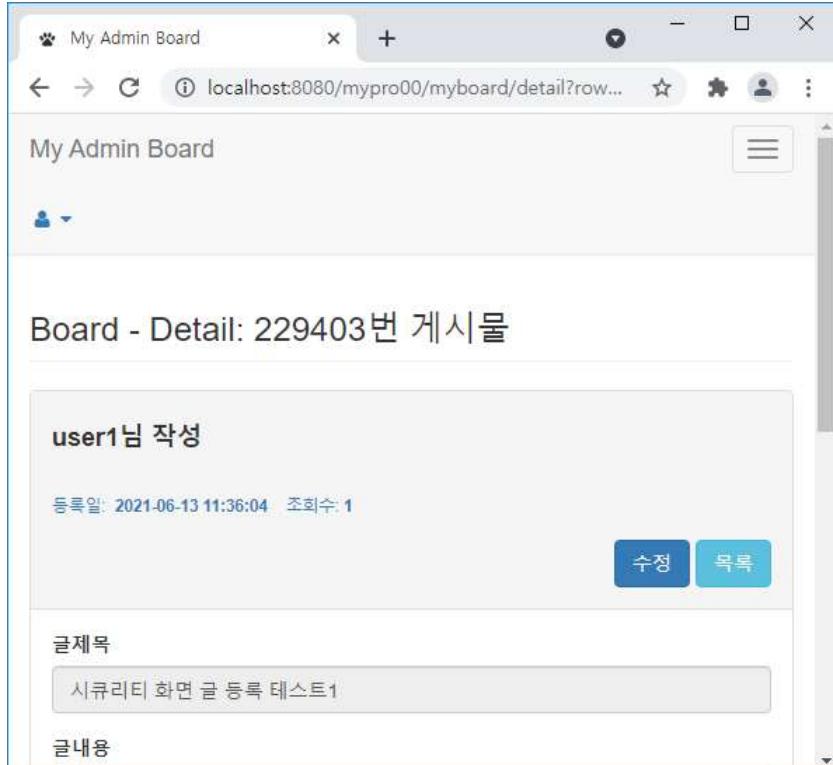
☞ <sec:authentication property="principal" var="principal"/> : Authentication 객체의 반환정보인 principal을 principal 변수를 통해서 사용합니다.

(권장) var 속성에 설정하는 변수이름은 principal로 설정해야 동작이 항상 잘 작동합니다.

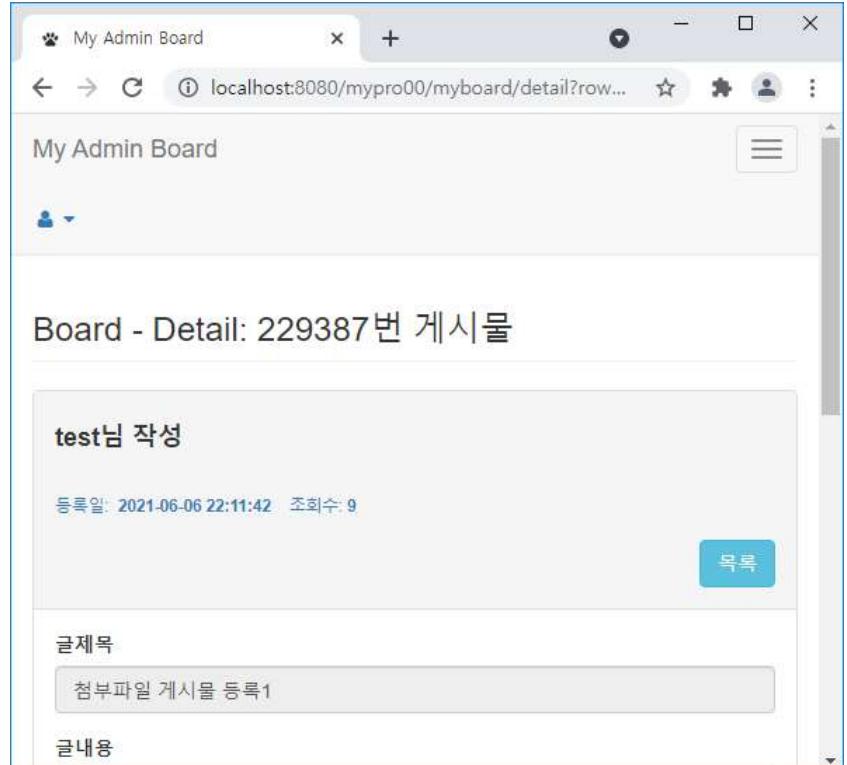
☞ <c:if test="\${principal.username eq board.bwriter}">: 로그인ID와 작성자가 동일할 경우를 확인합니다..

→ 다음은 브라우저에서 user1로 로그인하여 테스트 한 화면캡쳐입니다.

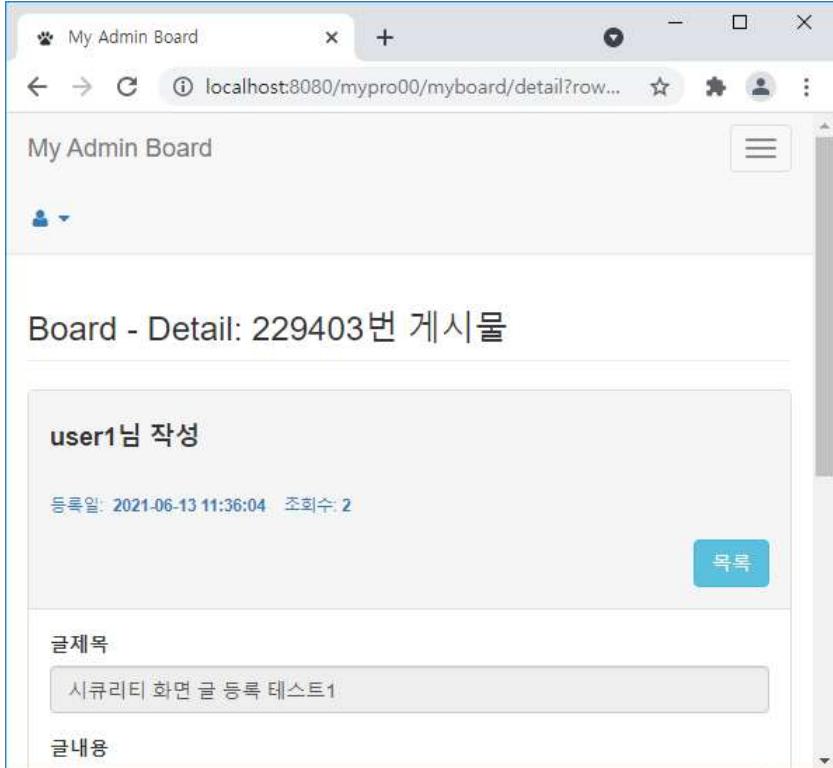
[user1이 작성한 게시물]



[user1이 작성하지 않은 게시물]



[로그인 하지 않은 경우]



(6-4-4) [댓글작성] 버튼과 댓글입력란 표시: 로그인 사용자일 경우에만 표시(JSP에서만 구현)

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 파일에 빨간색 코드 추가

...(생략)...

```
<div class="panel-heading">
  <p style="margin-bottom: 0px; font-size: 16px;">
    <strong style="padding-top: 2px;">
      <span>댓글&nbsp;<c:out value="${board.breplyCnt}"/>개</span>
      <span>&nbsp;</span>

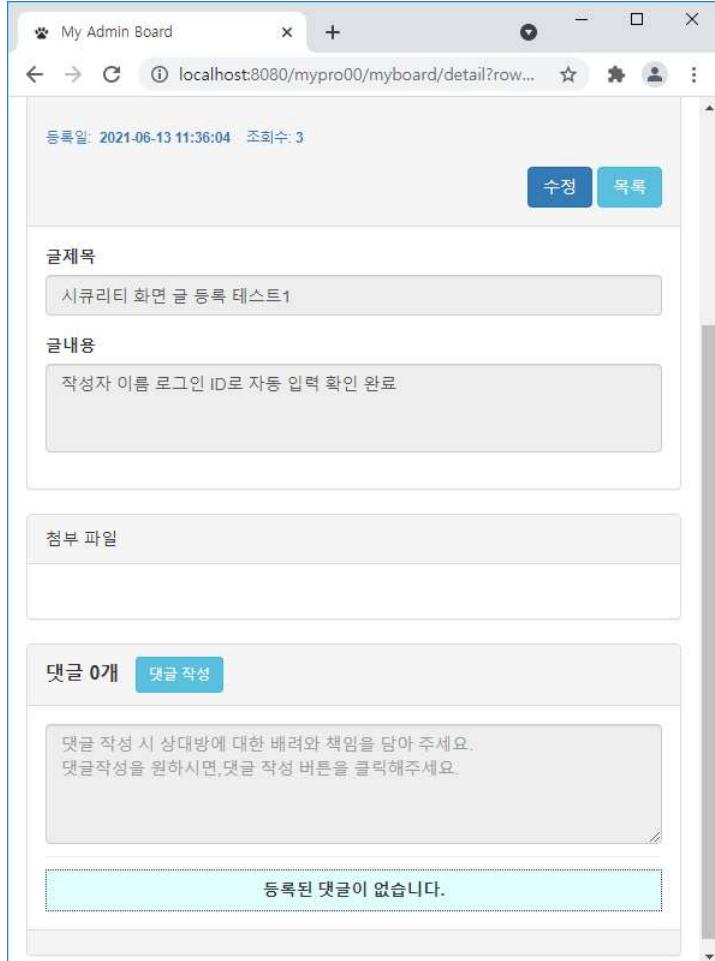
      <sec:authorize access="isAuthenticated()" ><%-- 추가: 댓글 작성 버튼, 로그인 사용자일 때만 표시 --%>
        <button type="button" id="btnChgCmtReg" class="btn btn-info btn-sm">댓글 작성</button>
      </sec:authorize>

      <button type="button" id="btnRegCmt" class="btn btn-warning btn-sm" style="display:none">댓글 등록</button>
      <button type="button" id="btnCancelRegCmt" class="btn btn-warning btn-sm" style="display:none">취소</button>
    </strong>
  </p>
</div> <%-- /.panel-heading --%>

<div class="panel-body">
<%-- 댓글 입력창 시작 --%>
<sec:authorize access="isAuthenticated()" ><%-- 추가: 댓글 입력창, 로그인 사용자일 때만 표시 -->
  <div class="form-group" style="margin-bottom: 5px;">
    <textarea class="form-control txtBoxCmt" name="rcontent"
              placeholder="댓글작성을 원하시면,&#10;댓글 작성 버튼을 클릭해주세요."
              readonly="readonly">
    </textarea>
  </div><%-- 댓글 입력창 div 끝 --%>
  <hr style="margin-top: 10px; margin-bottom: 10px;">
</sec:authorize>
...(생략)...
```

→ 다음은 브라우저에서 테스트 한 화면캡쳐입니다.

[user1 로 로그인 한 경우]



[로그인 하지 않은 경우]



(6-4-5) [답글 작성] 버튼 표시: 로그인 사용자일 경우에만 표시(JSP 에서만 구현)

☞ 답글 작성 버튼은, 댓글 목록을 ajax로 받아와서 showCmtList(page) JavaScript 함수를 통해 생성됩니다.

