# 개발을 위한 준비



번외 의존성 주입 Dependency Injection에 대하여.
의존성 : 하나의 객체가 다른 객체 없이 제대로 된 역할을 할 수 없다는 것을 의미.
하나의 객체가 다른 객체의 상태에 따라 영향을 받는 것.
상황 예시) 객체 A의 멤버로 참조 B를 갖는다 할 때. 해당멤버로 다양한 연산함.
객체 A는 실객체 B없이 동작이 불가능한 상황. A가 B에 의존적이다.

주입은 : 말그대로 외부에서 밀어 넣는 것.
일반 식당은 재료를 직접 구하지만(필요한 것을 스스로 구함)
프랜차이즈 식당은 본사에서 트럭으로 재료를 주입함.(외부에서 필요한것을 주입)

의존성 주입 : 어떤 객체가 필요한 객체를 외부에서 밀어 넣는 것.
그렇다면 왜 이 의존성 주입방식을 사용할까?
장점: 장사에 집중할 수있다. 자신의 역할에만 집중할수있다.

A객체에서 B객체를 직접 생성하여 사용하는 거시 아닌/ A객체에게 외부에서 필요한 B를 주입
필요한 것을 주입해주는 외부 존재하나가 더 필요함.

스프링은 이러한 구조를 만드는데 적합한 구조로 설계되어있다.

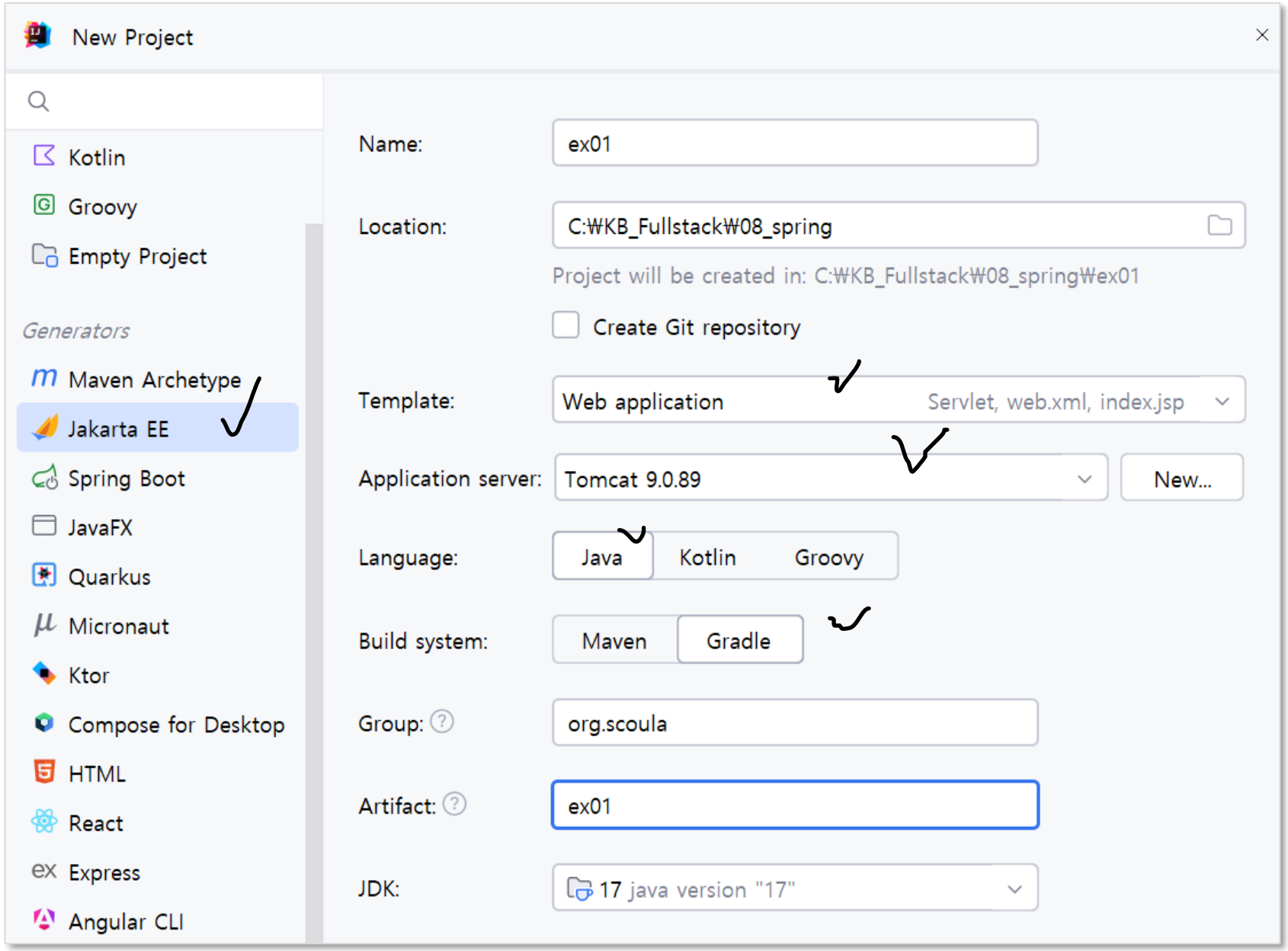
스프링에서는 application context 라는 존재가 필요한 객체를 생성하고
주입하는 구조.

주입하는것은 스프링이 아니까 우리 개발자들은 객체를 만들고 application context라는존재가
주입하기 쉽게 wiring작업을 하면된다.

실질적으로 application context는 객체들을 관리하고 이를 bean이라부르며
빈과 빈사이의 의존관계 정의(wiring)은
xml, annotation, java설정 방식을 이용하여 wiring하면된다!

