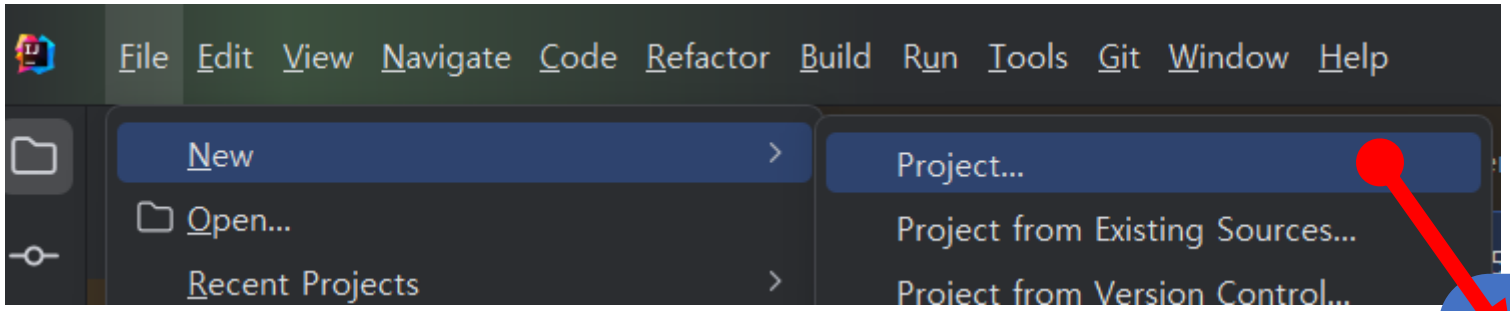
KB

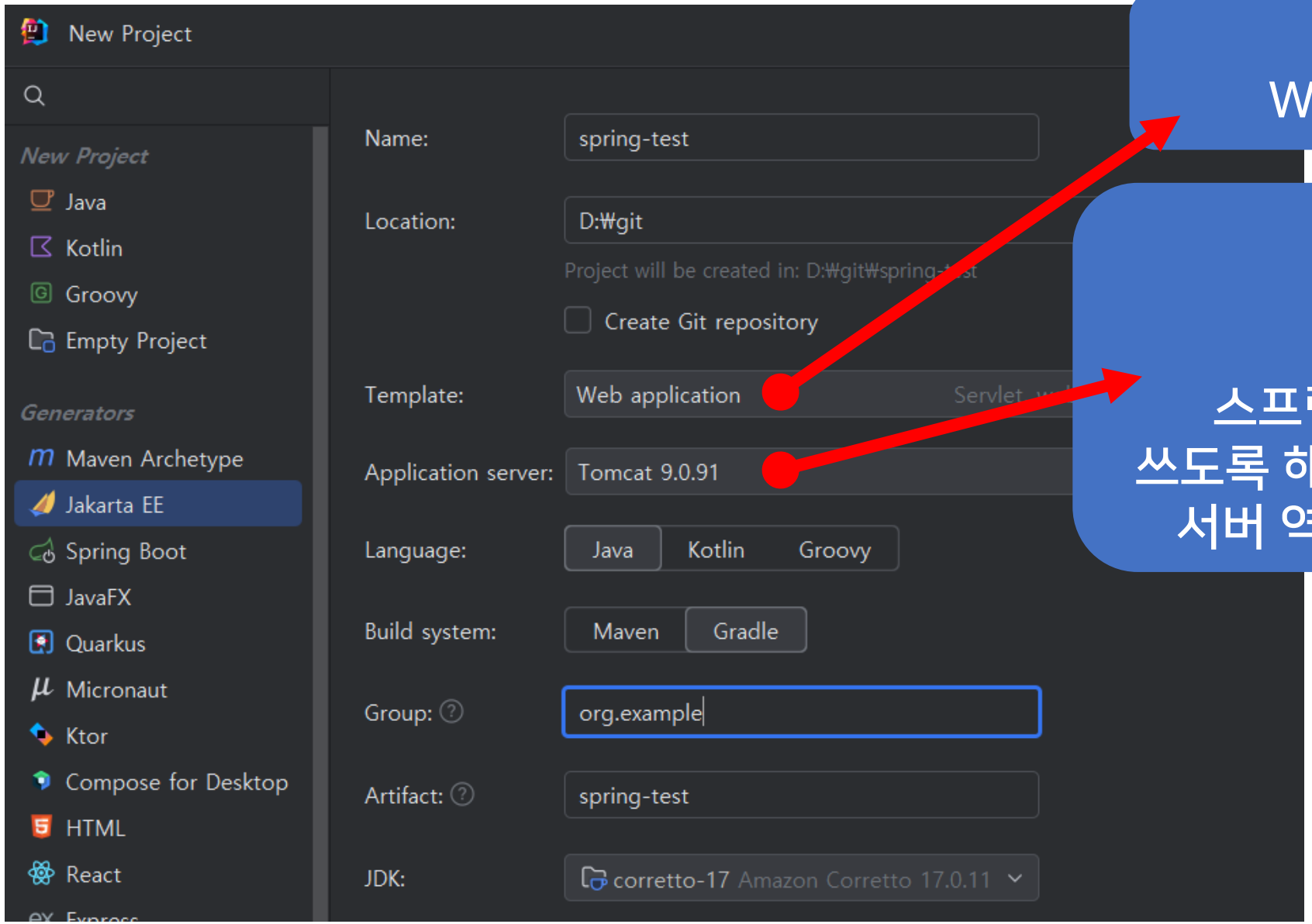
It's Your Life

with

Spring 프로젝트 세팅하기 with 설명



인테텔리제이에서 File → New → Project 선택



REST API (X)
Web applictiaon (O) 선택!

기존에 사용하던
Tomcat 서버 선택

스프링은 결국 서블릿을 편하게
쓰도록 해주는 친구(프레임워크)이므로
서버 역할은 여전히 톰캣이 합니다!

New Project

New Project
- Java
- Kotlin
- Groovy
- Empty Project

Generators
- Maven Archetype
- Jakarta EE
- Spring Boot
- JavaFX
- Quarkus
- Micronaut
- Ktor
- Compose for Desktop
- HTML
- React
- Express

Name: spring-test

Location: D:₩git

Project will be created in: D:₩git₩spring-test

Create Git repository

Template: Web application — Servlet, web.xml, index.jsp

Application server: Tomcat 9.0.91

Language: Java | Kotlin | Groovy

Build system: Maven | Gradle

Group: org.example

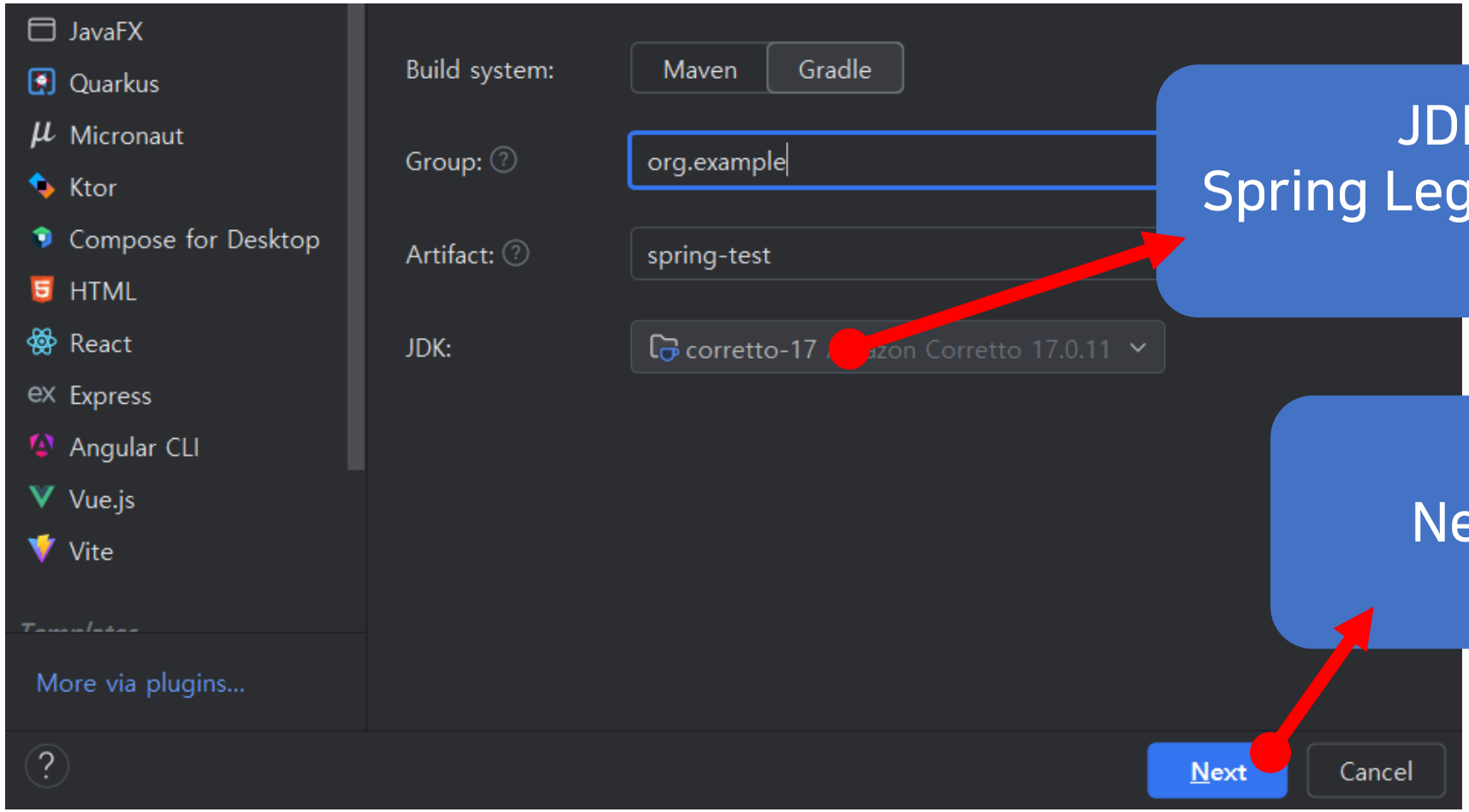
Artifact: spring-test

JDK: corretto-17 Amazon Corretto 17.0.11

언어는 JAVA 선택!

패키지 관리는 Gradle 선택

Group 은 아무거나 선택
Artifact 는 가급적 프로젝트 이름과 통일!



JDK 는 최신버전 사용시
Spring Legacy 와 충돌이 날 수 있으므로
17 버전을 선택

Next 눌러서 다음으로 진행!

Version: Jakarta EE 9.1

Dependencies:

- [ ] Batch (2.0.0)
- [ ] Bean Validation (3.0.0)
- [ ] Contexts and Dependency Injection (CDI) (3.0
- [ ] Concurrency Utils (2.0.0)
- [ ] Connector Architecture (JCA) (2.0.0)
- [ ] Enterprise Java Beans (EJB) (4.0.0)
- [ ] JSON Binding (JSON-B) (2.0.0)
- [ ] JSON Processing (JSON-P) (2.0.1)
- [ ] Message Service (JMS) (3.0.0)
- [ ] Model View Controller (MVC) (2.0.0)
- [ ] Persistence (JPA) (3.0.0)
- [ ] RESTful Web Services (JAX-RS) (3.0.0)
- [ ] Security (2.0.0)
- [ ] Server Faces (JSF) (3.0.0)
- [x] Servlet (5.0.0)
- [ ] Transaction (JTA) (2.0.0)

자바 EE(Enterprise Edition) 9 이상은 서블릭 5.0 을 기본으로 사용

New Project

Version: Java EE 8

우리는 스프링과의 호환성을 위해
자바 EE 8 버전 + Servlet 4.0 버전을
사용

Dependencies:

- [ ] Batch (1.0.1)
- [ ] Bean Validation (2.0.1.Final)
- [ ] Contexts and Dependency Injection (CDI) (2.0
- [ ] Concurrency Utils (1.1)
- [ ] Connector Architecture (JCA) (1.7.1)
- [ ] Enterprise Java Beans (EJB) (3.2.2)
- [ ] JSON Binding (JSON-B) (1.0)
- [ ] JSON Processing (JSON-P) (1.1.4)
- [ ] Message Service (JMS) (2.0.1)
- [ ] Model View Controller (MVC) (1.0.0)
- [ ] Persistence (JPA) (2.2)
- [ ] RESTful Web Services (JAX-R) (2.1.1)
- [ ] Security (1.0)
- [ ] Server Faces (JSF) (2.3)
- [x] Servlet (4.0.1)
- [ ] Transaction (JTA) (1.3)
- [ ] WebSocket (1.1)
- [ ] XML Web Services (JAX-WS) (2.3.1)
- Implementations
  - [ ] Apache CXF REST Server (JAX-RS) (3.4.10)

Full Platform

Includes most of the Jav EE specifications
Compatible servers: GlassFish 5.x, WebSphe
Liberty 18.0.0.2, Wildfly 14.x, JBoss Enterpris
WebLogic Serve 14.1.1.0.

