수업 목표

스프링 프레임워크를 이용한 개발을 위해서 필요한 프로그램들과 이에 관련된 설정들이 필요합니다. 이 장에서는 개발에 앞서 실습에 필요한 공통적인 환경을 준비합니다.

- JDK 17 설치
- Eclipse 설치와 Lombok 설정
- Tomcat 10 과 웹 프로젝트의 생성
- 스프링 프레임워크 실행
- Log4j2 의 설정
- Maria DB 의 설치와 설정

0.1. Spring Framework 6와 JDK

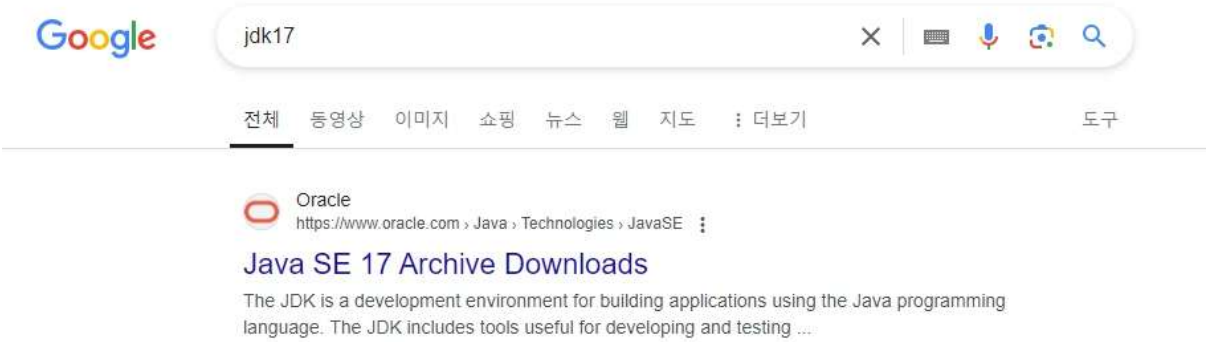
스프링 프레임워크의 경우 2024 년 10 월 현재 6.1.13 버전까지 배포된 상황입니다(최근에 스프링이라는 용어는 스프링 부트를 의미하는 경우가 많으므로 정확히는 스프링 6 레거시라고 표현합니다만 예제에서는 스프링으로 칭하도록 합니다.). 스프링 프레임워크는 사용하려는 버전에 따라서 JDK 나 WAS(Web Application Server)의 버전을 주의해야 합니다.

스프링 프레임워크 버전	JDK	Jakarta EE (과거 JavaEE)
6.x	17 이상	Tomcat 10.x 이상
5.x	8,11,17(5.3.13 부터 JDK17 지원)	Tomcat 9.0 이상
4.x	6,7,8	Tomcat 6,7
3.x	5,6,7	Tomcat 6,7

스프링 6 버전의 프로젝트를 생성하려면 반드시 JDK 17 이상을 이용해야 하므로 실습에 필요한 환경 역시 JDK 17 을 기준으로 모든 환경을 구성하도록 합니다.

JDK17 다운로드와 설치

JDK 의 설치는 검색 엔진을 이용해서 Open JDK 나 Oracle 이 제공하는 JDK 를 이용하면 됩니다. 다만 버전이 17 이상인 점에 주의합니다.



JDK 는 클릭만으로 설치할 수 있는 Installer 표시가 있는 항목을 이용하는 것이 편리합니다.

macOS x64 Compressed Archive	170.85 MB	https://download.oracle.com/java/17/archive/jdk-17.0.11_macos-x64_bin.tar.gz (sha256)
macOS x64 DMG Installer	170.26 MB	https://download.oracle.com/java/17/archive/jdk-17.0.11_macos-x64_bin.dmg (sha256)
Windows x64 Compressed Archive	172.83 MB	https://download.oracle.com/java/17/archive/jdk-17.0.11_windows-x64_bin.zip (sha256)
Windows x64 Installer	153.91 MB	https://download.oracle.com/java/17/archive/jdk-17.0.11_windows-x64_bin.exe (sha256)
Windows x64 MSI Installer	152.66 MB	https://download.oracle.com/java/17/archive/jdk-17.0.11_windows-x64_bin.msi (sha256)