## workspace 준비

- C:₩KB_Fullstack₩08_Spring

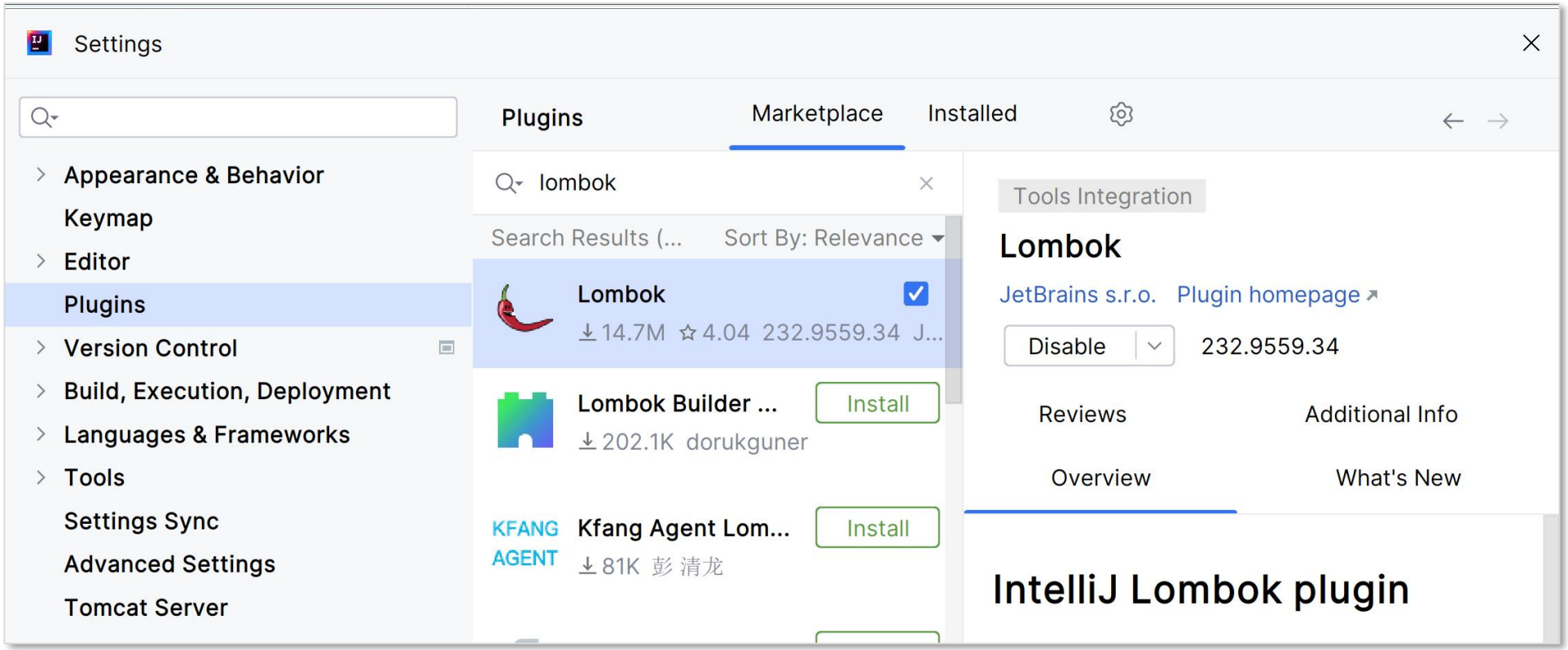

New Project
Kotlin
Groovy
Empty Project
Generators
Maven Archetype
Jakarta EE
Spring Boot
JavaFX
Quarkus
Micronaut
Ktor
Compose for Desktop
HTML
React
Express
Angular CLI
Name:
ex01
Location:
C:₩KB_Fullstack₩08_spring
Project will be created in: C:₩KB_Fullstack₩08_spring₩ex01
Create Git repository
Template:
Web application
Servlet, web.xml, index.jsp
Application server:
Tomcat 9.0.89
New...
Language:
Java
Kotlin
Groovy
Build system:
Maven
Gradle
Group:
org.scoula
Artifact:
ex01
JDK:
17 java version "17"