☞ 따라서, 스프링 시큐리티의 코드는 Javascript영역에 추가됩니다. <sec:authorize access="isAuthenticated()"> 태그를 추가합니다.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 파일에 빨간색 코드 추가

...(생략)...

```
<%--댓글 목록표시--%><%--댓글 목록표시 함수, ajax 클로저 모듈 함수를 호출함 --%>
function showCmtList(page){

    myCommentClsr.getCmtList(
        {bno: bnoValue, page: page || 1 },
        function(replyPagingCreator) { //ajax에서 실행할 callback 함수

    ...  
...(생략)
        if(replyPagingCreator.replyList[i].lvl > 1){
            str += '&ampnbsp 답글';
        }
        str += '' + '

';


```

```

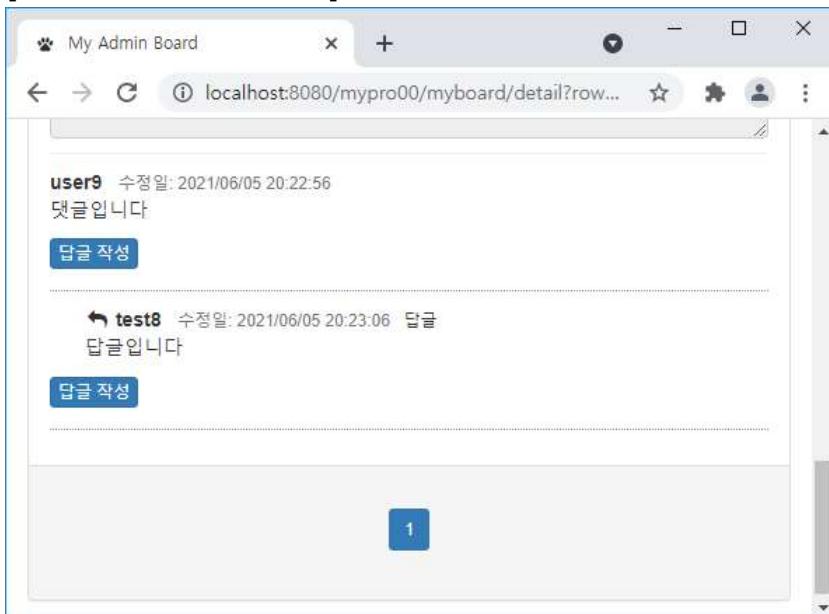
<%-- 추가: 답글추가 버튼, 로그인 사용자일 때만 표시 --%>
<sec:authorize access="isAuthenticated()" >
str +='      <div class="btmsReply" style="margin-bottom:10px">
+       <button type="button" style="display:inline-block" ' +
+           class="btn btn-primary btn-xs btnChgReplyReg">' +
+               <span>답글 작성</span></button>
+       </div>';
    </sec:authorize>

    str +='      </div>
+     '</li>';
} //for-End

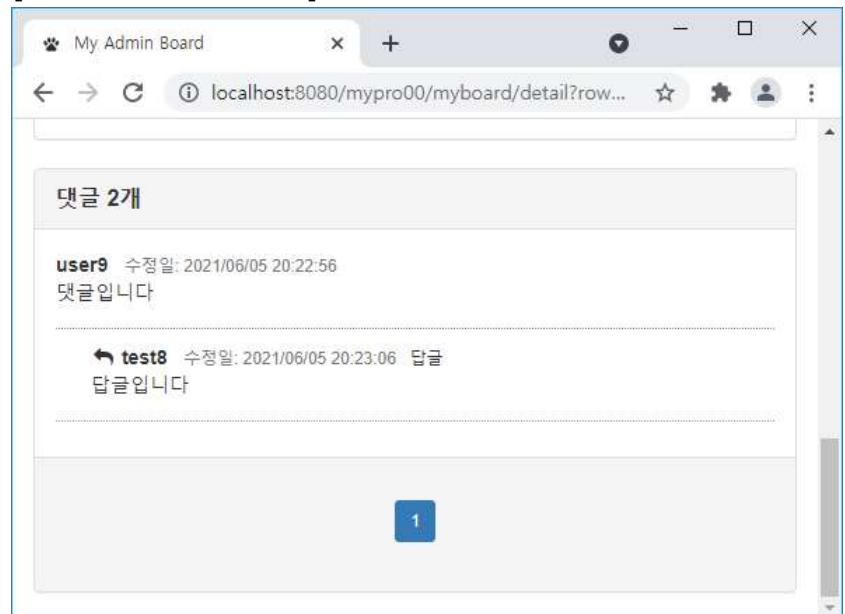
```

→ 다음은 브라우저에서 테스트 한 화면캡쳐입니다.

[user1로 로그인 한 경우]



[로그인 하지 않은 경우]



#### (6-4-6) 댓글/답글의 글내용 클릭 이벤트에 대한 제어 구현(JSP 에서만 구현, 실제 서버로의 전송은 아님)

☞ 로그인 사용자가 글 작성자일 경우에만 수정/삭제 버튼과 입력창이 표시

☞ 로그인 되지 않은 사용자일 경우: "로그인 후 수정이 가능합니다" 메시지 표시

☞ 로그인 했지만, 글 작성자가 아닌 경우: "작성자만 수정 가능합니다" 메시지 표시

☞ 2 가지 변수가 필요: 글작성자(rwritter)와 로그인 사용자 명(loginUser)

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 파일의 showCmtList() 함수에 빨간색 코드 추가

... (생략) ...

<%--로그인 사용자명 변수에 저장하는 코드는 댓글/답글 자바스크립트 코드 시작 부분으로 이동시킵니다.--%>

```

var loginUser = "";
<sec:authorize access="isAuthenticated()" >
    loginUser = '<sec:authentication property="principal.username"/>';<%--로그인 사용자명 변수에 저장--%>
</sec:authorize>

```

<%--댓글-답글 수정/삭제 화면 요소 표시: p 태그 클릭 이벤트 --%>

```

$(".chat").on("click", ".commentLi p", function(){

```

```

chgBeforeCmtBtn();<%--댓글 등록 상태 초기화--%>
chgBeforeReplyBtn();<%--다른 답글 등록 상태 초기화--%>
chgBeforeCmtRepBtns(); <%--다른 답글/댓글 수정 상태 초기화--%>

<%--로그인 하지 않은 경우--%>
if(!loginUser){
    alert("로그인 후, 수정이 가능합니다.");
    return ;
}

<%--작성자 변수에 저장--%>
var rwriter = $(this).data("rwriter");
console.log("rwriter: " + rwriter);
console.log("loginUser: " + loginUser);

<%--로그인 계정과 작성자가 다른 경우--%>
if(rwriter != loginUser){
    alert("작성자만 수정 가능합니다");
    return ;
}

$(this).parents("li").find(".btnChgReplyReg").attr("style", "display:none");

var rcontent = $(this).text();
console.log("선택된 댓글내용: " + rcontent);

var strTxtBoxReply =
    "<textarea class='form-control txtBoxMod' name='rcontent' style='margin-bottom:10px;'"
    + "placeholder='답글작성을 원하시면, 답글 작성 버튼을 클릭해주세요.'"
    + "></textarea>"
    + "<button type='button' class='btn btn-warning btn-sm btnModCmt'>수정</button> "
    + "<button type='button' class='btn btn-danger btn-sm btnDelCmt'>삭제</button> "
    + "<button type='button' class='btn btn-info btn-sm btnCancelCmt' style='margin-left: 4px;'>취소</button>";

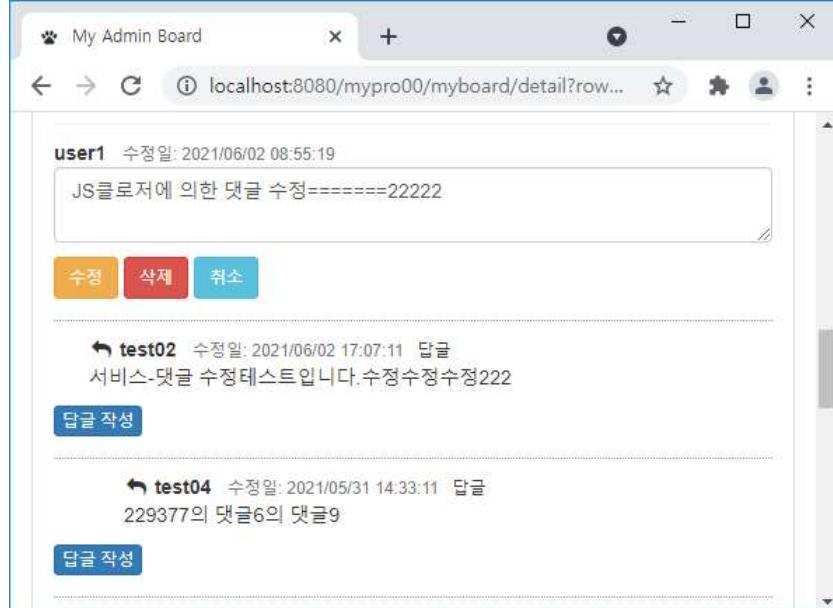
$(this).after(strTxtBoxReply);
$(".txtBoxMod").val(rcontent);
$(this).attr("style", "display:none");

});

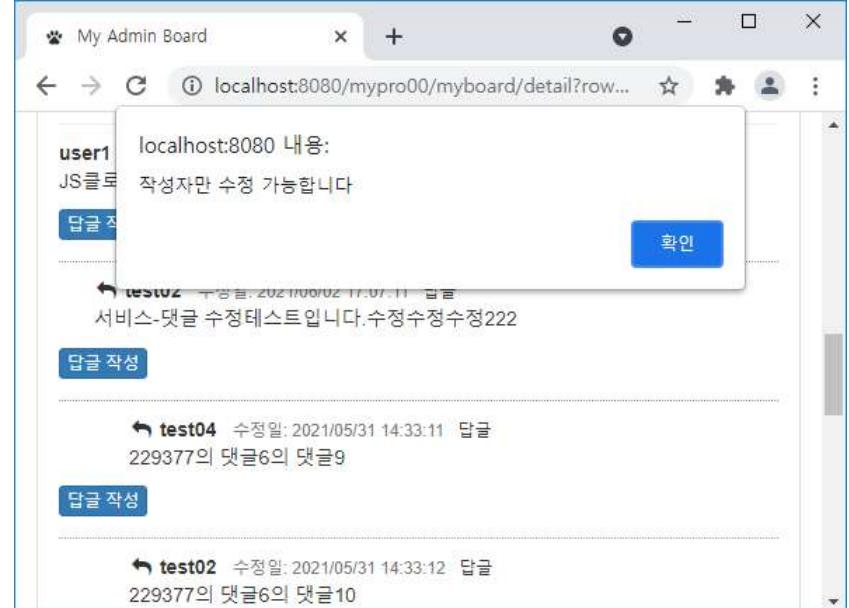
```

☞ 다음은 브라우저에서 테스트 한 화면캡쳐입니다.

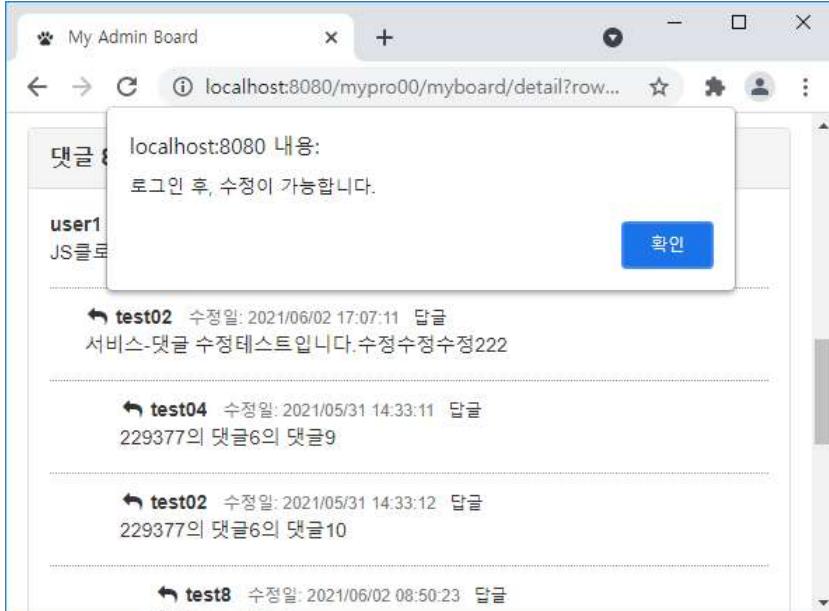
[user1 로 로그인하여 자신의 글을 클릭]



[user1 로 로그인하여 test02 의 글을 클릭]



### [로그인 하지 않은 경우]



### (6-4-7) Ajax 를 통한 댓글/답글의 등록 및 수정 시, CSRF 토큰값이 같이 전달되도록 구현(JSP 에서만 구현)

- ☞ 댓글 등록, 답글 등록, 댓글/답글의 수정 및 삭제 버튼 클릭에 의하여 Ajax로 서버에 데이터와 요청을 보낼 때 게시물 상세 페이지에 전달된 CSRF 토큰을 같이 보내야만 버튼 클릭의 작업이 서버에서 정상적으로 처리됩니다.
- ☞ 이 구현을 ajax 함수에 포함시킬 수 있지만, 다음처럼 구성하면, 게시물 상세 HTML에서 발생되는 모든 Ajax 요청에 대하여 csrf 토큰값이 요청 헤더에 자동으로 포함됩니다.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 파일의 댓글처리 자바 스크립트의 제일 위에 아래의 빨간색 코드를 추가합니다.

