**목차**

**1. 프로젝트 환경설정1**

1.1 프로젝트 생성2

1.2 라이브러리 2

1.3 View 환경설정 2

1.4 빌드 및 실행 2

**2. 스프링 웹 개발 기초4**

2.1 정적 콘텐츠2

2.2 MVC와 템플릿 엔진2

2.3 API2

**3. 회원 관리 예제 – 백엔드 개발4**

3.1 비즈니스 요구사항 정리2

3.2 회원 도메인과 리포지토리 만들기2

3.3 회원 리포지토리 테스트 케이스 작성2

3.4 회원 서비스 개발2

3.5 회원 서비스 테스트2

**4. 스프링 빈과 의존관계4**

4.1 컴포넌트 스캔과 자동 의존관계 설정2

4.2 자바 코드로 직접 스프링 빈 등록하기2

**5. 회원 관리 예제 – 웹 MVC 개발4**

5.1 회원 웹 기능 – 홈 화면 추가2

5.2 회원 웹 기능 - 등록2

5.2 회원 웹 기능 – 조회2

**6. 스프링 DB 접근 기술4**

6.1 H2 데이터베이스 설치2

6.2 순수 JDBC2

6.3 스프링 통합 테스트2

6.4 스프링 JDBC Template2

6.5 JPA2

6.6 스프링 데이터 JPA2

**7. AOP4**

7.1 AOP가 필요한 상황2

7.2 AOP 적용2

**8. 다음으로4**

# **1. 프로젝트 환경설정**

## **1.1 프로젝트 생성**

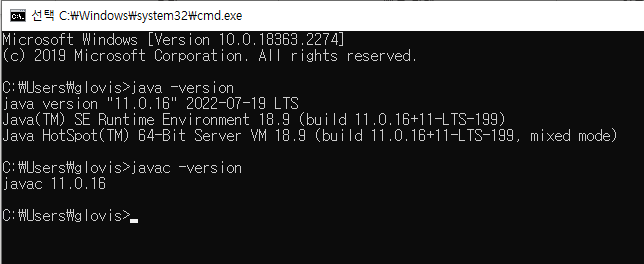
**■개발환경**

1. **IDE**

: Eclipse or **IntelliJ Community 버전 설치**

1. **JAVA**

: JAVA SE 11 (JDK 11.0.16) // 설정완료

****

* JDK 설치 참고 사이트 : [자바(JAVA) 11 설치 및 환경설정 | 자바 11 환경 변수 (velog.io)](https://velog.io/@sqk8657/%EC%9E%90%EB%B0%94JAVA-11-%EC%84%A4%EC%B9%98-%EB%B0%8F-%ED%99%98%EA%B2%BD%EC%84%A4%EC%A0%95-%EC%9E%90%EB%B0%94-11-%ED%99%98%EA%B2%BD-%EB%B3%80%EC%88%98)
* 자바 환경변수 설정 참고 사이트 : [[환경변수] JAVA\_HOME/CLASSPAT.. : 네이버블로그 (naver.com)](https://blog.naver.com/winter6120/221000738717)

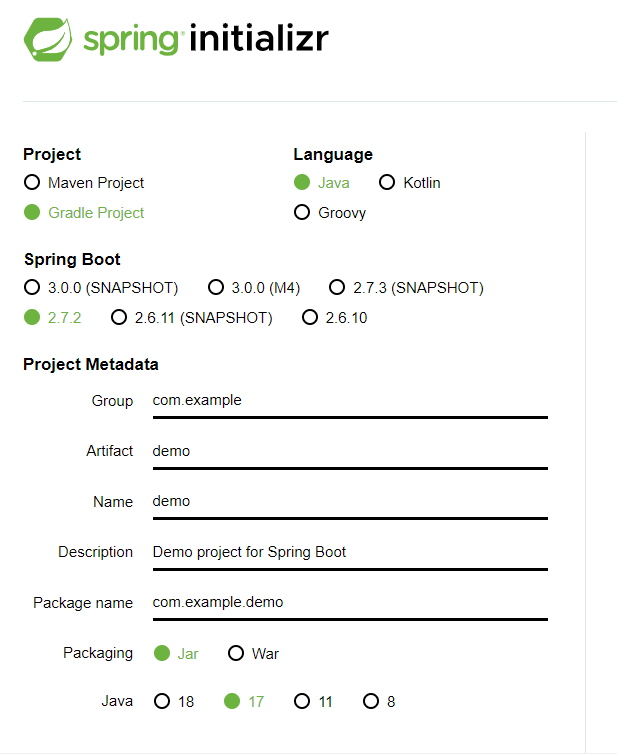
: JAVA\_HOME,

CLASSPATH,

PATH

1. 스프링 부트 스타터 사이트 이동> 스프링 프로젝트 생성

<https://start.spring.io>

****

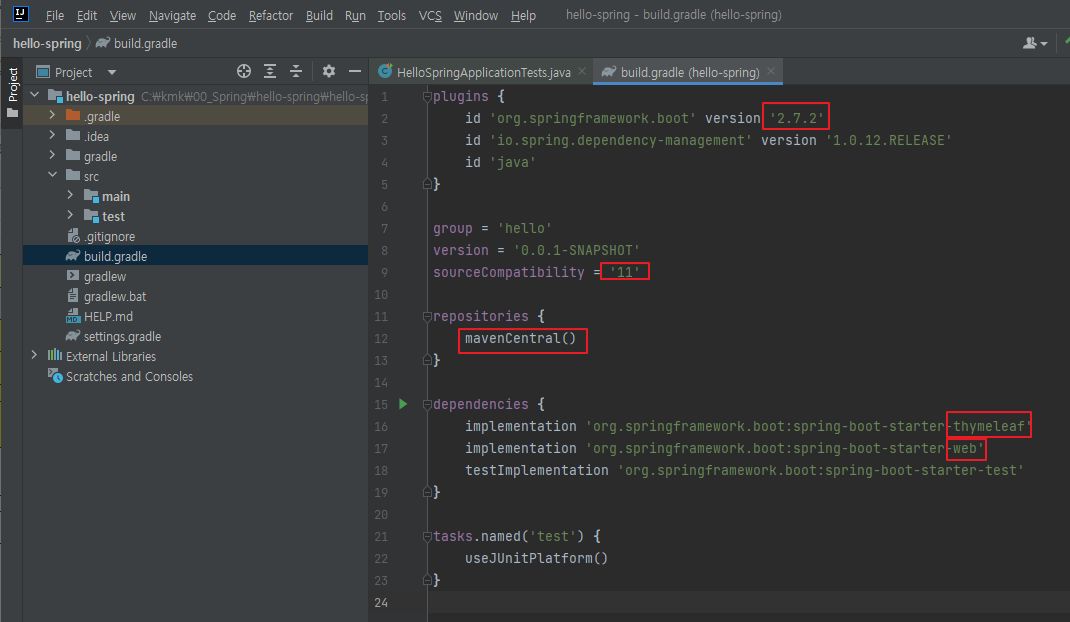
* Group : 보통 기업명
* Artifact : Build 되어 나오는 결과물 (프로젝트명)

****

* Dependencies : 스프링 프로젝트에서 어떤 라이브러리를 불러와 사용할 것인지 사전에 선택
  + [WEB] Spring Web
  + Thymeleaf : HTML 생성 엔진

1. **IntelliJ (설치완료)**

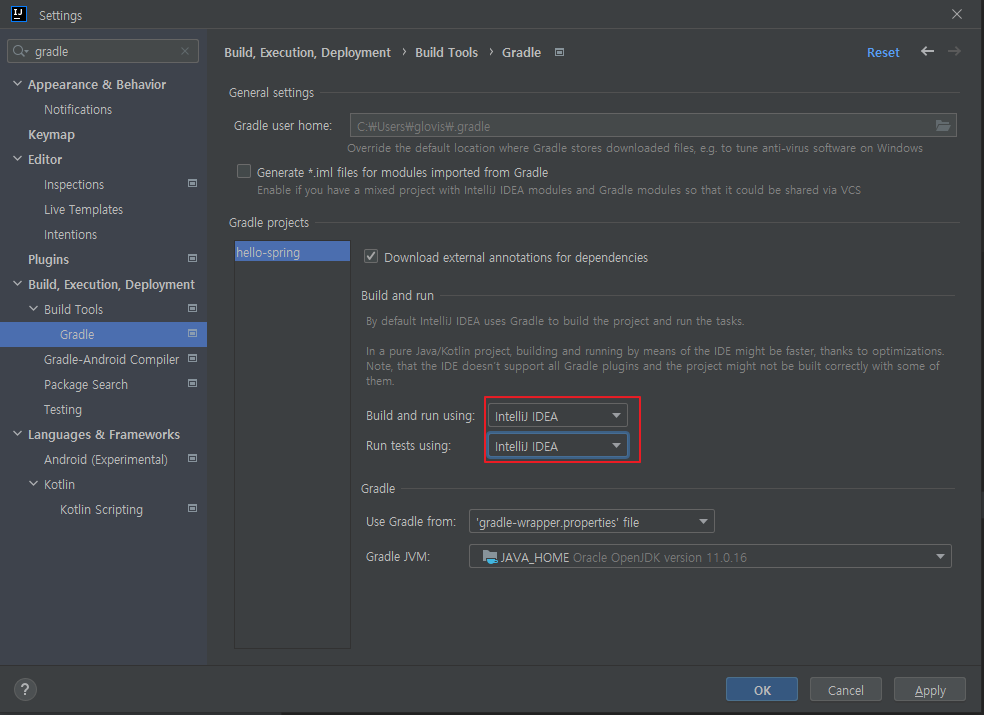
* 최초 프로젝트 생성 시 Gradle 자동 다운로드됨(네트워크 연결 필요)
* src 하위에 main, test 폴더가 자동으로 생성되는데 최근 개발 트렌드상 테스트가 중요한 부분으로 대두되어 IntelliJ 에서 test 폴더까지 자동으로 생성하게 셋팅되어 있음
* Build.gradle

****

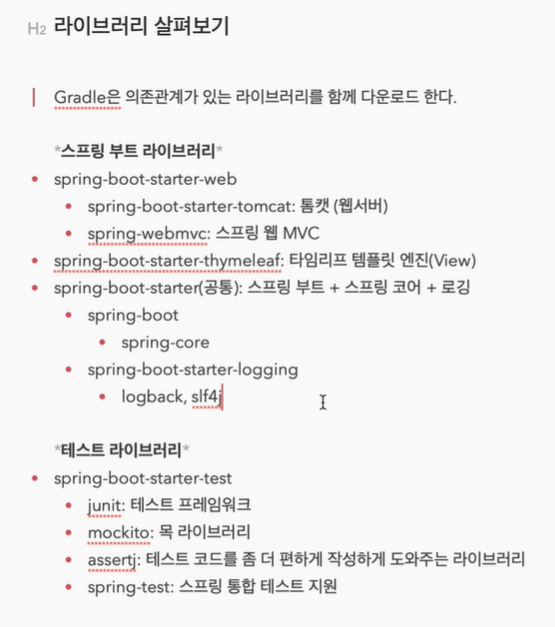
* mavenCentral() : dependencies 와 같은 라이브러리를 다운받은 저장소를 설정해준것
* Build.gradle, .gitignore 와 같은 파일은 start.spring.io 에서 자동생성되어 만들어짐

: 과거에는 해당 파일을 일일히 생성하여 소스 넣어줬어야 함

* Build & Run setting : Gradle 에서 IntelliJ 로 변경
* Gradle을 거치지 않고 IntelliJ 내부에서 Java를 실행하기 때문에 조금 더 빨리 실행됨

****

## **1.2 라이브러리**

****

\*Gradle : Compile, Test, Packaging, Deply & Run 작업을 자동화 시켜주는 Build Tool

**\*스프링에서의 Gradle?**

**- Compile : Java파일을 바이트 코드로 변환**

**- Test : App이 제대로 동작할지에 대한 테스트를 지원(Unit, UI 테스트 등)**

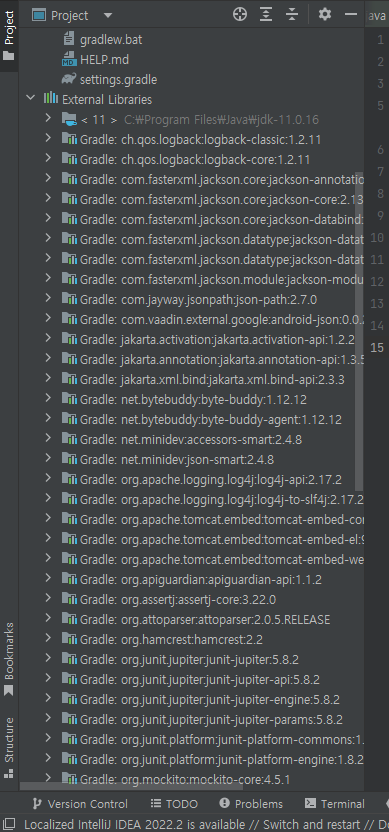
**- Packaging : 코드를 패키징해 aab 파일이나 apk 파일로 만드는 것**