- Version: Jakarta EE 8
  - Servlet 4.0.1

## Lombok Plugin 설치

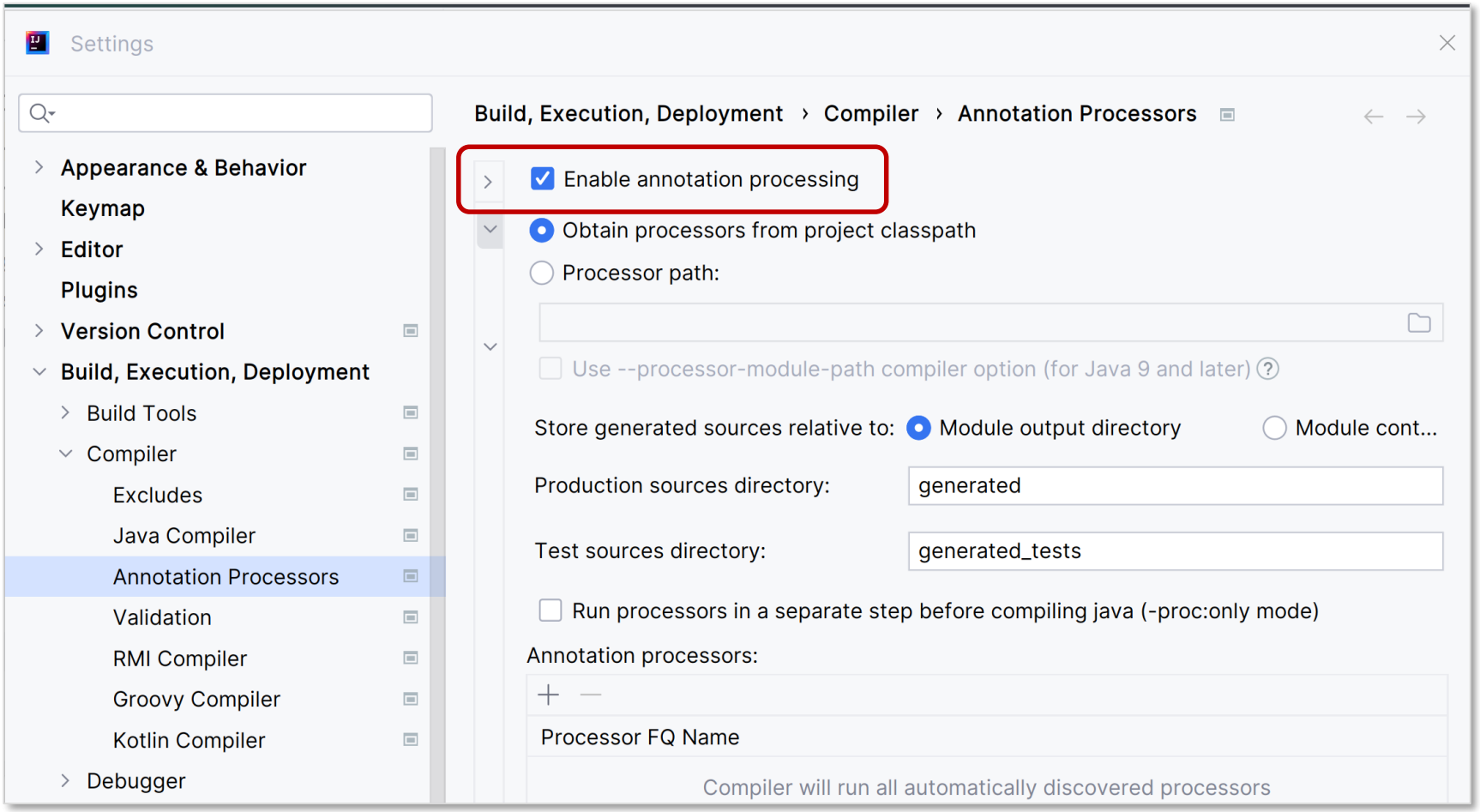
우린 준비되어있으니까 넘어가고ㅃ이

- File > Settings... > Plugins
  - Marketplace에서 Lombok 검색 후 설치



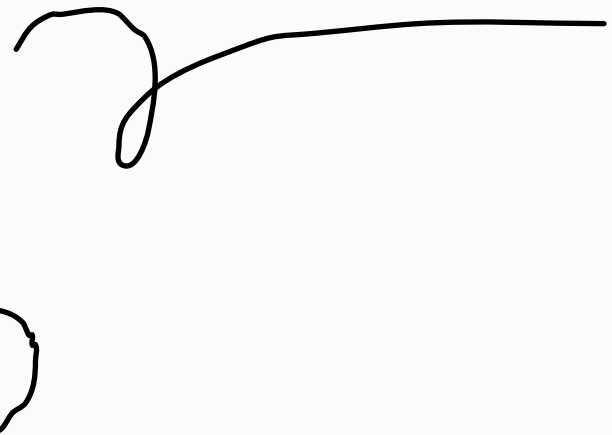
## Lombok 설정

- File > Setting... > Build, Exeuction, Deployment > Compiler > Annotation Processors
  - Enable annotation processing 체크 → **프로젝트 생성시 마다 해줘야 함**



## build.gradle

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

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

repositories {
  mavenCentral()
}

ext {
  junitVersion = '5.9.2'
  springVersion = '5.3.37'
  lombokVersion = '1.18.30'
}

sourceCompatibility = '1.17'
targetCompatibility = '1.17'

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



ext필드 안에 있는 것들은

나중에 dependencies 필드에서
${} 로 할 수 있음

## build.gradle

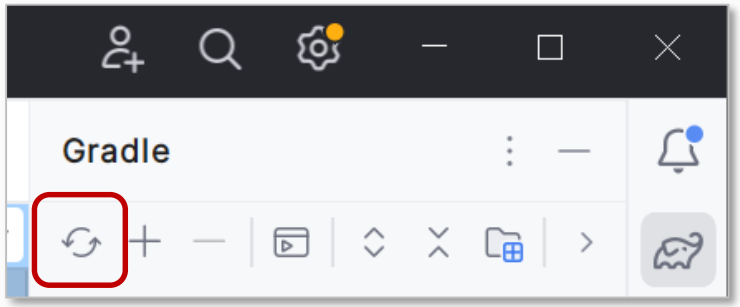
변수를 사용할때는 괄호 써야함

```
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'
  compileOnly 'javax.servlet.jsp:javax.servlet.jsp-api:2.3.3'
  implementation 'javax.servlet:jstl:1.2'

  // Log4j2 Logging
  implementation 'org.apache.logging.log4j:log4j-api:2.18.0'
  implementation 'org.apache.logging.log4j:log4j-core:2.18.0'
  implementation 'org.apache.logging.log4j:log4j-slf4j-impl:2.18.0'

  // xml내 한글 처리
  implementation 'xerces:xercesImpl:2.12.2'
```

로그 시스템 사용. 이걸 통해서만 출력

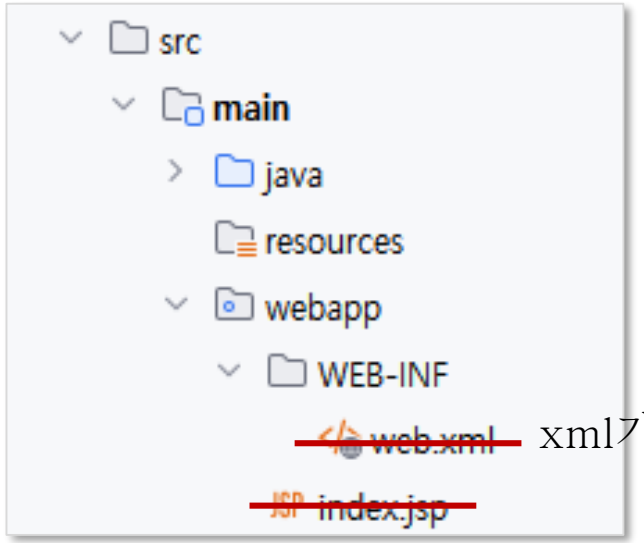
## build.gradle

```
  // Lombok
  compileOnly "org.projectlombok:lombok:${lombokVersion}"
  annotationProcessor "org.projectlombok:lombok:${lombokVersion}"

  // Jackson - Json 처리
  implementation 'com.fasterxml.jackson.core:jackson-databind:2.9.4'

  // 테스트
  testImplementation "org.springframework:spring-test:${springVersion}"
  testCompileOnly"org.projectlombok:lombok:${lombokVersion}"
  testAnnotationProcessor "org.projectlombok:lombok:${lombokVersion}"

  testImplementation("org.junit.jupiter:junit-jupiter-api:${junitVersion}")
  testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junitVersion}")
}


test {
  useJUnitPlatform()
}
```