...(생략)...

```
<%-- 댓글처리 자바 스크립트 시작//-----%>
<script type="text/javascript" src="${contextPath}/resources/js/mycomment.js"></script>
<script>

<%--로그인 사용자명 변수에 저장하는 코드는 댓글/답글 자바스크립트 코드 시작 부분으로 이동시킵니다. 앞 실습 --%
var loginUser = "";
<sec:authorize access="isAuthenticated()">
    loginUser = '<sec:authentication property="principal.username"/>';
</sec:authorize>

<%-- HTML에서 일어나는 모든 Ajax 전송 요청에 대하여 csrf 토큰값이 요청 헤더에 설정됨 --%
var csrfHeaderName = "${_csrf.headerName}";
var csrfTokenValue = "${_csrf.token}";
$(document).ajaxSend(function(e, xhr, options){
    xhr.setRequestHeader(csrfHeaderName, csrfTokenValue);
});

var bnoValue = '<c:out value="${board.bno}" />';
var commentUL = $(".chat");
var frmCmtPagingValue = $("#frmCmtPagingValue");

...(생략)...
```

#### (6-4-8) 댓글 등록 버튼 클릭 이벤트에 대한 제어 구현: JSP와 Controller에 모두 구현

- ☞ 로그인 사용자ID가 작성자로 서버로 전달되어 처리되어야 함, Ajax로 전송되므로 이를 고려해야 함
- ☞ 요청을 처리하는 서버의 메서드에, 로그인 유무를 확인하도록 구성.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 파일에 빨간색 코드 추가

```
<%-- 댓글처리 자바 스크립트 //--%>
<script type="text/javascript" src="${contextPath}/resources/js/mycomment.js"></script>

<script type="text/javascript">

<%--로그인 계정을 loginUser 변수에 저장,
   앞 실습으로 이미 구성되어 있음, 않은 경우 추가 --%>
var loginUser = "";
<sec:authorize access="isAuthenticated()">
    loginUser = '<sec:authentication property="principal.username"/>';
</sec:authorize>

<%-- HTML에서 일어나는 모든 Ajax 전송 요청에 대하여 csrf 토큰값이 요청 헤더에 설정됨
   앞 실습으로 이미 구성되어 있음, 않은 경우 추가 --%>
var csrfHeaderName = "${_csrf.headerName}";
var csrfTokenValue = "${_csrf.token}";
$(document).ajaxSend(function(e, xhr, options){
    xhr.setRequestHeader(csrfHeaderName, csrfTokenValue);
});

...(생략)...

<%-- 아래의 댓글 등록 버튼 클릭 jQuery 실행문 코드를 찾아서 빨간색 코드 추가 --%>
<%--댓글등록 버튼 클릭 이벤트 처리 --%>
$("#btnRegCmt").on("click", function(){
    <%--찾음--%>

    //var loginUser = "user9"; //이 실행문 주석 처리 또는 삭제 후, 아래 빨간색 코드 추가

    //로그인 유무 검증
    if(!loginUser){
        alert("로그인 후, 등록이 가능합니다.");
        return ;
    }
    console.log("댓글등록 시 loginUser: " + loginUser);

    var txtBoxCmt = $(".txtBoxCmt").val();
    var comment = { bno: bnoValue,
                    rcontent: txtBoxCmt,
                    rwriter: loginUser}; <%--로그인 사용자 값이 전달되어짐--%>

    console.log("댓글등록: 서버전송 객체내용: " + comment);

    myCommentClsr.registerCmt(
        comment,
        function(serverResult){
            $(".txtBoxCmt").val("");
            chgBeforeCmtBtn();

            alert("댓글이 등록되었습니다");

            showCmtList(-1); <%--댓글이 추가된 맨 마지막 페이지 표시--%>
        }
    );
});
```

→ src/main/java/com.spring5213.mypro00.MyReplyController.java 파일에 빨간색 코드 추가

```
//게시물에 대한 댓글 등록: rno 반환
@PreAuthorize("isAuthenticated()") // 추가
@PostMapping(value = "/{bno}/new",
    consumes = {"application/json; charset=UTF-8"},
    produces = { "text/plain; charset=UTF-8" })
public ResponseEntity<String> registerReplyForBoard( @PathVariable("bno") Long bno,
   @RequestBody MyReplyVO myReply) {

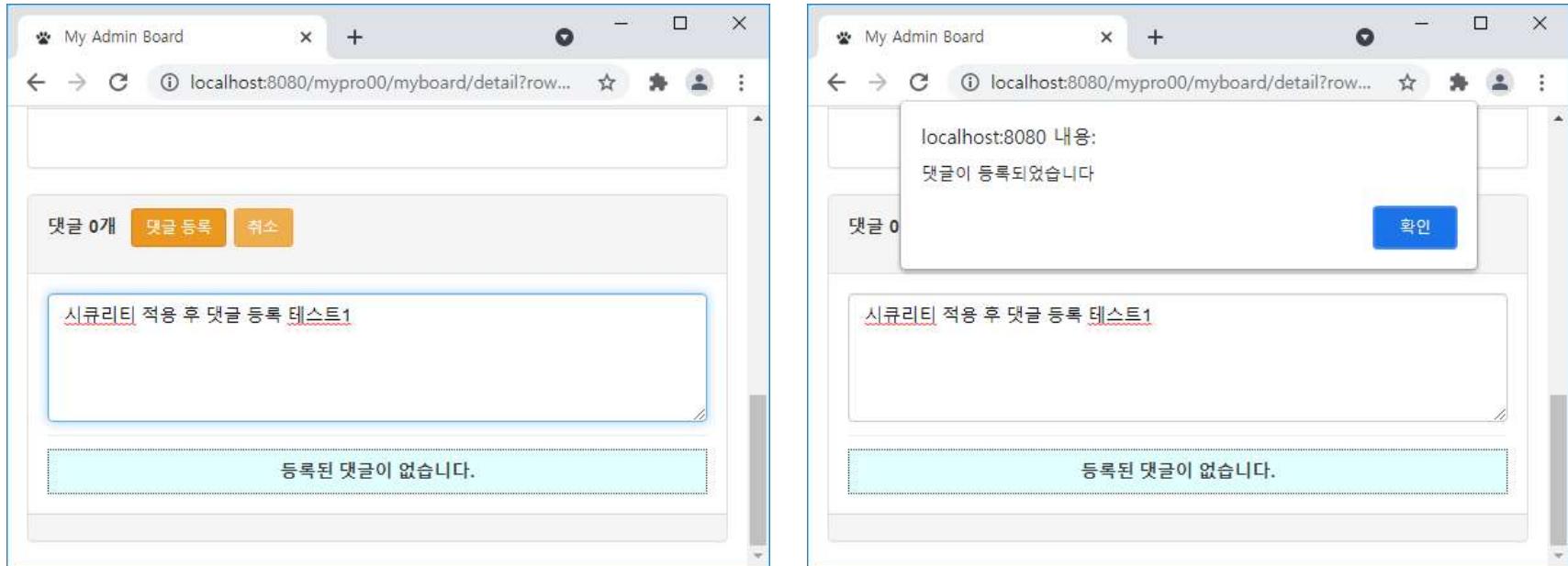
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-URI 추출 bno: " + bno);
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-myReply.getBno(): " + myReply.getBno());
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-서비스로 전달되는 MyReplyVO: " + myReply);

    Long registeredRno = myReplyService.registerReplyForBoard(myReply);
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-반환된 rno(registeredRno): " + registeredRno);
    log.info("댓글-컨트롤러-게시물에 대한 댓글 등록-myReply.getRno(): " + myReply.getRno());

    return registeredRno != null ? new ResponseEntity<>("게시물의 댓글 등록 성공", HttpStatus.OK)
        : new ResponseEntity<>("게시물의 댓글 등록 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}
```

→ 저장 → 이클립스에서 톰캣 서버가 자동으로 재기동 됩니다.

→ 다음은 브라우저에서 user1로 로그인하여 테스트 한 화면캡쳐입니다. 로그인 하지 않은 경우, 댓글 등록이 불가능합니다(앞에서 구현).



(6-4-9) 답글 등록 버튼 클릭 이벤트에 대한 제어 구현: JSP와 Controller에 모두 구현

☞ 로그인 사용자ID가 작성자로 서버로 전달되어 처리되어야 함, Ajax로 전송되므로 이를 고려해야 함

☞ 요청을 처리하는 서버의 메서드에, 로그인 유무를 확인하도록 구성.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 파일에 빨간색 코드 추가

```
<%-- 댓글처리 자바 스크립트 //--%>
<script type="text/javascript" src="${contextPath}/resources/js/mycomment.js"></script>
<script type="text/javascript">
```

```

<%--로그인 계정을 loginUser 변수에 저장,
앞 실습으로 이미 구성되어 있음, 않은 경우 추가 --%>
var loginUser = "";
<sec:authorize access="isAuthenticated()">
    loginUser = '<sec:authentication property="principal.username"/>';
</sec:authorize>

<%-- HTML에서 일어나는 모든 Ajax 전송 요청에 대하여 csrf 토큰값이 요청 헤더에 설정됨
앞 실습으로 이미 구성되어 있음, 않은 경우 추가 --%>
var csrfHeaderName = "${_csrf.headerName}";
var csrfTokenValue = "${_csrf.token}";
$(document).ajaxSend(function(e, xhr, options){
    xhr.setRequestHeader(csrfHeaderName, csrfTokenValue);
});

...(생략)...

<%-- 아래의 답글 등록 버튼 클릭 jQuery 실행문 코드를 찾아서 빨간색 코드 추가 --%>
<%--답글 등록 버튼 클릭 이벤트 처리: 답글이 달린 댓글이 있는 페이지 표시--%>
$(".chat").on("click", ".commentLi .btnRegReply" ,function(){
    //var loginUser = "test8";      //이 실행문 주석 처리 또는 삭제 후, 아래 빨간색 코드 추가
    //로그인 유무 검증
    <%--로그인 안 한 경우--%>
    if(!loginUser){
        alert("로그인 후, 답글 등록이 가능합니다.");
        return ;
    }
    console.log("답글 등록 시 loginUser: "+ loginUser);

    var pageNum = frmCmtPagingValue.find("input[name='pageNum']").val();
    console.log("답글 추가가 발생된 댓글 페이지 번호: " + pageNum);

    var txtBoxReply = $(this).prev().val();
    console.log("txtBoxReply: " + txtBoxReply);

    var prnoVal = $(this).closest("li").data("rno");
    console.log("prnoVal: " + prnoVal);

    var reply = { bno: bnoValue,
                  rcontent: txtBoxReply,
                  rwriter: loginUser,
                  prno: prnoVal };
    console.log("답글등록: 서버전송 객체내용: " + reply);

    myCommentClsr.registerReply(
        reply,
        function(serverResult){
            alert("답글이 등록되었습니다");

            showCmtList(pageNum);<%--댓글이 추가된 페이지 표시--%>
        }
    );
})

```

→ src/main/java/com.spring5213.mypro00.MyReplyController.java 파일에 빨간색 코드 추가

```

//댓글에 대한 답글 등록: rno 반환
@PreAuthorize("isAuthenticated()") // 추가
@PostMapping(value = "/{bno}/{prno}/new",
            consumes = {"application/json; charset=UTF-8"},
            produces = { "text/plain; charset=UTF-8" })
public ResponseEntity<String> registerReplyForReply(@PathVariable("bno") Long bno,
   @PathVariable("prno") Long prno,
   @RequestBody MyReplyVO myReply) {
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-URI 추출 bno: " + bno);
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-myReply.getBno: " + myReply.getBno());
    log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-URI 추출 bno: " + prno);
}

```

```

log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-myReply.getPrno: " + myReply.getPrno());
log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-전달된 MyReplyVO: " + myReply);

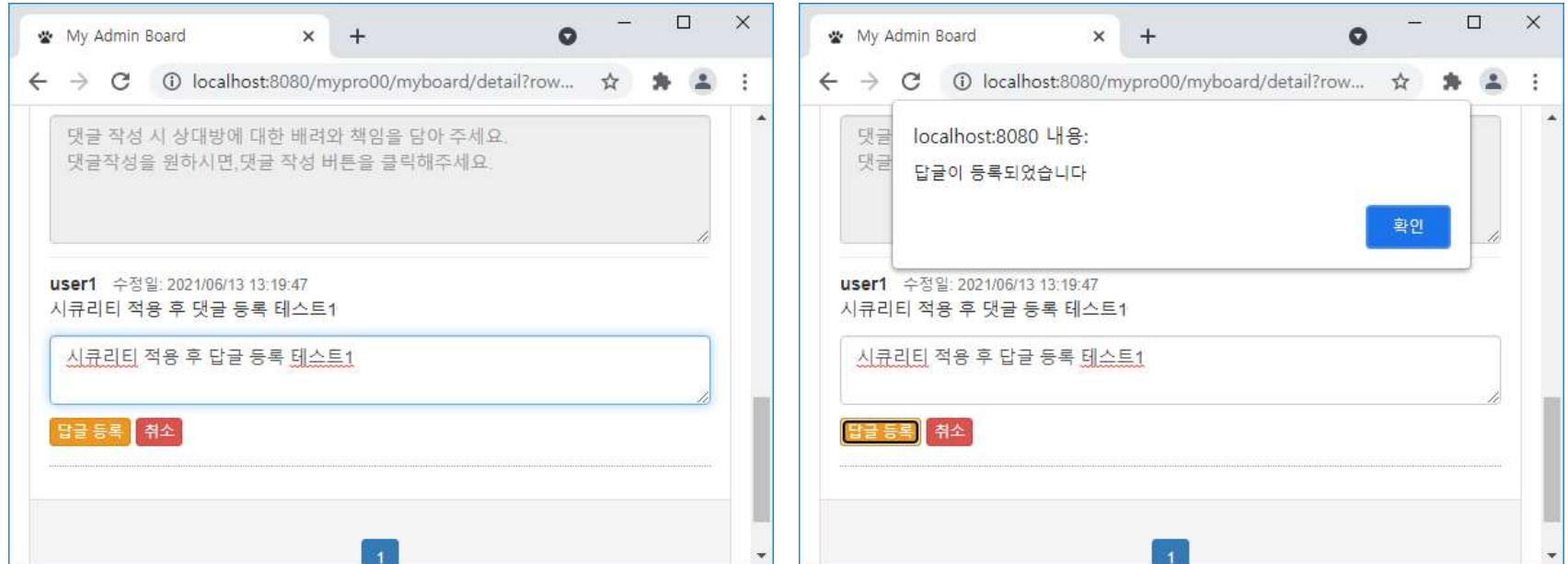
Long registerdRno = myReplyService.registerReplyForReply(myReply);
log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-반환된 rno(registerdRno): " + registerdRno);
log.info("댓글-컨트롤러 - 댓글에 대한 답글 등록-myReply.getRno: " + myReply.getRno());

return registerdRno != null
    ? new ResponseEntity<>("댓글에 대한 답글 등록 성공", HttpStatus.OK)
    : new ResponseEntity<>("댓글에 대한 답글 등록 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}

```

→ 저장 → 이클립스에서 톰캣 서버가 자동으로 재기동 됩니다.