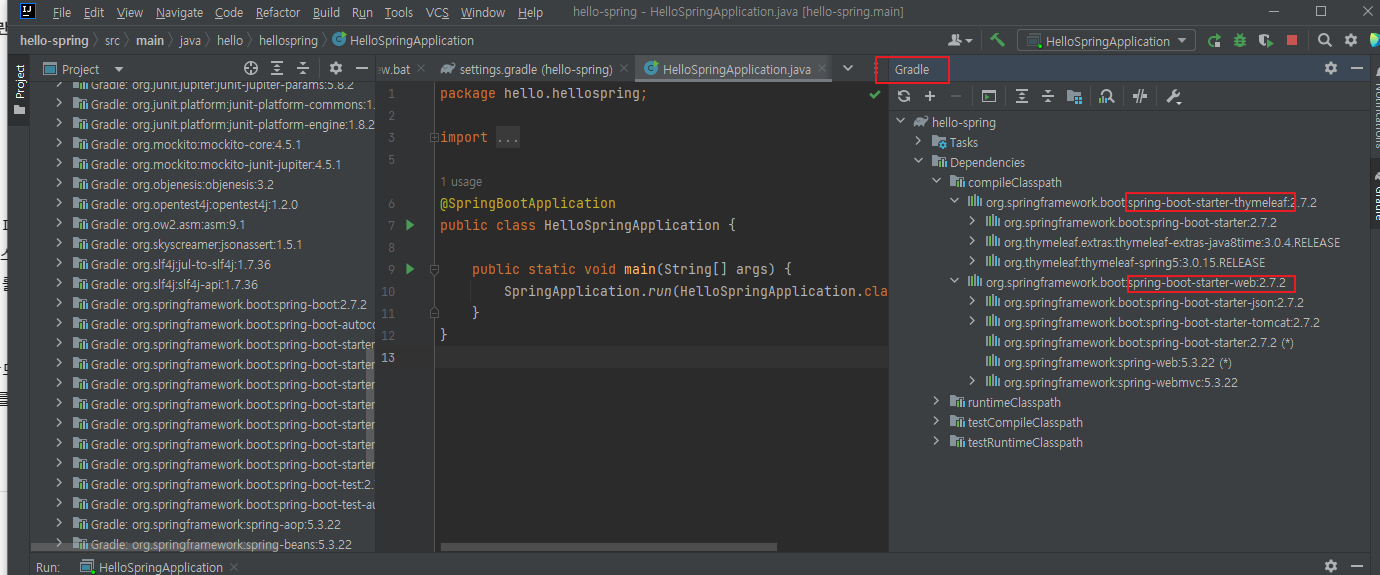
**- Deploy & Run : 코드를 App 으로 패키징해서 실제 기기에 넣어 실행할 수 있도록 하는 것**

**\*Gradle의 강점**

1. **직관적인 코드와 자동완성**
2. **다양한 Repository 사용 가능**
3. **각 작업에 필요한 라이브러리들만 가져오는 작업**

* 의존이 필요한 라이브러리들의 상호관계를 파악하여 불러옴

****

****

* 과거에는 web/was 서버에 tomcat 설치/셋팅 후 자바코드를 밀어넣는 방식이었으나 요즘은 개발툴(Spring 등) 내에 web/was 서버가 내장되어 있어 좀 더 편리하게 개발이 가능함

(메인 메서드 실행 시 톰캣서버 자동실행)

## **1.3 View 환경설정**

1. 웰컴페이지 생성(정적 페이지)

* Src\resources\static\index.html
* html 페이지 작성 후 톰캣 재기동/반영

* 스프링부트 템플릿 엔진
  + FreeMarker
  + Groovy
  + Thymeleaf
  + Mustache

위 4개 자동생성

(추가진행)

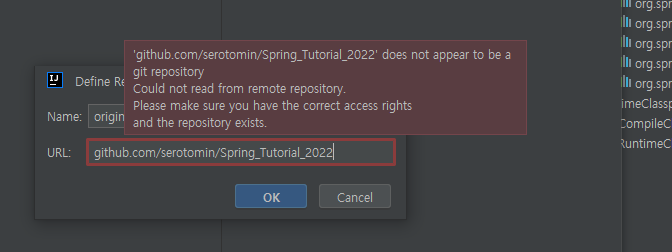
* Git 설치 & Github 연동

[[IntelliJ] IntelliJ + Github 연동하기 (tistory.com)](https://goddaehee.tistory.com/249)

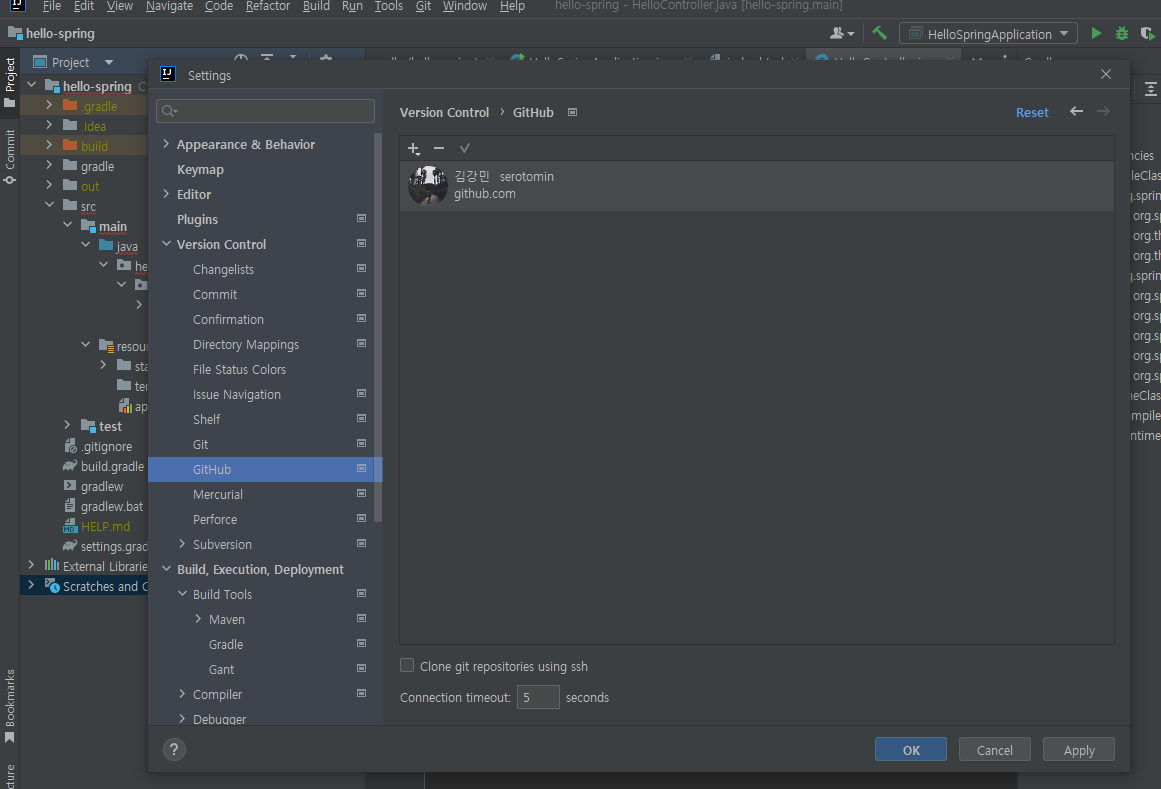
* Git 환경설정

[[IntelliJ] IntelliJ + Github 연동하기 (tistory.com)](https://goddaehee.tistory.com/249)

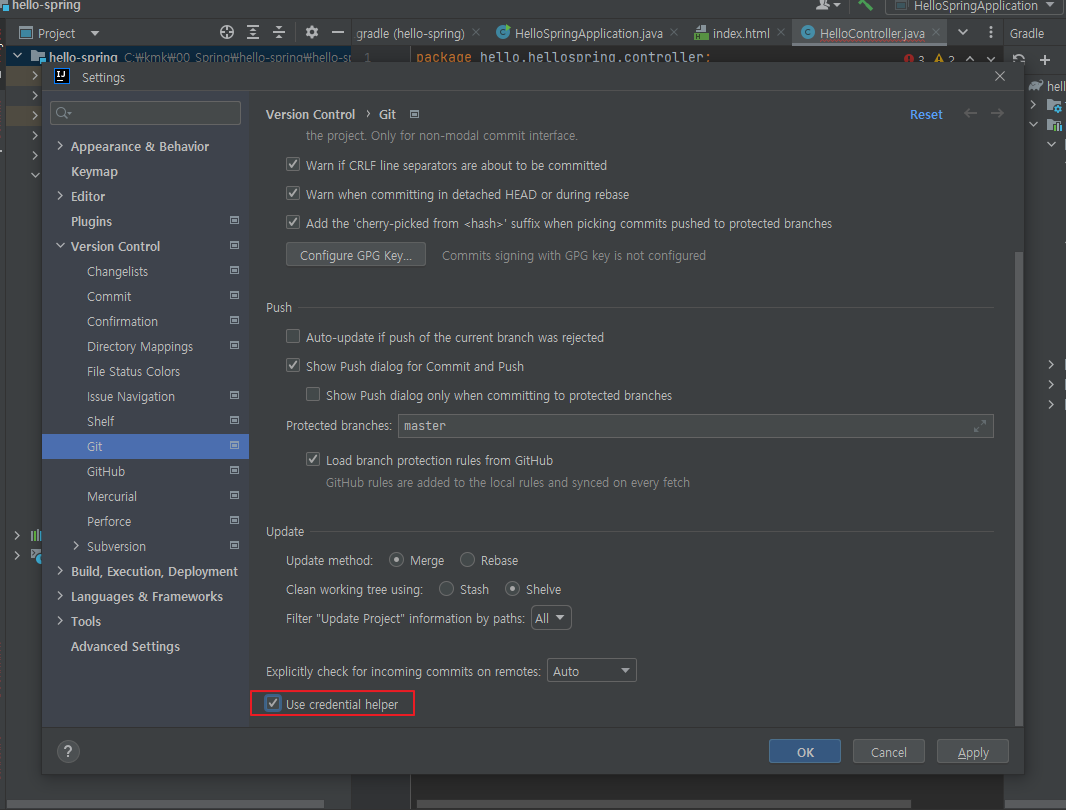
[문제]

****

GIT-PUSH 기본 저장소 지정 : 깃허브에 저장소를 퍼블릭으로 생성하였음에도 연동이 안됨

****

GitHub 계정 추가 후 확인 / 실패

****

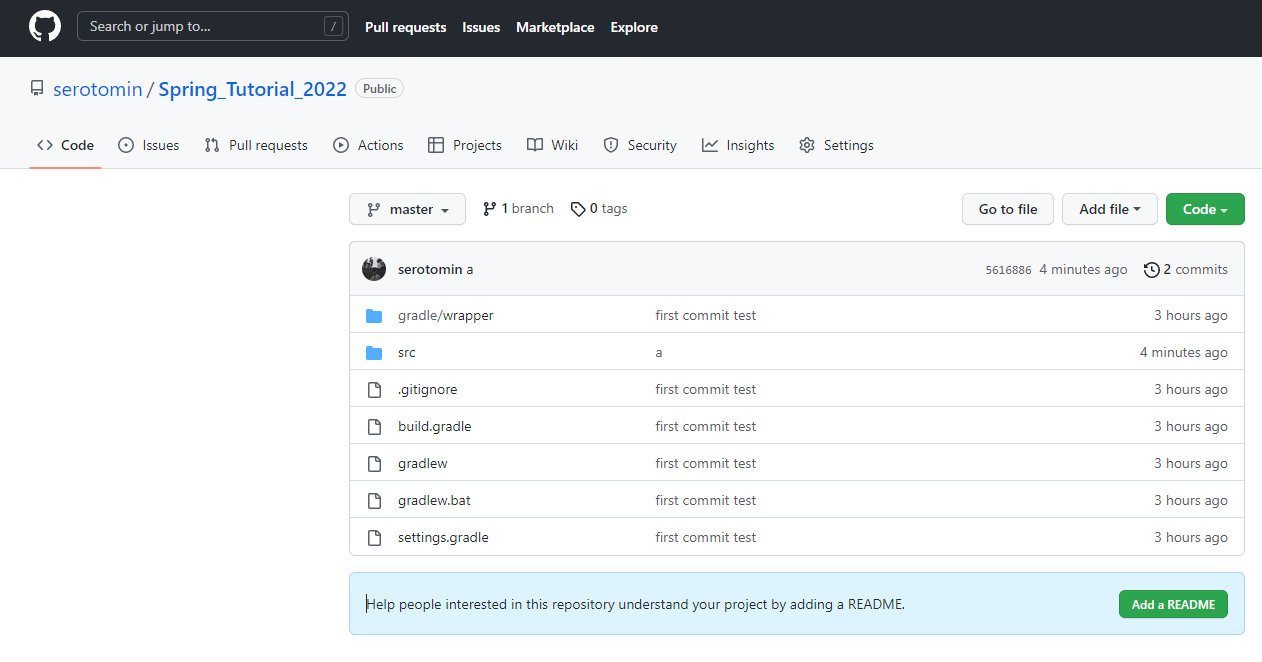
Settings > Version Control > Git > Use credential helper 체크