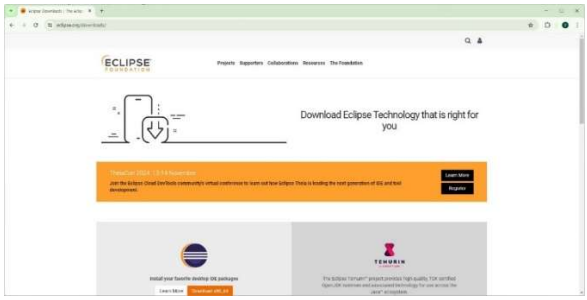
JDK 의 설치시에는 설치 경로를 주의 깊게 봐두도록 합니다. 설치 경로는 나중에 Eclipse 의 설정에 영향을 주기 때문에 메모장 등에 경로를 보관하는 것이 좋습니다.

0.2. Eclipse IDE 설정

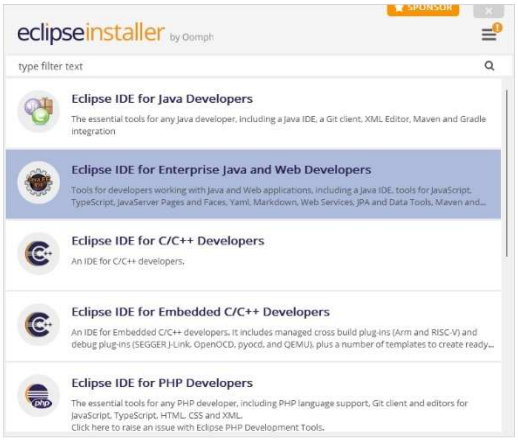
스프링 6 레거시 프로젝트는 대부분의 개발 도구에서 지원되지 않는 경우가 많습니다. 이것은 스프링 프레임워크가 공식적으로는 스프링 부트와 통합되었기 때문에 개발도구들 역시 스프링 레거시 방식의 지원을 중단했기 때문입니다.

스프링 6 레거시의 개발에 권장되는 개발 도구는 Eclipse 와 Intellij Ultimate 버전입니다. Eclipse 의 경우 무료라는 장점이 있으므로 처음 시작하는 분들에게는 적합합니다.

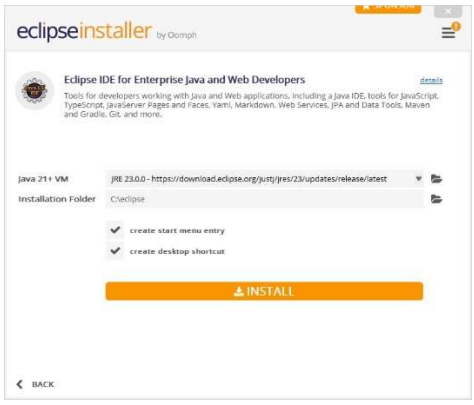
Eclipse 는 매년 6 월말을 기준으로 새로운 버전이 배포되므로, 최신 버전을 다운로드 하는 경우 약간의 설정을 변경해 줄 필요가 있습니다. eclipse.org 사이트를 이용하면 자신의 운영체제에 맞는 프로그램을 다운로드 할 수 있습니다.

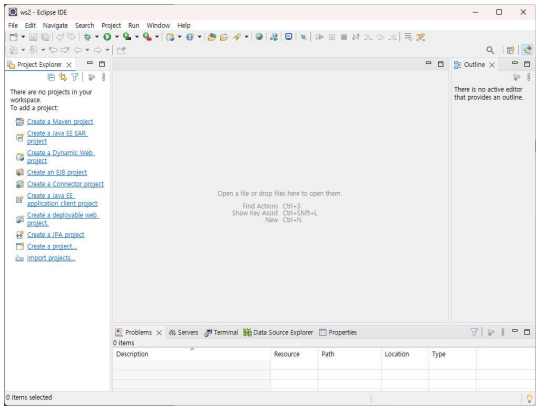


다운로드는 압축 파일의 형태로 이루어지므로 클릭만으로 설치가 가능합니다. 설치 과정에서는 반드시 'Enterprise Java and Web Developers' 항목을 선택해야만 웹 프로젝트를 생성할 수 있다는 점을 주의해야 합니다.



최신 버전의 Eclipse 의 경우 JDK 21 버전을 기준으로 설정되어 있기 때문에 아래의 화면과 같이 설치시에 자동으로 Java 21 이상의 실행환경(JRE)을 다운로드 하게 됩니다(조금 뒤에 JDK 17 버전의 세팅으로 변경할 것입니다.).