→ 다음은 브라우저에서 user1로 로그인하여 테스트 한 화면캡쳐입니다. 로그인 하지 않은 경우, 답글 등록이 불가능합니다(앞에서 구현).



#### (6-4-10) 댓글과 답글의 수정/삭제 버튼 클릭 이벤트에 대한 제어 구현: JSP와 Controller에 모두 구현

- ☞ 댓글과 답글의 수정/삭제 버튼은 댓글/답글 내용 클릭에 의해 표시된 버튼입니다.
- ☞ 작성자와 로그인 사용자ID가 동일한 한 경우에만 댓글/답글 수정-삭제가 서버로 전달되어 처리되어야 하며, 이 때 Ajax로 요청이 전송되므로 CSRF-токن값의 전송을 고려해야 합니다.
- ☞ 요청을 처리하는 서버의 메서드에, 로그인 유무를 확인하도록 구성합니다.

→ src/main/webapp/WEB-INF/views/myboard/detail.jsp 파일에 빨간색 코드 추가

```

<%-- 댓글처리 자바 스크립트 //--%>
<script type="text/javascript" src="${contextPath}/resources/js/mycomment.js"></script>
<script type="text/javascript">

<%--로그인 계정을 loginUser 변수에 저장,
  앞 실습으로 이미 구성되어 있음, 않은 경우 추가 --%>
var loginUser = "";
<sec:authorize access="isAuthenticated()">
  loginUser = '<sec:authentication property="principal.username"/>';

```

```

</sec:authorize>

<%-- HTML에서 일어나는 모든 Ajax 전송 요청에 대하여 csrf 토큰값이 요청 헤더에 설정됨
앞 실습으로 이미 구성되어 있음, 않은 경우 추가 --%>
var csrfHeaderName = "${_csrf.headerName}";
var csrfTokenValue = "${_csrf.token}";
$(document).ajaxSend(function(e, xhr, options){
    xhr.setRequestHeader(csrfHeaderName, csrfTokenValue);
});

...(생략)...

<%-- 아래의 댓글-답글 수정 버튼 클릭 jQuery 실행문 코드를 찾아서 빨간색 코드 추가 --%>
<%-- 댓글-답글 수정 처리: 수정 버튼 클릭 이벤트 --%>
$(".chat").on("click", ".commentLi .btnModCmt", function(){
    <%--로그인 안 한 경우--%>
    if(!loginUser){
        alert("로그인 후, 수정이 가능합니다.");
        return ;
    }

    <%--작성자 변수에 저장--%>
    var rwriterVal = $(this).siblings("p").data("rwriter");
    console.log("rwriterVal: " + rwriterVal);
    console.log("loginUser: " + loginUser);

    <%--로그인 계정과 작성자가 다른 경우--%>
    if(rwriterVal != loginUser){
        alert("작성자만 수정 가능합니다");
        return ;
    }

    var pageNum = frmCmtPagingValue.find("input[name='pageNum']").val();
    console.log("댓글/답글 수정이 페이지 번호: " + pageNum);

    var txtBoxComment = $(this).prev().val();
    console.log("txtBoxComment: " + txtBoxComment);

    var rnoVal = $(this).closest("li").data("rno");
    console.log("rnoVal: " + rnoVal);

    var comment = { bno: bnoValue,
                    rno: rnoVal,
                    rcontent: txtBoxComment,
                    rwriter: rwriterVal };
    console.log("답글등록: 서버전송 객체내용: " + comment);

    myCommentClsr.modifyCmtReply(
        comment,
        function(serverResult){
            alert("수정되었습니다");

            showCmtList(pageNum); <%-- 답글이 추가된 페이지 표시 --%>
        }
    );
});

<%--댓글-답글 삭제 처리: 삭제 버튼 클릭 이벤트 --%>
$(".chat").on("click", ".commentLi .btnDelCmt", function(){
    <%--로그인 하지 않은 경우--%>
    if(!loginUser){
        alert("로그인 후, 삭제가 가능합니다.");
        return ;
    }

    <%--작성자 변수에 저장--%>

```

```

var rwriterVal = $(this).siblings("p").data("rwriter");
console.log("rwriterVal: " + rwriterVal);
console.log("loginUser: " + loginUser);

<%--로그인 계정과 작성자가 다른 경우--%>
if(rwriterVal != loginUser){
    alert("작성자만 삭제 가능합니다");
    return ;
}

var delConfirm = confirm('삭제하시겠습니까?');

if(!delConfirm){
    alert('삭제가 취소되었습니다.');
    return ;
}

var pageNum = frmCmtPagingValue.find("input[name='pageNum']").val();
console.log("답글 삭제가 발생된 댓글 페이지 번호: " + pageNum);

var rnoVal = $(this).closest("li").data("rno");
console.log("rnoVal: " + rnoVal);

var myComment = { bno: bnoValue,
                 rno: rnoVal,
                 rwriter: rwriterVal };

console.log("답글삭제: 서버전송 객체내용: " + myComment);

myCommentClsr.removeCmtReply(
    myComment,
    function(serverResult){<%--서버에서 댓글저장 성공 시, 브라우저에서 실행될 콜백함수--%>
        alert("삭제되었습니다.");

        showCmtList(pageNum);
    }
);
});
});

```

→ src/main/java/com.spring5213.mypro00.MyReplyController.java 파일에 빨간색 코드 추가

```

//댓글-답글 수정
@PreAuthorize("isAuthenticated() && principal.username == #myReply.rwriter") //추가
@RequestMapping(value = "/{bno}/{rno}",
    method = { RequestMethod.PUT, RequestMethod.PATCH },
    consumes = "application/json; charset=UTF-8",
    produces = "text/plain; charset=UTF-8" )
public ResponseEntity<String> modifyReply(@PathVariable("bno") Long bno,
   @PathVariable("rno") Long rno,
   @RequestBody MyReplyVO myReply) {
    log.info("댓글-컨트롤러 - 댓글 조회-URI 추출 bno: " + bno);
    log.info("댓글-컨트롤러 - 댓글 조회-URI 추출 rno: " + rno);
    log.info("댓글-컨트롤러 - 게시물에 대한 댓글 수정-전달된 MyReplyVO: " + myReply);

    //put의 경우
    myReply.setBno(bno);
    myReply.setRno(rno);

    int modCnt = myReplyService.modifyReply(myReply);
    log.info("댓글-컨트롤러 댓글 수정 - 수정된 댓글 수: " + modCnt);

    return modCnt == 1
        ? new ResponseEntity<>("댓글-답글 수정 성공", HttpStatus.OK)
        : new ResponseEntity<>("댓글-답글 수정 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}

```

```

//댓글-답글 삭제
@PreAuthorize("isAuthenticated() && principal.username == #myReply.rwriter") //추가
@DeleteMapping(value = "/{bno}/{rno}", produces = { "text/plain; charset=UTF-8" })
public ResponseEntity<String> removeReply(@PathVariable("bno") Long bno,
   @PathVariable("rno") Long rno,
   @RequestBody MyReplyVO myReply) { //추가
    log.info("댓글-컨트롤러 - 댓글/답글 삭제-URI 추출 bno: " + bno);
    log.info("댓글-컨트롤러 - 댓글/답글 삭제-URI 추출 rno: " + rno);
    log.info("댓글-컨트롤러 - 댓글/답글 삭제-Ajax로 전달된 myReply: " + myReply); //추가

    return myReplyService.removeReply(bno, rno) == 1
        ? new ResponseEntity<>("댓글 삭제 성공", HttpStatus.OK)
        : new ResponseEntity<>("댓글 삭제 실패", HttpStatus.INTERNAL_SERVER_ERROR);
}

```

→ 저장 → 이클립스에서 톰캣 서버가 자동으로 재기동 됩니다.

→ 다음은 브라우저에서 user1로 로그인하여 테스트 한 화면캡쳐입니다.

로그인 하지 않은 경우, 댓글/답글의 수정/삭제가 불가능합니다(앞에서 구현).

→ 답글 수정 테스트

The figure consists of three screenshots of a web browser window titled 'My Admin Board'. Each screenshot shows a reply from 'user1' with a timestamp of 2021/06/13 14:05:54. In the first screenshot, the reply content is '시큐리티 적용 후 댓글 등록 테스트1' and the '수정' (Modify) button is highlighted. In the second screenshot, the reply content has been changed to '시큐리티 적용 후 댓글 등록 테스트1 수정 테스트1'. In the third screenshot, the reply content has been restored to '시큐리티 적용 후 댓글 등록 테스트1'. Each screenshot also shows a '답글 작성' (Reply Write) button at the bottom.

## → 답글 삭제 테스트

The figure consists of three screenshots of a web application titled "My Admin Board".  
1. The first screenshot shows a list of replies. One reply from "user1" is selected. Below it, there is another reply from "user1". At the bottom of the list, there is a blue "답글 작성" (Reply) button.  
2. The second screenshot shows a confirmation dialog box. It contains the message "localhost:8080 내용: 삭제되었습니다." (Content of localhost:8080: Deleted). There is a blue "확인" (Confirm) button.  
3. The third screenshot shows the same list of replies. The reply from "user1" has been removed, and the list now shows only one reply from "user1".

## (6-5) 게시물 수정 화면에서의 스프링 시큐리티 처리

☞ 게시물 수정 화면은 게시물 상세 화면에서 [수정] 버튼을 클릭해서 이동

→ 로그인 전제로 로그인한 계정이 게시물의 작성자인 경우에만 [수정]버튼 클릭이 가능하도록 이미 구현됨

☞ 게시물 수정 화면의 작업은, 정상적이라면 게시물 작성자가 로그인했을 때만 사용 가능함

→ 단, 서버에 의해서 로그인 상태가 해제된 경우의 처리를 위해 화면과 컨트롤러에 스프링 시큐리티 관련 구현이 필요함

☞ 게시물 수정 화면에서 스프링 시큐리티 구현내용

- 스프링 시큐리티의 태그 라이브러리 추가 및 CSRF 토큰 처리
- 수정/삭제 버튼에 대하여 로그인 유무와 작성자/로그인 계정 비교를 한 번 더 수행
- 버튼 클릭 시, 서버에서 작성자/로그인 계정 비교를 위해 작성자 정보가 같이 서버로 전송되도록 구성.

☞ 다음의 파일들이 수정됩니다.

- src/main/webapp/WEB-INF/views/myboard/modify.jsp 페이지
- src/main/java/com.spring5213.mypro00.controller.MyBoardController 클래스

(6-5-1) 스프링 시큐리티 표현식을 사용하기 위해 태그 라이브러리 참조 설정 및 CSRF 토큰 전송 설정

→ /src/main/webapp/WEB-INF/views/myboard/**modify.jsp** 파일에 빨간색 코드 추가

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %> <%--추가 --%>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<%@ include file="../myinclude/myheader.jsp" %>

...(생략)...

