목차

**1. 프로젝트 환경설정1**

1.1 프로젝트 생성2

1.2 라이브러리 2

1.3 프로젝트 생성 2

1.4 프로젝트 생성 2

장 제목 입력(수준 3)3

**장 제목 입력(수준 1)4**

장 제목 입력(수준 2)5

장 제목 입력(수준 3)6

ㄴㅇ

**01\_프로젝트 환경설정-프로젝트 생성**

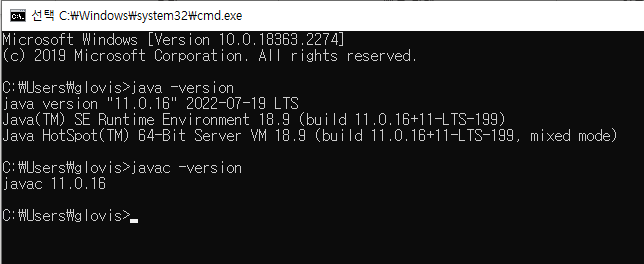
**■개발환경**

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)

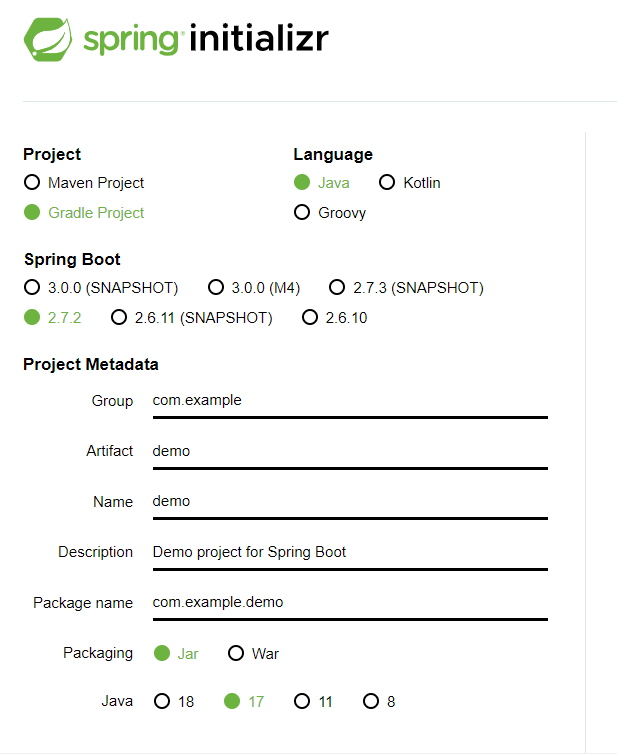
: JAVA\_HOME,

CLASSPATH,

PATH

1. 스프링 부트 스타터 사이트 이동> 스프링 프로젝트 생성

<https://start.spring.io>

****

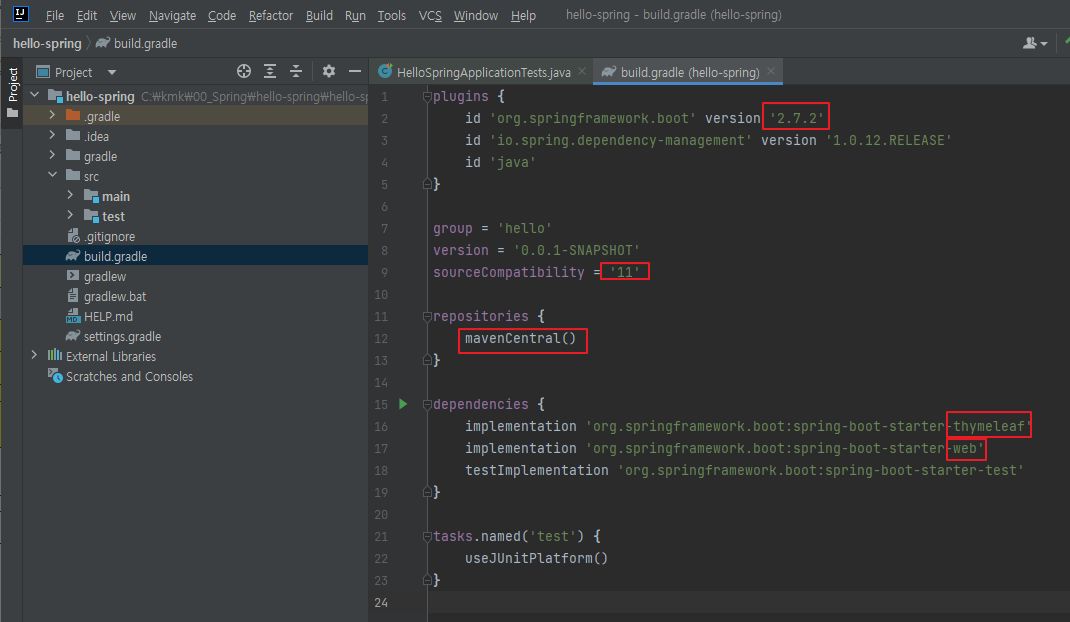
* Group : 보통 기업명
* Artifact : Build 되어 나오는 결과물 (프로젝트명)