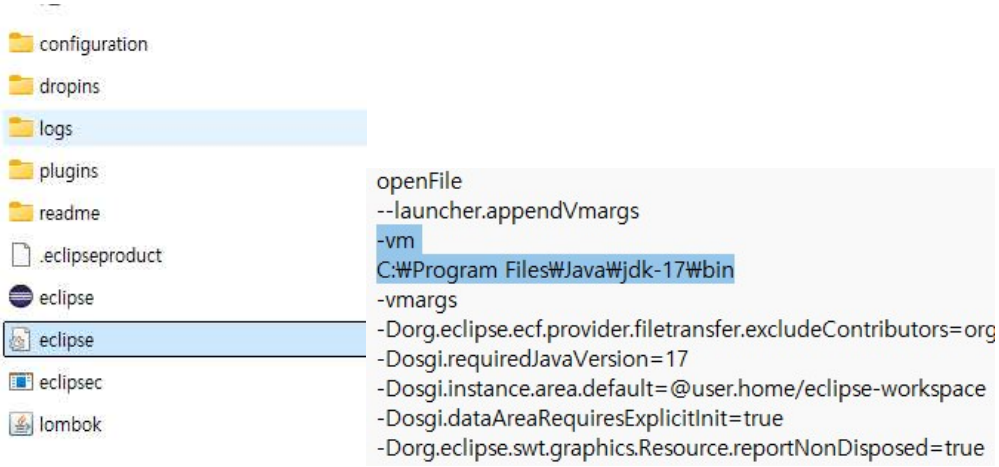




Eclipse 의 JDK 17 설정

설치된 Eclipse 의 실행 자체는 위의 그림과 같이 'Java 21 + VM' 환경이므로 별도의 설정이 없다면 프로젝트 생성과 실행에도 동일 환경을 사용하게 됩니다. 실습 환경에서는 JDK 17 환경을 이용할 것이므로 Eclipse 의 JDK 설정을 조정해 주는 것이 신규 프로젝트의 생성시에 별도의 추가적인 설정을 줄일 수 있습니다.

Eclipse 가 설치된 폴더를 확인해 보면 eclipse.ini 파일이 존재하는데 이를 메모장 등으로 열어서 '-vm' 경로를 설치된 JDK 17 의 bin 폴더로 수정합니다.



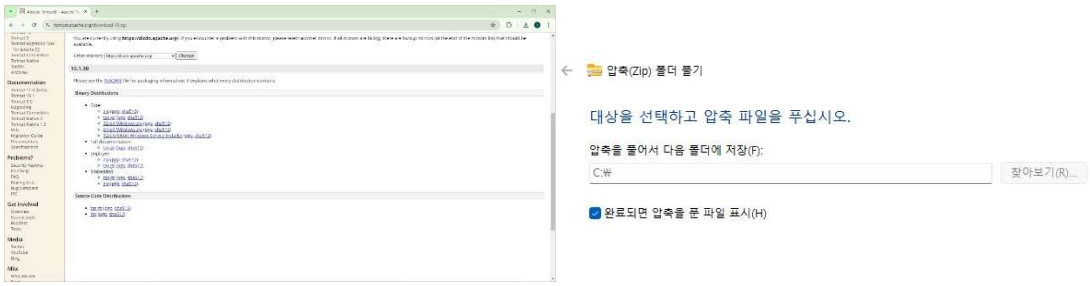
조금 아래쪽에는 -Dosgi.requiredJavaVersion 항목을 아래와 17 버전으로 설정합니다.

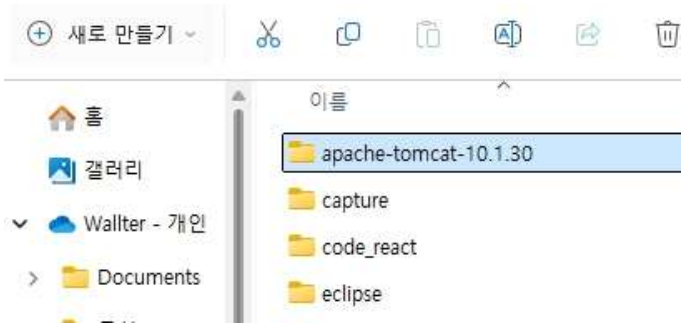
```
-vmargs
-Dorg.eclipse.ecf.provider.filetransfer.excludeContributors=org.ecl
-Dosgi.requiredJavaVersion=17
-Dosgi.instance.area.default=@user.home/eclipse-workspace
-Dosgi.dataAreaRequiresExplicitInit=true
-Dorg.eclipse.swt.graphics.Resource.reportNonDisposed=true
-Declipse.e4.inject.javax.warning=false
-Dsun.java.command=Eclipse
```

설정을 변경한 후에 Eclipse 가 정상적으로 실행되는지 확인합니다.