로도 안됨

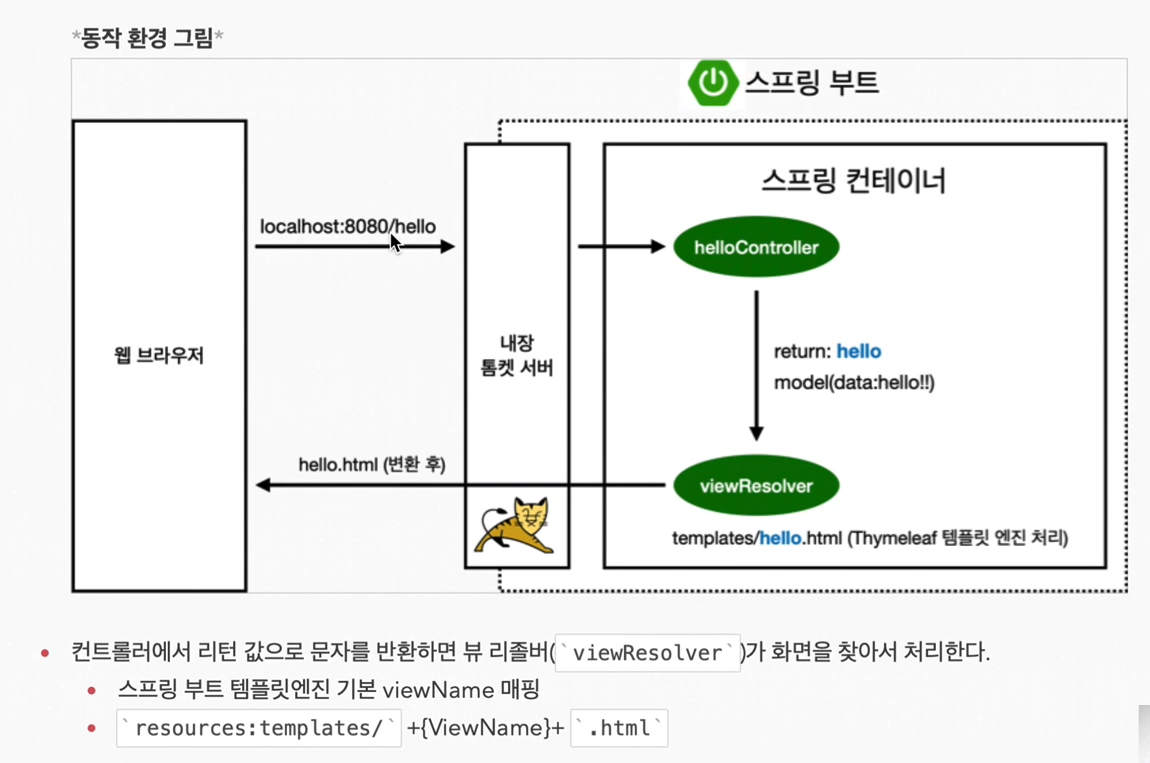
아래 명령으로 해결

Git config --global http.sslVerify false

출처: <https://tweety1121.tistory.com/entry/unable-to-access-ssl-certificate-problem-self-signed-certificate-in-certificate-chain-intellij> [Pli's 개발일기:티스토리]

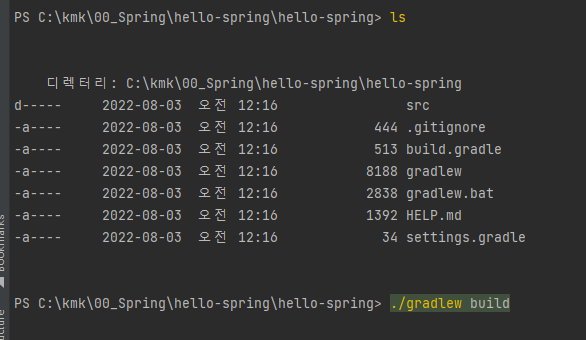
****

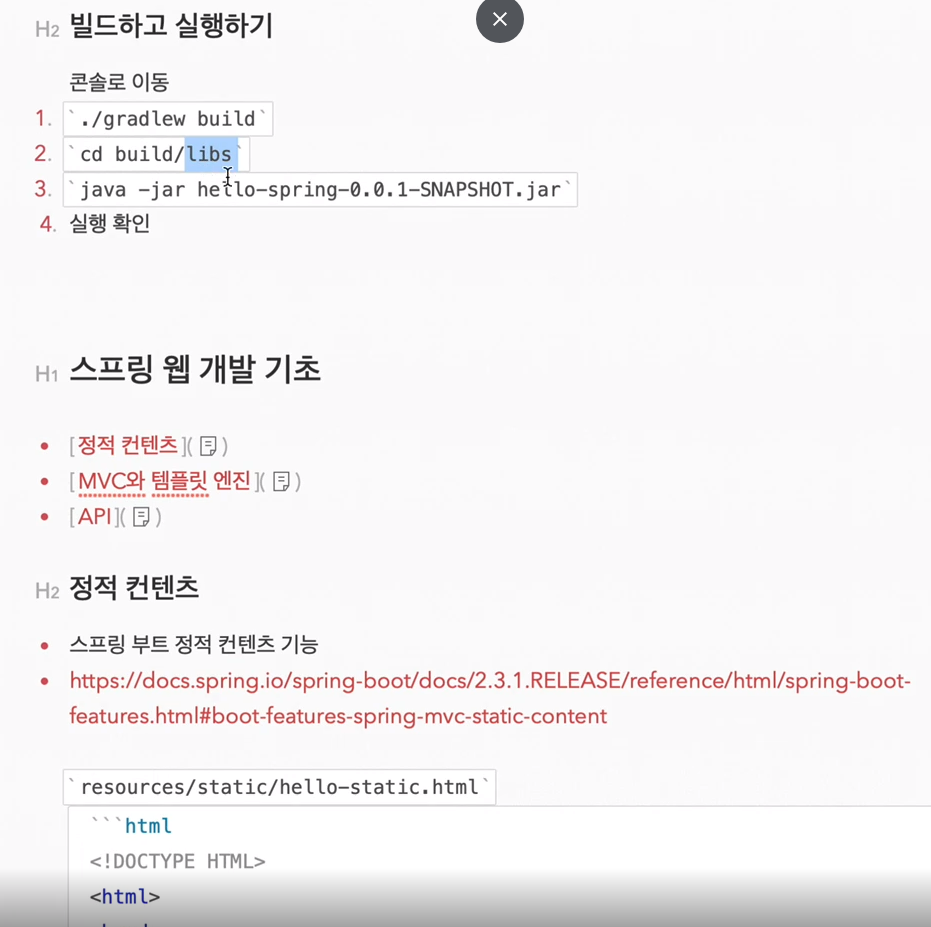
소스코드 github push 완료

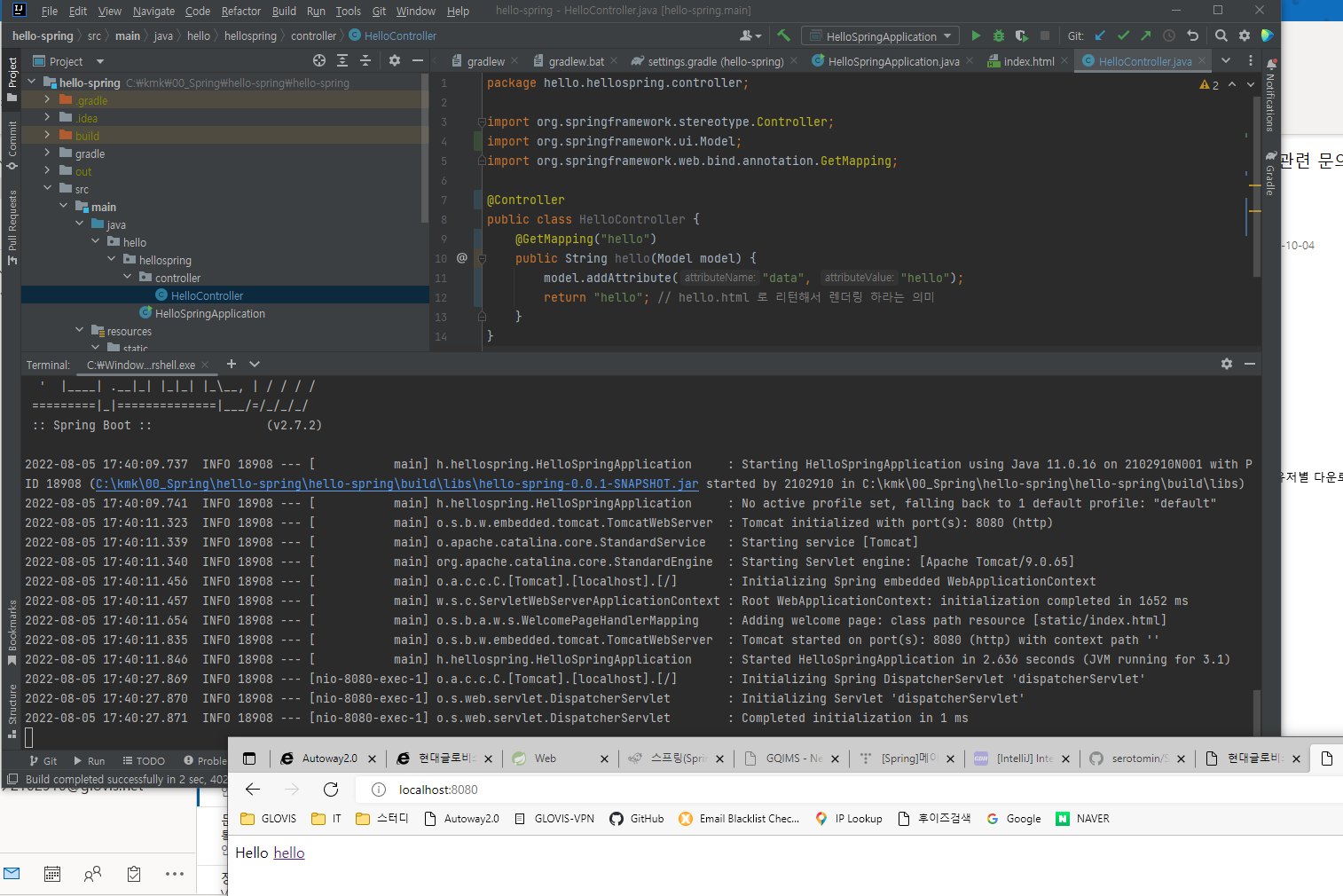
****

{ViewName} == hello 로 치환

## **1.4 빌드 및 실행**

****

****

****

# **2. 스프링 웹 개발 기초**

## **2.1 정적 콘텐츠**

* 1. 정적 콘텐츠(Static Content)
  2. (예시)서버에서 하는 것 없이 파일을 웹브라우저에서 그대로 내려주는 것
  3. 스프링부트는 기본적으로 정적 콘텐츠 기능을 제공함
  4. Resources\static\hello-static.html
  5. 시스템 생성 대체 텍스트:
     *정적 컨텐츠 이미지* 
     스프링 부트 

* 1. MVC와 템플릿 엔진
  2. 템플릿 엔진 예 : JSP, PHP
  3. 서버에서 html을 변형 후 클라이언트에게 전달하는 것
  4. API
  + (예시)최근에는 JSON이라는 데이터 포맷으로 전달해줌
  + Vue, react, 서버끼리 통신할 때 등등