****

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

1. **IntelliJ (설치완료)**

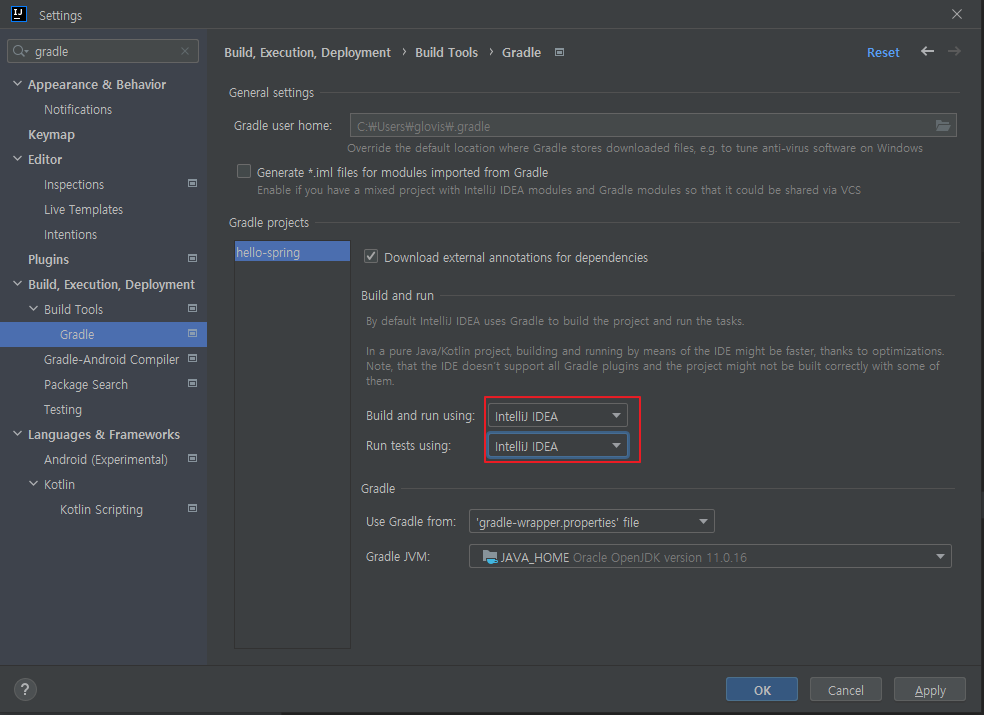
* 최초 프로젝트 생성 시 Gradle 자동 다운로드됨(네트워크 연결 필요)
* src 하위에 main, test 폴더가 자동으로 생성되는데 최근 개발 트렌드상 테스트가 중요한 부분으로 대두되어 IntelliJ 에서 test 폴더까지 자동으로 생성하게 셋팅되어 있음
* Build.gradle

****

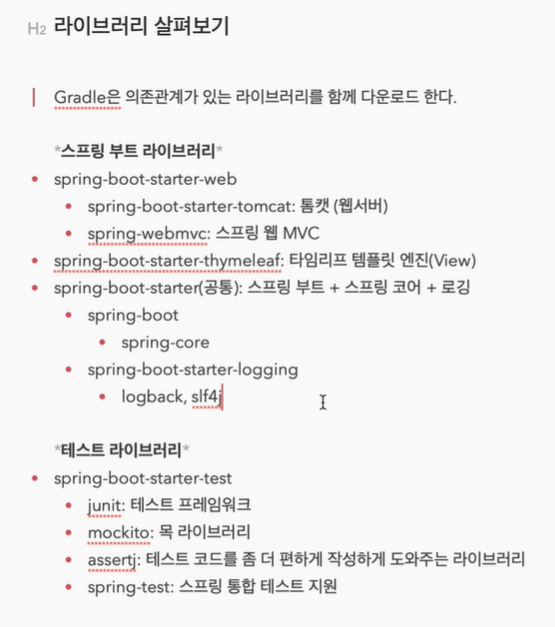
* mavenCentral() : dependencies 와 같은 라이브러리를 다운받은 저장소를 설정해준것
* Build.gradle, .gitignore 와 같은 파일은 start.spring.io 에서 자동생성되어 만들어짐

: 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

****

**01\_프로젝트 환경설정-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)**

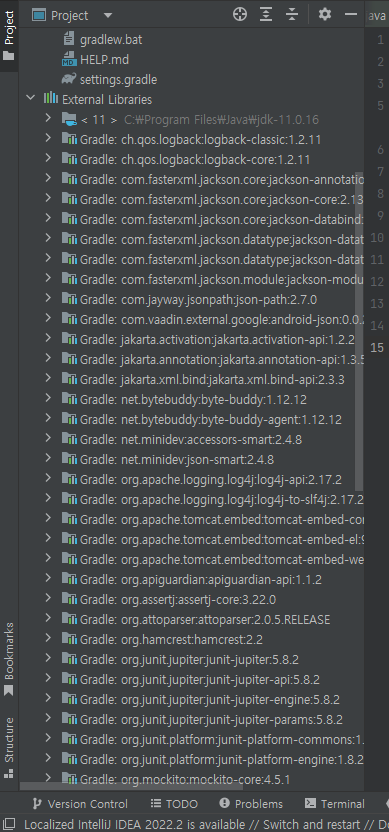
**- Packaging : 코드를 패키징해 aab 파일이나 apk 파일로 만드는 것**