## 디렉토리 구조

- java
  - Java 클래스 배치
- resources
  - Java 클래스를 제외한 모든 파일(설정, xml 등)
- webapp
  - Web URL로 접근하는 root 디렉토리
  - WEB-INF
    - 웹 설정, 스프링 View 배치 디렉토리
    - index.jsp, web.xml 삭제



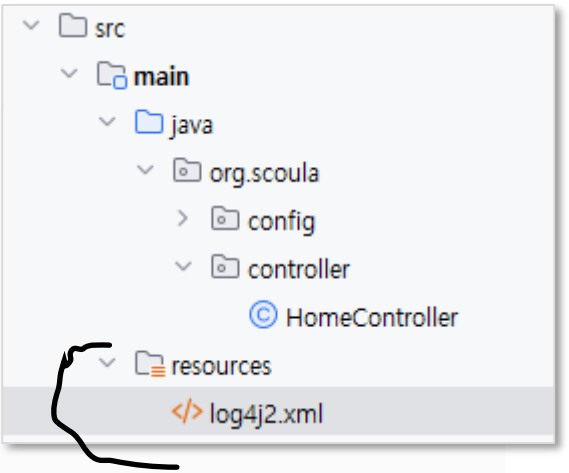

xml기반 안할거야

프론트 컨트롤러 패턴 사용할거야. 이미 스프링이 적용해놔

web0inf밑에 views밑에 jsp들 만들고 배치할 예정

로그 시스템 설정 xml

## resources/log4j2.xml




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

    <!-- Appender, Layout 설정 -->
    <Appenders>
        <Console name="console" target="SYSTEM_OUT">
            <PatternLayout charset="UTF-8" pattern=" %-5level %c(%M:%L) - %m%n"/>
        </Console>
    </Appenders>

    <!-- Logger 설정 -->
    <Loggers>
        <Root level="INFO">
            <AppenderRef ref="console"/>
        </Root>
        <Logger name="org.scoula" level="INFO" additivity="false" >
            <AppenderRef ref="console"/>
        </Logger>
        <Logger name="org.springframework" level="INFO" additivity="false">
            <AppenderRef ref="console"/>
        </Logger>
    </Loggers>

</Configuration>
```

메세지 출력 패턴

출력장치

출력 레벨

## 로그의 레벨 설정

- 너무 많은 로그 메시지 출력 → 로그 레벨 이용하여 수정 가능
- 로그레벨

심각

- FATAL : 가장 크리티컬한 에러가 일어 났을 때만 로깅
- ERROR : 일반 에러가 일어 났을 때만 로깅
- WARN : 에러는 아니지만 주의할 필요가 있을 때만 로깅
- INFO : 일반 정보를 나타낼 때 로깅 (INFO + ERROR)
- DEBUG : 일반 정보를 상세히 나타낼 때 로깅 (디버깅 정보)

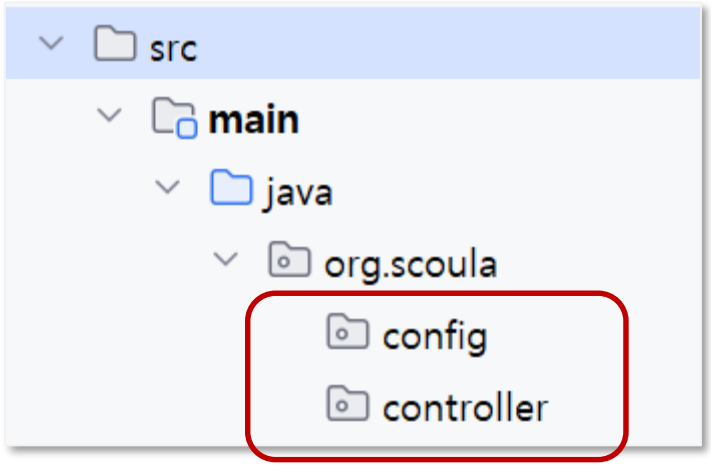
낮은

- TRACE : 경로추적을 위해 사용

현재 레벨보다 높은 것들은 출력됨

## 패키지 생성

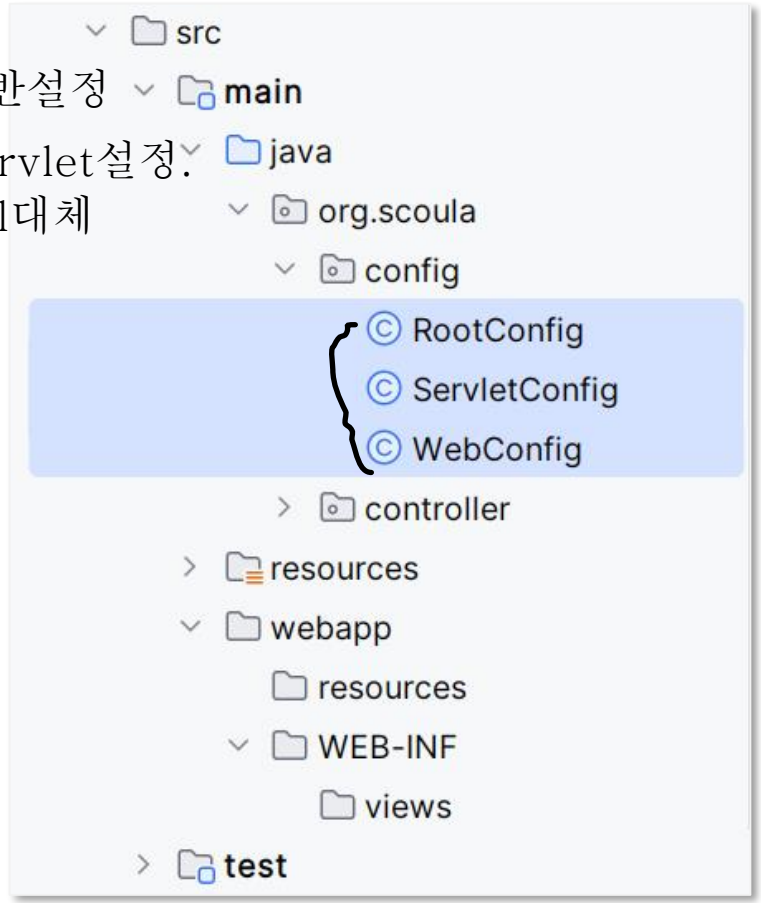
- src/main/java에 패키지 생성
  - org.scoula
    - config : 스프링 설정 패키지
    - controller : 기본 컨트롤러 패키지



rootconfig.java 일반설정 범용데이터 관리, 디비 연결 설정.
servletconfig.java frontcontroller설정 컨트롤러 관리
webconfig.java web.xml 설정 앱관련 설정 필터링 캐릭터셋 등등

## 설정 파일 만들기

- org.scoula.config
  - RootConfig.java → 일반설정
  - ServletConfig.java → dispatchservlet설정.
  - WebConfig.java → web.xml대체



## RootConfig.java