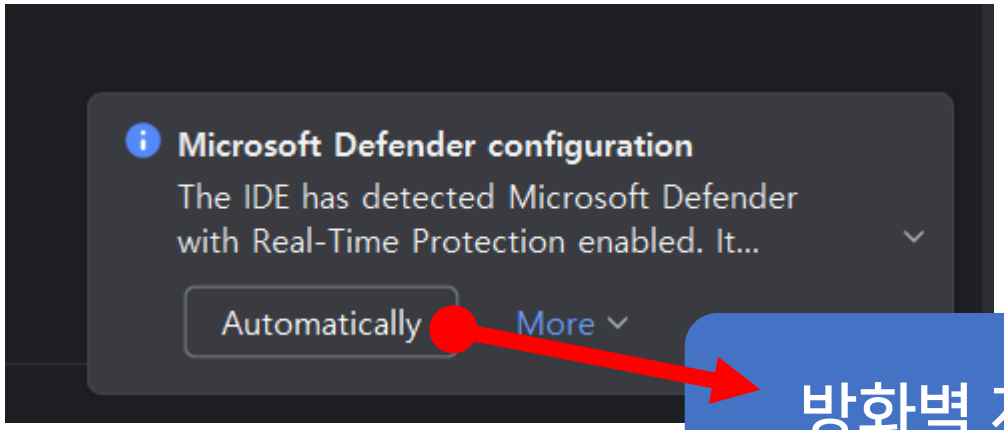
Web site ↗ pecification ↗

Added dependencies:

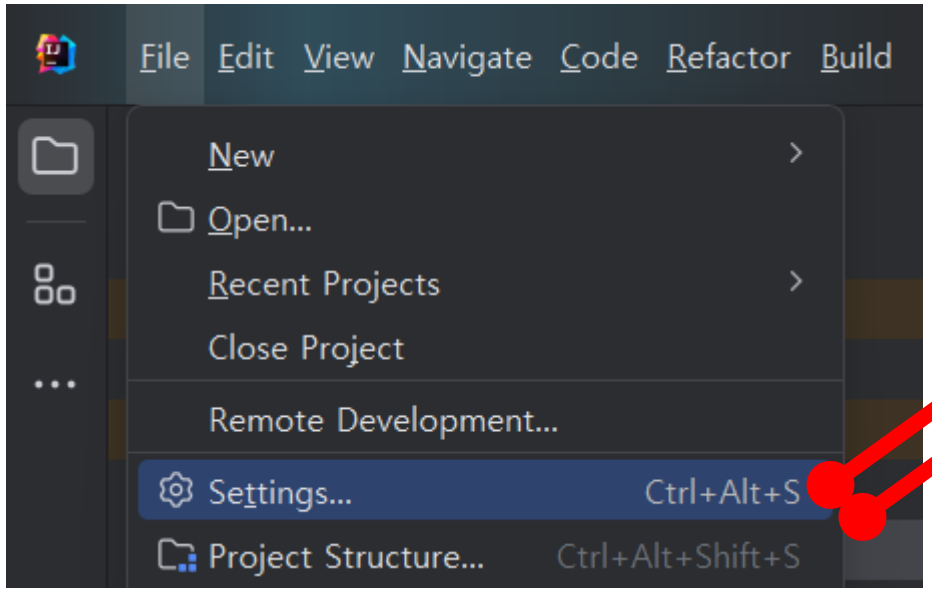
× Servlet

Create 눌러서 생성!

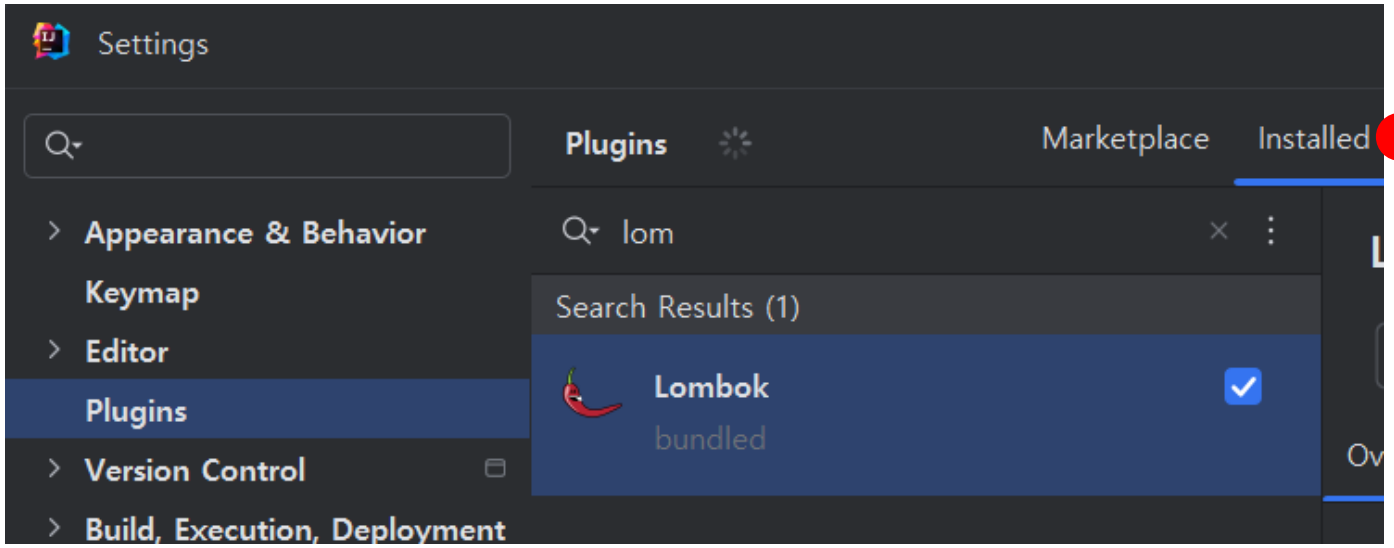
Previous Create



방화벽 자동 우회 등록하기!

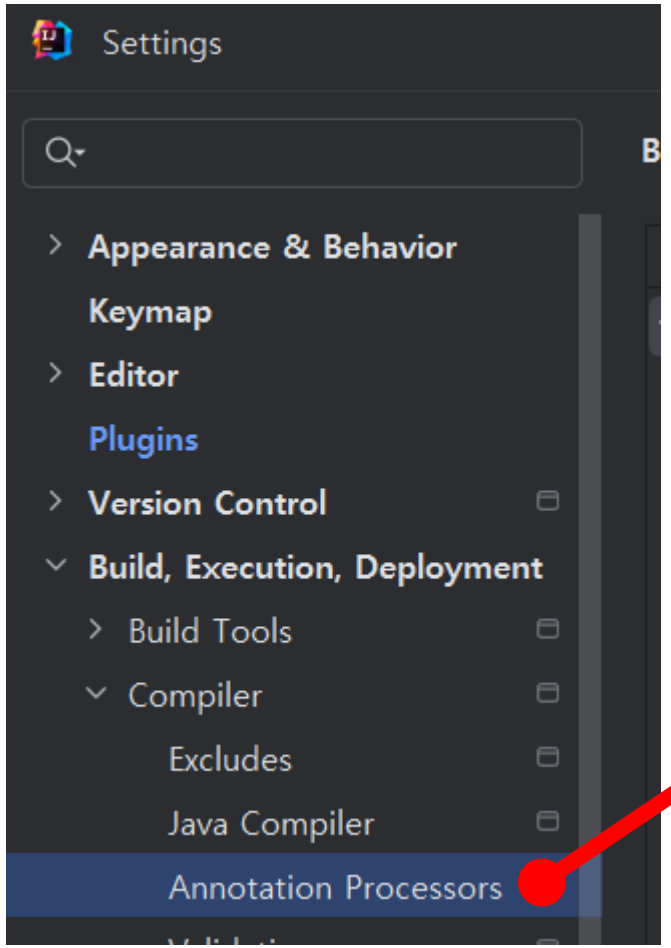


File → Settings 로 이동



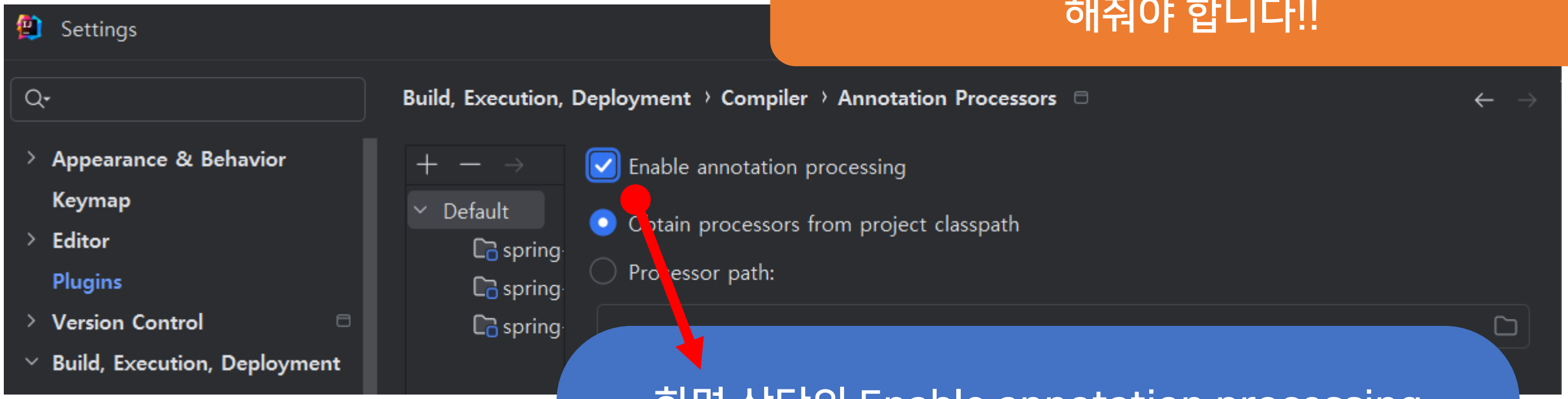
Plugins → Installed 선택 후
Lombok 설치 확인

→ 설치가 안되어 있으면 설치 하기!



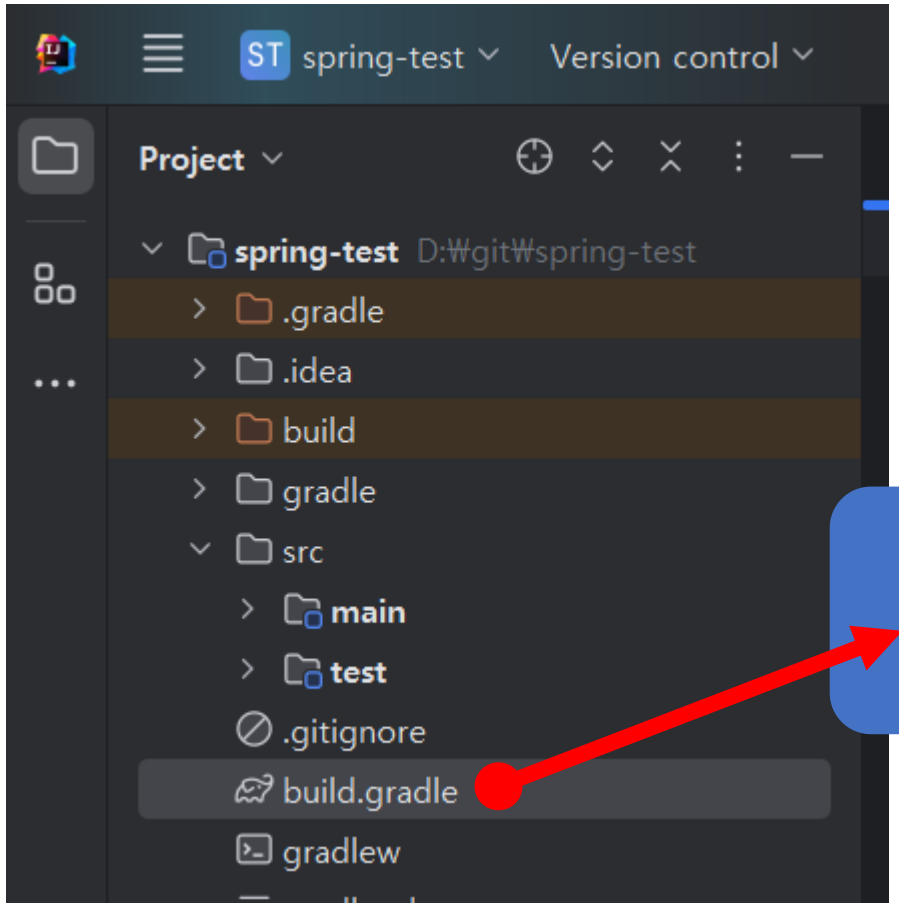
Build, Execution, Deployment 선택 →
Compiler 선택 →
Annotation Processors 선택

요 작업은 스프링 프로젝트를 생성할 때마다 해줘야 합니다!!



화면 상단의 Enable annotation processing 체크 상태로 변경 후, OK 누르기!

→ 우리는 어노테이션으로 추가적인 코드나 설정 파일을 생성하고, 검사 등을 수행해야 하므로 해당 옵션을 켜서 인텔리제이가 해당 작업을 수행하도록 설정이 필요!