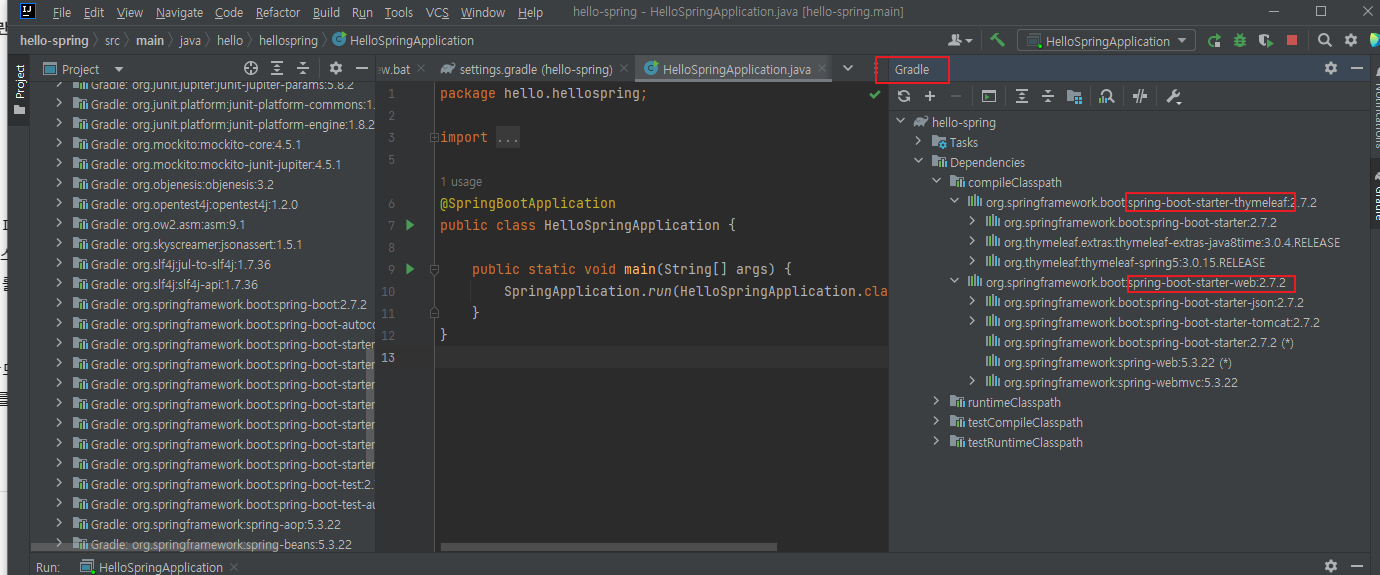
**- Deploy & Run : 코드를 App 으로 패키징해서 실제 기기에 넣어 실행할 수 있도록 하는 것**

**\*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)

**01\_프로젝트 환경설정-View**

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)