## **2.2 MVC와 템플릿 엔진**

■MVC pattern

* Model, View, Controller 을 분리
* 시스템 생성 대체 텍스트:
  H MVC와 템플릿 엔진 
  MVC: Model, View, Controller 
  •Controller' 
  java 
  @Controuer 
  public class HelloController { 
  @GetMapping ( ) 
  public String helloMvc(@RequestParam("name") String name, Model model) { 
  model.addAttribute(••name", name); 
  return "hello—template"• 
  •View' 
  resources/template/heuo—temp html 
  •html 
  ehtml 
  e-body» 
  <9 th:text:" 'hello • + $1nameH*1ello! empty•z/p 
  c/bodp 
  e/html* 
  실행 
  I 
  http://localhost:8080/hello-mvc?name—spring 
* View : 화면을 그리는데 역량 집중
* Model, Controller : 비즈니스 로직, 설계 관련된 역량을 집중

* IntelliJ 팁
* [CTRL] + P : 파라미터 정보 확인

## **2.3 API**

* API
* Hello-string 메서드와 hello-mvc 메서드의 차이를 보면 페이지 소스만 봐도 알 수 있음

\*Hello-string : <http://localhost:8080/hello-string?name=spring> //html 태그 없이 문자만

\*Hello-mvc : <http://localhost:8080/hello-mvc?name=spring>!! //html 태그 그대로 조회

* Intelli J 팁
* [CTRL] + [SHIFT] + [ENTER] : 코드 끝부분 괄호닫기 전 입력하면 ; + 개행 자동으로 됨



* JSON 방식
* 소스확인 : <http://localhost:8080/hello-api?name=Spring>
* JSON 방식? : Key-Value로 이루어진 구조
* 시스템 생성 대체 텍스트:
  •@ResponseB0dy 시용 원리 
  콤켓 서버 
  @ResponseBOdy 를 사용 
  HTTP의 BODY에 문자 내용을 직접 반환 
  0 스프링 부트 
  스프링 컨테이너 
  @ResponseBody 
  HttpM essageCon `, erter 
  S:•lngCO 
  viewReso Iver 대신에 HttpMessageConverter 가 동작 
  기본 문자처리: 51 ringHtt*1essageConverter 
  기본 객지처리: MappingJackson2HttpHessageConverter 
  byte 처리 등등 기타 여러 HttpMessageConverter가 기본으로 등록되어 있음 
  참고: 를라이언트의 HTTP Accept 하더와 서버의 컨트를러 한한 타입 점보 물을 조합해서 
  HttpHessageConverter 가 선택된다. 더 자세한 내용은 스프링 MVC 감의너서 설명하겠다. 
* 스프링에서 hello-api 메소드 확인
* @ResponseBody 가 있으면 HTTP응답에 데이터를 넘겨야 된다고 판단하지만, Hello라는 객체가 있기 때문에 추가 판단을 하게됨
* 객체가 있으면 기본이 JSON 방식으로 데이터를 만들어서 HTTP응답에 리턴을 하게됨
* @ResponseBody가 있으면 viewResolver 대신에 HttpMessageConverter 가 동작하고 객체면 JsonConverter가 기본으로 동작하게 되고, 객체를 JSON 포맷으로 바꿈
* 스프링 기본 객체처리 : MappingJackson2HttpMessageConverter 라이브러리 사용

* 정적 콘텐츠 : 파일을 그대로 내림
* MVC : Model, View, Controller 분리하여 클라이언트에게 전달(템플릿 엔진 방식)
* API : 일반적으로 객체 반환

시스템 생성 대체 텍스트:
*MVC, 템플릿 엔진 이미지* 
localhost:8080/hello-mvc 
0 스프링 부트 
스프링 컨테이너 
return: hello-template 
templates/hello-template.html (Thymeleaf 팅를릿 연진 처리) 
월 브라우저 
내장 
통켓 서버 
HTML (변환 후) 

* 웹브라우저에서 로컬호스트 8080에 hello-mvc를 넘기면 스프링부트 안에 내장 톰캣서버로 넘기고 다시 스프링 컨테이너로 넘김
* 스프링 컨트롤러 내 해당 메서드를 호출해줌
* 스프링이 viewResolver (화면관련 동작, 추후 강의 확인) 뷰 리졸버가 hello-template.html 찾아서 thymeleaft 템플릿 엔진에 내장된 기능으로 화면 렌더링을 함

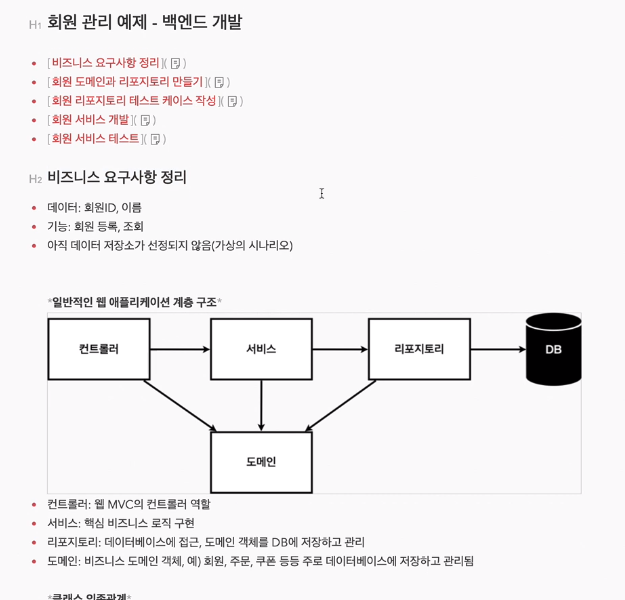
# **3. 회원관리 예제 – 백엔드 개발**

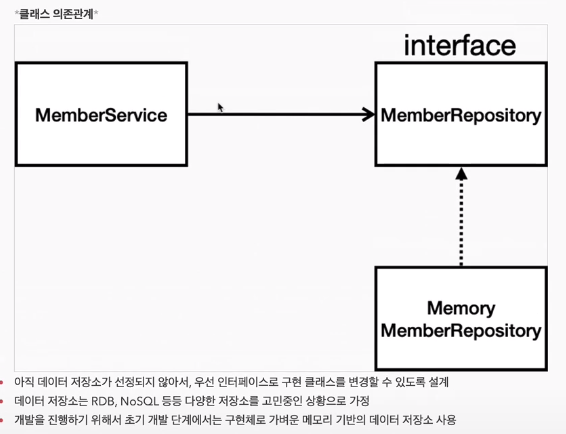
## **3.1 비즈니스 요구사항 정리**

- Controller : 외부 요청 수집

- Service : 비즈니스 로직 정립

- Repository : 데이터 저장





- DBMS 가 정해지지 않았다는 가정하에 진행하기 때문에, 단순 메모리 형태로 선개발 후 적용하는 방향으로 진행

## **3.2 회원 도메인과 리포지토리 만들기**

1) hello.hellosping.domain 패키지 생성

- hello.hellospring.domain 패키지에 Member 클래스 생성

: getter, setter 자동 생성(Generate)

2) hello.hellospring.repository 패키지 생성

- repository.MemberReposirory 인터페이스 생성

3) repository.MemoryMemberRepository 클래스 생성

생성

- [CTRL] + I or [ALT] + [ENTER] : Implement 가능한 메서드 목록을 확인하여 구현하기 위한 코드를 자동 생성

- [CTRL] + [SPACE] : 라이브러리 import

## **3.3 회원 리포지토리 테스트 케이스 작성**

1. 자바 - Junit 프레임워크로 테스트 코드 작성/실행

1) test.java.hello.hellospring.repository 패키지 생성

2) repository.MemoryMemberRepositoryTest 클래스 생성

- 다른곳에서 쓰이지 않기 때문에 public 클래스로 생성하지 않음

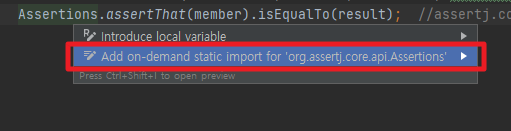
- @test : Import -> Junit Jupiter API



Assertions.*assertEquals*(result, member); // result, member가 같은지 체크 Junit.jupiter API 활용(Assertions) / 같으면 : 정상실행 , 다르면 : 오류발생

Assertions.*assertThat*(member).isEqualTo(result);

**[ALT] + [ENTER] : Assertions 클릭 후 alt + enter**



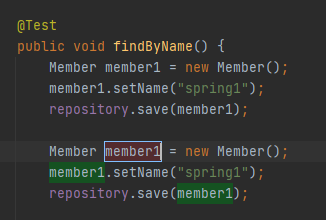
*assertThat*(member).isEqualTo(result);

위와 같이 간소화되고 아래처럼 static 으로 import됨

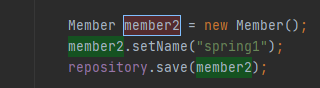
import static org.assertj.core.api.Assertions.\*;

**[SHIFT] + [F6]** : 아래처럼 코드 복붙 후 밑에 있는 변수 값을 일괄적으로 바꿀 때 사용

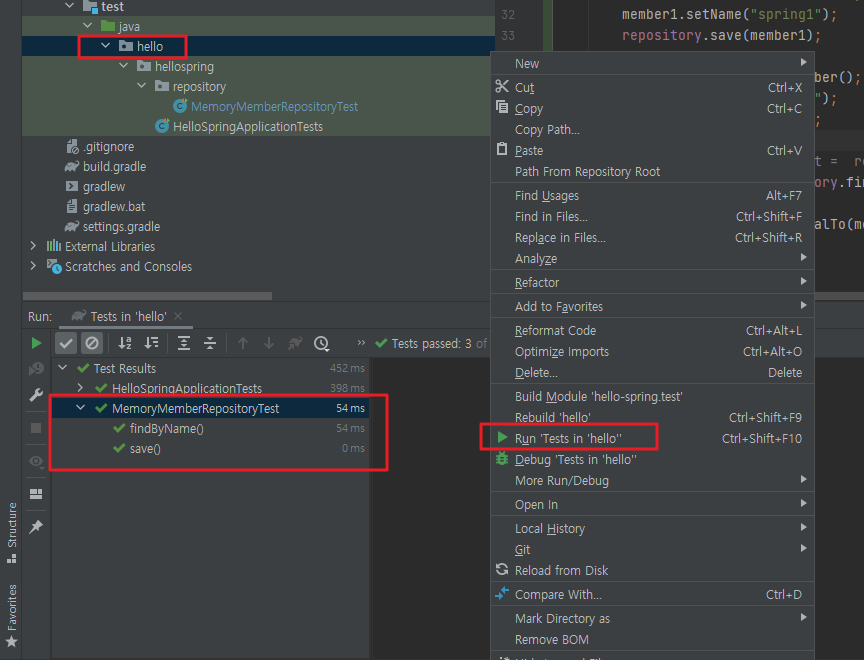
[적용 전]



[적용 후]

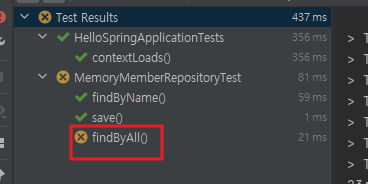


클래스 레벨에서 테스트



메모리 초기화 없이 클래스 전체 실행하게 되면 findByAll()은 에러가 뜸

**\*\*테스트는 서로 순서, 의존관계 없이 설계가 되어야 함(주의)**



\*메모리 초기화 코드 추가

1) main ~ MemoryMemberRepository 클래스에 clearStore() 메소드 추가

2) test ~ MemoryMemberRepository 클래스에 afterEach() 메소드 추가

\*테스트를 먼저하고 MemoryMemberRepository를 만들면?

-> 테스트 주도개발 = TDD(Test-Driben Development)

## **3.4 회원 서비스 개발**

1) hello.hellospring.service 패키지 생성

- 서비스 패키지 내에 생성되는 클래스, 메소드 네임은 비즈니스 용어 주로 사용(join, findMembers 등)

2) service.MemberService 클래스 생성

- 회원가입을 위한 클래스.