프로젝트 세팅을 위해 build.gradle 열기

build.gradle (spring-test) × HelloServlet.java index.jsp

```
plugins {
    id 'java'
    id 'war'
}

group 'org.example'
version '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

ext {
    junitVersion = '5.9.2'
}

sourceCompatibility = '1.8'
targetCompatibility = '1.8'

tasks.withType(JavaCompile) {
    options.encoding = 'UTF-8'
}
```

스프링 프로젝트 세팅을 위해 대량의
라이브러리 등록이 필요하기 때문에
하나하나 등록하기는 귀찮으므로

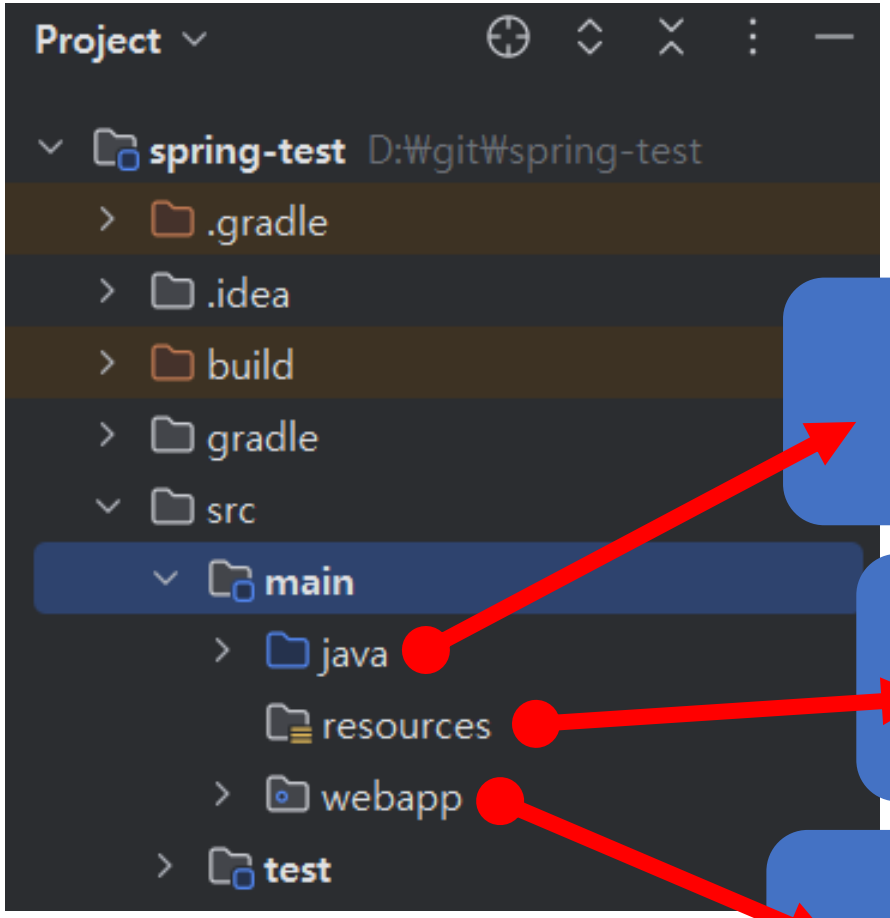
아래의 github 주소에 접속하여
build.grade 코드를 복붙!
(Thx for 민준님!)

https://github.com/xenosign/kb-spring-lecture/blob/main/build.gradle

```
tasks.withType(JavaCompile) {
    options.encoding = 'UTF-8'
}

dependencies {
    // 스프링
    implementation("org.springframework:spring-context:${springVersion}") {
        exclude group: 'commons-logging', module: 'commons-logging'
    }
    implementation("org.springframework:spring-webmvc:${springVersion}")
    implementation('javax.inject:javax.inject:1')

    // AOP
    implementation('org.aspectj:aspectjrt:1.9.20')
    implementation('org.aspectj:aspectjweaver:1.9.20')

    // JSP, SERVLET, JSTL
    implementation('javax.servlet:javax.servlet-api:4.0.1')
    compileOnly('javax.servlet.jsp:jsp-api:2.1')
    implementation('javax.servlet:jstl:1.2')

    // Logging
    implementation('org.slf4j:slf4j-api:2.0.9')
    runtimeOnly('org.slf4j:jcl-over-slf4j:2.0.9')
    runtimeOnly('org.slf4j:slf4j-log4j12:2.0.9')
    implementation('log4j:log4j:1.2.17')
```

복붙이 완료 되었으면 Sync 진행

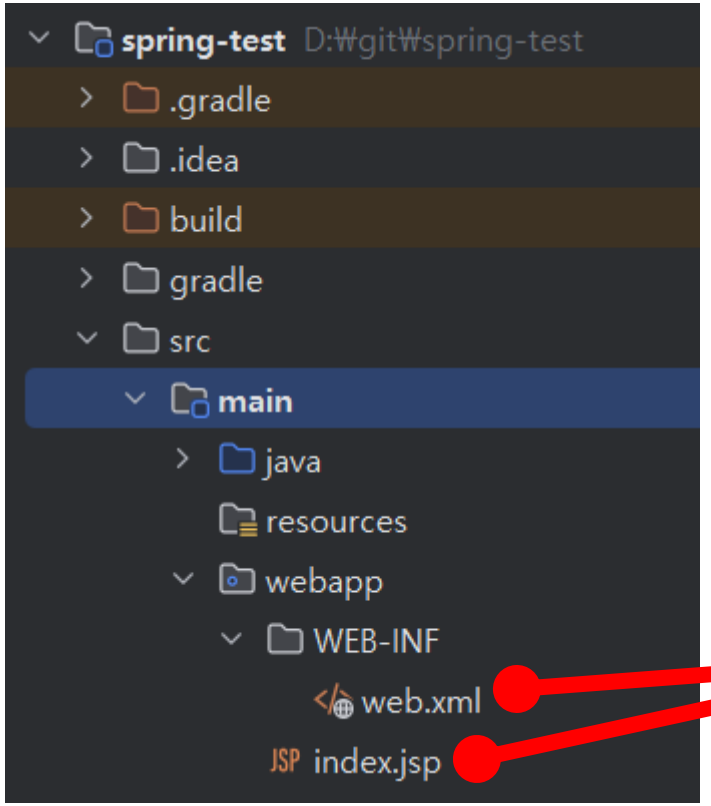
스프링 프로젝트의 폴더 구조



컨트롤러 등의 소스 코드 배치

프로젝트 세팅을 위한 설정 파일 배치

프론트(View) 설정 파일, View 파일 배치



이제 스프링을 통하지 않고서는
View 에 접근이 불가능 해야 하므로
webapp 폴더의 index.jsp 파일은 삭제!

웹 관련 설정도 xml 보다는 java 코드를 사용할
것이므로 web.xml 도 삭제!

왜 XML 을 안쓰고
Java 로 쓸까요!?

+ 빌드 세팅은
Maven 을 안쓰고
Gradle 로 바뀌었을까요!?

2019학년도 대학수학능력시험 문제지

제 2 교시

수학 영역(가형)

홀수형

5지선다형

1. 두 벡터 $\vec{a}=(1,-2)$, $\vec{b}=(-1,4)$에 대하여 벡터 $\vec{a}+2\vec{b}$의 모든 성분의 합은? [2점]

① 1 ② 2 ③ 3 ④ 4 ⑤ 5

3. 좌표공간의 두 점 $A(2,a,-2)$, $B(5,-2,1)$에 대하여 선분 AB를 2 : 1로 내분하는 점이 x축 위에 있을 때, a의 값은? [2점]

① 1 ② 2 ③ 3 ④ 4 ⑤ 5

Q. △ABC와 △DEF가 어째서 합동인지 설명하시오.

```xml
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0
         version="4.0">
    <!-- DispatcherServlet 설정 -->
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring/servlet-context.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

XML 로 구성 된
distpatchcer(A.K.A FrontContoroller)
설정 코드

```java
@Override  no usages  kdtTetz
protected String[] getServletMappings() {
    return new String[] { "/" };
}
@Override  no usages  kdtTetz
public void configureViewResolvers(ViewResolverRegistry registry){
    InternalResourceViewResolver bean = new InternalResourceViewResolver();
    bean.setViewClass(JstlView.class);
    bean.setPrefix("/WEB-INF/views/");
    bean.setSuffix(".jsp");
    registry.viewResolver(bean);
}
```

같은 코드의 Java + @Annotation 버전!


게임의 법칙
박은지
이제 알겠어

```
dependencies {
    // 스프링
    implementation("org.springframework:spring-context:${springVersion}") {
        exclude group: 'commons-logging', module: 'commons-logging'
    }
    implementation("org.springframework:spring-webmvc:${springVersion}")
    implementation('javax.inject:javax.inject:1')

    // AOP
    implementation('org.aspectj:aspectjrt:1.9.20')
    implementation('org.aspectj:aspectjweaver:1.9.20')

    // JSP, SERVLET, JSTL
    implementation('javax.servlet:javax.servlet-api:4.0.1')
    compileOnly('javax.servlet.jsp:jsp-api:2.1')
    implementation('javax.servlet:jstl:1.2')
```

build.gradle 의 의존성 코드

```xml
<dependencies>
    <!-- 스프링 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>javax.inject</groupId>
        <artifactId>javax.inject</artifactId>
        <version>1</version>
    </dependency>
```

같은 코드의 Maven 버전


나는 괜찮은데
그 어릴 때부터 습관입니다

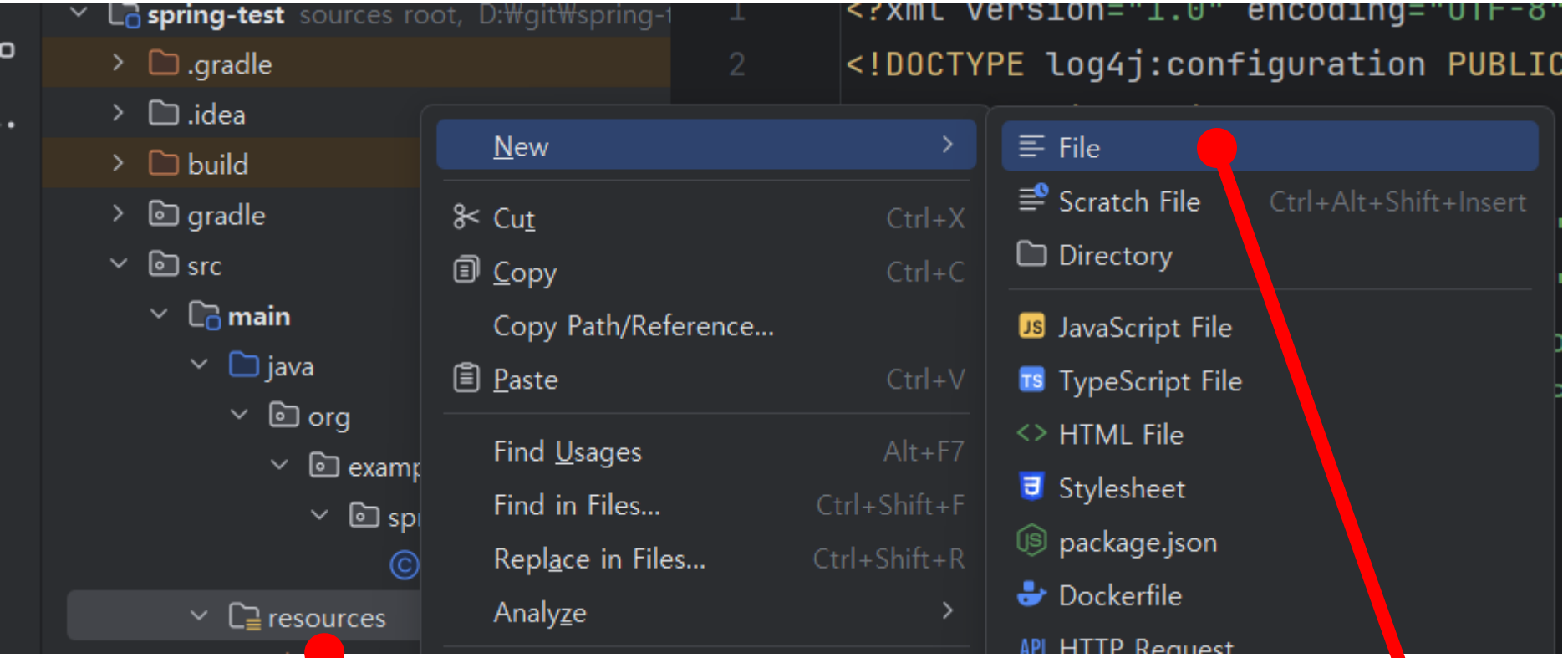

괜찮은데!


- 아직 두 장 남았다

```xml
<!-- AOP -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.9.20</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.20</version>
</dependency>
```

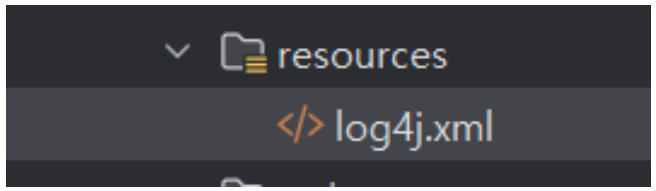
```xml
<!-- JSP, SERVLET, JSTL -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
</dependency>
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.1</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```


아~
그런거구나,
완벽히
이해했어.
알아냈어..
완벽히!
그런거였나....
과연...



설정에 대한 작업이므로
resources 폴더에 파일 추가!

resources → New → File 선택
→ log4j.xml 파일 생성

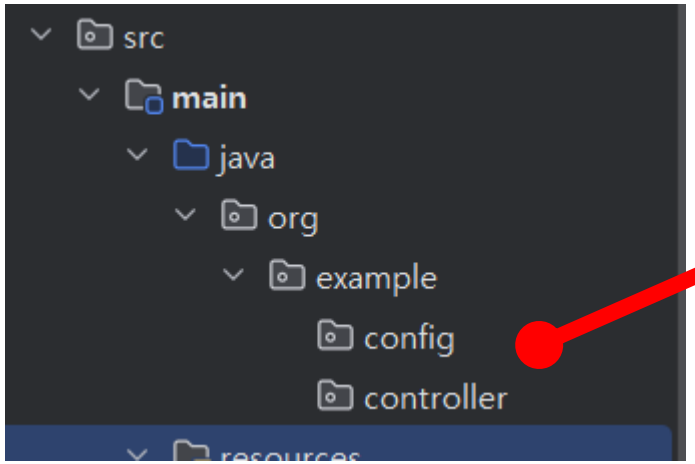


아래 링크의 코드를 복붙!