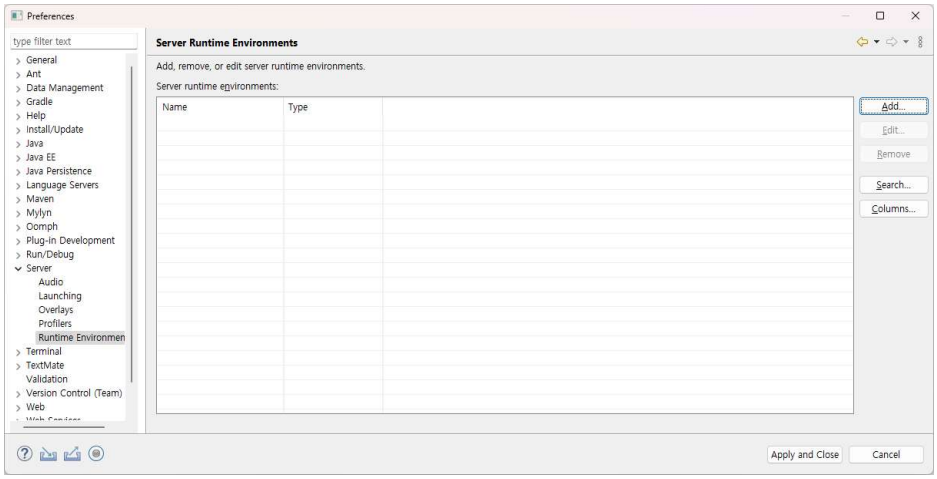
Tomcat 10 설치

스프링 6 레거시의 경우 반드시 Tomcat 은 10.0 이상 버전을 이용해야만 합니다. 실습 예제에서는 10.1.x 를 사용하도록 합니다. Tomcat 의 다운로드는 https://tomcat.apache.org/ 에서 받을 수 있고, 다운로드시에는 반드시 zip 파일을 형태로 받아서 찾기 쉬운 경로에 압축을 풀어줍니다.