    <sec:csrfInput/> <%-- </form> 종료 태그 위에 csrf 토큰 input 태그 추가, 자동 완성 기능 사용 --%>
</form>
```

(6-5-2) 게시물 수정, 삭제 버튼 표시: 로그인 한 작성자일 경우에만 표시: JSP에서만 구현

→ src/main/webapp/WEB-INF/views/myboard/**modify.jsp** 파일에 다음의 빨간색 코드 추가

```
...(생략)...

<sec:authorize access="isAuthenticated()" ><%-- 추가: 로그인 유무 확인 --%>
    <sec:authentication property="principal" var="principal"/><%-- 추가: 로그인 계정 변수화 --%>
        <c:if test="${principal.username eq board.bwriter}"><%-- 추가: 로그인 계정과 작성자 비교 --%>
            <button type="button" id="btnModify" data-oper="modify" class="btn btn-default btn-frmModify">수정
            </button>
            <button type="button" id="btnRemove" data-oper="remove" class="btn btn-danger btn-frmModify">삭제
            </button>
        </c:if><%-- 추가 --%>
</sec:authorize><%-- 추가 --%>

    <button type="button" id="btnList" data-oper="list" class="btn btn-info btn-frmModify">취소(목록)</button>

...(생략)...
```

☞ <sec:authentication> 태그는 <sec:authorize>와 </sec:authorize> 사이에서 사용해야 합니다.

☞ <sec:authorize access="isAuthenticated()"> 와 </sec:authorize> 사이의 코드는 로그인되었을 때만 처리되어 표시됩니다.

☞ <sec:authentication property="principal" var="principal"/>로 Authentication 객체의 반환정보인 principal이 principal 이름의 변수를 통해서 사용됩니다.

(권장)var에 설정하는 변수이름은 principal로 설정해야 동작이 항상 잘 작동합니다.

☞ <c:if test="\${principal.username eq board.bwriter}">에 의해 로그인ID와 작성자가 동일할 경우에만 버튼이 표시됩니다.

→ 다음은 브라우저에서 user1로 로그인하여 테스트 한 화면캡쳐입니다.

[자신이 작성한 글인 경우]

The screenshot shows two browser windows side-by-side. The left window displays the 'Board - Detail: 229403 번 게시물' page for a post by 'user1'. It includes fields for '글제목' (Subject) containing '시큐리티 화면 글 등록 테스트1' and '글내용' (Content) containing '작성자 이름 로그인 ID로 자동 입력 확인 완료'. The right window shows the '게시글 수정-삭제' (Modify-Delete) page for the same post, with the '글번호' (Post Number) set to '229403' and the '글제목' field also containing '시큐리티 화면 글 등록 테스트1'. Both windows have a blue header bar with the title 'My Admin Board' and a navigation bar with icons.

[다른 사용자가 작성한 글인 경우: 상세페이지에 수정버튼 자체가 없습니다.]

This screenshot shows a single browser window displaying the 'Board - Detail: 229387 번 게시물' page for a post by 'test'. The page structure is identical to the one in the first screenshot, featuring a subject ('첨부파일 게시물 등록1') and content ('11111'). Unlike the first screenshot, there is no separate modification page; the update functionality is integrated directly into the detail view.

### (6-5-3) Ajax 를 통한 첨부파일 수정 시, CSRF 토큰값이 같이 전달되도록 구현

☞ 게시물 수정 페이지에서의 첨부파일 수정(첨부파일 추가, 첨부파일 삭제)도, 댓글에서와 마찬가지로, 파일 유형의 `input` 요소의 내용이 변경되거나 삭제 버튼을 클릭하면 Ajax로 서버에 요청이 전송될 때, 게시물 수정 페이지에 전달된 CSRF 토큰을 같이 보내야만 작업이 서버에서 정상적으로 처리됩니다.

☞ 댓글에서의 처리와 동일하게 게시물 수정 HTML에서 발생되는 모든 Ajax 요청에 대하여 csrf 토큰값이 요청 헤더에 자동으로 포함되도록 구현합니다.

→ `src/main/webapp/WEB-INF/views/myboard/modify.jsp` 파일의 첨부파일 관련 자바 스크립트의 제일 위에 아래의 빨간색 코드를 추가합니다.

...(생략)...

```
<script><%-- 첨부파일 관련 자바스크립트 시작//%--%>
<%-- HTML에서 일어나는 모든 Ajax 전송 요청에 대하여 csrf 토큰값이 요청 헤더에 설정됨 --%>
var csrfHeaderName = "${_csrf.headerName}";
var csrfTokenValue = "${_csrf.token}";
$(document).ajaxSend(function(e, xhr, options){
    xhr.setRequestHeader(csrfHeaderName, csrfTokenValue);
});

var bnoValue = '<c:out value="${board.bno}" />';

...(생략)...
```

### (6-5-4) 게시물 수정 버튼 클릭 이벤트에 대한 제어 구현: JSP 와 Controller 에 모두 구현

☞ 다음의 요구사항이 구현되어야 합니다.

- 취소(목록) 버튼은, 로그인 유무와 상관없이, 구성된 동작(`form` 비움, 게시물 상세 화면으로 다시 이동)이 항상 동작되어야 함.
- 수정, 삭제 버튼은 로그인 유무, 작성자와 로그인 계정이 동일한 지 확인을 거쳐서 동작되어야 함.

☞ 작성자와 로그인 계정이 동일한지를 컨트롤러에서 확인하는 것은, `form` 을 통해 전달된 데이터 중, `bwriter` 값을 이용합니다.

☞ 현재 컨트롤러의 삭제 처리 메서드는 `bno` 값만 처리하도록 구현되어 있습니다.

따라서, 컨트롤러의 삭제 메서드에 작성자(`bwriter`) 값도 처리할 수 있도록 코드를 수정합니다.

→ `src/main/webapp/WEB-INF/views/myboard/modify.jsp` 파일의 수정버튼, 삭제 버튼, 취소 버튼 클릭 이벤트 처리 jQuery 코드를 다음의 빨간색 코드처럼 전체적으로 수정하면서 스프링 시큐리티 코드를 추가합니다.

```
<script>

//form의 수정/삭제/목록보기 버튼에 따른 동작 제어
var frmModify = $("#frmModify");

var loginUser = "";    <%-- 여기서부터 추가 시작 --%>

<sec:authorize access="isAuthenticated()">
    loginUser = '<sec:authentication property="principal.username"/>';
```

```

</sec:authorize>

$('.btn-frmModify').on("click", function(e){

    var operation = $(this).data("oper"); //각 버튼의 data-xxx 속성에 설정된 값을 저장
    var bwriterVal = '<c:out value="${board.bwriter}" />';<%-- 추가 --%>
    alert("operation: " + operation + ", bwriterVal: " + bwriterVal);<%-- 기존코드 삭제 후, 추가 --%>

<%--취소(목록)을 먼저 처리 후, 수정 삭제 처리 순서로 if 절 수정 --%>
    if(operation == "list"){ //게시물 목록 화면 요청
        var pageNumInput = $("input[name='pageNum']").clone();
        var rowAmountInput = $("input[name='rowAmountPerPage']").clone();
        var scopeInput = $("input[name='scope']").clone();
        var keywordInput = $("input[name='keyword']").clone();

        frmModify.empty();

        frmModify.append(pageNumInput);
        frmModify.append(rowAmountInput);
        frmModify.append(scopeInput);
        frmModify.append(keywordInput);
        frmModify.attr("action", "${contextPath}/myboard/list").attr("method", "get");
    } else {
        <%--로그인 안 한 경우--%>
        if(!loginUser){
            alert("로그인 후, 수정/삭제가 가능합니다.");
            return ;
        }

        <%--로그인 계정과 작성자가 다른 경우--%>
        if(bwriterVal != loginUser){
            alert("작성자만 수정/삭제가 가능합니다");
            return ;
        }

        if(operation == "modify"){ //게시물 수정 요청

            var strFilesInputHidden = "";
            //업로드 결과의 li 요소 선택하여 각각에 대하여 다음을 처리
            $(".fileUploadResult ul li").each(function(i, obj){
                var objLi = $(obj);
                strFilesInputHidden
                    += "<input type='hidden' name='attachFileList["+i+"].uuid' "
                    + "      value='" + objLi.data("uuid") + "'>"
                    + "<input type='hidden' name='attachFileList["+i+"].uploadPath' "
                    + "      value='" + objLi.data("uploadpath") + "'>"
                    + "<input type='hidden' name='attachFileList["+i+"].fileName' "
                    + "      value='" + objLi.data("filename") + "'>"
                    + "<input type='hidden' name='attachFileList["+i+"].fileType' "
                    + "      value='" + objLi.data("filetype") + "'> ";
            });
            console.log(strFilesInputHidden);//테스트 후, 주석처리할 것
            frmModify.append(strFilesInputHidden); //form에 추가
            frmModify.attr("action", "${contextPath}/myboard/modify");

        } else if(operation == "remove"){ //게시물 삭제 요청
            //frmModify.attr("action", "${contextPath}/myboard/delete");
            frmModify.attr("action", "${contextPath}/myboard/remove");
        }
    }
    frmModify.submit() ; //요청 전송
});

</script>

...(생략)...

```

→ src/main/java/com.spring5213.mypro00.controller.MyBoardController.java 파일에 다음의 빨간색 코드 추가

→ 삭제 처리 메서드는 추가적으로 bwriter 를 처리할 매개변수를 추가해야 합니다.

//게시물 수정 처리

```
@PreAuthorize("isAuthenticated() && principal.username == #myBoard.bwriter")
@PostMapping("/modify")
public String modifyBoard(MyBoardVO myBoard,
                         RedirectAttributes redirectAttr,
                         MyBoardPagingDTO myBoardPagingDTO){
    log.info("컨트롤러 - 게시물 수정 전달된 myBoard 값: " + myBoard);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    if (myBoardService.modifyBoard(myBoard)) {
        redirectAttr.addFlashAttribute("result", "successModify");
    }

    redirectAttr.addAttribute("bno", myBoard.getBno());
    redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
    redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
    redirectAttr.addAttribute("scope", myBoardPagingDTO.getScope());
    redirectAttr.addAttribute("keyword", myBoardPagingDTO.getKeyword());

    log.info("컨트롤러 - 전달된 redirectAttr: " + redirectAttr);

    return "redirect:/myboard/detailmod";
}
```

//게시물 삭제 - 실제 삭제는 않됨

```
@PreAuthorize("isAuthenticated() && principal.username == #bwriter")
@PostMapping("/delete")
public String setBoardDeleted(@RequestParam("bno") Long bno,
                             @RequestParam("bwriter") String bwriter, ← 추가
                             RedirectAttributes redirectAttr,
                             MyBoardPagingDTO myBoardPagingDTO){
    log.info("컨트롤러 - 게시물 삭제(bdelFlag값변경 글번호): " + bno);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    if (myBoardService.setBoardDeleted(bno)) {
        redirectAttr.addFlashAttribute("result", "successRemove");
    }

    return "redirect:/myboard/list" + myBoardPagingDTO.addPagingParamsToURI();
}
```

//게시물 삭제 - 실제 삭제 됨

```
@PreAuthorize("isAuthenticated() && principal.username == #bwriter")
@PostMapping("/remove")
public String removeBoard(@RequestParam("bno") Long bno,
                         @RequestParam("bwriter") String bwriter, ← 추가
                         RedirectAttributes redirectAttr,
                         MyBoardPagingDTO myBoardPagingDTO){
    log.info("컨트롤러 = 게시물 삭제: 삭제되는 글번호: " + bno);
    log.info("컨트롤러 - 전달된 MyBoardPagingDTO: " + myBoardPagingDTO);

    //첨부파일 정보를 저장할 리스트 객체 생성
    List<MyBoardAttachFileVO> attachFileList = myBoardService.getAttachFilesByBno(bno);
```

//게시물 삭제가 성공한 후,

```
if(myBoardService.removeBoard(bno)) {
    //첨부파일 삭제(MyBoardController에 새로 추가된 메서드)
    removeAttachFiles(attachFileList);
    redirectAttr.addFlashAttribute("result", "successRemove");
} else {
    redirectAttr.addFlashAttribute("result", "failRemove");
}
```