```
package org.scoula.config;

import org.springframework.context.annotation.Configuration;

@Configuration
public class RootConfig {

}
```

보통 디비 설정이 들어간다

## ServletConfig.java

front controller에 대한 설정. dispatcher servlet에 대한 설정.

```
package org.scoula.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@EnableWebMvc
@ComponentScan(basePackages = {"org.scoula.controller"})     // Spring MVC용 컴포넌트 등록을 위한 스캔 패키지
public class ServletConfig implements WebMvcConfigurer {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry
            .addResourceHandler("/resources/**")     // url이 /resources/로 시작하는 모든 경로
            .addResourceLocations("/resources/");    // webapp/resources/경로로 매핑
    }
```

* 그 레벨에서 뭐든 상관 없다.
** 그레벨 포함 모든 하위까진 뭐든 상관없다

/resource 요청이면
정적 처리

요 경로에서 찾아라.
js css image 등등이 있는 곳

## ServletConfig.java

```
    // jsp view resolver 설정                                    요청 처리및 해석 어떻게 할 것인지
    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {
        InternalResourceViewResolver bean = new InternalResourceViewResolver();  jsp 엔진 사용

        bean.setViewClass(JstlView.class);jsp엔진 결정
        bean.setPrefix("/WEB-INF/views/");
        bean.setSuffix(".jsp");

        registry.viewResolver(bean);jsp로 고
    }

}
```

## WebConfig.java

톰캣에게 앱에대해서 알리는 역할

```
package org.scoula.config;

import org.springframework.web.filter.CharacterEncodingFilter;
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

import javax.servlet.Filter;

public class WebConfig extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { RootConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { ServletConfig.class };
    }

    // 스프링의 FrontController인 DispatcherServlet이 담당할 Url 매핑 패턴, / : 모든 요청에 대해 매핑
    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
```

이 친구가 web.xml를 대체하는 함

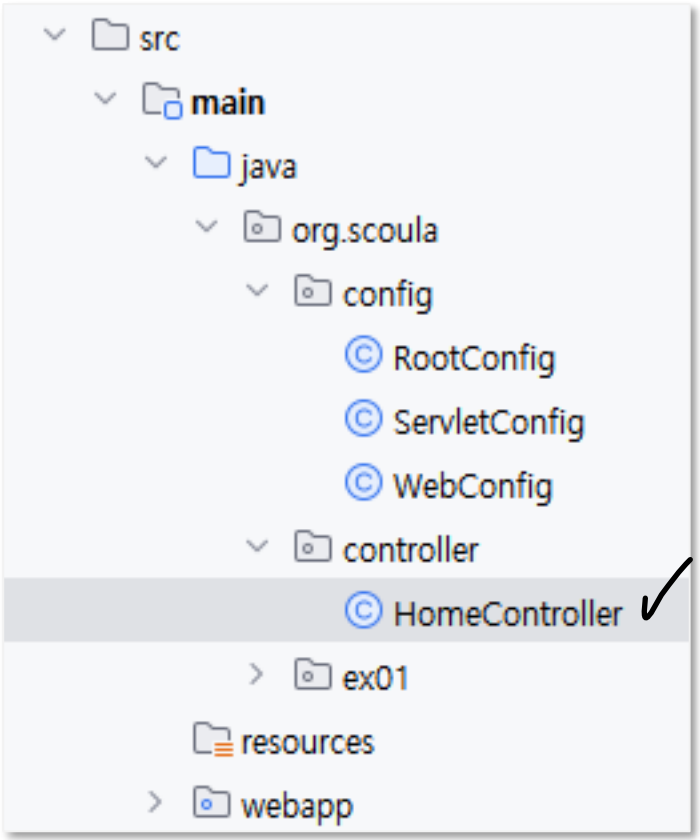
모든 요청이 dispatch servlet으로

## WebConfig.java

```
    // POST body 문자 인코딩 필터 설정 - UTF-8 설정
    protected Filter[] getServletFilters() {
        CharacterEncodingFilter characterEncodingFilter = new CharacterEncodingFilter();

        characterEncodingFilter.setEncoding("UTF-8");
        characterEncodingFilter.setForceEncoding(true);

        return new Filter[] {characterEncodingFilter};
    }

}
```

## 컨트롤러 만들기

- org.scoula.controller
  - HomeController.java



## HomeController.java

@Controller는 @Component의 자식이다