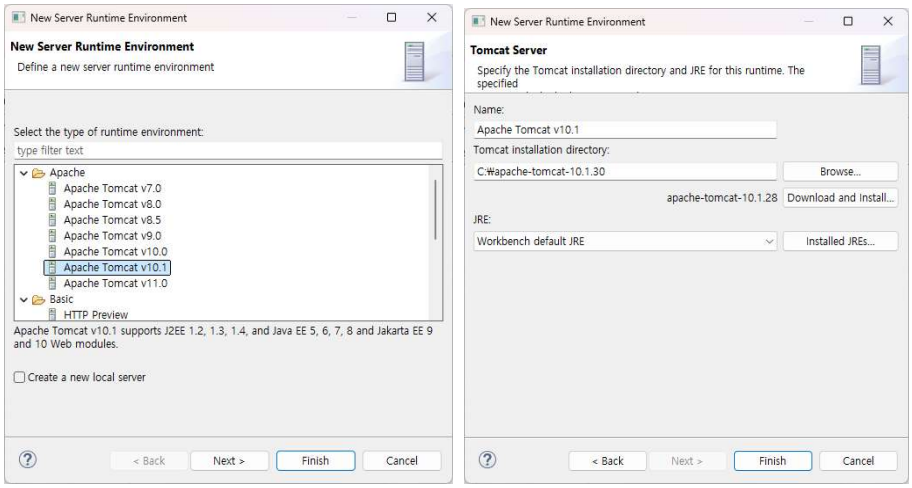




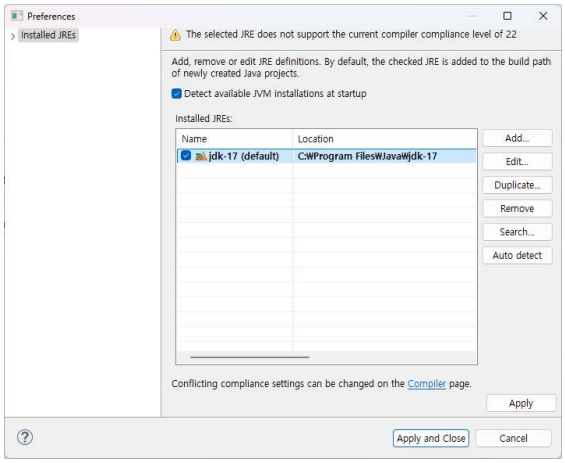
Eclipse 는 Tomcat 을 그대로 활용하는 것이 아니라 필요한 설정을 복사해서 사용하므로 별도로 설정이 필요합니다. Eclipse 의 메뉴에서 'Window -> Preference -> Server' 항목내에 있는 'Runtime Environments'항목을 선택합니다.



'Add' 항목을 이용해서 Tomcat 의 버전과 설치 경로를 지정합니다.

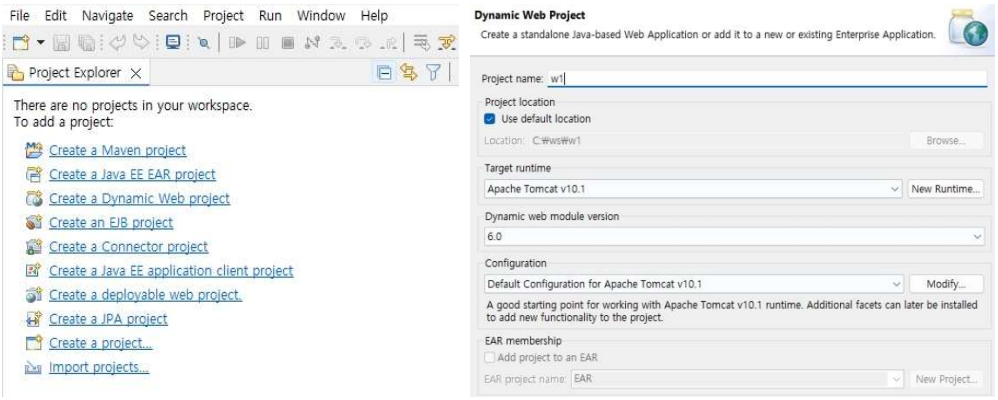


Tomcat 실행에 사용하는 JRE 는 JDK 17 디렉토리를 설정해 줍니다.



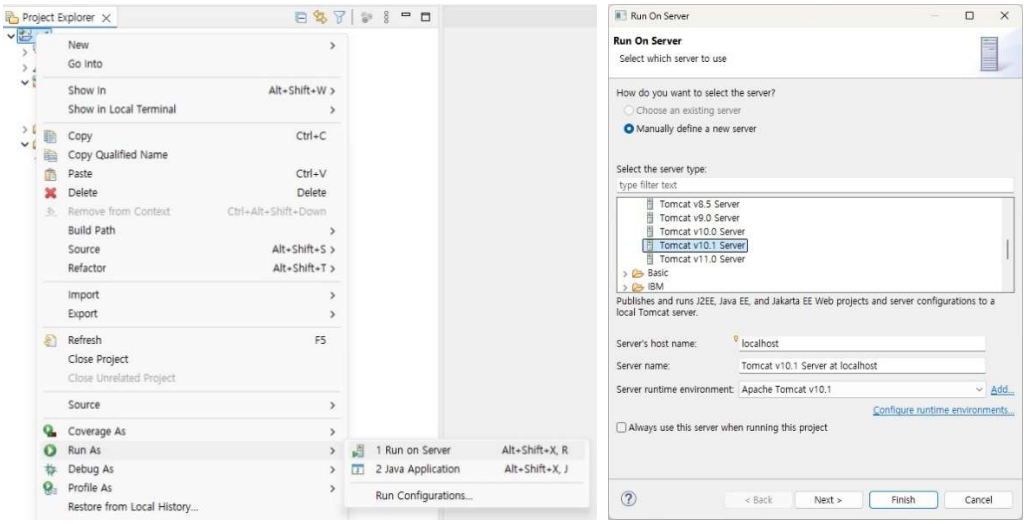
Dynamic Web Project 생성하기

Eclipse 와 Tomcat 에 대한 설치가 완료되었다면 'Dynamic Web Project' 메뉴를 통해서 프로젝트를 작성합니다. 생성하는 프로젝트의 이름은 'w1'으로 지정합니다. 우선은 Tomcat 의 실행을 확인하는 용도이기 때문에 별도의 다른 설정은 추가하지 않아도 됩니다.