```

//RedirectAttribute 유형의 객체에 전달할 페이지 데이터를 속성으로 바인딩
redirectAttr.addAttribute("pageNum", myBoardPagingDTO.getPageNum());
redirectAttr.addAttribute("rowAmountPerPage", myBoardPagingDTO.getRowAmountPerPage());
redirectAttr.addAttribute("scope", myBoardPagingDTO.getScope());
redirectAttr.addAttribute("keyword", myBoardPagingDTO.getKeyword());
log.info("화면으로 전달될 redirectAttr: " + redirectAttr);

return "redirect:/myboard/list";
}

```

☞ 다음은 브라우저에서 user1로 로그인하여 테스트 한 화면캡쳐입니다.

[게시물 상세 페이지, 수정 버튼 클릭]

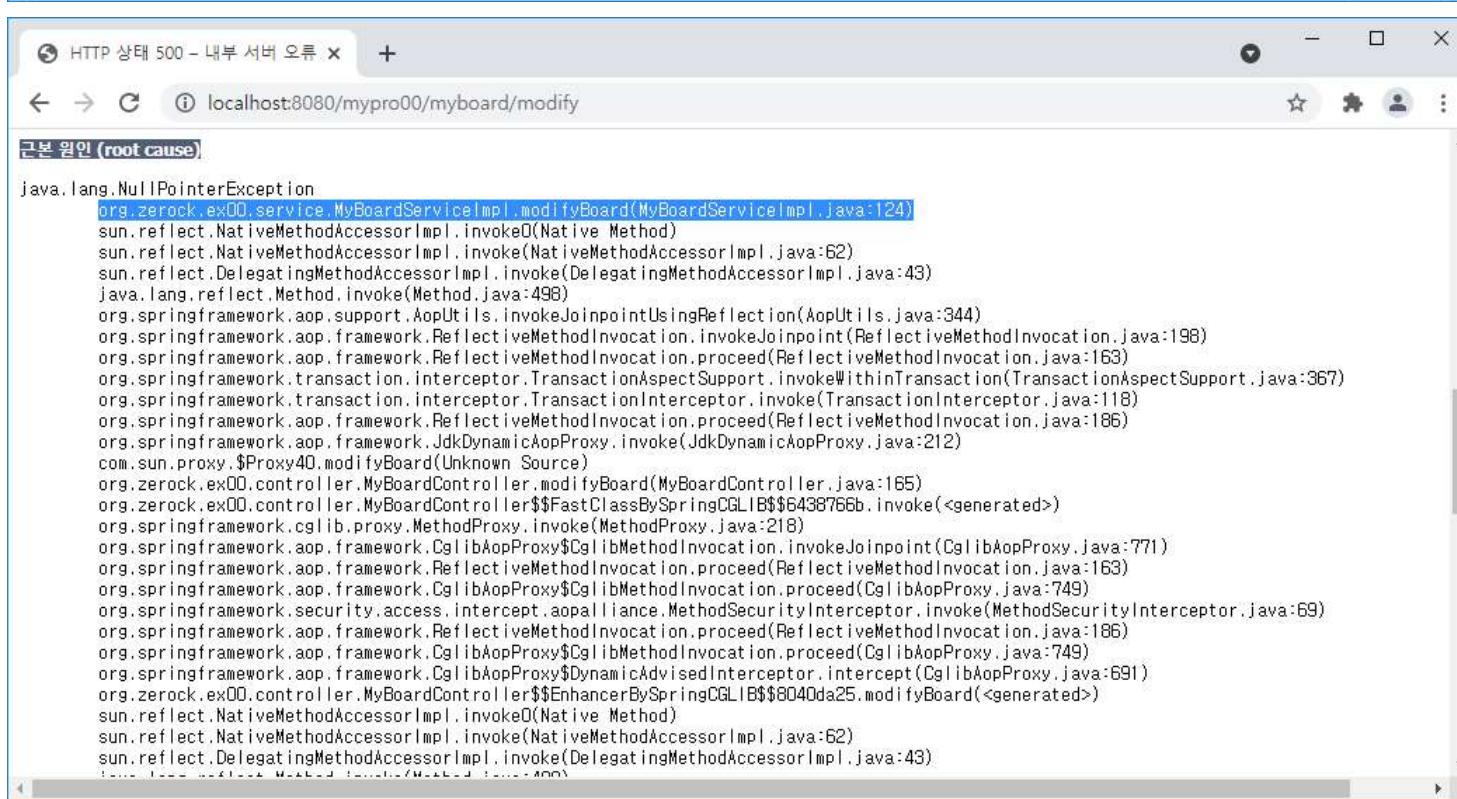
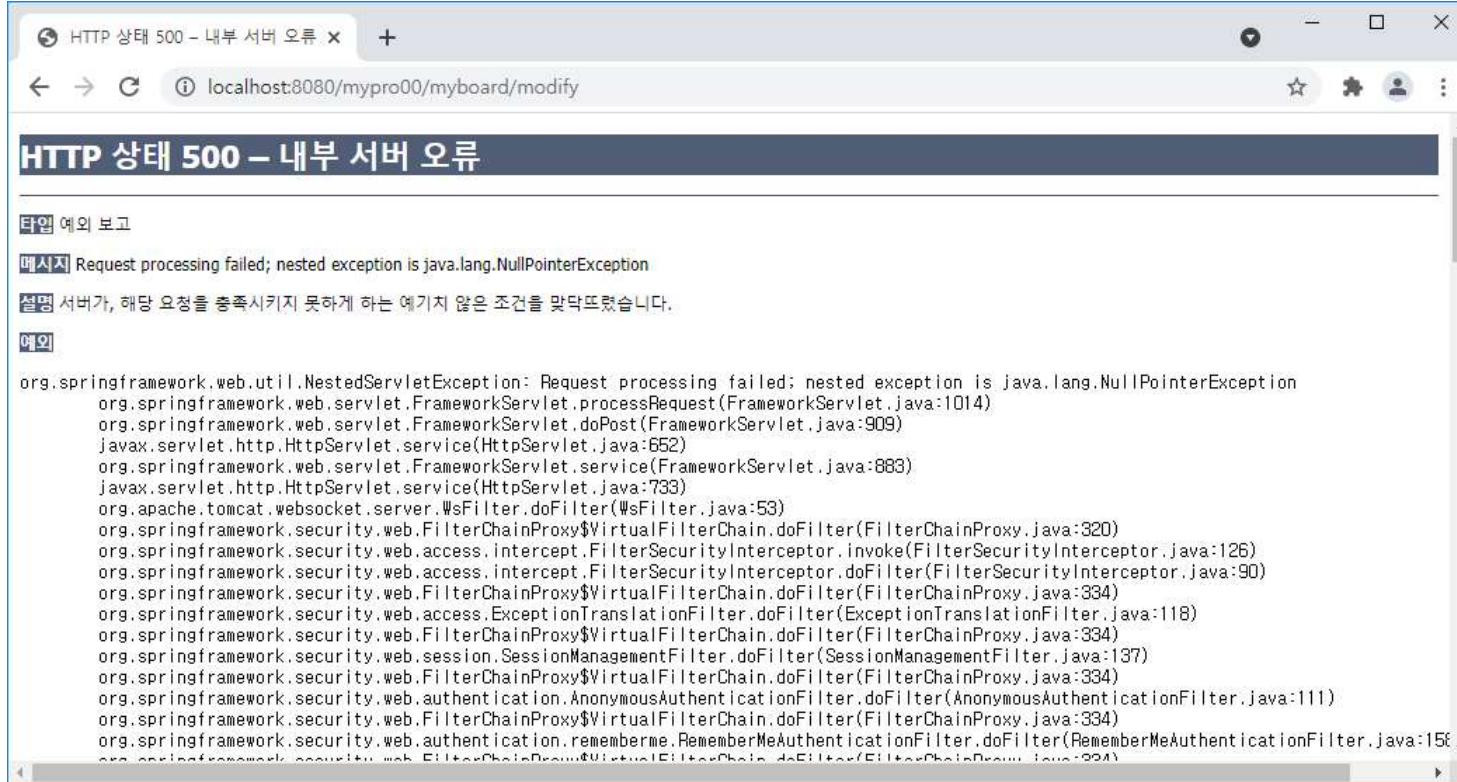
This screenshot shows the 'Board - Detail' page for post number 229403. It includes fields for title ('user1님 작성'), date ('등록일: 2021-06-13 11:36:04'), and views ('조회수: 18'). Below these are fields for subject ('글제목') and content ('글내용'). A blue '수정' (Modify) button is visible at the bottom right of the main content area.

[내용 수정(첨부파일 없음) 수정버튼 클릭]

This screenshot shows the '게시글 수정-삭제' form for post number 229403. It contains fields for subject ('글제목'), content ('글내용'), author ('작성자'), and a note about the latest update ('최종수정일 [등록일시]: 2021/06/13 11:36:04'). The blue '수정' (Modify) button is highlighted.

This screenshot shows the '게시글 수정-삭제' form again. A modal dialog is open, displaying the URL 'localhost:8080 내용:' and the operation 'operation: modify, bwriterVal: user1'. The '확인' (Confirm) button in the dialog is highlighted.

첨부파일 없이 수정한 경우, 다음의 오류가 발생되었습니다. 이건 미처 생각하지 못했습니다.



## 콘솔내용

```
INFO : org.zerock.ex00.controller.MyBoardController - 컨트롤러 - 게시물 수정 전달된 myBoard 값: MyBoardVO(bno=229403, btitle=시큐리티 게시물 수정 테스트1, bcontent=시큐리티 게시물 수정 테스트 글 내용, bwriter=user1, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null, attachFileList=null)
INFO : org.zerock.ex00.controller.MyBoardController - 컨트롤러 - 전달된 MyBoardPagingDTO: MyBoardPagingDTO(pageNum=1, rowAmountPerPage=10, scope=, keyword=)
INFO : org.zerock.ex00.service.MyBoardServiceImpl - BoardService.modify()에 전달된 MyBoardVO: MyBoardVO(bno=229403, btitle=시큐리티 게시물 수정 테스트1, bcontent=시큐리티 게시물 수정 테스트 글 내용, bwriter=user1, bviewsCnt=0, breplyCnt=0, bdelFlag=0, bregDate=null, bmodDate=null, attachFileList=null)
INFO : jdbc.sqlonly - DELETE FROM book_ex.tbl_myAttachFiles WHERE bno = 229403 ←첨부파일이 없는 게시물이므로, 0개행이 삭제됨, 오류없음

INFO : jdbc.sqltiming - DELETE FROM book_ex.tbl_myAttachFiles WHERE bno = 229403 ←첨부파일이 없는 게시물이므로, 0개행이 삭제됨, 오류없음
{executed in 3 msec}
INFO : jdbc.sqlonly - UPDATE book_ex.tbl_myboard SET btitle = '시큐리티 게시물 수정 테스트1', bcontent = '시큐리티 게시물 수정 테스트 글 내용', bmodDate = DEFAULT WHERE bno = 229403

INFO : jdbc.sqltiming - UPDATE book_ex.tbl_myboard SET btitle = '시큐리티 게시물 수정 테스트1', bcontent = '시큐리티 게시물 수정 테스트 글 내용', bmodDate = DEFAULT WHERE bno = 229403
{executed in 2 msec}

6월 13, 2021 2:53:14 오후 org.apache.catalina.core.StandardWrapperValve invoke
심각: 경로 [/mypro00]의 컨텍스트 내의 서블릿 [appServlet]을(를) 위한 Servlet.service() 호출이, 근본 원인(root cause)과 함께, 예외 [Request processing failed; nested exception is java.lang.NullPointerException]을(를) 발생시켰습니다.
java.lang.NullPointerException ←오류가 NullPointerException입니다
at org.zerock.ex00.service.MyBoardServiceImpl.modifyBoard(MyBoardServiceImpl.java:124) ← MyBoardServiceImpl 124행에서 오류 발생. 더블클릭 → 이를클립스에 소스 표시
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:344)
```

```

at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:198)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:367)
at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:118)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186)
at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:212)
at com.sun.proxy.$Proxy40.modifyBoard(Unknown Source)
at org.zerock.ex00.controller.MyBoardController.modifyBoard(MyBoardController.java:165) ← MyBoardController.java 165행에서 오류 발생.
at org.zerock.ex00.controller.MyBoardController$$FastClassBySpringCGLIB$$6438766b.invoke(<generated>)

...(생략)...

```

[오류 해결을 위한 조치과정]

→ 해당 MyBoardServiceImpl.java 클래스의 오류 발생 코드를 확인합니다.