- 회원가입 전 id 중복여부 체크(findByName 활용)

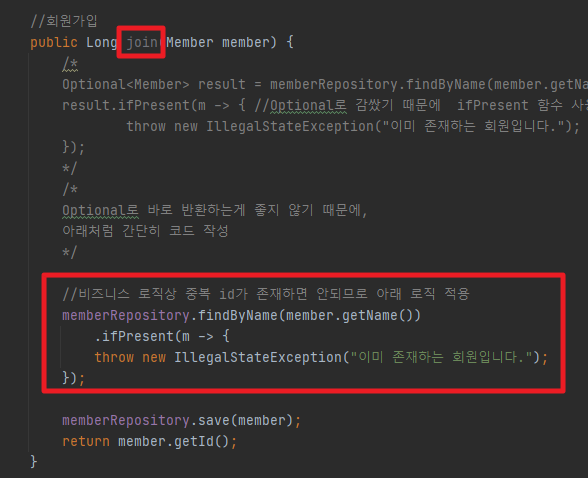
- 이상없으면 save 및 return

3) Extract Method (ctrl + alt + v -> Method 검색)

- validateDuplicateMember // Extract Method 생성

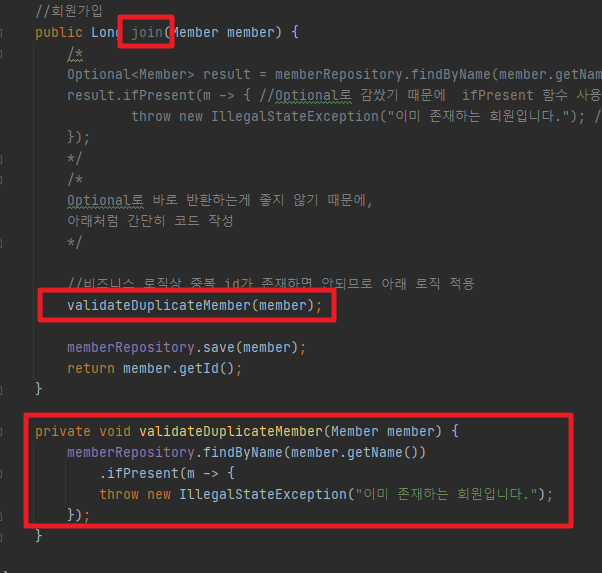
[생성 전]

- join 내부에 존재



[생성 후]

- join 과 validateDuplicateMember 별도 분리



**(참고) Optional 단축키 : [CTRL] + [ALT] + V**

[적용 전]

memberRepository.findByName(member.getName());

[적용 후]

Optional<Member> byName = memberRepository.findByName(member.getName());

**(참고) Refactor 단축키 : [CTRL] + [ALT] + V**

-

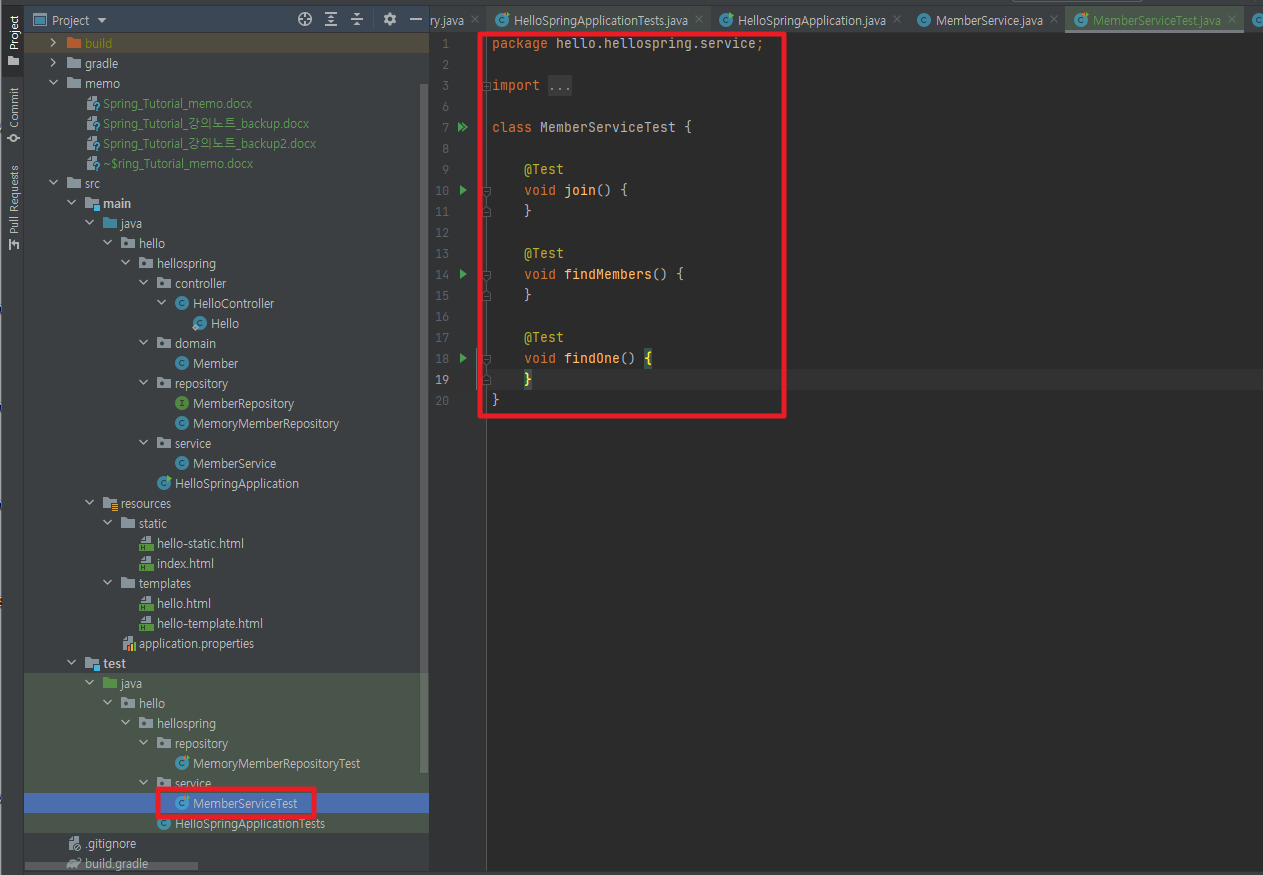
## **3.5 회원 서비스 테스트**

**1) MemberService 신규 테스트 생성**

**- [CTRL] + [SHIFT] + T**



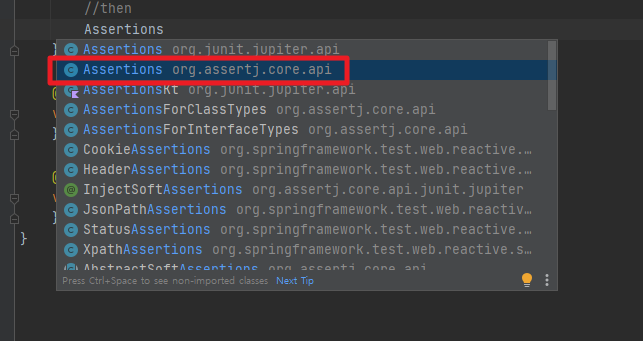
Test 경로에 자동생성됨



테스트는 외국인과 협업하는 경우가 아니라면 직관적으로 한국어를 기재하는 경우도 많음

2) given ~ when ~ then 으로 테스트

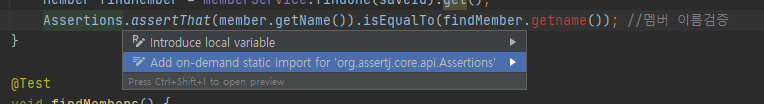
<https://hseungyeon.tistory.com/328>



\*Static Import : 클래스에 대한 인스턴스 생성없이 메소드 바로 사용  
**- 단축키 : [Alt] + [Enter]**

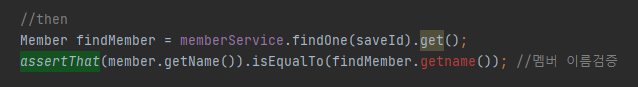
<https://offbyone.tistory.com/283>

[적용 전]

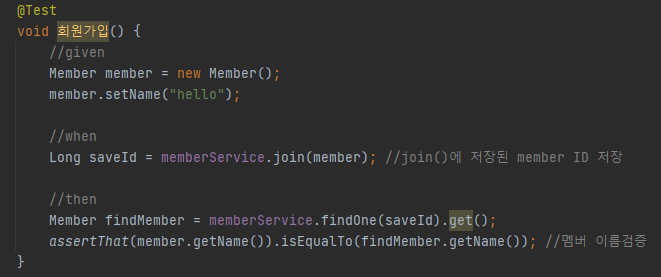


[적용 후]

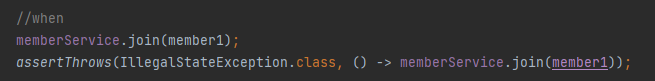




아래와 같은 테스트 코드는 너무 단순하다..테스트는 예외 플로우가 매우 중요하여 보통은 아래처럼 테스트 하지 않음



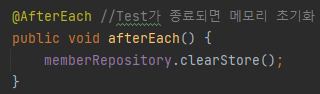
본 프로젝트의 회원가입 로직에서 중요한 핵심은 중복회원 검증인데, 해당 부분 예외처리도 정상적으로 되는지 아래 코드와 같이 간결하게 정리하여 테스트가 필요하다.



**(참고) Optional 단축키 : [CTRL] + [ALT] + V**

3) 멤버 메모리 초기화 로직 추가

- MemberRepository 생성 // 메모리 초기화



**(참고) Re Run : [SHFIT] + [F10]**

- 현재 MemberService에 있는 Repository와, 테스트케이스에 있는 Repository 인스턴스가 서로 다른데 각각 생성되는게 문제

3.5..정리

(1) @BeforeEach를 통해 MemoryMemberRepository 인스턴스 생성

(2) memberService 에 memberRepository 주입

-> 같은 memberRepository 하나의 memory에 생성

-> 외부에서 멤버 리포지터리 주입

-> 멤버 서비스 입장에서 같은 메모리 멤버 리포지토리 사용

**: DI(Dependency Injection) 의 일종임**

# **4. 스프링 빈과 의존관계**

## **4.1 컴포넌트 스캔과 자동 의존관계 설정**

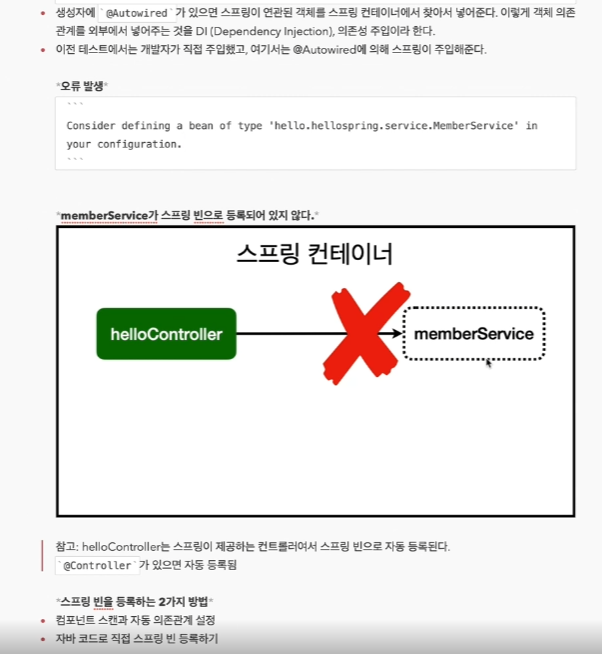
\*회원 컨트롤러가 회원서비스와 회원 리포지토리를 사용할 수 있게 의존관계 준비

1) MemberController 생성

@Controller를 생성해두면 스프링 컨테이너가 최초 동작 시 해당 객체 생성해서 스프링 빈이 관리하고 있음

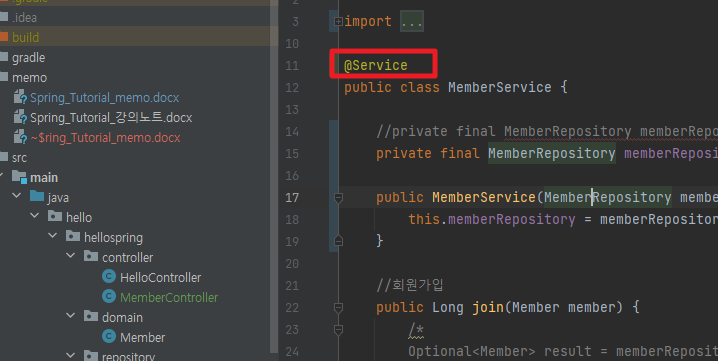


e.g) helloController // 스프링 컨테이너 내 스프링 빈



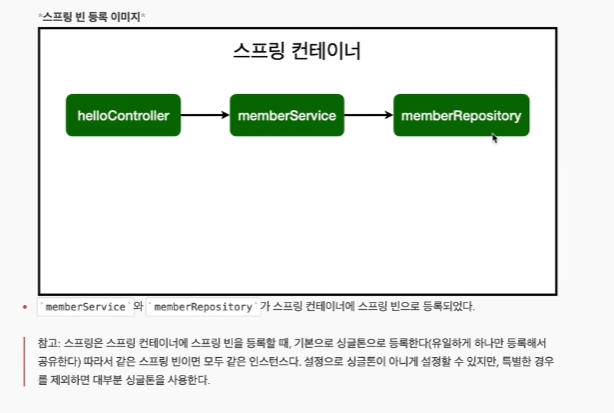
- MemberService 클래스에 @Service 추가 : 스프링이 스프링 컨테이너에 등록