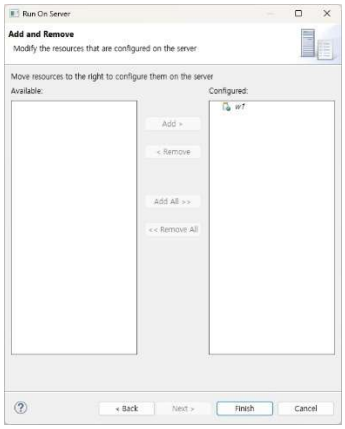


작성된 프로젝트는 아래와 같이 보이게 됩니다.

w1 프로젝트를 선택한 상태에서 마우스 오른쪽 버튼을 클릭하면 'Run As -> Run on Server'항목을 확인할 수 있고 이를 통해서 실행합니다.



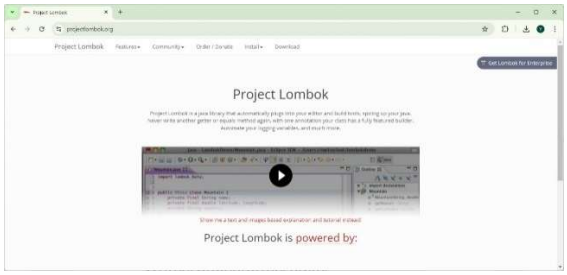
서버의 실행 후에는 자동으로 브라우저가 실행되면서 'localhost:8080/w1'이라는 경로로 이동합니다. 아직은 아무것도 작성된 코드가 없으므로 404 에러 메시지가 나오게 됩니다.

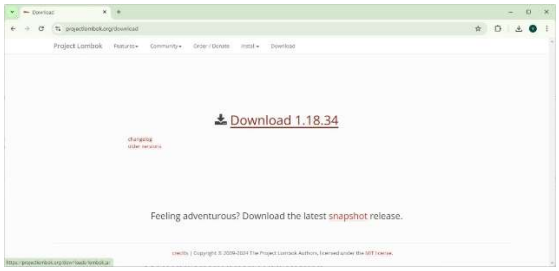


Lombok 라이브러리 설정

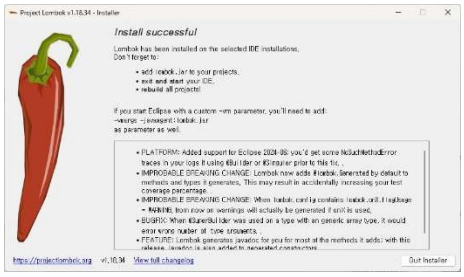
Lombok 은 Java 개발시에 자주 사용하는 getter/setter, 생성자, toString( )과 같은 번거로운 코드들을 컴파일시에 생성해 주는 편리한 라이브러리입니다. Lombok 을 이용하면 간결한 코드와 약간의 어노테이션들로 필요한 코드를 생성할 수 있으므로 Eclipse 에 관련 설정을 추가해주도록 합니다.

Lombok 은 project.lombok.org 사이트를 통해서 다운로드하고 lombok.jar 파일을 실행해서 Eclipse 가 설치된 경로에 추가하면 됩니다.

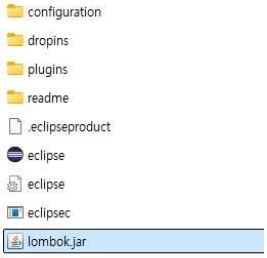




만일 Eclipse 의 경로를 찾지 못한다면 직접 Eclipse 가 설치된 경로를 지정해서 추가합니다.



lombok.jar 파일이 설치되면 Eclipse 의 실행 경로에 아래 그림과 같이 lombok.jar 파일이 추가되고 eclipse.ini 파일의 마지막부분에 lombok.jar 파일이 추가된 것을 확인할 수 있습니다.