```

//게시물 수정 처리
@Transactional
@Override
public boolean modifyBoard(MyBoardVO myBoard){
    log.info("BoardService.modify()에 전달된 MyBoardVO: " + myBoard);

    Long bno = myBoard.getBno();

    //게시물 변경 시, 기존 첨부파일을 모두 삭제되어 전달된 첨부파일 정보가 없는 경우도 있으므로
    //첨부파일 삭제는 무조건 처리합니다.
    myBoardAttachFileMapper.deleteAttachFilesByBno(bno);

    //게시물 수정: tbl_myboard 테이블에 수정
    boolean boardModifyResult = myBoardMapper.updateMyBoard(myBoard) == 1;

    //게시물 수정이 성공하고, 첨부파일이 있는 경우에만 다음작업 수행
    //첨부파일 정보 저장: tbl_myAttachFiles 테이블에 저장
    if (boardModifyResult && myBoard.getAttachFileList().size() > 0) { ← 오류가 발생된 124행입니다
        myBoard.getAttachFileList().forEach(
            attachFile -> {
                attachFile.setBno(bno);
                myBoardAttachFileMapper.insertAttachFile(attachFile);
            }
        );
    }

    return boardModifyResult;
}

```

→ 정확한 NullPointerException 오류의 원인을 찾기 위해 다음의 빨간색 코드 2 줄을, 오류 발생 행 위에 삽입 → 저장

```

//게시물 수정: tbl_myboard 테이블에 수정
boolean boardModifyResult = myBoardMapper.updateMyBoard(myBoard) == 1;
log.info("boardModifyResult: " + boardModifyResult);
log.info("myBoard.getAttachFileList().size(): " + myBoard.getAttachFileList().size()); //오류가 발생될 거 같습니다.
//게시물 수정이 성공하고, 첨부파일이 있는 경우에만 다음작업 수행
//첨부파일 정보 저장: tbl_myAttachFiles 테이블에 저장
if (boardModifyResult && myBoard.getAttachFileList().size() > 0) { ← 오류가 발생된 124행입니다
    myBoard.getAttachFileList().forEach(
        attachFile -> {
            attachFile.setBno(bno);
            myBoardAttachFileMapper.insertAttachFile(attachFile);
        }
    );
}

```

→ 조금 기다리면 자동으로 톰캣 서버가 재기동 됨

→ 오류 재현을 위해, 브라우저에서 user1로 로그인하여 동일한 게시물에 대하여 수정 테스트를 시도합니다.

☞ 다음은 오류 발생 시, 이클립스 콘솔에서 오류를 확인 후, MyBoardServiceImpl 의 오류난 행(122 행)의 소스로 이동합니다.

6월 13, 2021 3:19:13 오후 org.apache.catalina.core.StandardWrapperValve invoke

심각: 경로 [/mypro00]의 컨텍스트 내의 서블릿 [appServlet]을(를) 위한 Servlet.service() 호출이, 근본 원인(root cause)과 함께, 예외 [Request processing

```
failed; nested exception is java.lang.NullPointerException]을(를) 발생시켰습니다.  
java.lang.NullPointerException  
at org.zerock.ex00.service.MyBoardServiceImpl.modifyBoard(MyBoardServiceImpl.java:122) ←오류가 발생된 원인, 더블클릭  
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
...(생략)...
```

→ 새로 추가한 코드 중 `myBoard.getAttachFileList().size()`가 포함된 코드에서 오류가 발생되었습니다,

[오류 발생 이유]

예상대로 새로 추가한 `log.info();` 라인에서 오류가 발생되었습니다. 첨부파일이 없는 경우, `MyBoard.attachFileList` 필드는 Null 상태이므로 `size()` 메서드가 읽을 수 없기 때문에 `NullPointerException`이 발생된 것입니다.

→ 이를 고려해서 코드를 다음처럼 수정합니다.

```
//게시물 수정 처리  
@Transactional  
@Override  
public boolean modifyBoard(MyBoardVO myBoard){  
    log.info("BoardService.modify()에 전달된 MyBoardVO: " + myBoard);  
    //게시물 정보 수정, 수정된 행수가 1 이면 True.  
    //return myBoardMapper.updateMyBoard(myBoard) == 1;  
  
    Long bno = myBoard.getBno();  
  
    //게시물 변경 시, 기존 첨부파일을 모두 삭제되어 전달된 첨부파일 정보가 없는 경우도 있으므로  
    //첨부파일 삭제는 무조건 처리합니다.  
    myBoardAttachFileMapper.deleteAttachFilesByBno(bno);  
  
    //게시물 수정: tbl_myboard 테이블에 수정  
    boolean boardModifyResult = myBoardMapper.updateMyBoard(myBoard) == 1 ;  
  
    //게시물 수정이 성공하고, 첨부파일이 있는 경우에만 다음작업 수행  
    //첨부파일 정보 저장: tbl_myAttachFiles 테이블에 저장  
    //if (boardModifyResult && myBoard.getAttachFileList().size() > 0) { //첨부파일이 없을 경우 오류발생  
    if (boardModifyResult && myBoard.getAttachFileList() != null) { //수정코드 ← 수정됨  
        myBoard.getAttachFileList().forEach(  
            attachFile -> {  
                attachFile.setBno(bno);  
                myBoardAttachFileMapper.insertAttachFile(attachFile);  
            }  
        );  
    }  
  
    return boardModifyResult;  
}
```

→ 수정 → 저장 → 톰캣 서버가 재기동 될때까지 기다림

→ 웹 브라우저에서 user1로 로그인하여 다시 동일한 게시물 수정을 테스트를 시도합니다.

☞ 앞에서의 수정 작업은 오류로 인하여 수정 작업이 롤백된 것도 확인할 수 있습니다.

The image consists of four screenshots of a web browser window titled "My Admin Board".

- Screenshot 1 (Top Left):** Shows the "Board - Detail: 229403 번 게시물" page. It displays the post details: "user1님 작성" (written by user1), "등록일: 2021-06-13 11:36:04 조회수: 27" (registered on 2021-06-13 11:36:04, viewed 27 times). Below this are fields for "글제목" (title) containing "시큐리티 화면 글 등록 테스트1" and "글내용" (content) containing "작성자 이름 로그인 ID로 자동 입력 확인 완료". At the bottom are "수정" (edit) and "목록" (list) buttons.
- Screenshot 2 (Top Right):** Shows the "게시글 수정-삭제" (modification-deletion) form. It has fields for "글번호" (post number) set to 229403, "글제목" (title) set to "시큐리티 화면 글 수정 테스트1", and "글내용" (content) set to "첨부파일이 없을 경우 글 수정 테스트1". It also shows the original author "user1" and registration date "2021/06/13 11:36:04". At the bottom are "수정" (edit), "삭제" (delete), and "취소(목록)" (cancel/list) buttons.
- Screenshot 3 (Bottom Left):** Shows a confirmation dialog box with the message "localhost:8080 내용: operation: modify, bwriterVal: user1". It has a "확인" (confirm) button.
- Screenshot 4 (Bottom Right):** Shows the "Board - Detail: 229403 번 게시물" page again, identical to Screenshot 1, confirming that the update was rolled back.

← 잘 동작 합니다.

→ 브라우저에서 user1로 로그인하여 게시물 삭제도 테스트합니다.

The image displays five browser windows showing the 'My Admin Board' application interface. The windows are arranged in two columns and one row.

- Window 1 (Top Left):** Shows the 'Board - Detail' page for post number 229403. It includes fields for 글제목 (Title), 글내용 (Content), and 작성자 (Author). Buttons at the bottom include '수정' (Edit) and '목록' (List).
- Window 2 (Top Right):** Shows the '게시글 수정-삭제' (Post Modify/Delete) page for post number 229403. It includes fields for 글번호 (Post Number), 글제목 (Title), 글내용 (Content), and 작성자 (Author). Buttons at the bottom include '수정' (Edit), '삭제' (Delete), and '취소(목록)' (Cancel/Back to List).
- Window 3 (Bottom Left):** Shows the '게시글 삭제' (Post Delete) confirmation dialog. It displays the post number 229403 and the message 'localhost:8080 내용: operation: remove, bwriterVal: user1'. A '확인' (Confirm) button is visible.
- Window 4 (Bottom Middle):** Shows the '처리결과' (Processing Result) page, which displays the message '글이 삭제되었습니다.' (The article has been deleted.). A '확인' (Confirm) button is visible.
- Window 5 (Bottom Right):** Shows the '게시글 목록' (Post List) page. It includes search filters for '검색 범위' (Search Range) and '검색어를 입력하세요' (Enter search term), and a '검색' (Search) button. Below is a table listing posts:

번호	제목	작성자	작성일	수정일	조회수
229402	시큐리티 화면 글 등록 테스트1 [댓글수: 0]	user1	2021/06/12	2021/06/12 12:53:52	3
229401	화면 새글등록 테스트1 [댓글수: 0]	user1	2021/06/12	2021/06/12 12:50:33	0
229387	첨부파일 게시물 등록1 [댓글수: 0]	test	2021/06/06	2021/06/06 22:13:12	10
229385	첨부파일 게시물 등록	test	2021/06/06	2021/06/06	16

## (6-6) Bootstrap Template의 로그아웃 아이콘에 접속상태 표시 구현

- ☞ 현재 myheader.jsp에 구성된 내용에 의하여 브라우저 페이지 상단 오른편에 있는 사용자 아이콘을 클릭하면, 로그인 항목과 로그아웃 항목이 같이 표시됩니다.
- ☞ 이를 하나의 항목으로 통합하여, 스프링 시큐리티에 의해, 로그인 상태에서는 로그아웃 페이지로 이동하고, 로그아웃 상태에서는 로그인 페이지로 이동하도록 수정합니다.
- ☞ 그리고, 로그아웃 시에 사용자가 로그아웃을 확인해주는 과정을 추가하고 로그아웃 후에는 home.jsp 페이지가 표시될 때 로그아웃 메시지가 alert로 표시 되도록 수정합니다.

### (6-6-1) 로그아웃 아이콘에 로그인 상태 반영(myheader.jsp 파일)

#### 1) 스프링 시큐리티 표현식을 사용하기 위해 태그 라이브러리 참조 설정

→ src/main/webapp/WEB-INF/views/myinclude/myheader.jsp 파일에 다음의 빨간색 코드 추가

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %><%-- 추가 --%>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

...(생략)...
```

#### 2) 로그아웃 아이콘에 로그인 상태 반영

→ src/main/webapp/WEB-INF/views/myinclude/myheader.jsp 파일에 다음의 빨간색 코드 추가

```
...(생략)...

<ul class="nav navbar-top-links navbar-right">
    <li class="dropdown">
        <a class="dropdown-toggle" data-toggle="dropdown"><!-- href 속성 삭제 -->
            <i class="fa fa-user fa-fw"></i>
            <sec:authorize access="isAuthenticated()">
                <sec:authentication property="principal.username"/><span>님 접속</span>
            </sec:authorize>
            <sec:authorize access="isAnonymous()"><span>로그인</span>
            </sec:authorize>
            <i class="fa fa-caret-down"></i>
        </a>
        <ul class="dropdown-menu"><%-- 클래스 이름 중 dropdown-user 삭제 --%>
            <%-->
            <li><a id="myLogin" href="${contextPath}/login">
                <i class="fa fa-sign-in fa-fw"></i> Sign in</a><%-- a 링크 수정, Sign in 으로 수정 -->
            </li>
            <li class="divider"></li>
            <li><a id="myLogout" href="${contextPath}/logout">
                <i class="fa fa-sign-out fa-fw"></i> Sign out</a><%-- a 링크 수정, Sign out 으로 수정 -->
            </li> <%-->
        </ul>
    </li>
</ul>
```

```

<sec:authorize access="isAuthenticated()">
    <li><a id="myLogout" href="${contextPath}/logout">
        <i class="fa fa-sign-out fa-fw"></i> Sign out</a>
    </li>
</sec:authorize>
<sec:authorize access="isAnonymous()">
    <li><a id="myLogin" href="${contextPath}/login">
        <i class="fa fa-sign-in fa-fw"></i> Sign in</a>
    </li>
</sec:authorize>
</ul><!-- /.dropdown-user -->
</li><!-- /.dropdown -->
</ul><!-- /.navbar-top-links -->

...(생략)...