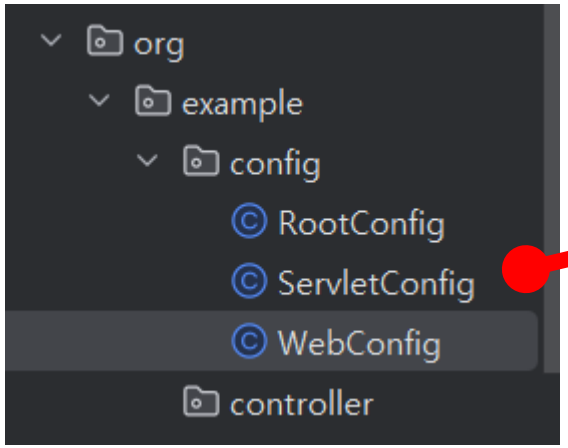
https://github.com/xenosign/kb-spring-lecture/blob/main/src/main/resources/log4j.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
    <!-- Appenders-->
    <appender name="console" class="org.apache.log4j.ConsoleAppender">
        <param name="Target" value="System.out" />
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-5p: %c - %m%n" />
        </layout>
    </appender>
    <!-- Application Loggers -->
    <logger name="org.example">
        <level value="info" />
    </logger>
    <!-- 3rdparty Loggers -->
    <logger name="org.springframework.core">
        <level value="info" />
    </logger>
    <logger name="org.springframework.beans">
        <level value="info" />
    </logger>
    <logger name="org.springframework.context">
        <level value="info" />
    </logger>
    <logger name="org.springframework.web">
        <level value="info" />
```

해당 주소는 오래 된 것 &
Log4j 의 설정에는 따로 필요
없으므로 삭제한 코드를 첨부



스프링 설정을 위한 config 패키지와
실제로 기능 역할을 담당할 controller 패키지를 생성

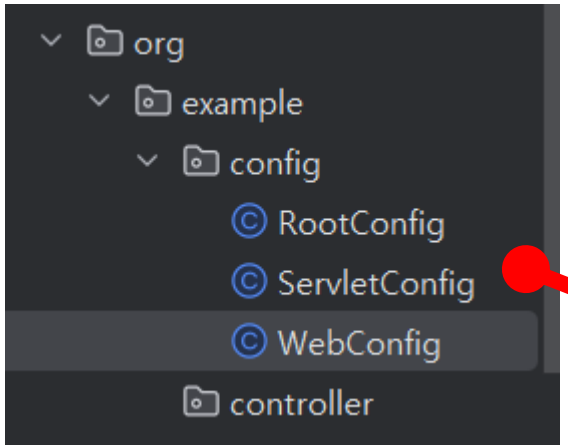


스프링 서비스의 설정을 위한 파일 3개를 생성

RootConfig : 어플리케이션의 전역 설정 파일로
데이터베이스 연결, 비즈니스 레이어 빈 설정 등등

ServletConfig : 서블릿의 컨텍스트(요청 주소) 설정

WebConfig : 글로벌 웹에 관련된 설정으로
CORS, 필터 등을 설정, 위의 두 설정 파일을 불러와서
최종적으로 적용시킨다!



RootConfig
https://github.com/xenosign/kb-spring-lecture/blob/main/src/main/java/org/example/config/RootConfig.java

ServletConfig
https://github.com/xenosign/kb-spring-lecture/blob/main/src/main/java/org/example/config/ServletConfig.java

WebConfig
https://github.com/xenosign/kb-spring-lecture/blob/main/src/main/java/org/example/config/WebConfig.java

RootConfig

```java
import org.springframework.context.annotation.Configuration;

@Configuration
public class RootConfig {
}
```

아직은 별다른 설정이 없기 때문에 빈 상태

ServletConfig

```java
@EnableWebMvc  no usages
@ComponentScan(basePackages = {"org.example.controller"})
public class ServletConfig implements WebMvcConfigurer {
    // Spring MVC용 컴포넌트 등록을 위한 스캔 패키지
    @Override  no usages
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry
                .addResourceHandler( ...pathPatterns: "/resources/**")
                .addResourceLocations("/resources/");
    }

    // jsp view resolver 설정
    @Override  no usages
    public void configureViewResolvers(ViewResolverRegistry registry){
        InternalResourceViewResolver bean = new InternalResourceViewResolver();
        bean.setViewClass(JstlView.class);
        bean.setPrefix("/WEB-INF/views/");
        bean.setSuffix(".jsp");
        registry.viewResolver(bean);
    }
}
```

정적 자원을 어디서 찾을지
설정하는 부분 설정 파일등을
찾을 때 /resources 폴더에서
찾으라는 설정

ServletConfig

```java
@EnableWebMvc  no usages
@ComponentScan(basePackages = {"org.example.controller"})
public class ServletConfig implements WebMvcConfigurer {
    // Spring MVC용 컴포넌트 등록을 위한 스캔 패키지
    @Override  no usages
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry
                .addResourceHandler( ...pathPatterns: "/resources/**")
                .addResourceLocations("/resources/");
    }

    // jsp view resolver 설정
    @Override  no usages
    public void configureViewResolvers(ViewResolverRegistry registry){
        InternalResourceViewResolver bean = new InternalResourceViewResolver();
        bean.setViewClass(JstlView.class);
        bean.setPrefix("/WEB-INF/views/");
        bean.setSuffix(".jsp");
        registry.viewResolver(bean);
    }
}
```

실제로 컨트롤러가 뷰파일을
요청하면 어느 폴더의
어떤 확장자를 선택 해야하는지
View resolver 설정

WebConfig

```java
public class WebConfig extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override  no usages
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { RootConfig.class };
    }
    @Override  no usages
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { ServletConfig.class };
    }

    @Override  no usages
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
```

RootConfig 의 기본 설정 불러오기

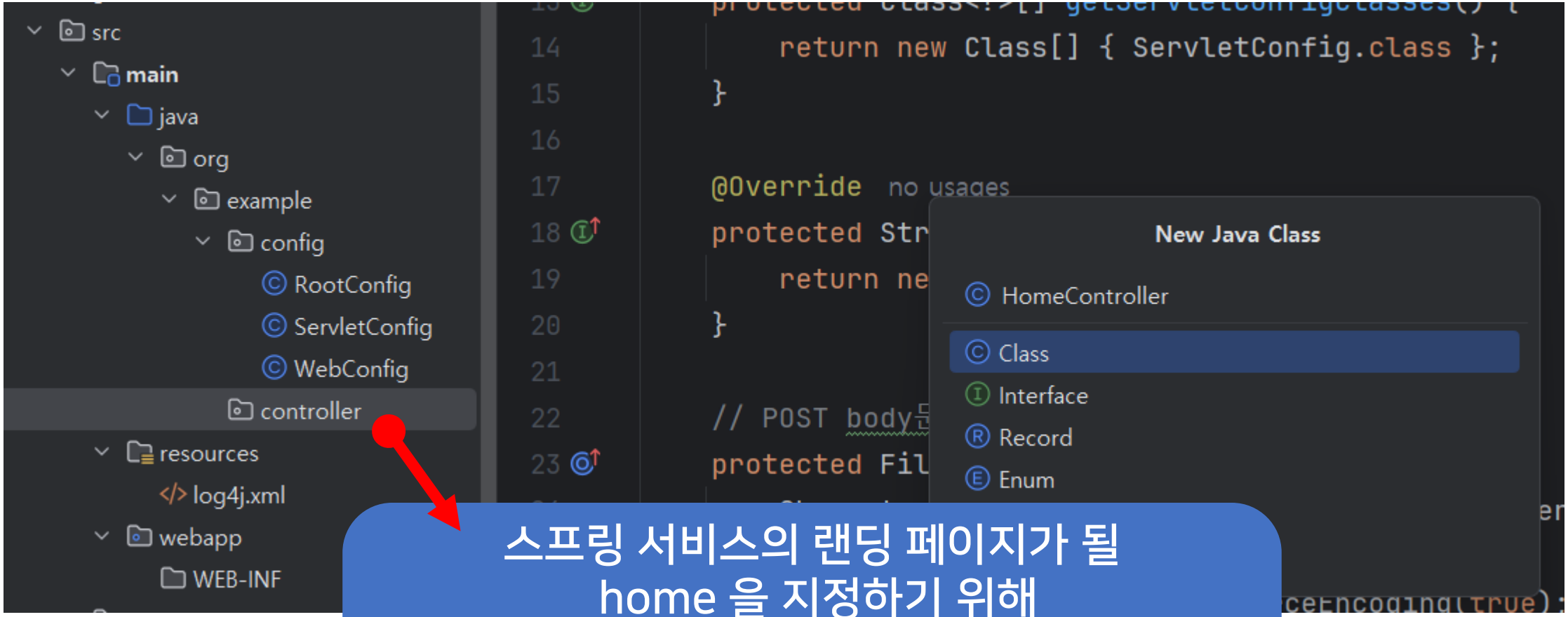
ServletConfig 의 서블릿 설정 로드

스프링의 FrontController 인
DispatcherServlet 이 관리를 시작할
URL 패턴을 설정하는 코드

```java
// POST body문자 인코딩 필터설정, UTF-8설정
protected Filter[] getServletFilters() {  no usages
    CharacterEncodingFilter characterEncodingFilter= new CharacterEncodingFilter();
    characterEncodingFilter.setEncoding("UTF-8");
    characterEncodingFilter.setForceEncoding(true);
    return new Filter[] {characterEncodingFilter};
}
```

스프링 서비스에서 기본적으로 사용하는
전역 필터 설정!

인코딩 필터의 기능을 기본으로 설정



스프링 서비스의 랜딩 페이지가 될
home 을 지정하기 위해
HomeController 컨트롤러 클래스 작성

코드는 아래의 주소의 것을 카피!

https://github.com/xenosign/kb-spring-lecture/blob/main/src/main/java/org/example/controller/HomeController.java

@Controller 어노테이션으로
스프링의 컨트롤러임을 등록 & 명시

```java
@Controller
@Slf4j
public class HomeController {
    @GetMapping("/")
    public String home() {
        log.info("================> HomController /");
        return "index";
    }
}
```

롬복 라이브러리로 서버 로그 객체 자동 생성

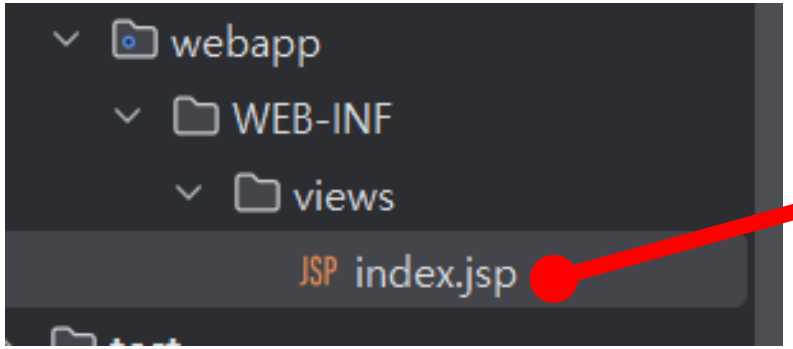
기본 주소 요청인 / 를 아래의 home 메서드에
연결처리

```java
@Controller
@Slf4j
public class HomeController {
    @GetMapping("/")
    public String home() {
        log.info("================> HomController /");
        return "index";
    }
}
```

기본 주소 요청인 / 를 아래의 home 메서드에 연결처리

@Sl4fj 어노테이션으로 자동 생성된 로그 객체(log) 를 사용해서 로그 출력

View resolver 설정에 의해 리턴 된 문자열과 동일한 jsp 파일을 View 로 제공하므로 기본 페이지 역할을 할 index(.jsp) 를 리턴

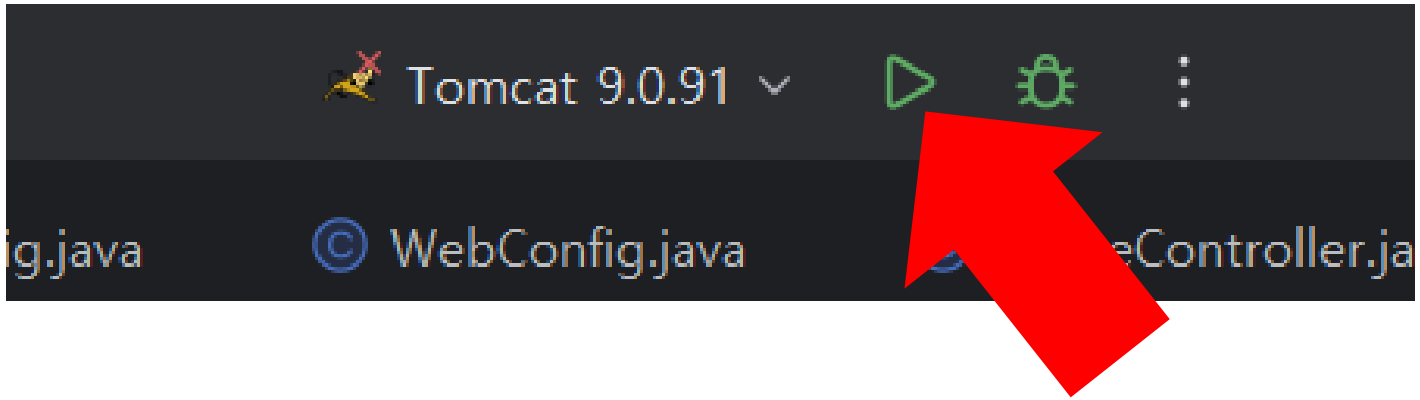


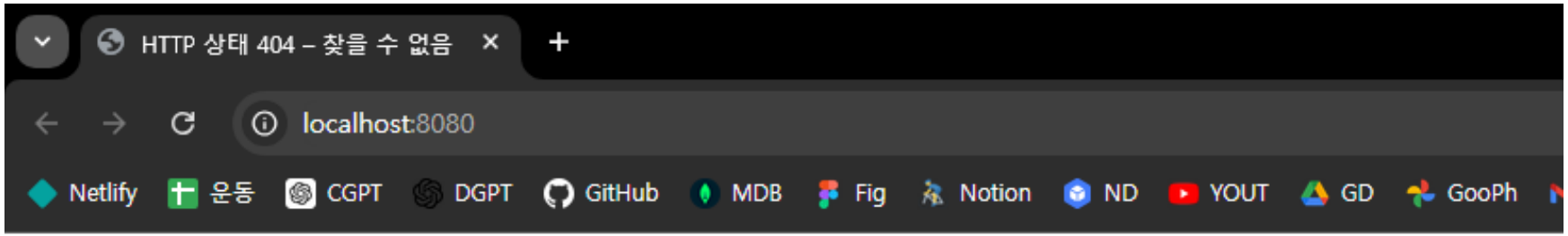
설정 값에 등록한 대로
WEB-INF 의 하위에 views 폴더를 만들고
index.jsp 파일 생성

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Spring</title>
</head>
<body>
    <h1>Hello, Spring world!</h1>
</body>
</html>
```