- MemoryMemberRepository에 @Repository 추가



- MemberService 도 @Autowired

- DI : @Autowired를 생성자와 함께 쓰면 스프링에 등록되어 있는 멤버 서비스를 주입주고, Repository와도 연결.(컴포넌트 스캔 방식



- @Component > @Controller, @Service, @Repository Annotation 활용

- 위 그림처럼 @를 활용하여 컨트롤러, 서비스, 저장소를 선언하고 @Autowared로 연결해줌

- 기본적으로 hello.hellospring 패키지 내에서만 컴포넌트 스캔 대상이 됨



\*싱글톤 : 유일하게 하나만 등록/연결하여 사용(helloController, memberService, memberRepository)

## **4.2 자바 코드로 직접 스프링 빈 등록하기**

1) 멤버 서비스와 리포지토리에서 @Service, @Repository 어노테이션 제거

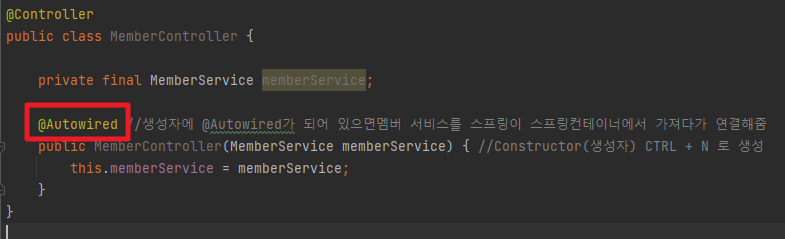
2) 컨트롤러는 유지

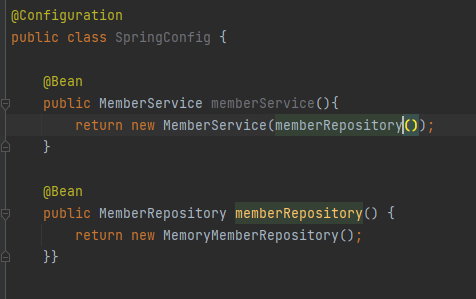
3) service.SpringConfig 클래스 생성

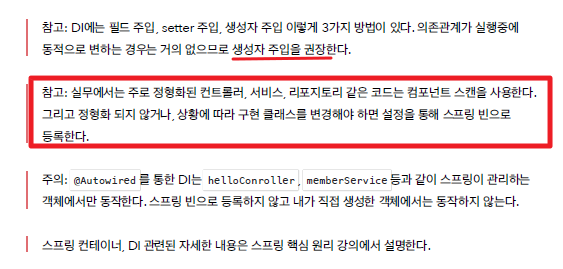
- @Configuration : 스프링이 시작할 때 해당 어노테이션 확인 후 컨테이너에 넣어줌

- @Bean : 인스턴스 스프링 빈에 등록

컨트롤러는 @Autowired 로 선언되어 있지만 그래도 스프링이 @Bean로 등록한 MemberService를 찾아 연결해준다.







\*이후 강의들에서 MemberRepository를 DB로 변경해야 되기 때문에 스프링 빈으로 등록해놓음

# **5. 회원 관리 예제 – 웹 MVC 개발**

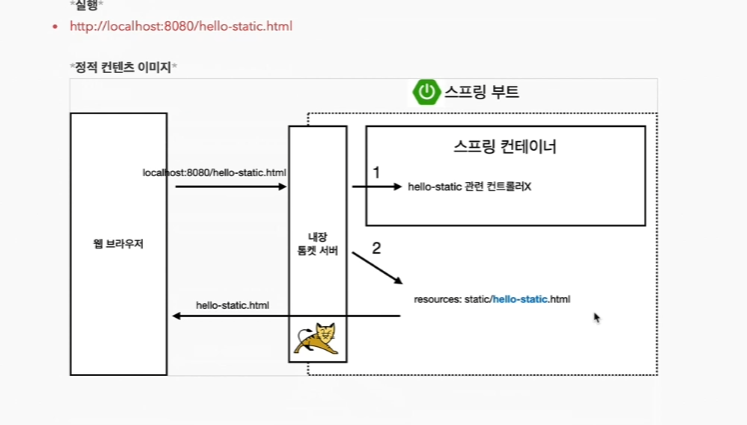
## **5.1 회원 웹 기능 - 홈 화면 추가**

\*컨트롤러를 통해 홈 화면 추가.

1) hello.hellospirng.controller.HomeController 생성

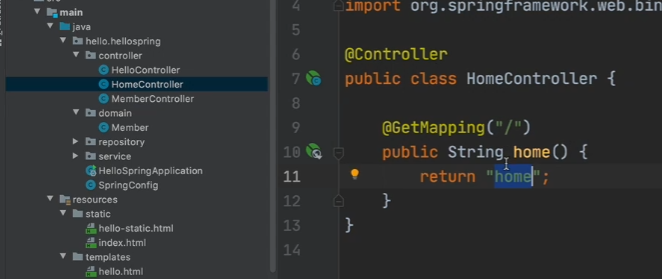
2) resources.template.home.html 생성

\*\*index.html 정적 페이지를 생성하였는데 home.html과 차이는 무엇인지?



위 그림을 보면 스프링 내 1)스프링 컨테이너에서 웰컴페이지 관련 컨트롤러가 있는지 확인한다. 없으면 2) 정적 페이지인 hello-static.html 파일을 찾는다.

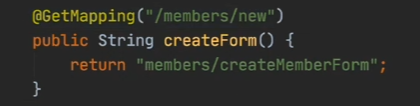
아래 이미지와 같이 HoemController가 생성되어 있기 때문에 정적페이지를 표현하지 않는다.



## **5.2 회원 웹 기능 - 등록**

\*등록/조회는 멤버컨트롤러 내에 맵핑을 추가함

1) controller.MemberController.CreateForm()

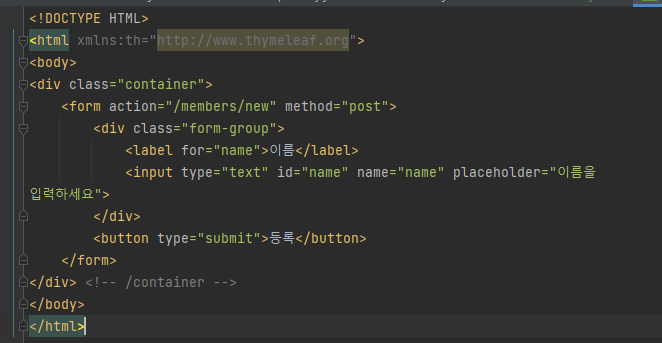


- GET방식은 주로 조회할 때 사용하는데 위의 코드는 localhost:8080/member/new URL 페이지(createMemberForm.html)만 표현해줌

2) template

- members 디렉토리 생성 : 회원가입 등록, 조회 등의 페이지 등 관련있는 페이지들 생성

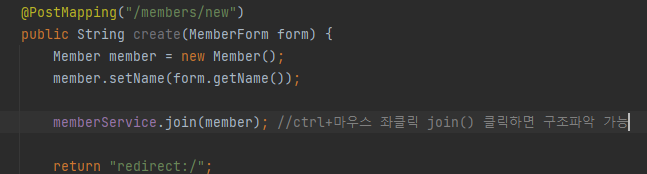
- createMemberForm.html 생성



- Name=**”name” //서버로 넘기는 key**

**-** input에 데이터 입력 후 버튼을 누르면 /member/new에 POST 방식으로 MemberController의 @PostMapping(“/members/new”)로 전달됨

- POST 방식은 주로 데이터를 폼에 넣고 서버에 넘길 때 주로 사용 (등록)



3) controller.MemberForm 생성

- [CTRL+N] : generate > Getter and Setter

- MemberForm의 name과 createMemberForm 의 name 매칭시켜줌

4) MemberController > create() 인스턴스 생성

## **5.3 회원 웹 기능 – 조회**

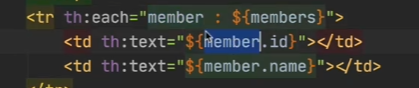
1) Membercontroller에 List()인스턴스 생성

2) MemberList.html 생성

- thymeleaft 템플릿 엔진이 본격적으로 동작하는 페이지

${members}

: ${ } 모델안에 있는 값을 꺼내어 오는 것 // Controller



순서대로 멤버 객체 값을 꺼내어 넣고, Member의 getId(), getName()을 통해 id, name을 반환받는다. (자바 프로퍼티 접근방식 == getter, setter)

@GetMapping("/members")  
public String list(Model model) {  
 List<Member> members = memberService.findMembers();  
 model.addAttribute("members", members);  
 return "members/memberList";  
}

위와 같이 모델안에 리스트 형태로 회원정보를 저장해둠

**\*\*현재까지는 PC메모리에 값을 저장해둠**

# **6. 스프링 DB 접근 기술**

## **6.1 H2 데이터베이스 설치**

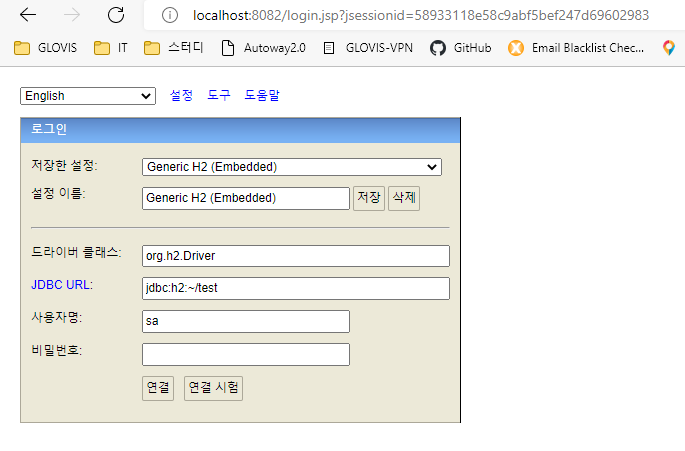
1) H2 DB v1.4.200 설치

- 추천도서 : JPA 프로그래밍

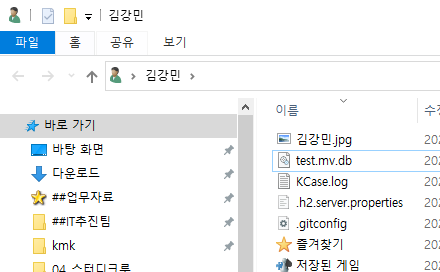
- JPA 요약 : 객체를 DB에 쿼리없이 저장/관리할 수 있음



2) H2 DB 설정 (웹 콘솔)



- 로컬에서 DB연결 후 test.mv.db 파일생성 확인

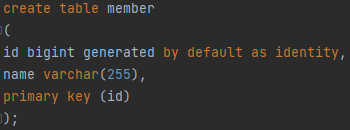


- 위의 JDBC 설정은 파일(test.mv.db)로 접근하는 방법이기 때문에 오류가 발생할 우려가 있음

- 아래와 같이 JDBC URL 설정 변경하여 소켓을 통해 연결

**jdbc:h2:tcp://localhost/~/test**

3) 테이블 생성



- 데이터타입 : bigint == long

- generated : 값을 셋팅하지 않아도 DB에서 ID 시퀀스를 자동으로 채번함(ID : 1씩 증가)

## **6.2 순수 JDBC**

1) build.gradle 파일에 jdbc, h2 데이터베이스 관련 라이브러리 추가

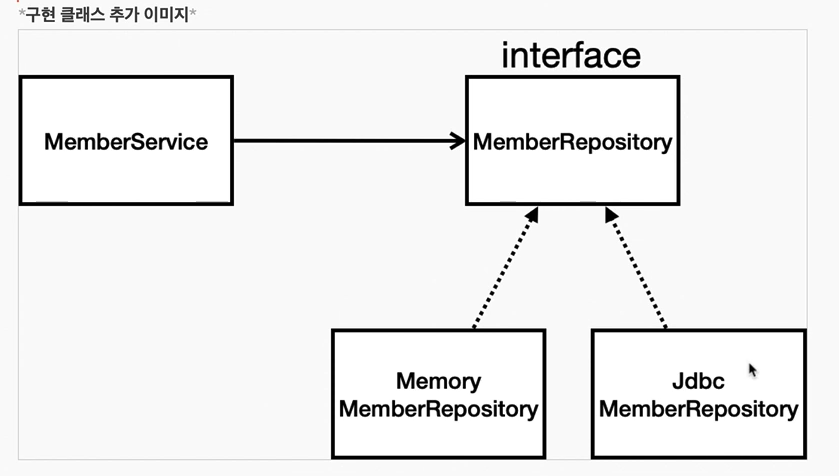
2) resources/application.properties에 스프링 부트 데이터베이스 연결설정 추가