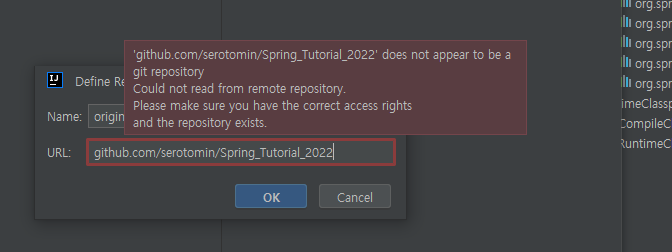
* Git 설치 & Github 연동

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

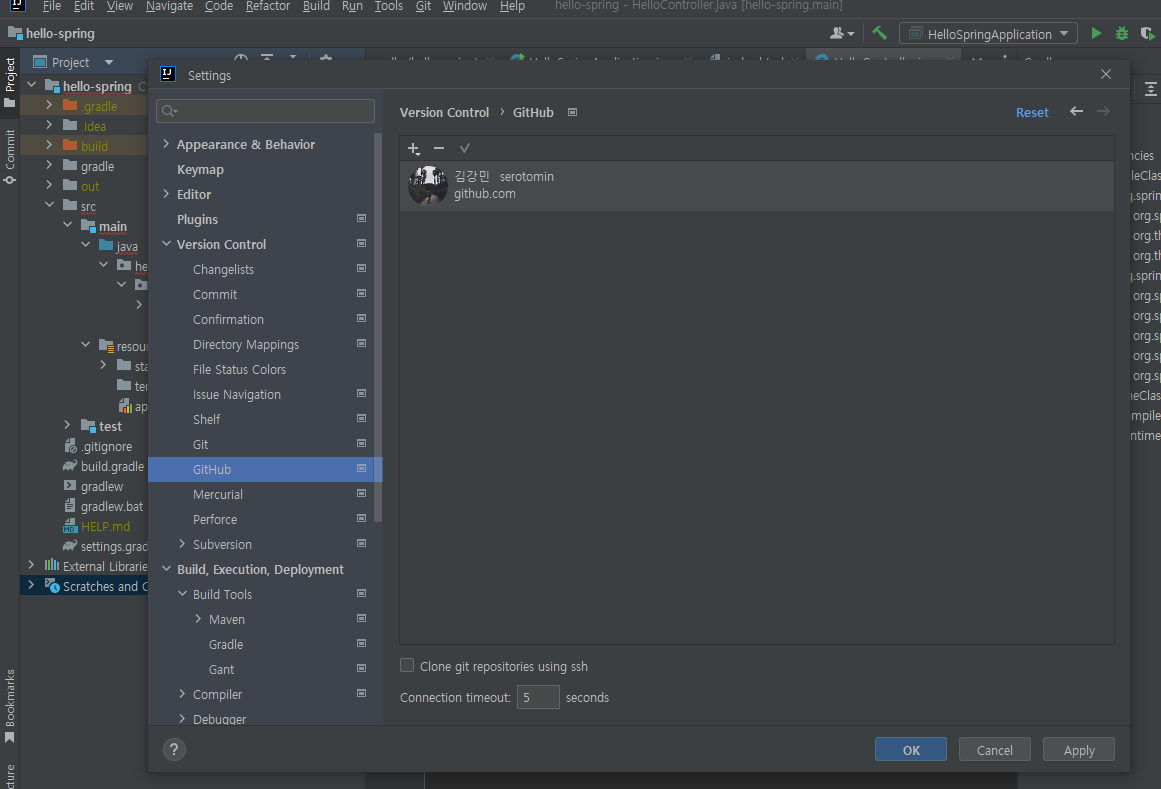
* Git 환경설정

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

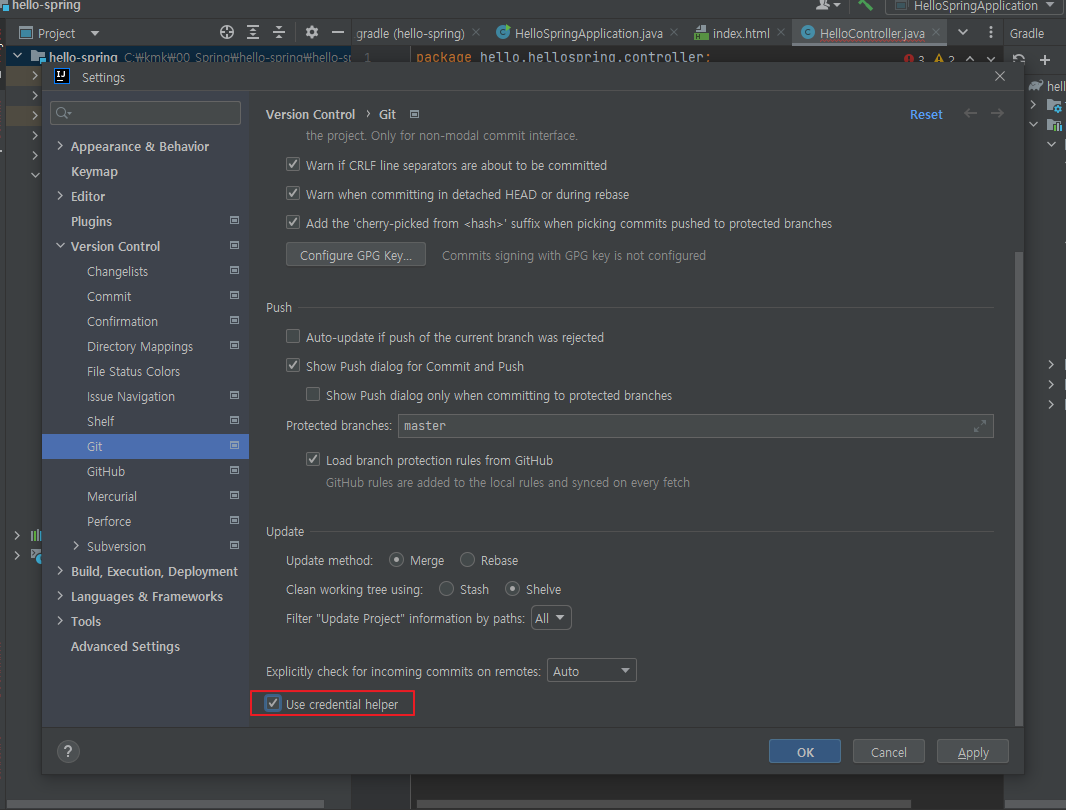
[문제]