404


spring
자~ 이제 시작이야(내꿈을)


Tomcat 9.0.91
g.java
WebConfig.java
eController.ja

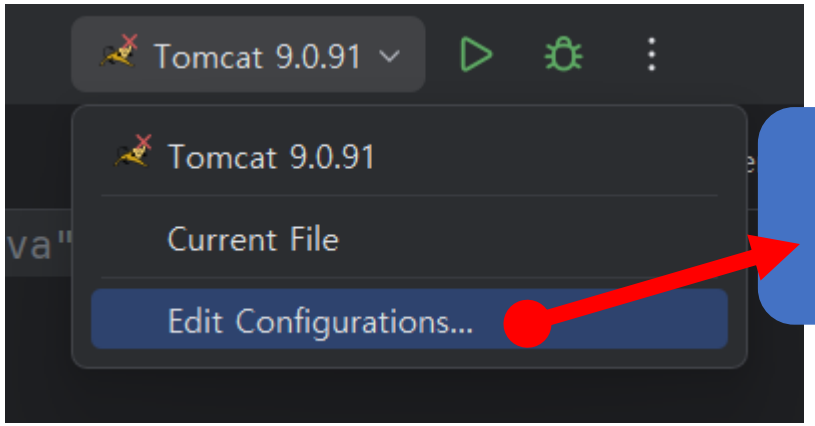


HTTP 상태 404 – 찾을 수 없음

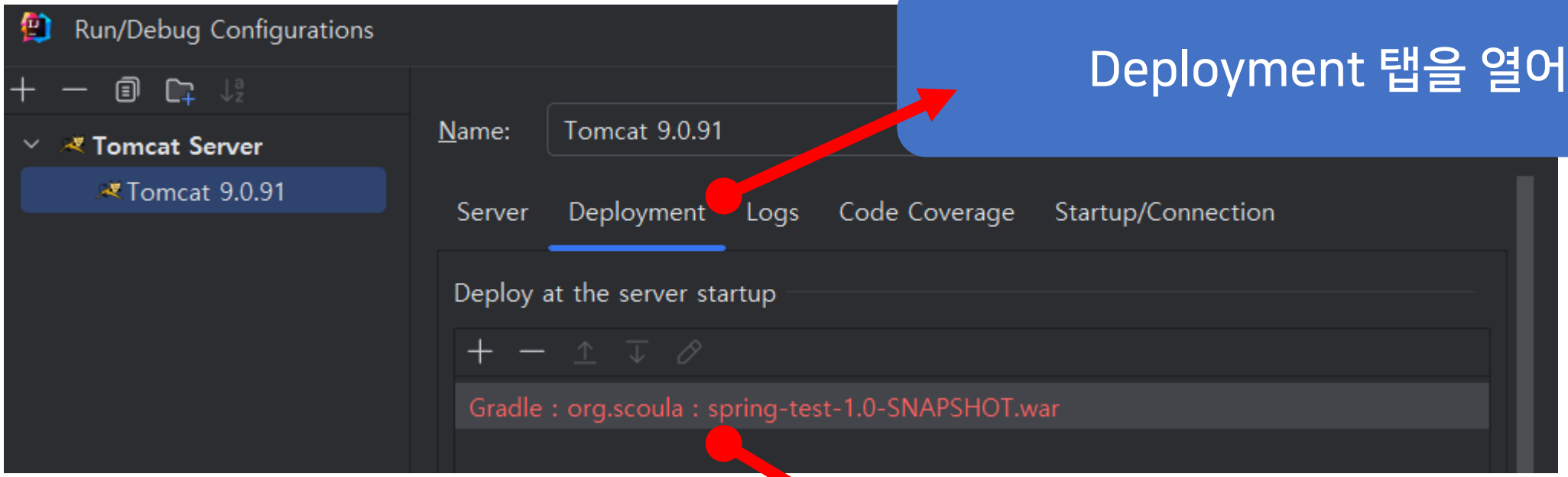
타입 상태 보고

설명 Origin 서버가 대상 리소스를 위한 현재의 representation을 찾지 못했거나, 그것이 존재하는지를

Apache Tomcat/9.0.91

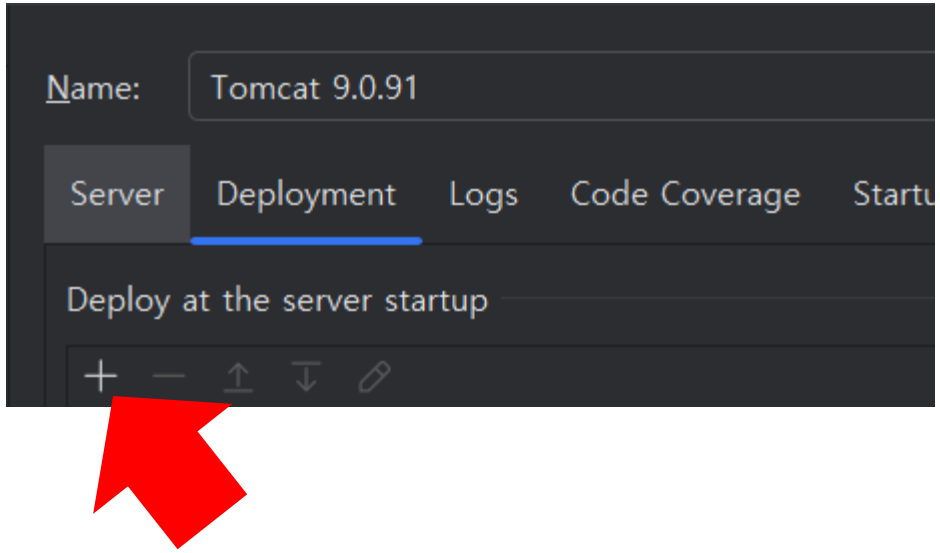


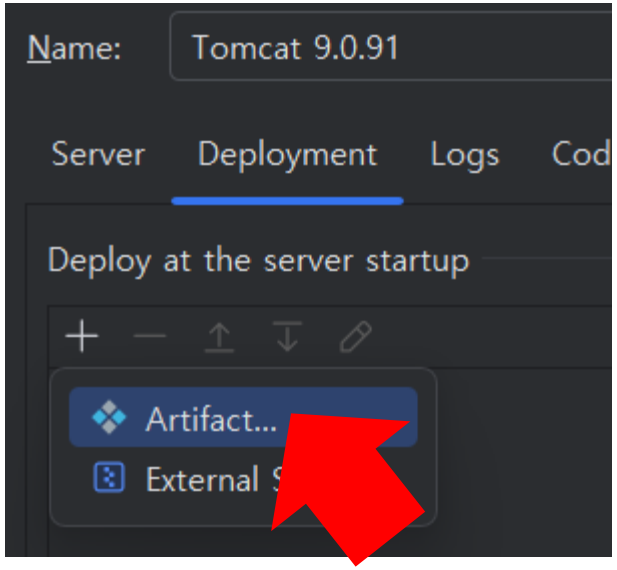
서버 설정 열어주기!

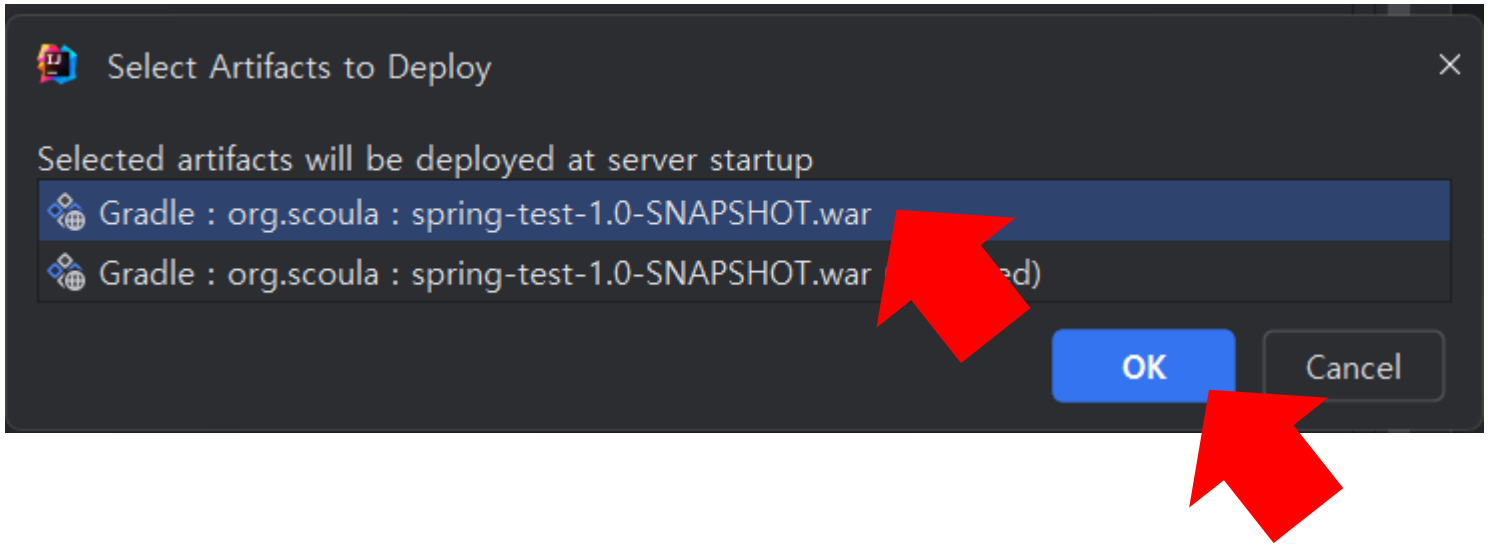


Deployment 탭을 열어 주세요!

기존에 자동 생성된 ~~.war 를 삭제!


Name:
Tomcat 9.0.91
Server
Deployment
Logs
Code Coverage
Deploy at the server startup


Name:
Tomcat 9.0.91
Server
Deployment
Logs
Deploy at the server startup
Artifact...


Select Artifacts to Deploy
Selected artifacts will be deployed at server startup
Gradle : org.scoula : spring-test-1.0-SNAPSHOT.war
Gradle : org.scoula : spring-test-1.0-SNAPSHOT.war
OK
Cancel

Name: Tomcat 9.0.91

Store as project file

Server | Deployment | Logs | Code Coverage | Startup/Connection

Deploy at the server startup

Gradle : org.scoula : spring-test-1.0-SNAPSHOT.war

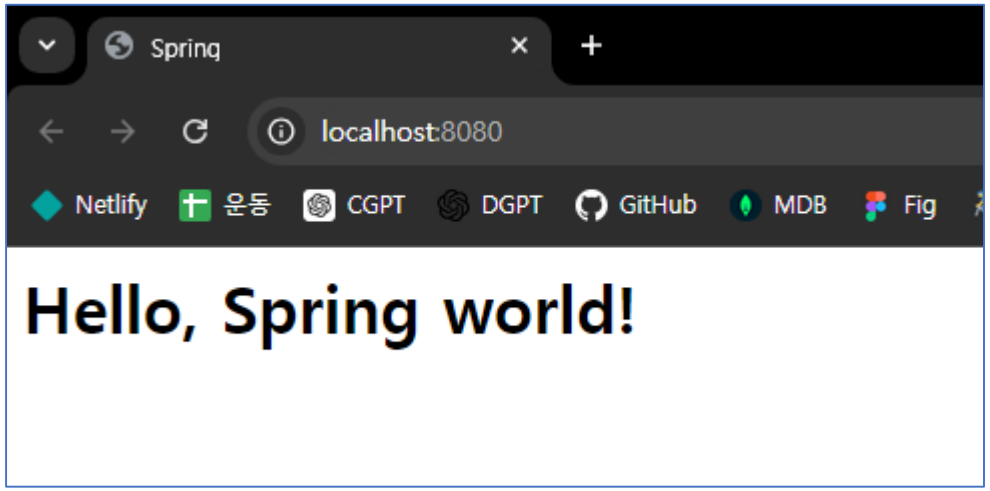
Application context: /

Run | OK | Cancel | Apply

자동으로 등록된 컨텍스트는 삭제하고
기본 주소인 "/" 만 남겨 주세요!

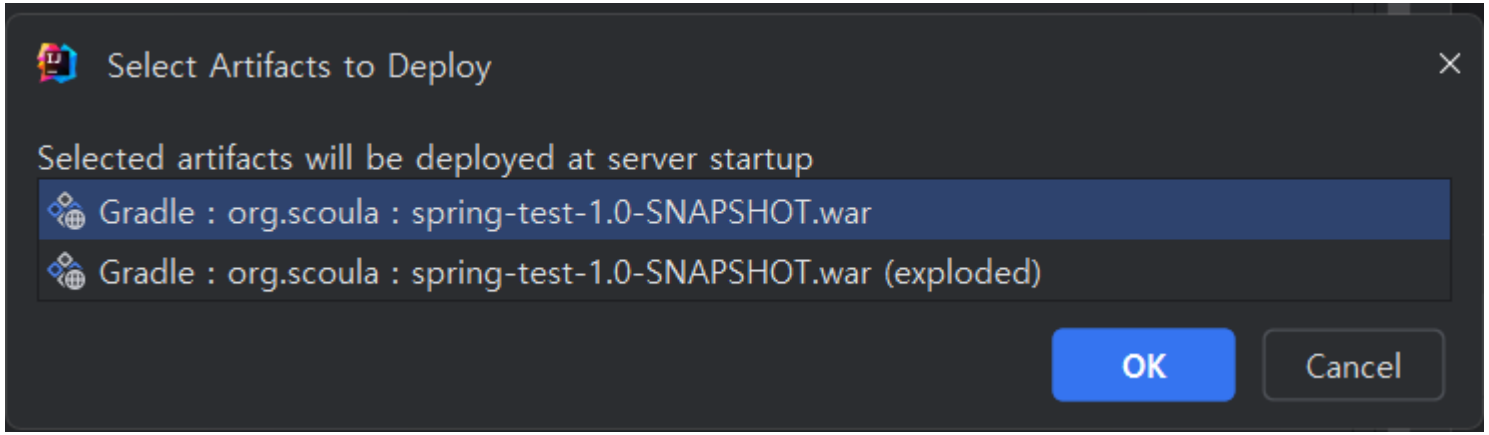
삭제 후, OK 클릭!


spring
자~ 이제 시작이야(내꿈을)


Spring
localhost:8080
Netlify
운동
CGPT
DGPT
GitHub
MDB
Fig
Hello, Spring world!