3) JdbcMemberRepository 생성

: 구현체

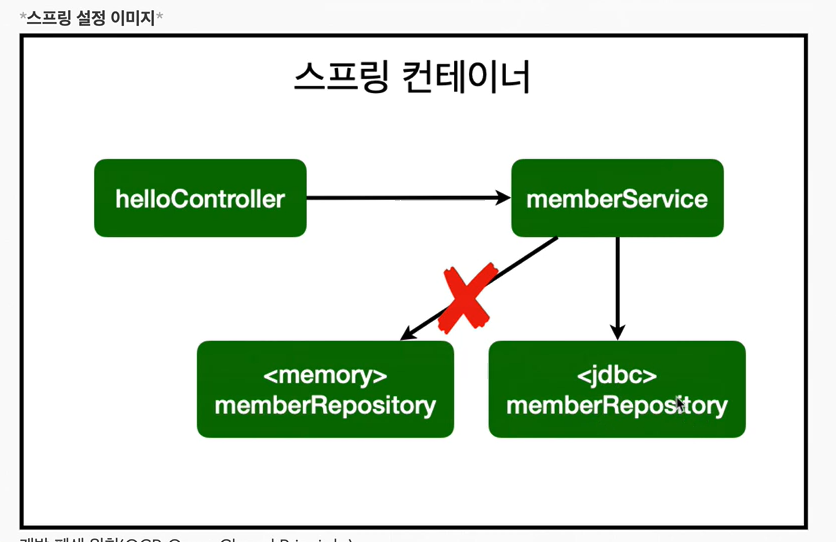
- Implement 자동생성 : ALT + ENTER

4) Spring Config 수정



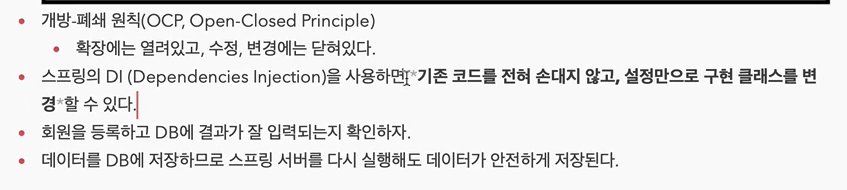
- MemberService -> MemberRepository 의존

- MemberRepository의 구현체 MemoryMemberRepository, JdbcMemberRepository



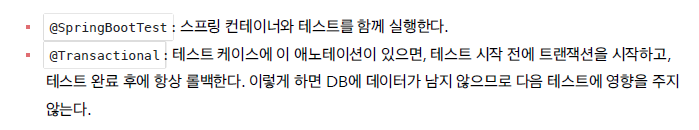
**- 스프링 컨테이너에서 JDBC버전의 memberRepository로 변경함**

**- 다형성의 개념을 잘 활용하면 주요기능을 변경해도 전체소스코드 수정 없이 변경이 가능하다.**



## **6.3 스프링 통합 테스트**

\*DB까지 연결하여 테스트



1) MemberServiceTest

- 기존에는 자바 JVM 안에서 테스트가 끝남(메모리 활용)

2) MemberServiceIntegrationTest

- MemberServiceTest 복사 후 생성

<<아래는 기존 멤버 서비스 테스트와의 코드 차이점>>

- @SpringBootTest, @Transactional 어노테이션 삽입

- @Trancsactional을 통해 테스트 이후 DB ROLLBACK 이 가능하다.

: 테스트 메서드 회원가입(), 중복\_회원\_예외() 동작 전/후 마다 트랜잭션 생성/종료

**\*\*MemberServiceTest와 같이 스프링 컨테이너 없이 단위단위로 쪼개서 테스트 하는게 바람직한 테스트 방법이다.**

## **6.4 스프링 JdbcTemplate (라이브러리)**

1) 순수 JDBC와 동일하게 build.gradle 파일에 implementation 해주면 됨

implementation 'org.springframework.boot:spring-boot-starter-jdbc'

2) 스프링 JdbcTemplate, MyBatis

: JDBC API에서 보았던 반복 코드를 대부분 제거해준다. 하지만 SQL은 직접 작성해야함

\*실무에서도 많이 사용

3)JdbcTemplateMemberRepository 생성

\*단축키 **[CTRL] + I or [ALT] + [ENTER]** : Implement 가능한 메서드 목록을 확인하여 구현하기 위한 코드를 자동 생성

\*단축키 **[ALT] + [ENTER] :** 람다 형태로 변환

- memberRowMapper 메서드 생성

: 리스트 형태로 멤버 객체 생성 및 id, name 저장

- jdbcTemplate 메서드 생성(findById, findByName, findAll)

: 결과값 memberRowMapper에서 반환받아 리스트형태로 result에 저장

4) Spring Config 수정

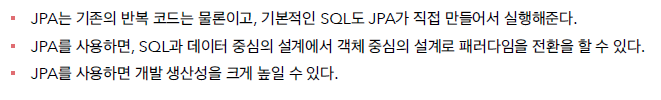
@Bean  
public MemberRepository memberRepository() {  
 //return new MemoryMemberRepository();  
 //return new JdbcMemberRepository(dataSource);  
 return new JdbcTemplateMemberRepository(dataSource);  
}

5) 테스트 해보기

- MemberServiceIntegraionTest 실행

\*\*순수JDBC코드와 비교해보기

## **6.5 JPA (Java Persistence API)**



1) build.gradle에 라이브러리 추가

//implementation 'org.springframework.boot:spring-boot-starter-jdbc' jpa라이브러리 추가 후 주석처리  
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

\*data-jpa : jdbc 및 jpa 모두 포함하기 때문에 이전에 정의한 jdbc 라이브러리는 주석처리하였음

2) application.properties 에 JPA 관련 설정 추가

spring.jpa.show-sql=true

- 위 설정으로 JPA의 SQL을 볼 수 있음

spring.jpa.hibernate.ddl-auto=none

- JPA를 사용하면 회원객체를 보고 테이블까지 만들지만, 하지만 현재 테이블이 생성되어 있고 그 테이블을 사용할 것이기 때문에 none으로 설정함(미사용)

라이브러리를 다운받고 설정까지 완료하면 엔티티를 맵핑해주어야 함

3) JPA는 표준 인터페이스 구현체는 여러 업체의 것이 있지만 본 수업에서는 hibernate 사용

4) JPA는 ORM(Object Relational database Mapping) 기술이다.

5) 엔티티 및 PK 추가

\* domain.Member

@Entity  
public class Member {  
  
 @Id @GeneratedValue(strategy = GenerationType.*IDENTITY*)

@GeneratedValue를 통해 시퀀스를 증가시키려면 위와 같이 IDENTITY를 추가하여야 함

6) JpaMemberRepository 생성

**\*생성자 생성 단축키 : [ALT] + [INSERT]**

public class JpaMemberRepository implements MemberRepository{  
  
 private final EntityManager em;  
  
 public JpaMemberRepository(EntityManager em) {  
 this.em = em;  
 }

- JPA는 EntityManager로 모든 동작을 한다고 이해

implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

- 위 gradle 설정으로 빌드 시 Springboot가 자동으로 EntityManager를 생성해줌

- 만들어진 EntityManager로 주입(Injection)을 받을 수 있도록 추가 작업을 해주면 됨

7) JpaMemberRepository 메서드 재정의

public Member save(Member member) {  
 em.persist(member);  
 return member;  
}

위와 같이 작성하면 JPA가 Insert쿼리 만들어서 DB에 집어놓고 ID까지 멤버에 set 해줌

**\*인라인 단축키 : [CTRL] + [N]**

public List<Member> findAll() {  
 return em.createQuery("select m from Member m", Member.class)  
 .getResultList();  
}

**- JPQL (Java Persistence Query Language) : 객체를 대상으로 쿼리문을 실행할 수 있음**

**- PK 기반이 아닌 건들은 JPQL 형태로 코드를 작성해줘야 함**

- “select m from Member m” : 멤버 엔티티 대상으로 쿼리 실행

-> select의 대상이 m (객체)인 점이 특징이다.

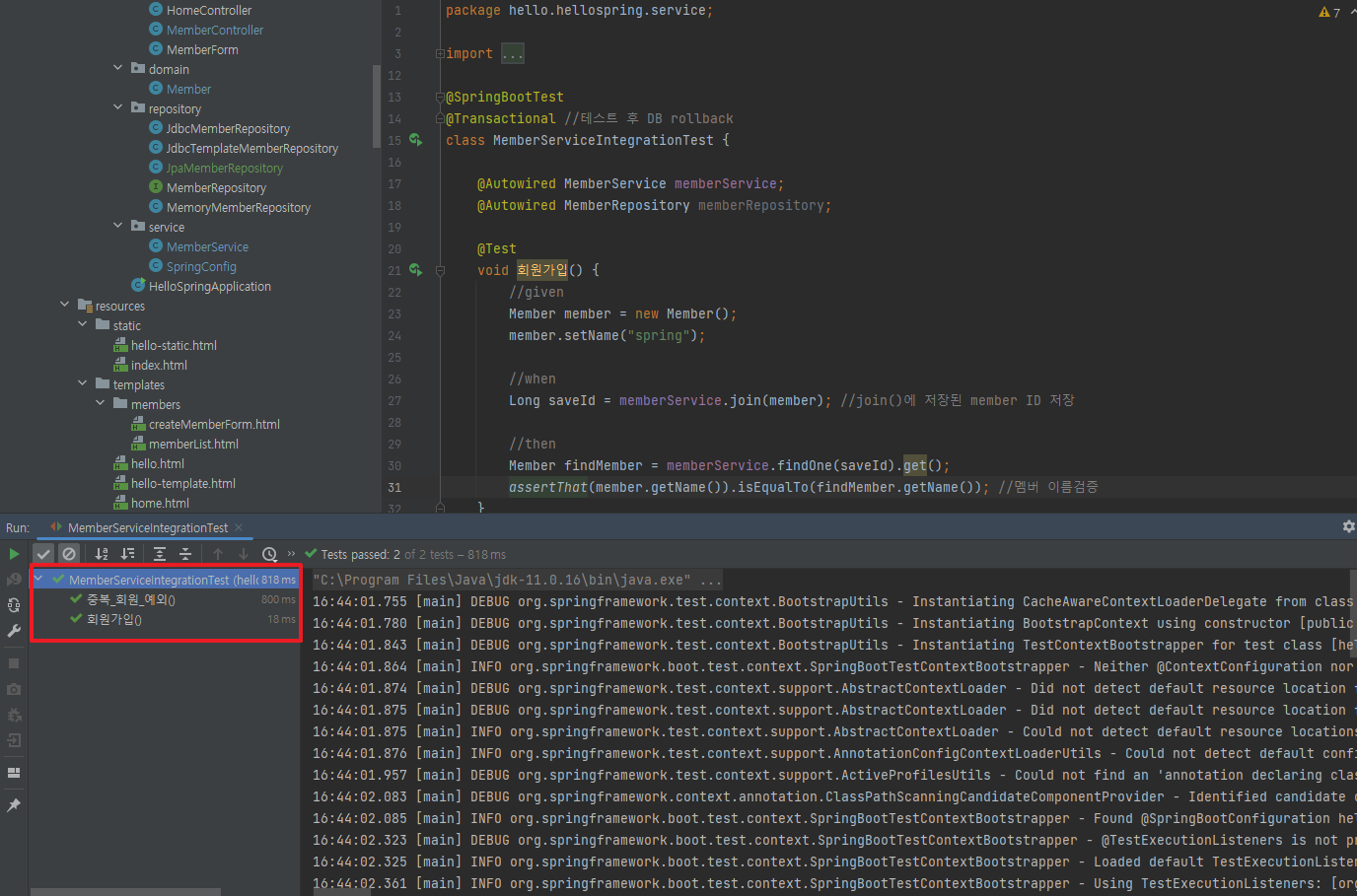
8) MemberService에 @Transactional 어노테이션 추가

- JPA 사용 시 꼭 @Transcational 어노테이션 선언을 해주어야 한다.