****

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

****

GitHub 계정 추가 후 확인 / 실패

****

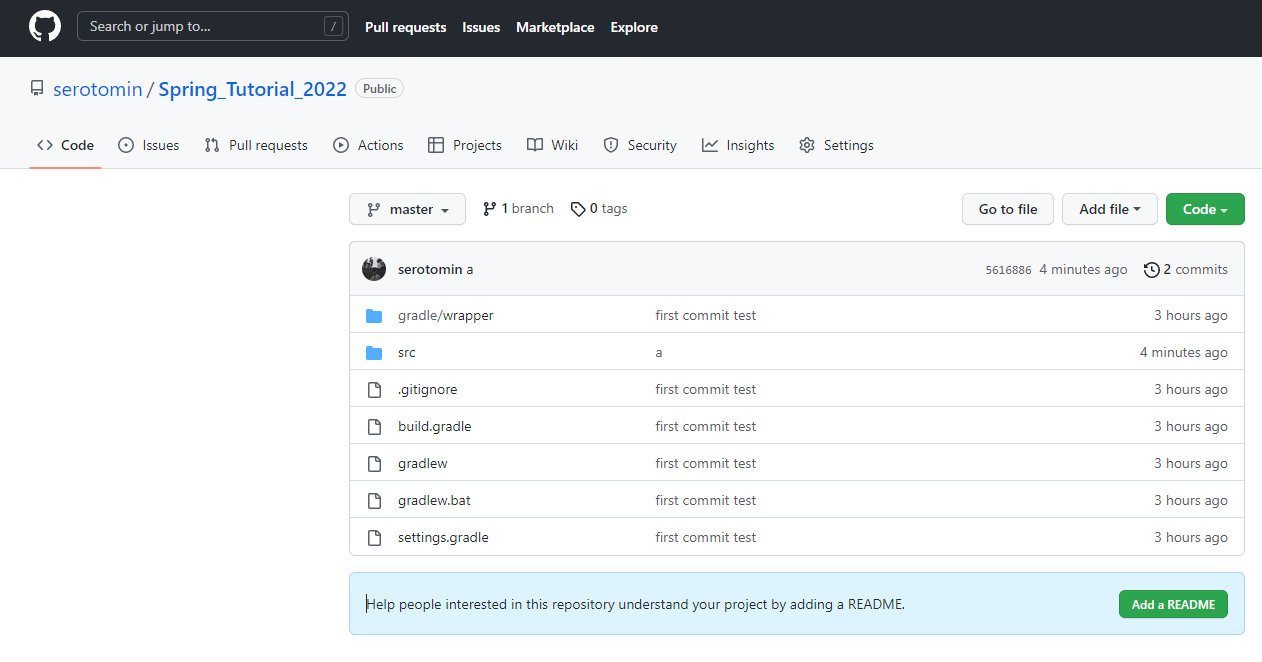
Settings > Version Control > Git > Use credential helper 체크

로도 안됨

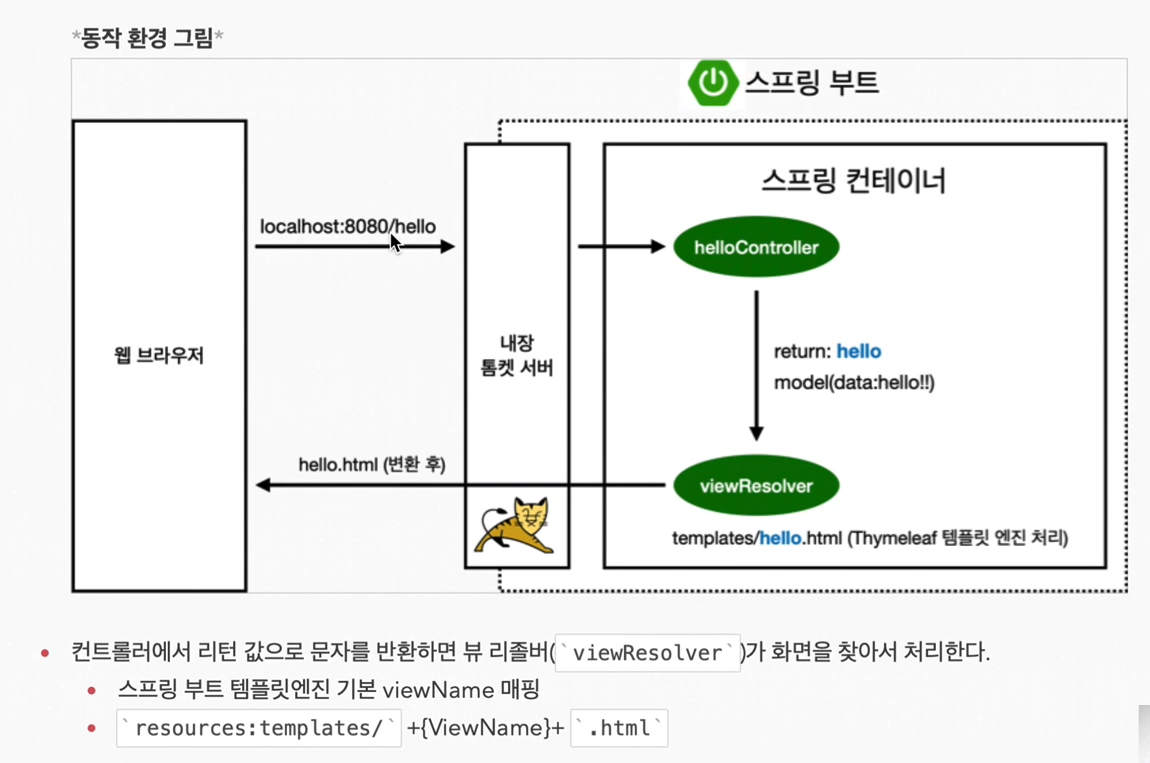
아래 명령으로 해결

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]