```
package org.scoula.controller;

import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller 빈등록하고 컨트롤러 인식함
@Log4j2
public class HomeController {

    @GetMapping("/")        이 요청오면 이 메소드 호출해라
    public String home() {
        log.info("================> HomController /");
        return "index";    // View의 이름
    }
                접두어+"반환값"+접미어 ==> jsp경로 결정됨
}
```

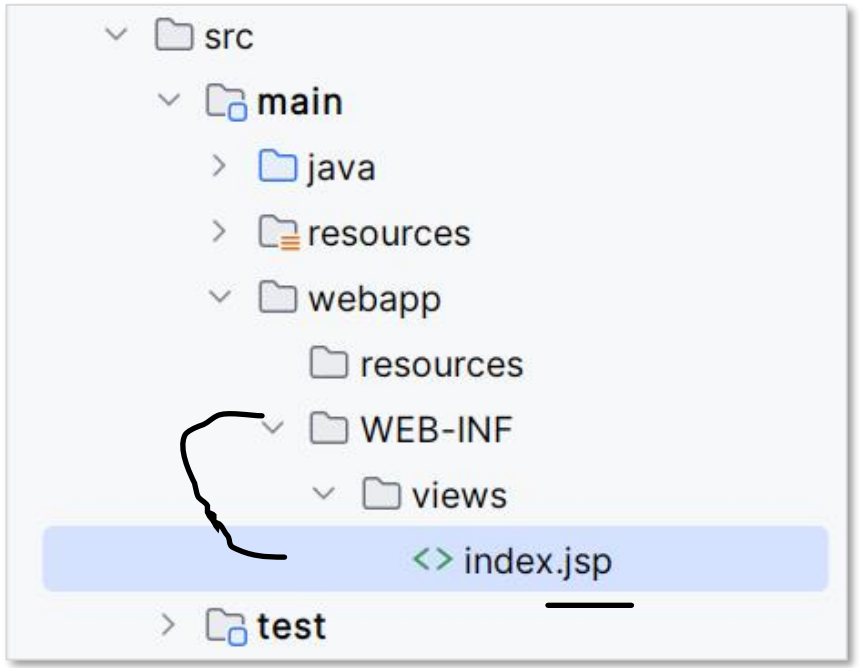
지난번에 getMap메서드와 대응한다.
이번엔 어노테이션으로

접두어 접미어는 아까 servletconfig파일(dispatchservlet설정)에서 정해놨죠?

## Jsp 파일 만들기

- src/main/webapp
  - WEB-INF/views/index.jsp

```
@Override
public void configureViewResolvers(ViewResolverRegistry registry) {
  InternalResourceViewResolver bean = new InternalResourceViewResolver();

  bean.setViewClass(JstlView.class);
  bean.setPrefix("/WEB-INF/views/");
  bean.setSuffix(".jsp");