```

(6-6-2) 로그아웃 결정 메시지 표시 jQuery 를 </body> 태그 위에 추가

→ src/main/webapp/WEB-INF/views/common/myLogout.jsp 파일에서 form 에 대하여 다음의 빨간색 코드를 추가하고,

</body> 태그 위에 <script> 요소의 코드를 추가

...(생략)...

```

<form role="form" id="frmMyLogout" action="${contextPath}/logout" method='post'>
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
    <fieldset>
        <div>
            <button type='button' class="btn btn-lg btn-success btn-block" id="btnMyLogout">Log Out</button>
        </div>
    </fieldset>

</form>

...(생략)...

<script>
    var frmMyLogout = $("#frmMyLogout");
    $("#btnMyLogout").on("click", function(e){
        var logoutDecision = confirm("로그아웃 하시겠습니까?");

        if (!logoutDecision){
            alert("로그아웃이 취소되었습니다.");
            return ;
        } else {
            frmMyLogout.submit();
        }
    });
</script>

</body>
</html>

```

(6-6-3) 로그아웃으로 home.jsp로 이동한 경우, 로그아웃 메시지 표시

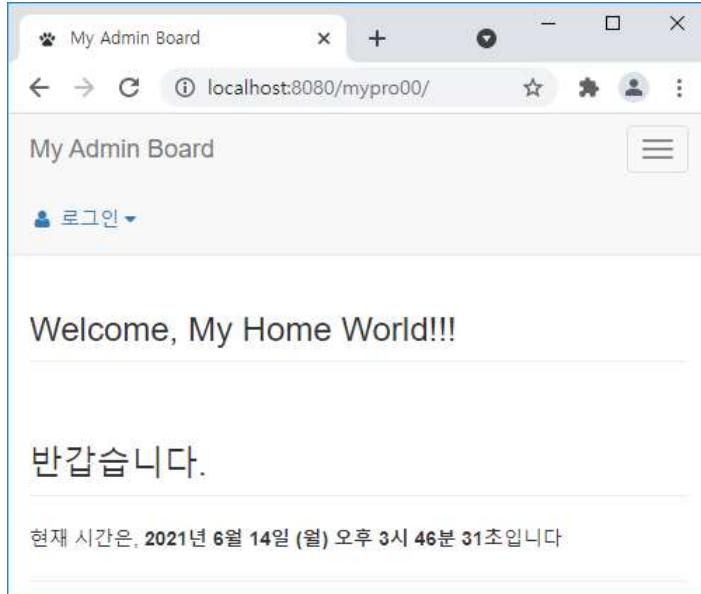
→ src/main/webapp/WEB-INF/views/home.jsp 파일에 다음의 빨간색 코드 추가

```
<%-- 로그아웃으로 홈페이지 이동한 경우에만 실행됨 --%>
<c:if test="${param.logout != null}">
    <script type="text/javascript">
        $(document).ready(function(){
            alert("로그아웃하였습니다.");
        });
    </script>
</c:if>

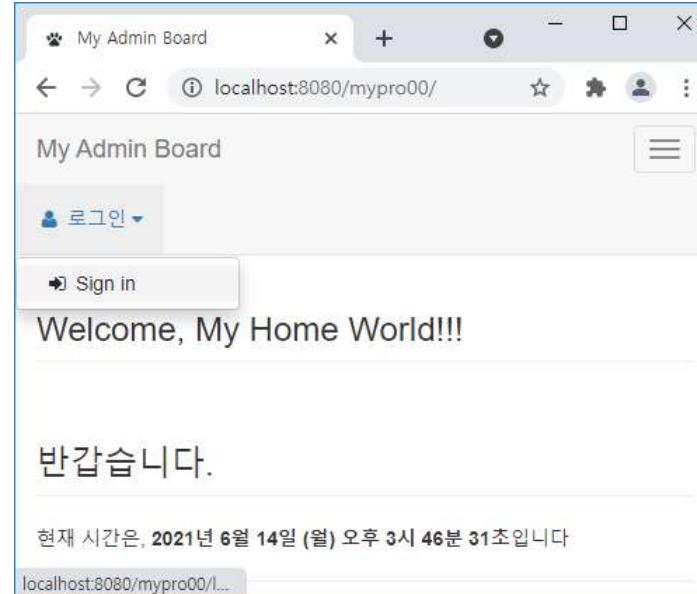
<%@ include file=".myinclude/myfooter.jsp" %>
```

☞ 다음은 웹브라우저에서 테스트한 결과입니다.

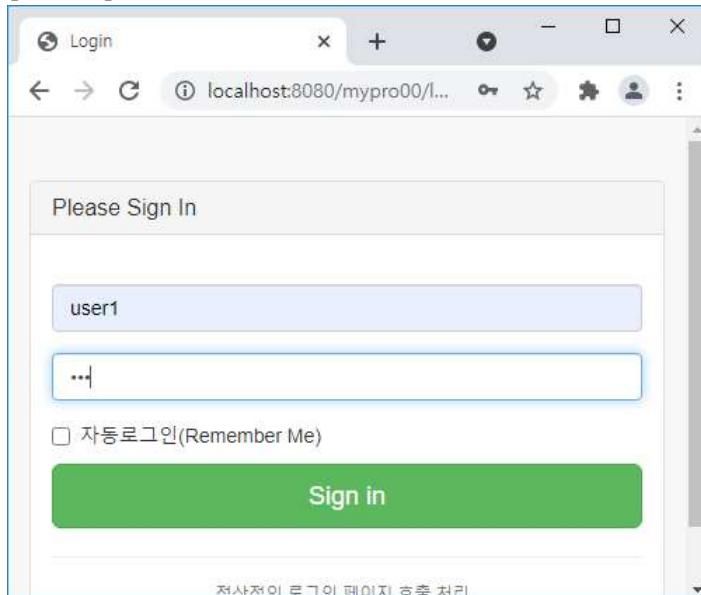
[/mypro00 요청]



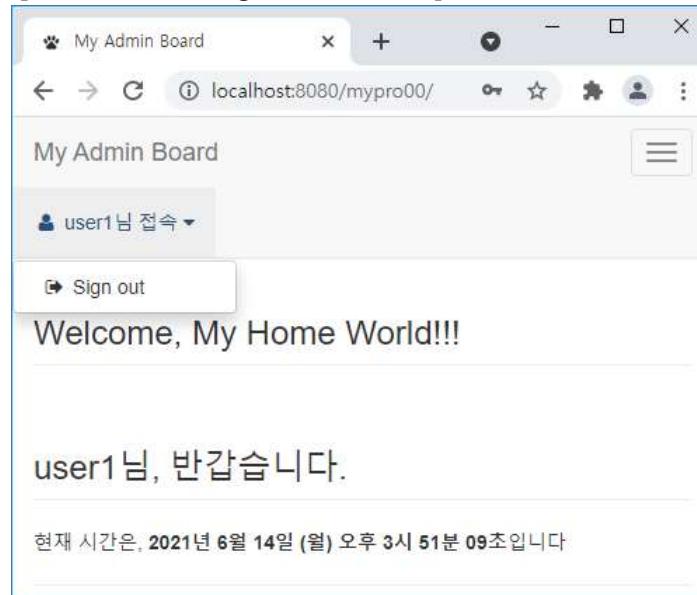
[로그인 안된 상태: Sign in 만 표시]



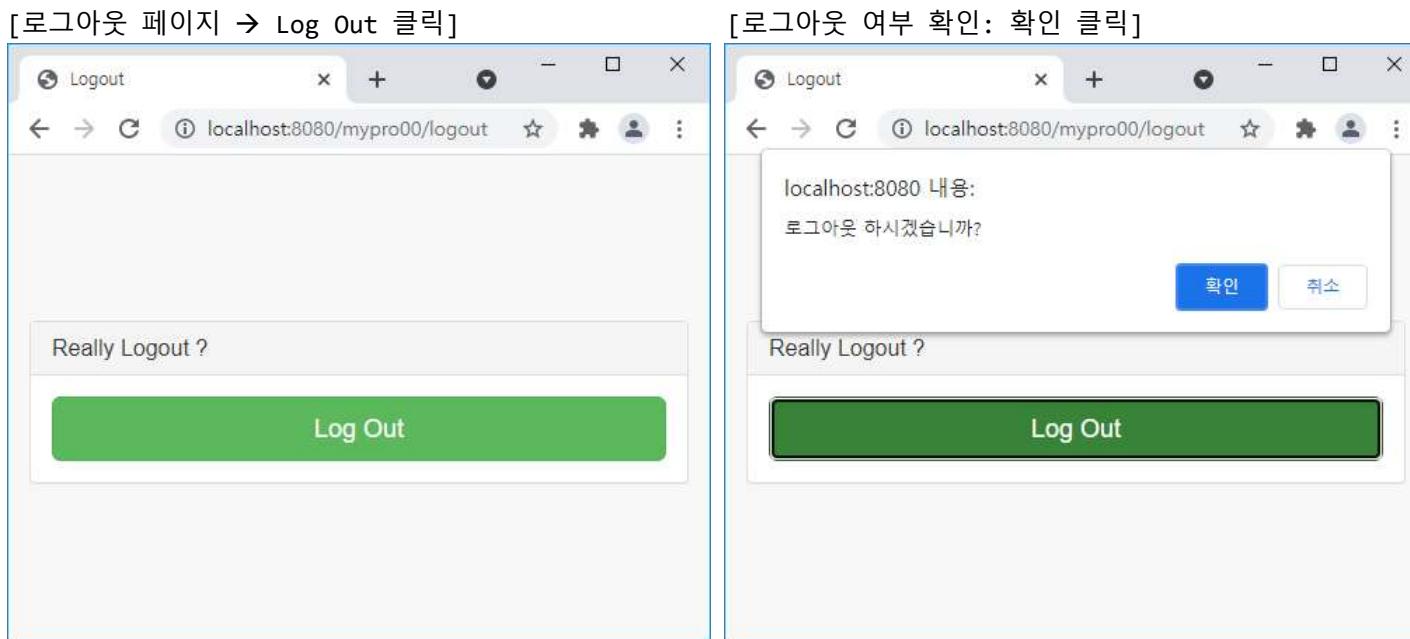
[로그인]



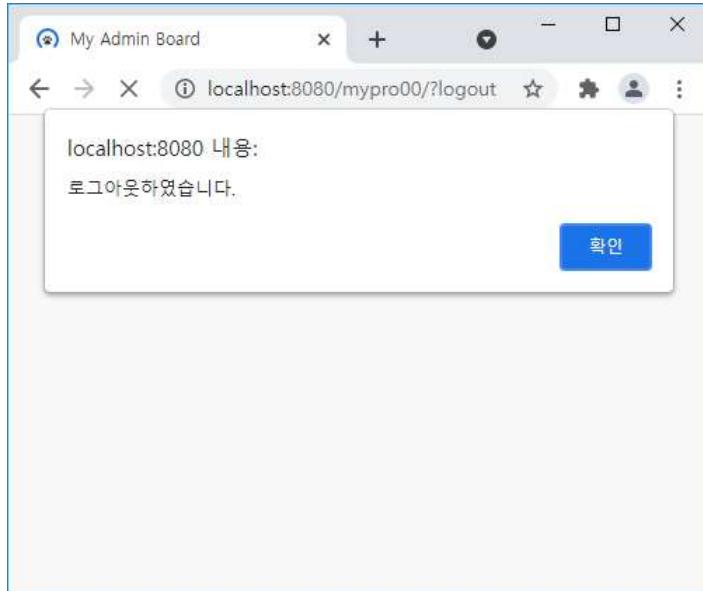
[로그인 상태: Sign out 만 표시]



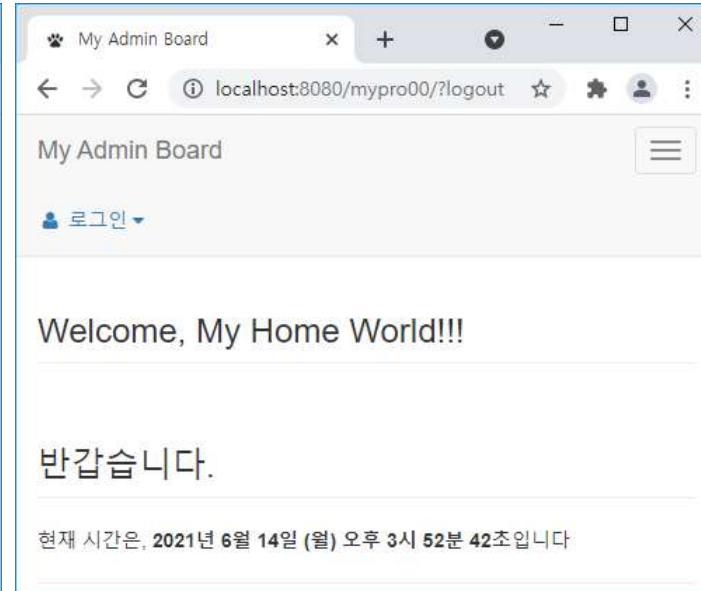
→ 로그인 상태에서, 로그아웃 시 테스트 시 메시지 표시 테스트(앞의 테스트에서 계속 이어짐)



[/mypro00/?logout 이동, 로그아웃 메시지 표시됨]



[로그아웃 후, /mypro00/?logout 화면]



[문제] "로그아웃하였습니다" 메시지가 표시되지 않는 경우, 이유를 해결해보시기 바랍니다.

→ 위의 그림에서 웹 브라우저에 표시된 URL에 힌트가 있습니다.

[답] security-context.xml 파일에 정의된 <security:logout> 요소의 logout-success-url 속성에 설정된 값을

"//"에서 "/?logout"로 변경하면 home.jsp로 생성된 HTML 내용을 웹브라우저가 전달받을 때,  
alert()에 설정한 메시지가 표시됩니다(아래 참고).

```
<security:logout logout-url="/logout" logout-success-url="/?logout" />
```