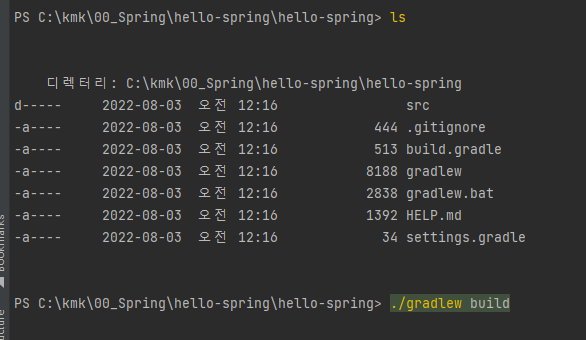
****

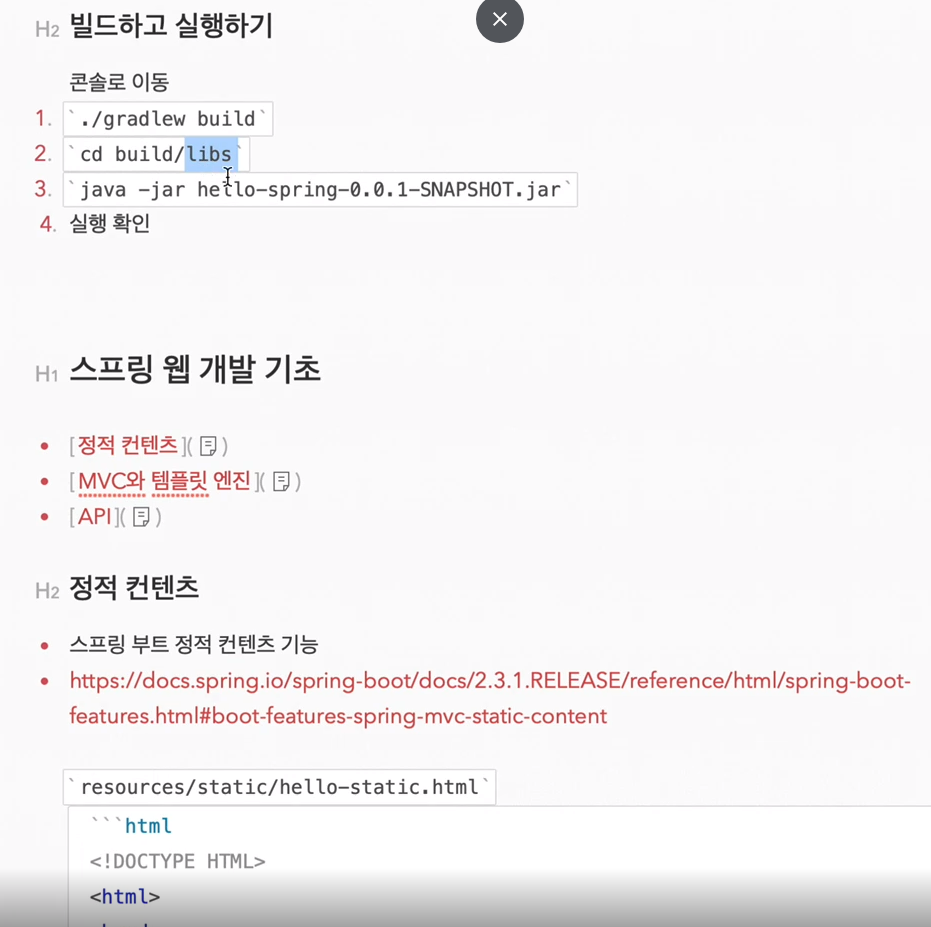
소스코드 github push 완료

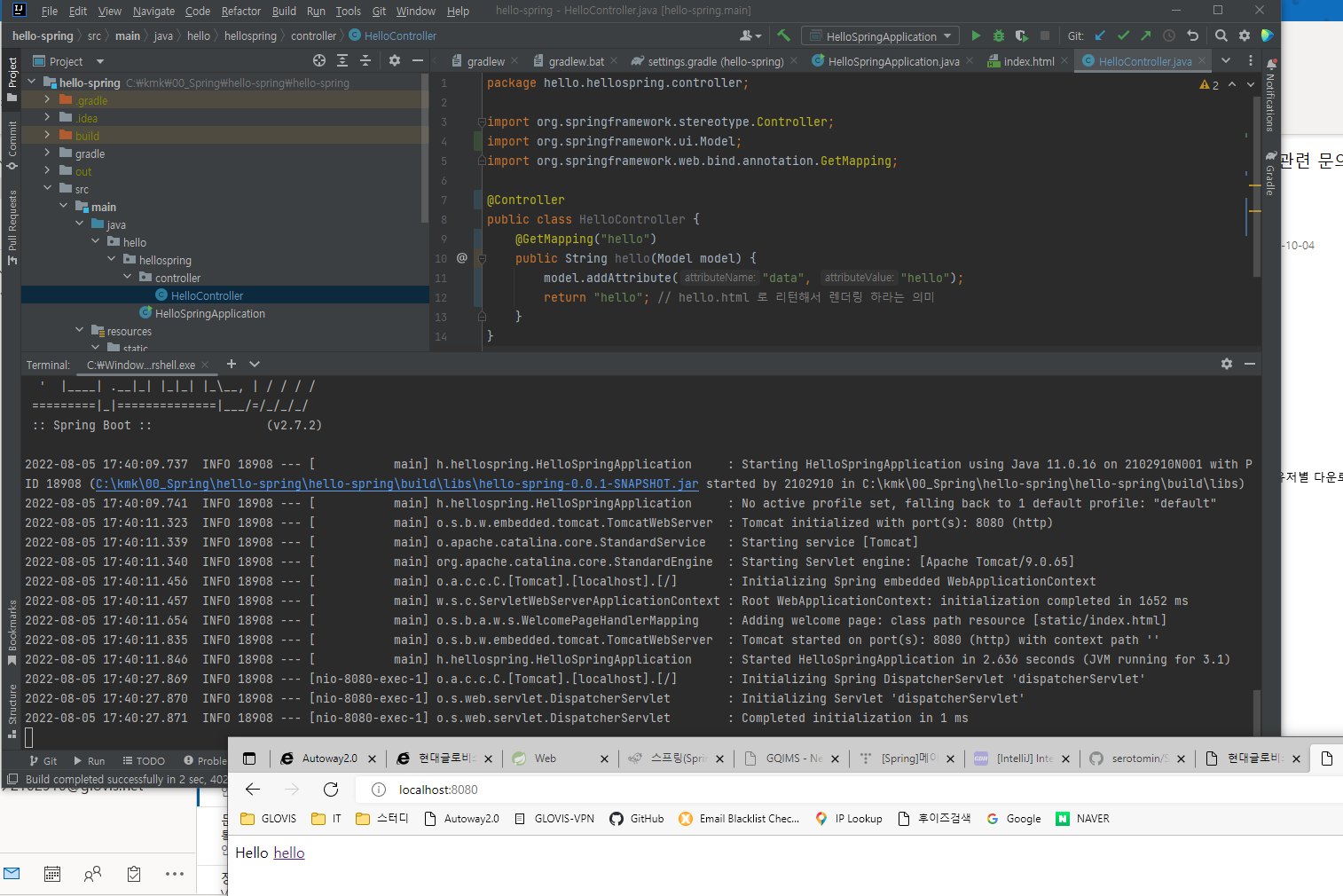
****

{ViewName} == hello 로 치환

**01\_프로젝트 환경설정-빌드 및 실행**

****

****

****

**02\_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

**02\_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

**02\_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



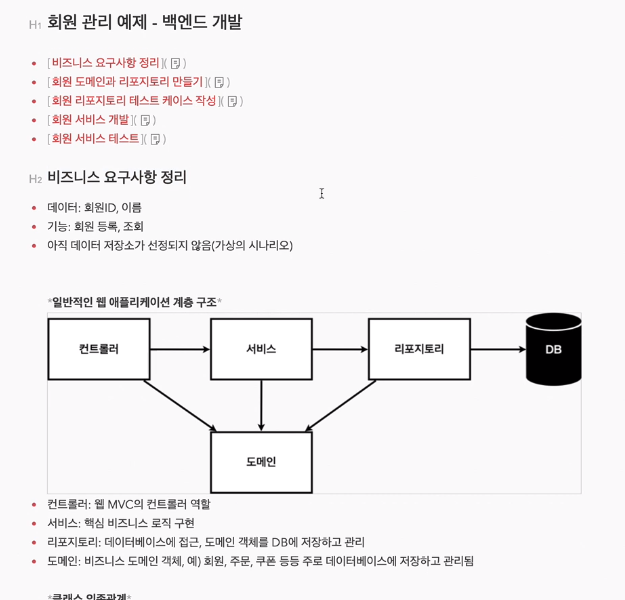
* JSON 방식
* 소스확인 : <http://localhost:8080/hello-api?name=Spring>
* JSON 방식? : Key-Value로 이루어진 구조
* 시스템 생성 대체 텍스트:
  •@ResponseB0dy 시용 원리 
  콤켓 서버 
  @ResponseBOdy 를 사용 
  HTTP의 BODY에 문자 내용을 직접 반환 
  0 스프링 부트 
  스프링 컨테이너 
  @ResponseBody 
  HttpM essageCon `, erter 
  S:•lngCO 
  viewReso Iver 대신에 HttpMessageConverter 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