  registry.viewResolver(bean);
}
```



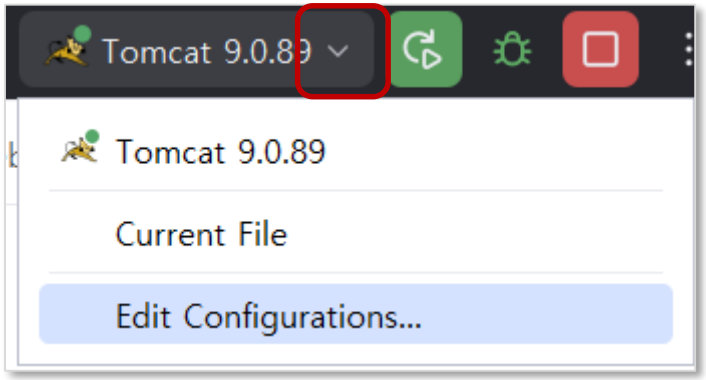

- controller에서 index라는 view 이름을 리턴한 경우
  /WEB-INF/views/ + view 이름 + .jsp
  → /WEB-INF/views/index.jsp

## views/index.jsp

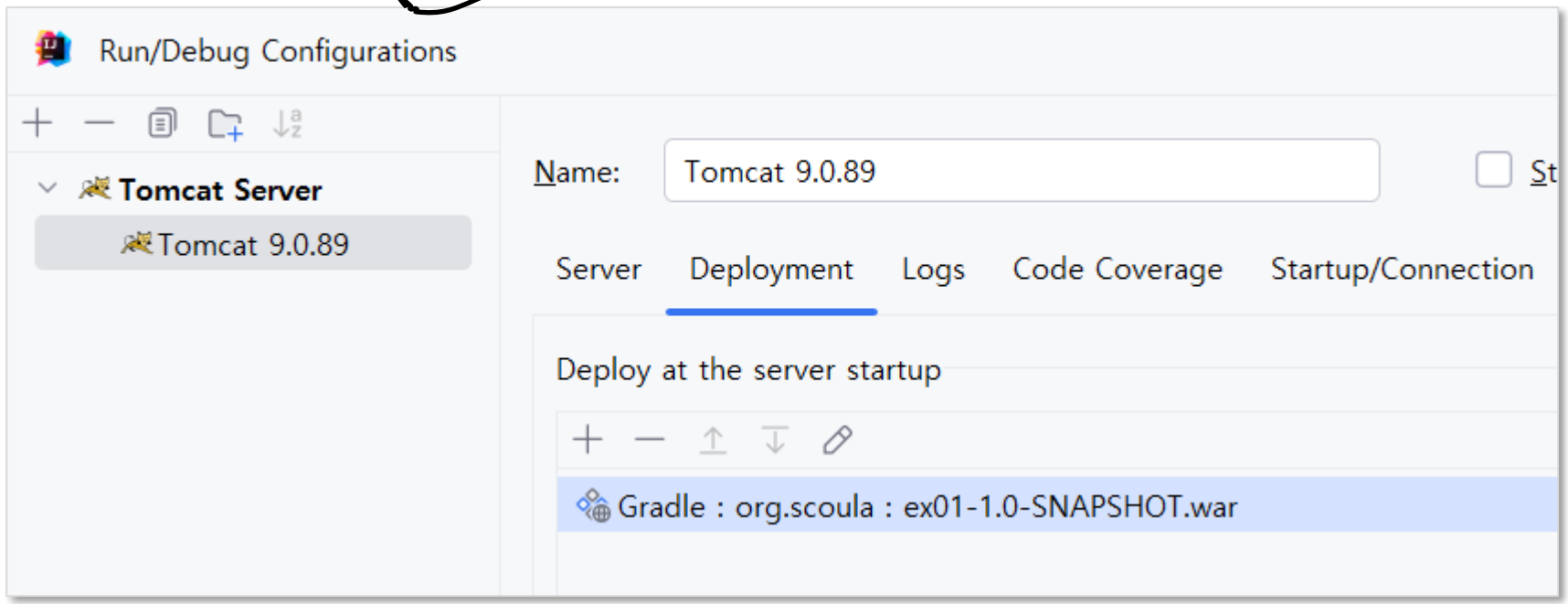
```
<!DOCTYPE html>
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <h1>환영합니다.</h1>
</body>
</html>
```

## 톰캣 설정

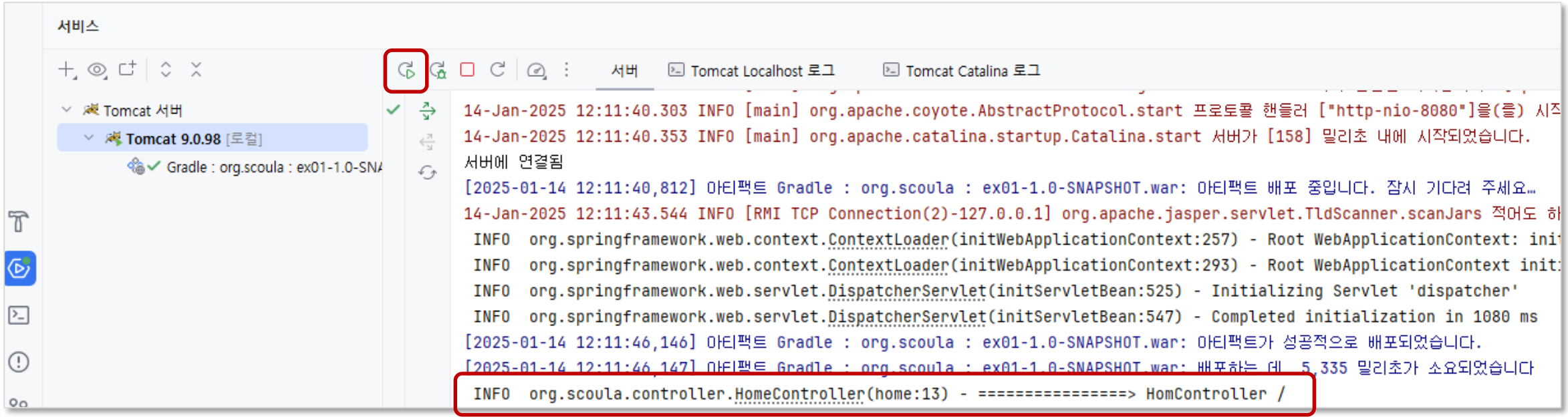
실행 설정 수정




- Application context: /

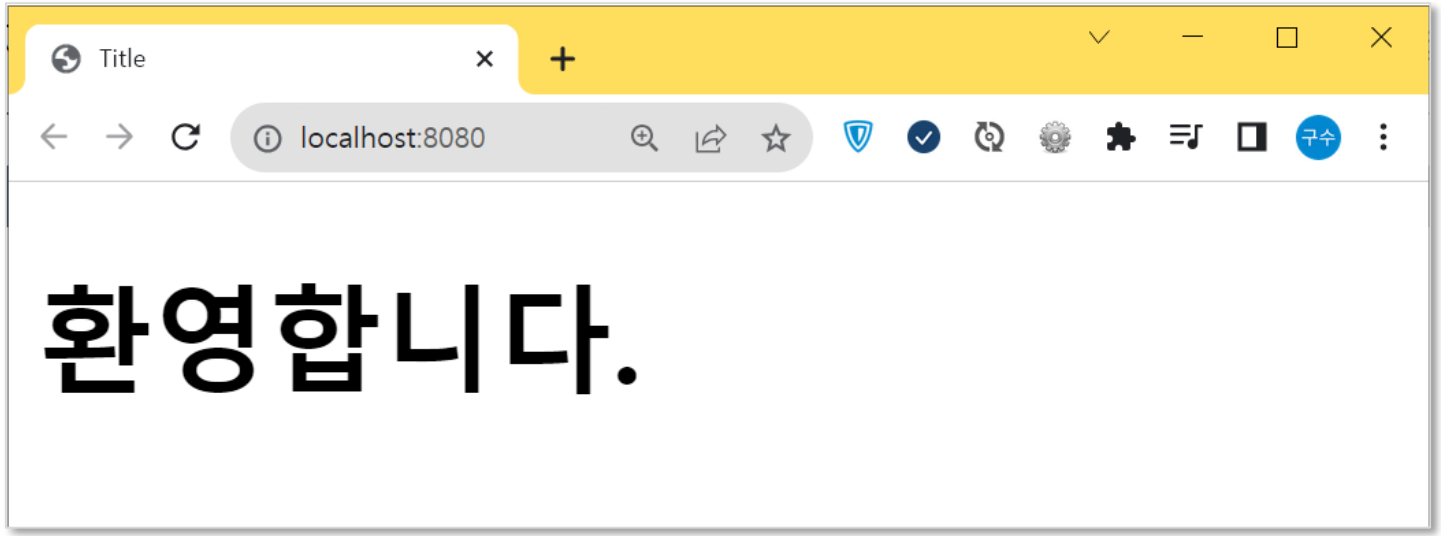


## 애플리케이션 실행


서비스
서버
Tomcat Localhost 로그
Tomcat Catalina 로그
Tomcat 서버
Tomcat 9.0.98 [로컬]
Gradle : org.scoula : ex01-1.0-SNA
14-Jan-2025 12:11:40.303 INFO [main] org.apache.coyote.AbstractProtocol.start 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작
14-Jan-2025 12:11:40.353 INFO [main] org.apache.catalina.startup.Catalina.start 서버가 [158] 밀리초 내에 시작되었습니다.
서버에 연결됨
[2025-01-14 12:11:40,812] 아티팩트 Gradle : org.scoula : ex01-1.0-SNAPSHOT.war: 아티팩트 배포 중입니다. 잠시 기다려 주세요…
14-Jan-2025 12:11:43.544 INFO [RMI TCP Connection(2)-127.0.0.1] org.apache.jasper.servlet.TldScanner.scanJars 적어도 하
INFO org.springframework.web.context.ContextLoader(initWebApplicationContext:257) - Root WebApplicationContext: ini