그란데 말입니다

요기 아래의 war 에 대한
설정은 왜 해줘야 하나요!?




spring
WAR
파일

WAR(Web Application Archive)

- 자바로 웹 어플리케이션을 배포하기위한 아카이브 파일
- WAR 내부에는 서블릿, JSP, HTML 그리고 모든 라이브러리 리소스를 포함
- 결국 톰캣 서버에는 해당 WAR 가 올라가고, 해당 WAR 에 의해 최종 서비스가 결정

자 설명을 들으셨으니 생각해 봅시다!

그럼 왜 다시 만들었을까요?

프로젝트 템플릿

저장하기

프로젝트 최상위 폴더 우클릭 →
Open In → Explorer 선택!

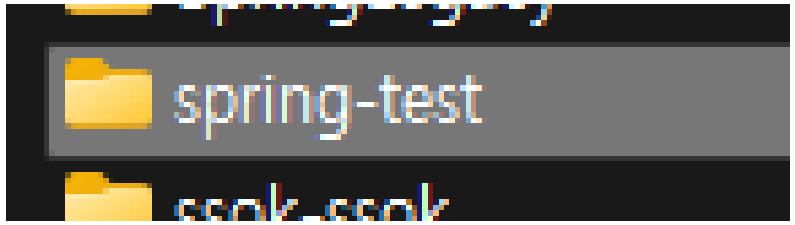
Project

New
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference...
Paste Ctrl+V
Find Usages Alt+F7
Find in Files... Ctrl+Shift+F
Replace in Files... Ctrl+Shift+R
Analyze
Refactor
Bookmarks
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Remove Module Delete
Build Module 'SpringLegacy'
Rebuild Module 'SpringLegacy' Ctrl+Shift+F9
Run 'Tests in 'SpringLega...' Ctrl+Shift+F10
Debug 'Tests in 'SpringLega...'
More Run/Debug
Open In
Explorer
Open In Associated Application

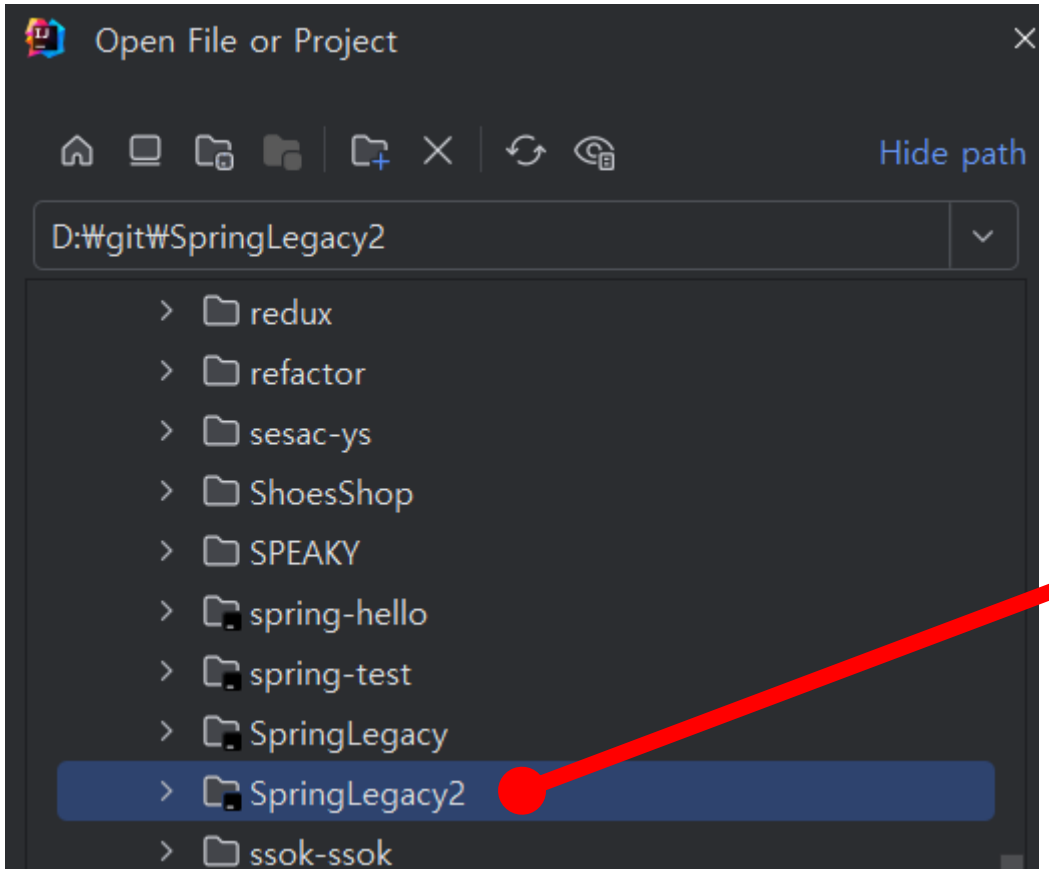
ad>
y>
<h1>Hello, Spring world!</
dy>
ml>

Services
Server
Tomca
: org.scoula : spring

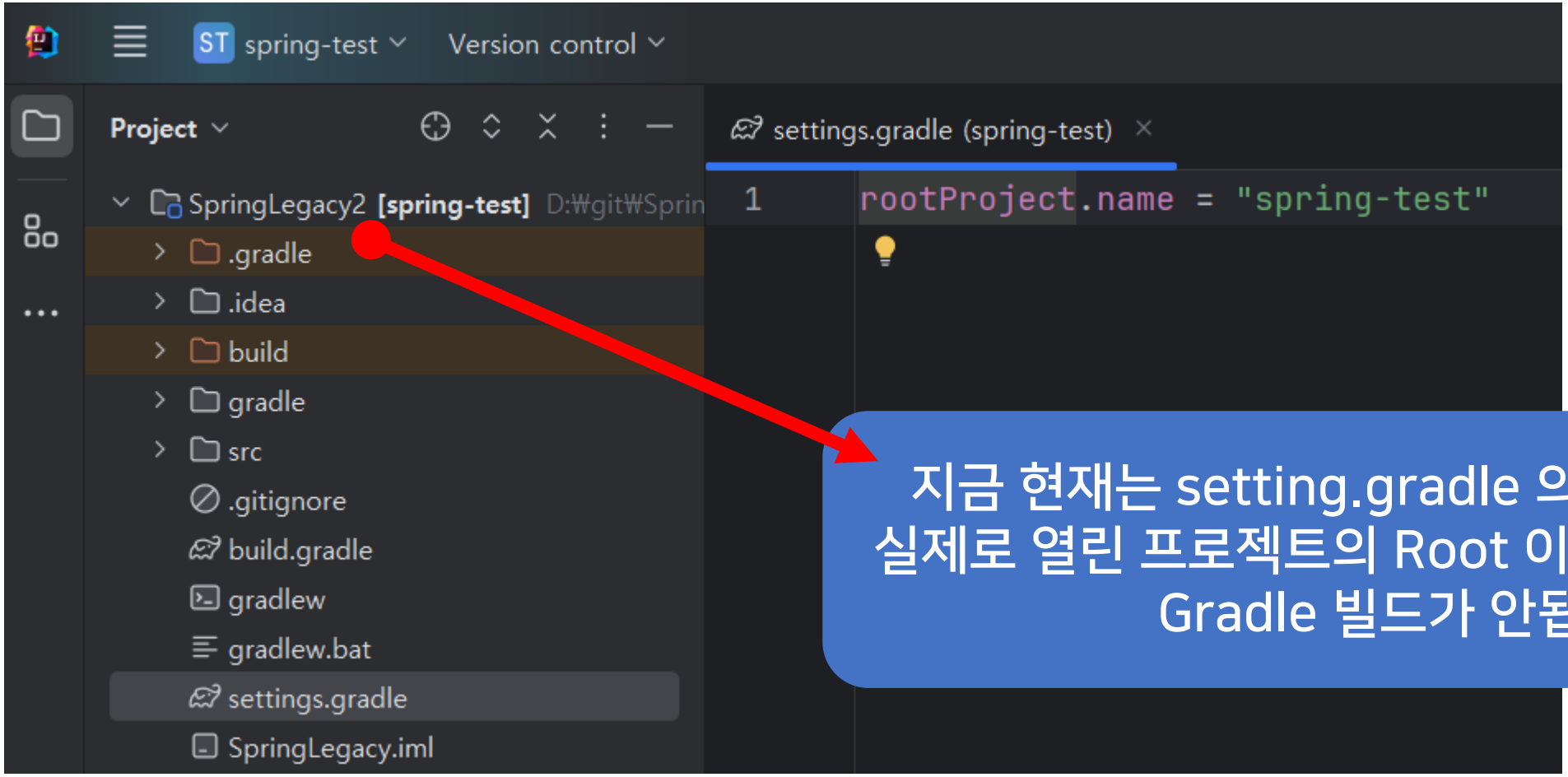
File → New Projects Setup
→ Save Project as Template 선택



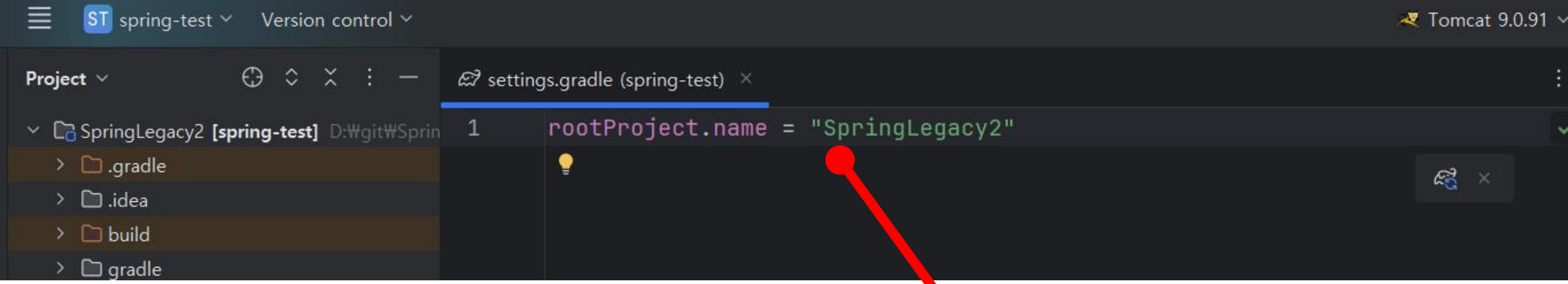
프로젝트 폴더 명을
SpringLegacy 로 변경
(저는 이미 만든 프로젝트가 있어서
2로 만들었습니다!)



→ 인텔리제이를 통해서 변경한 폴더를 열어주기!

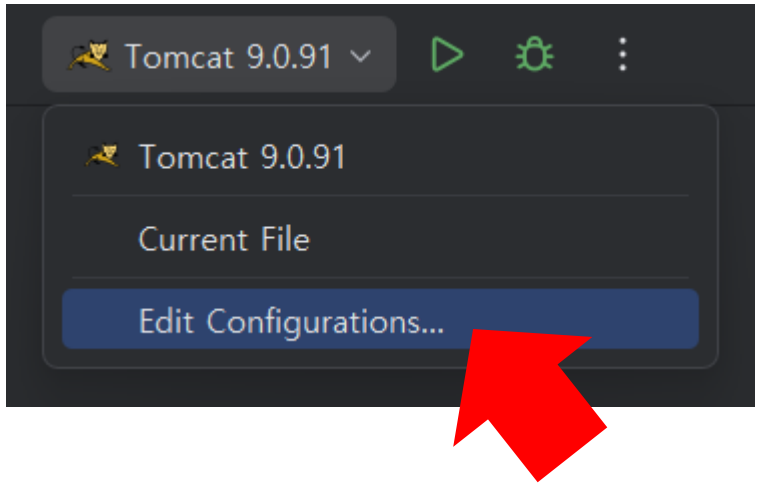


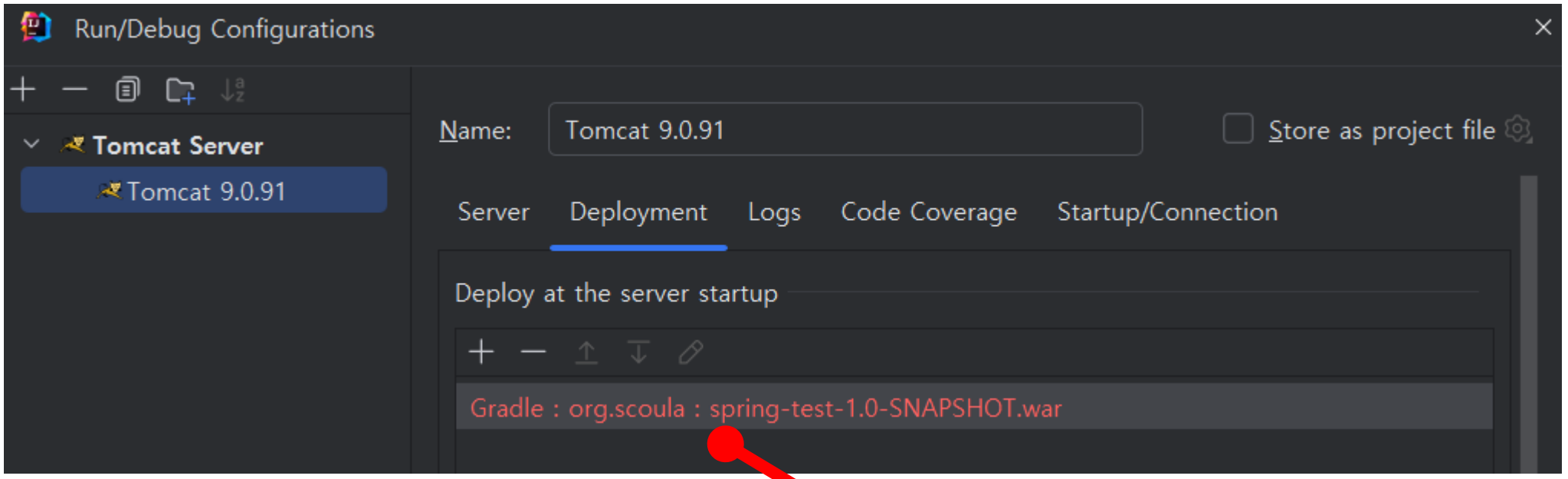
지금 현재는 setting.gradle 의 프로젝트 이름과
실제로 열린 프로젝트의 Root 이름이 다르기 때문에
Gradle 빌드가 안됩니다!