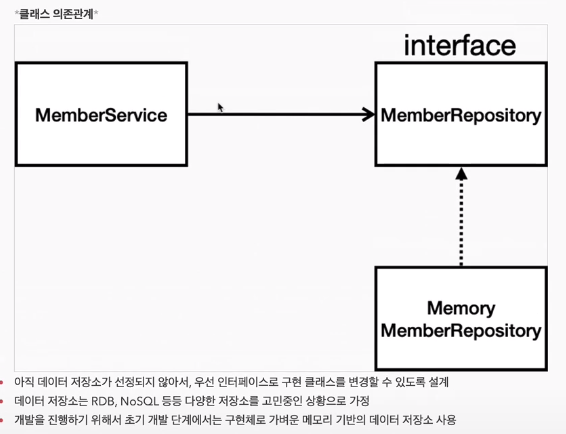
* 정적 콘텐츠 : 파일을 그대로 내림
* MVC : Model, View, Controller 분리하여 클라이언트에게 전달(템플릿 엔진 방식)
* API : 일반적으로 객체 반환

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

**03\_비즈니스 요구사항 정리**





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

**03\_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

**03\_회원 리포지토리 테스트 케이스 작성**

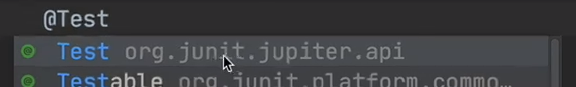
1. 자바 - Junit 프레임워크로 테스트 코드 작성/실행

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

2) repository.MemoryMemberRepositoryTest 클래스 생성

- 다른곳에서 쓰이지 않기 때문에 public 클래스로 생성하지 않음

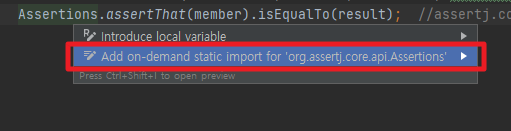
- @test : Import -> Junit Jupiter API



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**



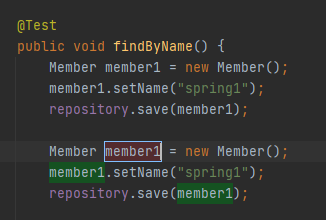
*assertThat*(member).isEqualTo(result);

위와 같이 간소화되고 아래처럼 static 으로 import됨

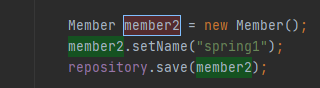
import static org.assertj.core.api.Assertions.\*;

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

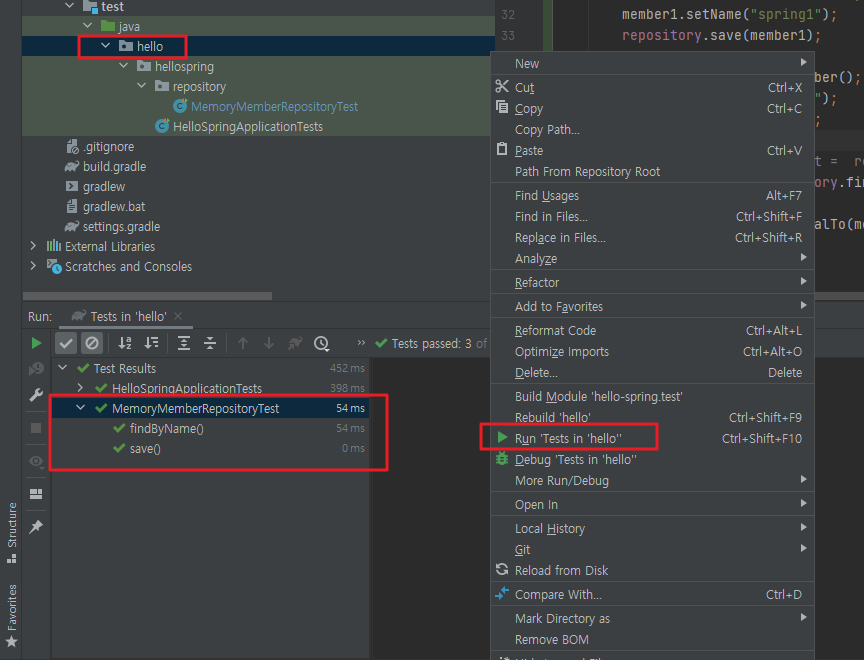
[적용 전]



[적용 후]

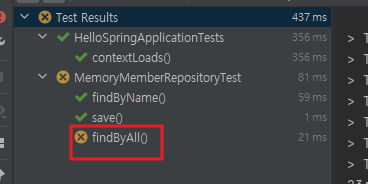


클래스 레벨에서 테스트



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)

**03\_회원 서비스 개발**

- ㅇ

**03\_회원 서비스 테스트**

- ㅇ