INFO org.springframework.web.context.ContextLoader(initWebApplicationContext:293) - Root WebApplicationContext init
INFO org.springframework.web.servlet.DispatcherServlet(initServletBean:525) - Initializing Servlet 'dispatcher'
INFO org.springframework.web.servlet.DispatcherServlet(initServletBean:547) - Completed initialization in 1080 ms
[2025-01-14 12:11:46,146] 아티팩트 Gradle : org.scoula : ex01-1.0-SNAPSHOT.war: 아티팩트가 성공적으로 배포되었습니다.
[2025-01-14 12:11:46,147] 아티팩트 Gradle : org.scoula : ex01-1.0-SNAPSHOT.war: 배포하는 데 5,335 밀리초가 소요되었습니다
INFO org.scoula.controller.HomeController(home:13) - ================> HomController /

## 확인

- http://localhost:8080/



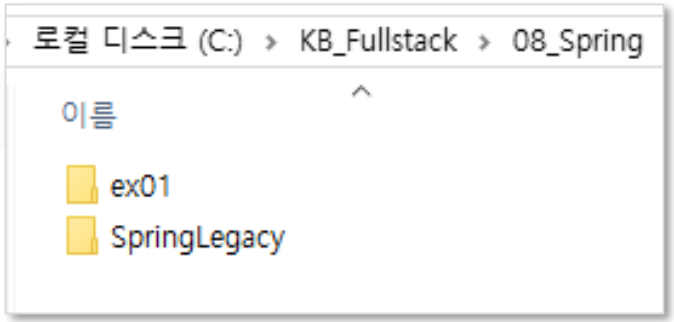
# 개발을 위한 준비

## 프로젝트 템플릿 만들기

- 탐색기에서 ex01 프로젝트 폴더를 SpringLegacy이름으로 복사 후 SpringLegacy 프로젝트 열기
- SpringLegacy/settings.gradle

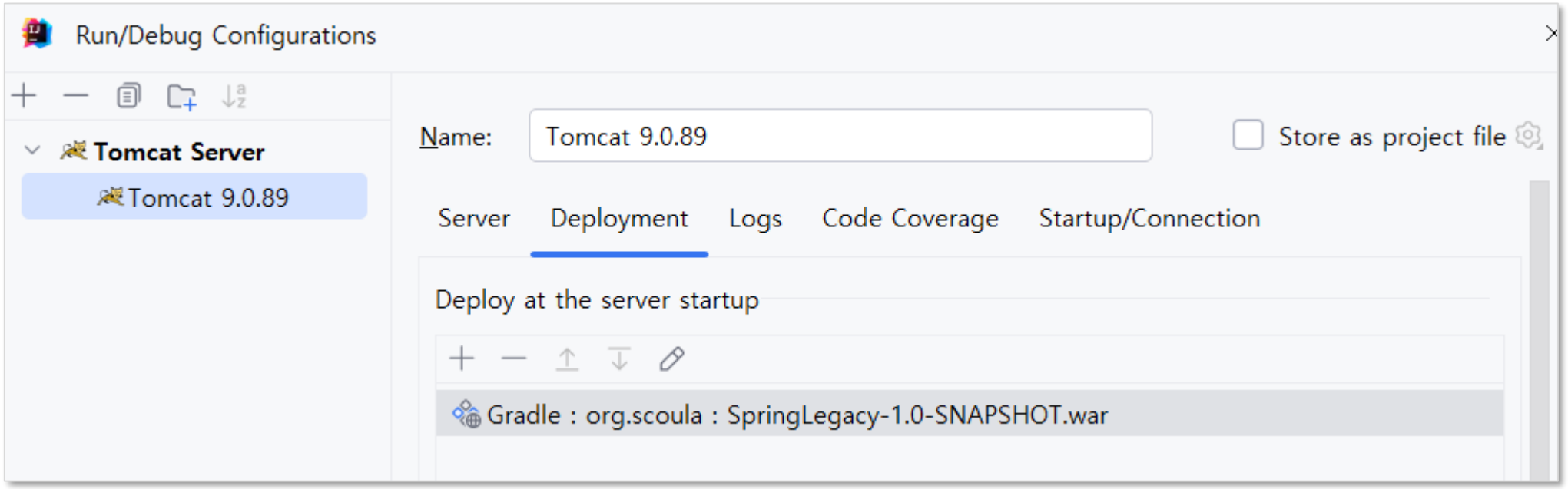
```
rootProject.name = "SpringLegacy"
```

  - 수정 후 Sync



## 톰캣 설정 수정 → ArtifactId 변경

```
rootProject.name = "SpringLegacy"
```



## 프로젝트 템플릿 만들기

- File > New Projects Setup > Save Project as Template...