9) SpringConfig 수정

- EntityManager 생성 후 반환

10) 통합테스트 (MemberServiceIntegration)



- 테스트 성공



- hibernate(구현체)가 select 및 insert into 쿼리문 실행

- 정상적으로 실행 후 테스트 케이스 DB 롤백

## **6.6 스프링 데이터 JPA**

**\*학습순서 : JPA 학습 -> 스프링 데이터 JPA (JPA를 좀 더 쉽게 스프링에서 사용)**

1) SpringDataJpaMemberRepository 인터페이스 생성

- extends 로 다중상속(자바에서 인터페이스는 다중상속 가능)

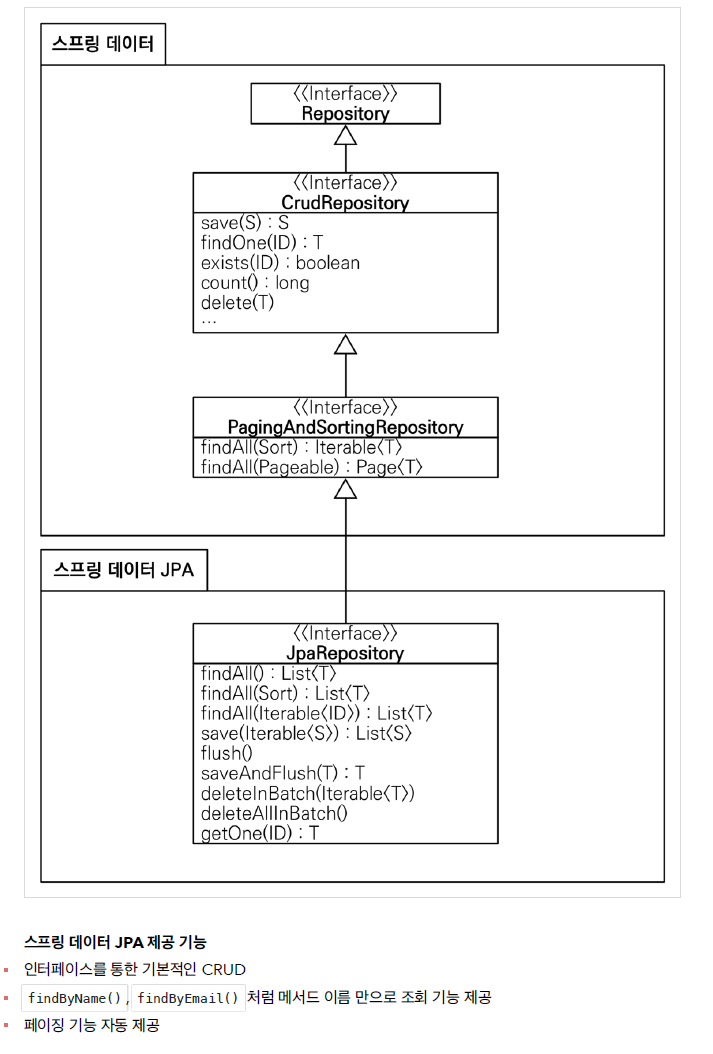
- 인터페이스에 JpaRepository가 있으면 스프링 데이터 JPA가 스프링빈에 구현체를 만들어서 자동으로 등록해줌

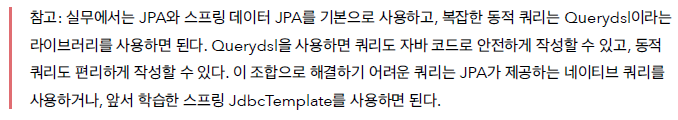
2) SpringConfig에 MemberRepository 생성 후 생성자에 주입

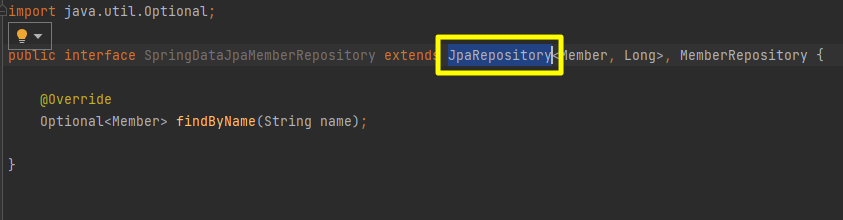
- 스프링 데이터 구현체를 만들어 놓은게 등록이 됨

- 스프링 데이터 JPA 사용시 아래와 같이 코드가 간결해짐

private final MemberRepository memberRepository;  
  
//@Autowired //생성자가 하나인 경우 생략가능  
public SpringConfig(MemberRepository memberRepository) {  
 this.memberRepository = memberRepository;  
}  
  
@Bean  
public MemberService memberService(){  
 return new MemberService(memberRepository);  
}

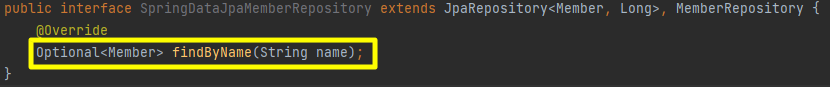






**단축키 [CTRL] + [B] : JPA 리포지토리 구조 확인**

- findAll() 등 기본 메서드들이 내장되어 있음

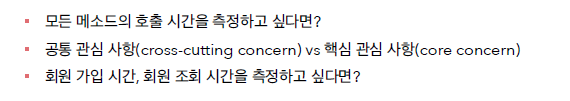


- 각 프로젝트별로 사용되는 변수명들은 모두 다르기 때문에 JPA 리포지토리에서 사전에 공통화해놓는 것은 100% 불가하다. 그래서 진행하고 있는 프로젝트의 객체나 변수명을 사용하려면 위와 같이 findByName 을 활용하여 선언한다. (findBynameAndId() 등등 많음)

그러면, 예를들어 JPQL 쿼리를 JPA에서 “select m from **member** m where **m.name** = ?” 과 같이 번역하여 실행한다.

# **7. AOP (Aspect Oriented Programming)**

## **7.1 AOP가 필요한 상황**



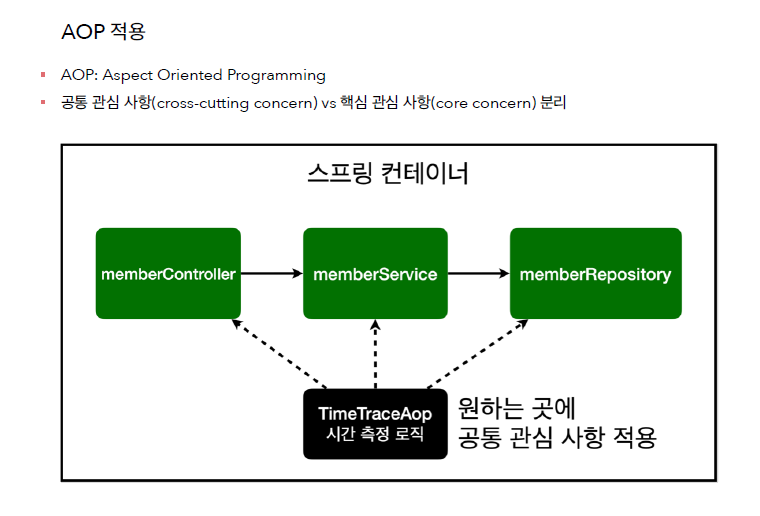
1) AOP 적용 전

아래와 같이 모든 메소드에 노가다성(?)으로 작업을 해야 한다.

public Long join(Member member) {  
  
  
 //AOP  
 long start = System.*currentTimeMillis*();  
  
 try {  
 validateDuplicateMember(member); //중복 회원 검증  
 memberRepository.save(member);  
 return member.getId();  
 } finally {  
 long finish = System.*currentTimeMillis*();  
 long timeMs = finish - start;  
 System.*out*.println("join = " + timeMs + "ms");  
 }  
}

메소드가 1000개~10000개 이상인 프로젝트에서 실행시간 확인을 위해 저렇게 비생산적인 작업을 해야 하는가?..

## **7.2 AOP 적용**



1) aop 패키지 생성  
2) TimeTraceAop 클래스 생성

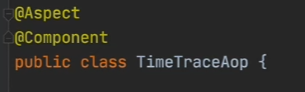
- @Aspect 어노테이션 추가

- TimeTraceAop 스프링 빈에 등록할수도 있고 컴포넌트 스캔을 활용할 수도 있음

[1-스프링빈 적용]

@Bean  
public TimeTraceAop TimeTraceAop() {  
 return new TimeTraceAop();  
}

[2-컴포넌트스캔 적용]



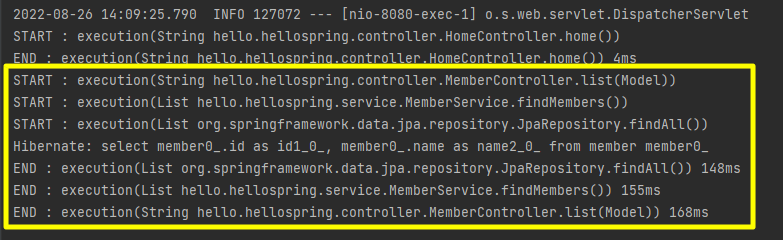
- 본강의에서는 스프링빈에 등록은 안하고, 컴포넌트 스캔에 등록하여 사용

- 어디에 사용할지 타겟팅을 해주어야 함

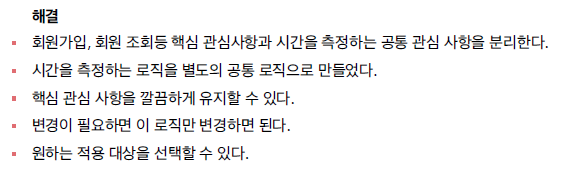
@Around("execution(\* hello.hellospring..\*(..))")

Hello.hellospring 하위 패키지에 있는 모든 메소드에 적용

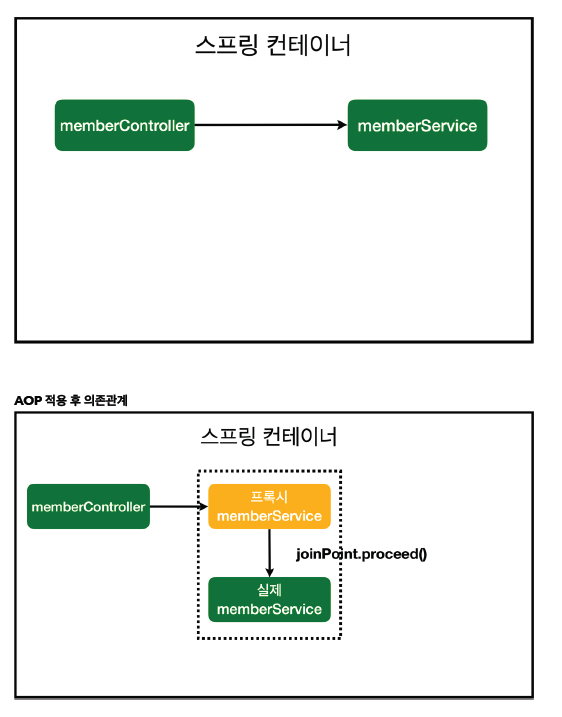
4) 실제 서버 올린 후 AOP 로그 확인



- Repository, Service, Controller 별 실행시간을 확인할 수 있다.

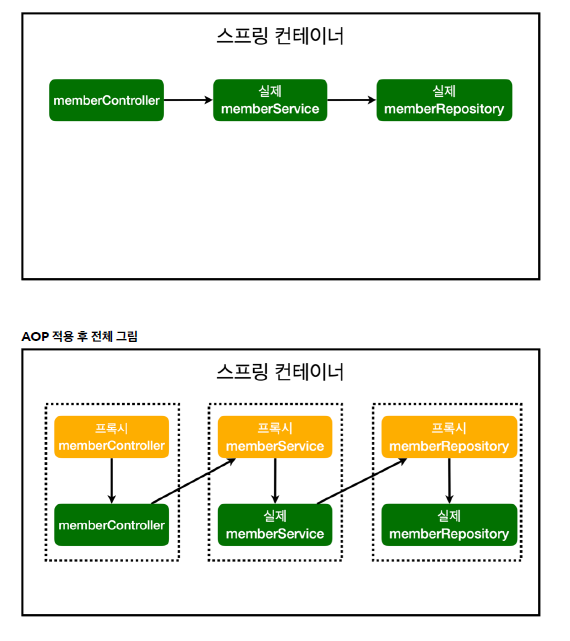


AOP적용 전 의존관계



- 프록시 방식의 AOP : 가짜 멤버서비스로 이해.. = 복제

AOP 적용 전 전체 그림



# **8. 다음으로**