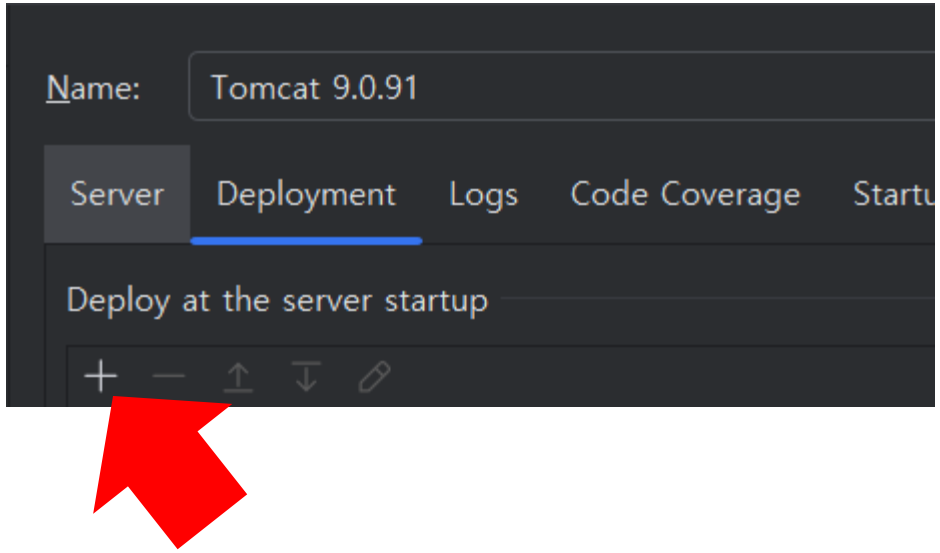
실제로 열린 프로젝트의 Root 이름으로
setting.gradle 의 프로젝트 이름을 변경!

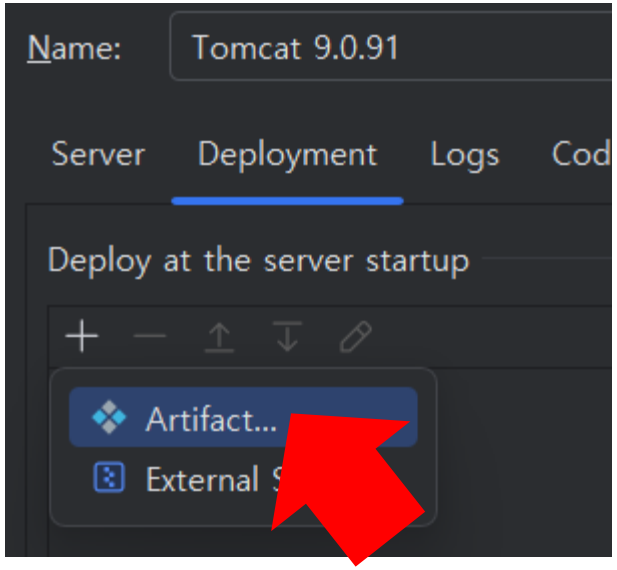
→ 코끼리 눌러서 Sync 실행!

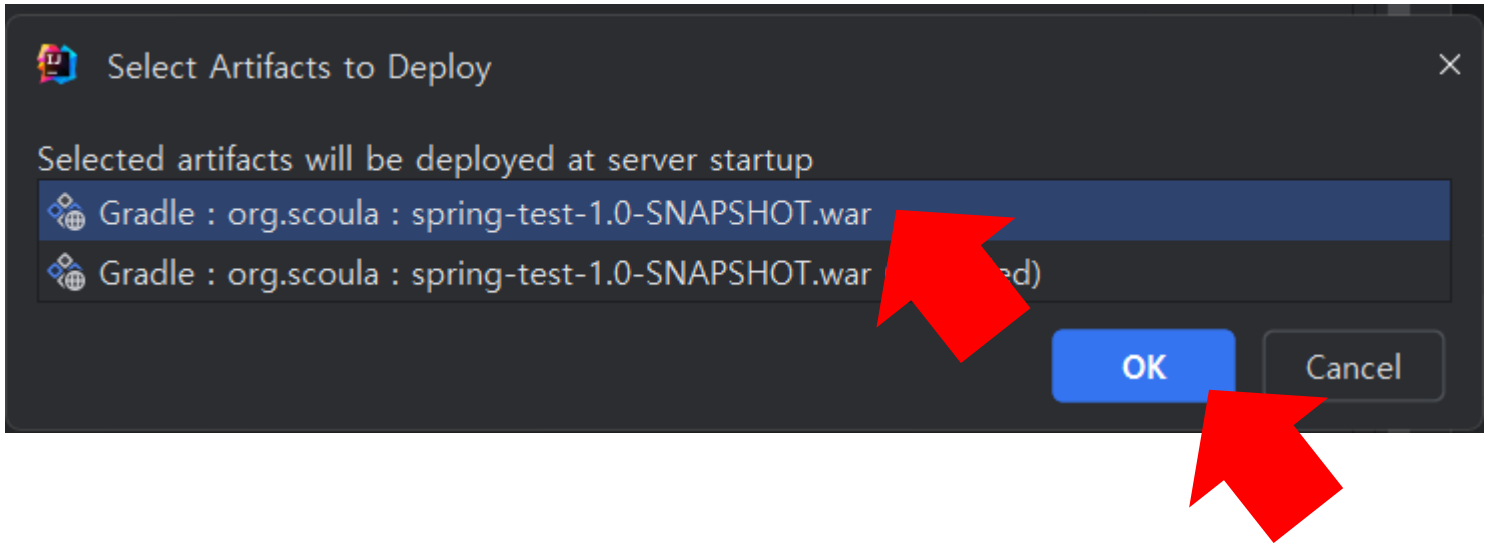

Tomcat 9.0.91
Tomcat 9.0.91
Current File
Edit Configurations...



프로젝트 설정이 변경 된 것이므로
기존 WAR 파일을 제거하고 다시 생성!


Name:
Tomcat 9.0.91
Server
Deployment
Logs
Code Coverage
Deploy at the server startup


Name:
Tomcat 9.0.91
Server
Deployment
Logs
Deploy at the server startup
Artifact...


Select Artifacts to Deploy
Selected artifacts will be deployed at server startup
Gradle : org.scoula : spring-test-1.0-SNAPSHOT.war
Gradle : org.scoula : spring-test-1.0-SNAPSHOT.war
OK
Cancel

Name: Tomcat 9.0.91

Store as project file

Server | Deployment | Logs | Code Coverage | Startup/Connection

Deploy at the server startup

Gradle : org.scoula : spring-test-1.0-SNAPSHOT.war

Application context: /

Run | OK | Cancel | Apply

자동으로 등록된 컨텍스트는 삭제하고
기본 주소인 "/" 만 남겨 주세요!

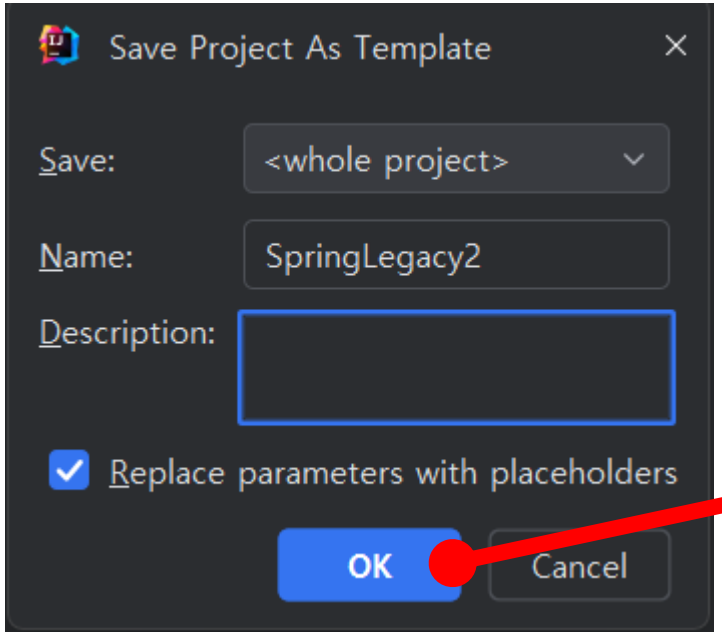
삭제 후, OK 클릭!

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help

- New
- Open...
- Recent Projects
- Close Project
- Remote Development...
- Settings... Ctrl+Alt+S
- Project Structure... Ctrl+Alt+Shift+S
- File Properties
- Local History
- Save All Ctrl+S
- Reload All from Disk Ctrl+Alt+Y
- Repair IDE
- Invalidate Caches...
- Manage IDE Settings
- New Projects Setup
- Save File as Template...
- Export
- Print...
- Power Save Mode

- Settings for New Projects...
- Run Configuration Templates or New Projects...
- Structure...
- Save Project as Template...
- Manage Project Templates...

템플릿 저장을 위해 클릭!



원하는 이름으로 저장 후, OK 클릭!

New Project

Generators
- Maven Archetype
- Jakarta EE
- Spring Boot
- JavaFX
- Quarkus
- Micronaut
- Ktor
- Compose for Desktop
- HTML
- React
- Express
- Angular CLI
- Vue.js
- Vite

Templates
- SpringLegacy3
- SpringLegacy2
- SpringLegacy

More via plugins...

Name: demo

Location: D:₩git

Project will be created in: D:₩git₩demo

Create Git repository

Template: REST service JAX-RS resource

Application server: Tomcat 9.0.91 New...

Language: Java Kotlin Groovy

Build system: Maven Gradle

Group: org.example

Artifact: demo

JDK:

Next Cancel

이제 새프로젝트 생성 시,
저장한 템플릿 확인 가능!

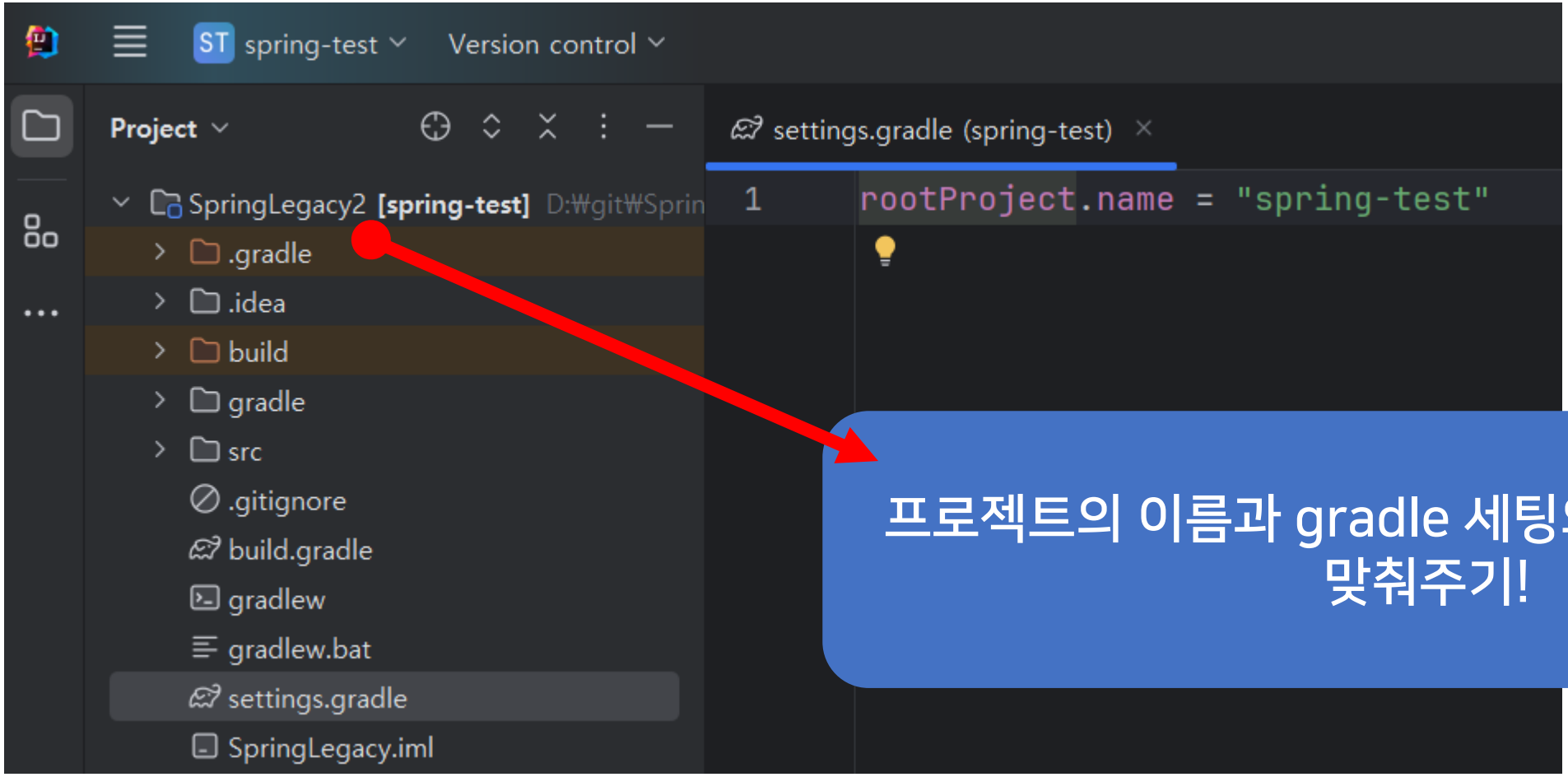
템플릿으로 새로운

프로젝트를 만들면!?

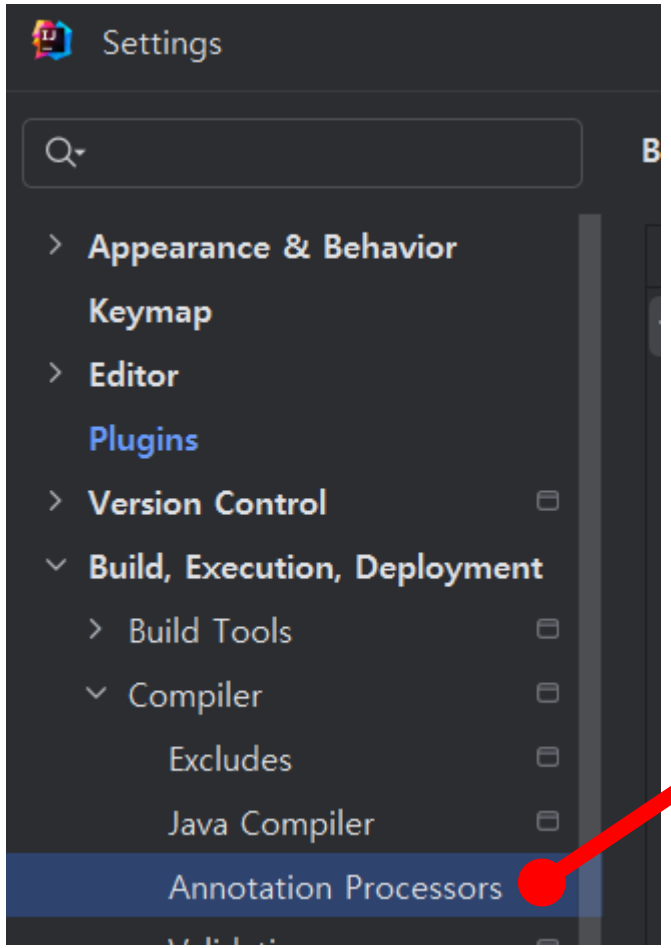
1. 프로젝트 이름과 settings.gradle 의 이름 맞추기

2. Annotation Processing 옵션 확인

3. 서버 설정 → Deployment 에 가서 새로운 WAR 파일 생성

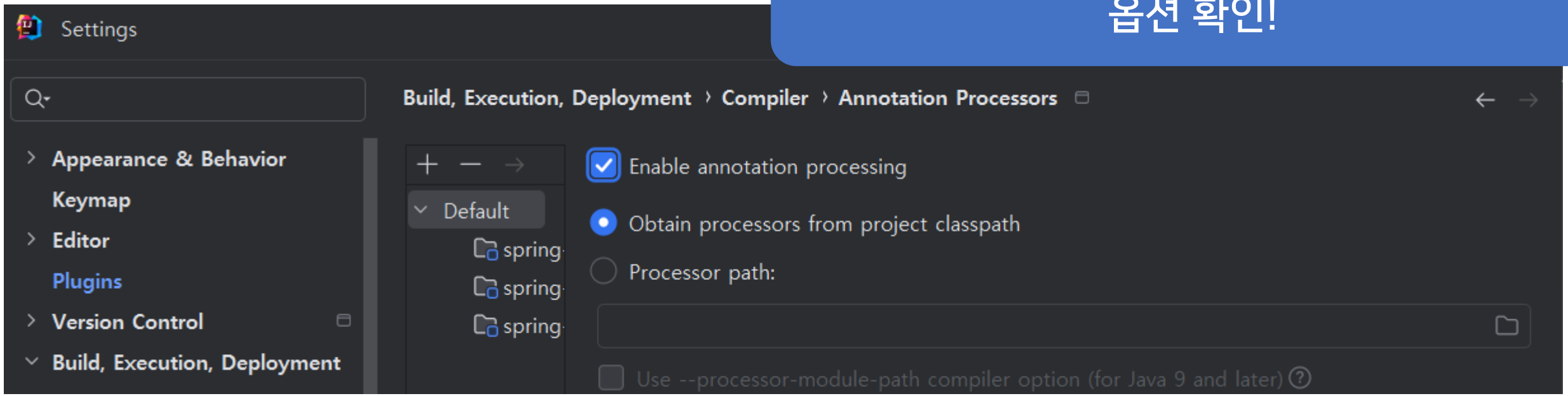


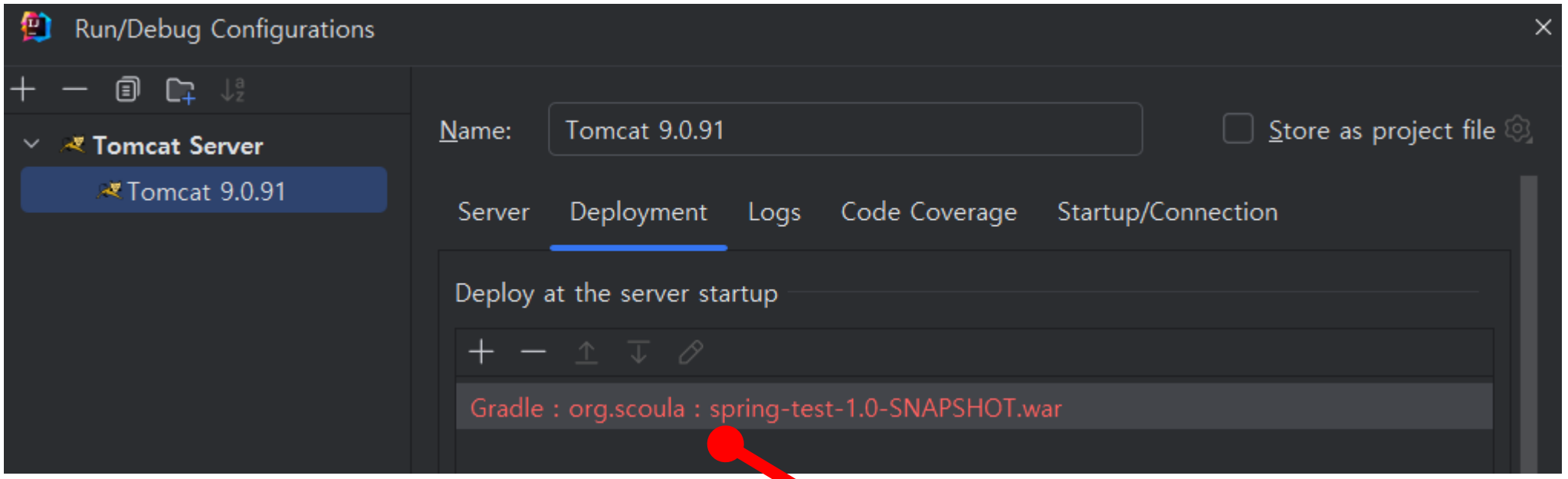
프로젝트의 이름과 gradle 세팅의 이름을 동일하게 맞춰주기!



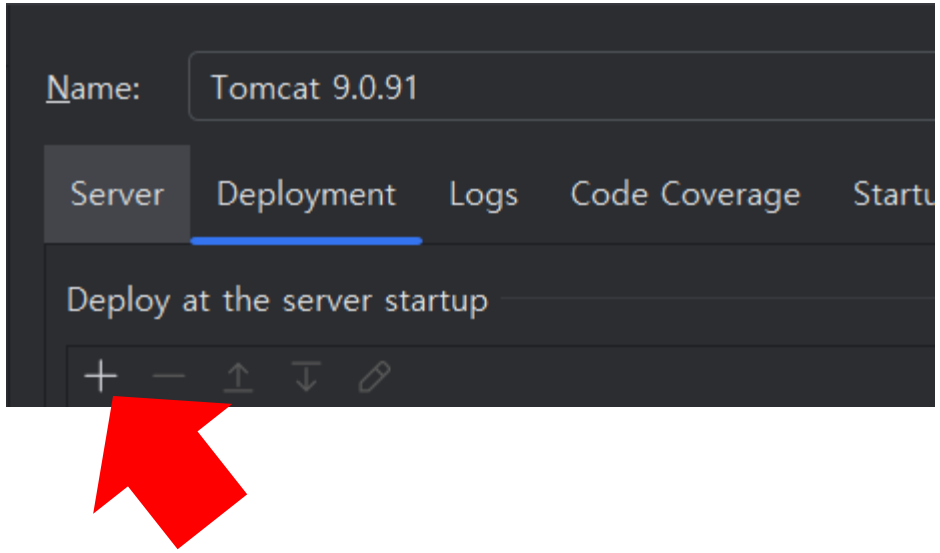
Build, Execution, Deployment 선택 →
Compiler 선택 →
Annotation Processors 선택

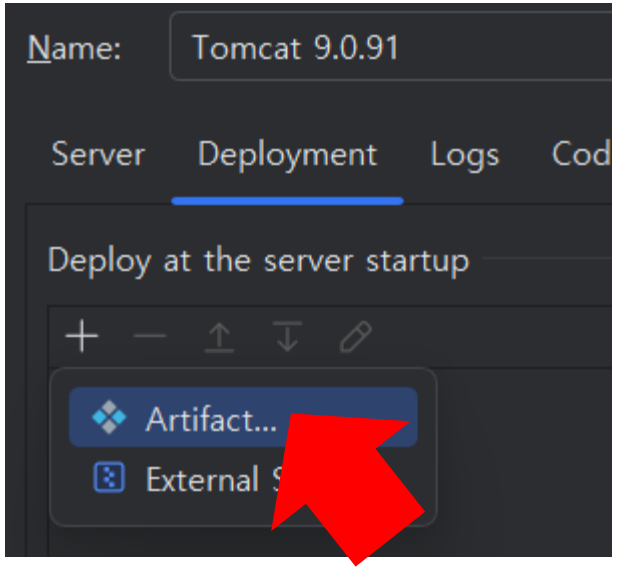
Enable annotation processing
옵션 확인!

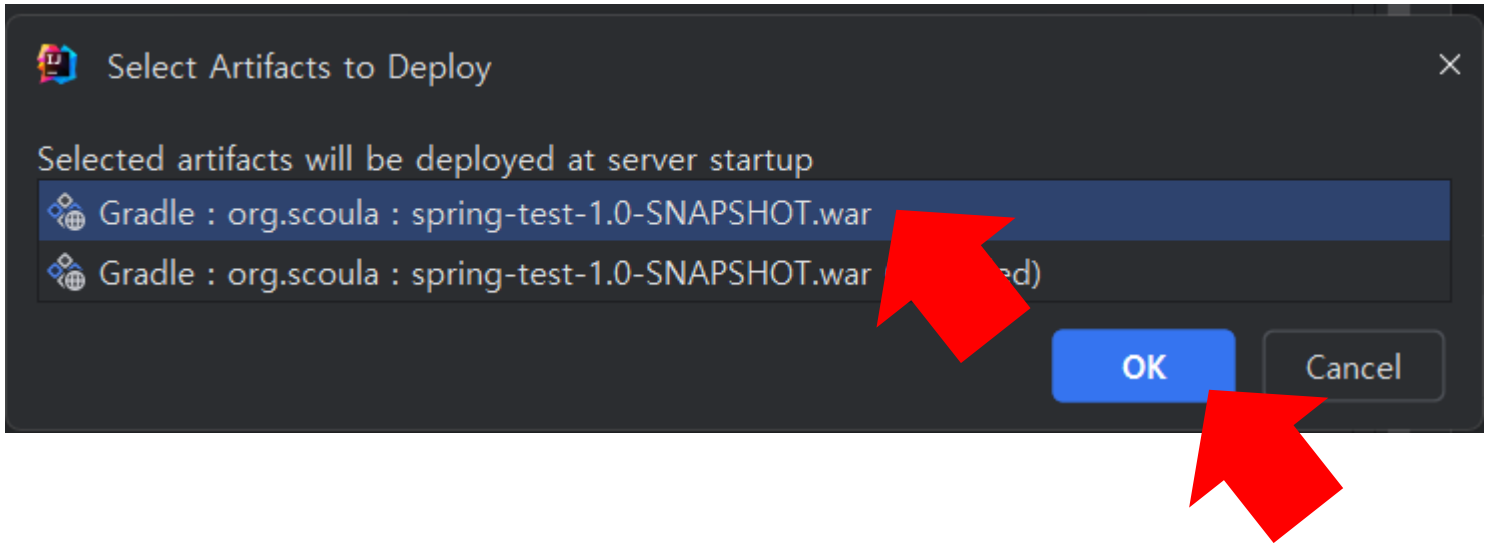




프로젝트 설정이 변경 된 것이므로
기존 WAR 파일을 제거하고 다시 생성!


Name:
Tomcat 9.0.91
Server
Deployment
Logs
Code Coverage
Deploy at the server startup


Name:
Tomcat 9.0.91
Server
Deployment
Logs
Deploy at the server startup
Artifact...


Select Artifacts to Deploy
Selected artifacts will be deployed at server startup
Gradle : org.scoula : spring-test-1.0-SNAPSHOT.war
Gradle : org.scoula : spring-test-1.0-SNAPSHOT.war
OK
Cancel

Name: Tomcat 9.0.91

Store as project file

Server | Deployment | Logs | Code Coverage | Startup/Connection

Deploy at the server startup

Gradle : org.scoula : spring-test-1.0-SNAPSHOT.war

Application context: /

Run | OK | Cancel | Apply

자동으로 등록된 컨텍스트는 삭제하고
기본 주소인 "/" 만 남겨 주세요!

삭제 후, OK 클릭!