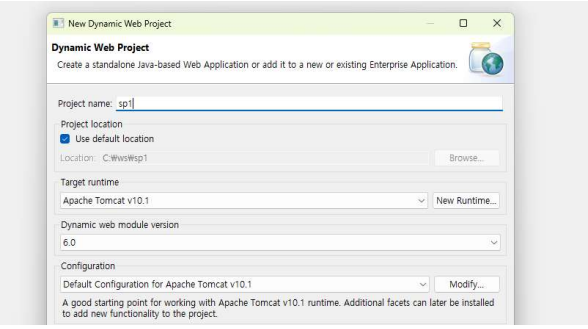

```
--add-modules=ALL-SYSTEM
-Djava.security.manager=allow
-Declipse.p2.max.threads=10
-Doomph.update.url=https://download.eclipse.org/oomph/updates/milestone/latest
-Doomph.redirection.index.redirection=index:/->https://raw.githubusercontent.com/
--add-opens=java.base/java.lang=ALL-UNNAMED
-javaagent:C:₩eclipse₩eclipse₩lombok.jar
```

0.3. Spring 6 Legacy Project

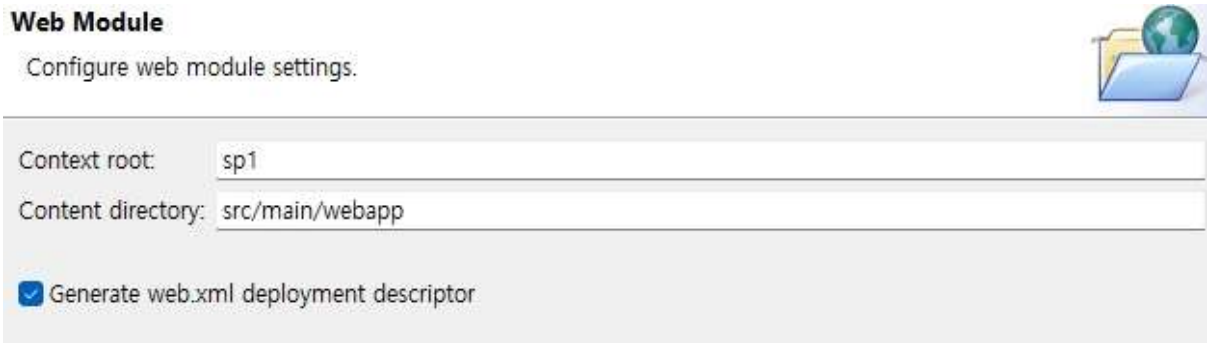
개발 환경에 대한 최종적인 점검은 스프링 6 버전의 레거시 프로젝트를 생성해서 실행해 보는 것입니다. 흔히 레거시 프로젝트라고 부르는 경우 주로 XML 설정을 이용해서 스프링을 사용하기 때문에 실습 예제 역시 XML 기반 설정을 이용하도록 합니다.

스프링 6 레거시 프로젝트의 생성

프로젝트의 생성은 Eclipse 의 Dynamic Web project 메뉴를 이용해서 새로운 프로젝트를 생성합니다. 생성하는 프로젝트는 'sp1'이라는 이름으로 생성합니다.

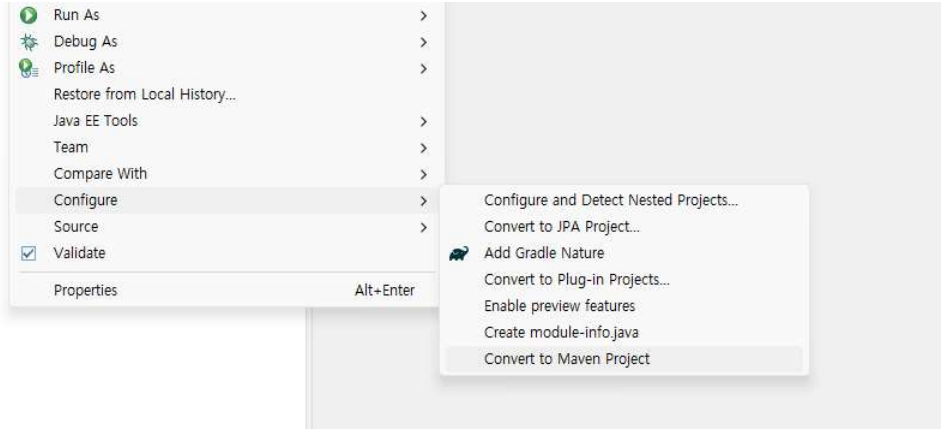


프로젝트의 생성과정에서는 XML 기반의 설정에 필요한 web.xml 을 생성하도록 지정합니다(나중에 추가할 수도 있지만 처음 프로젝트의 생성단계에서 추가하는 것이 편리합니다.).

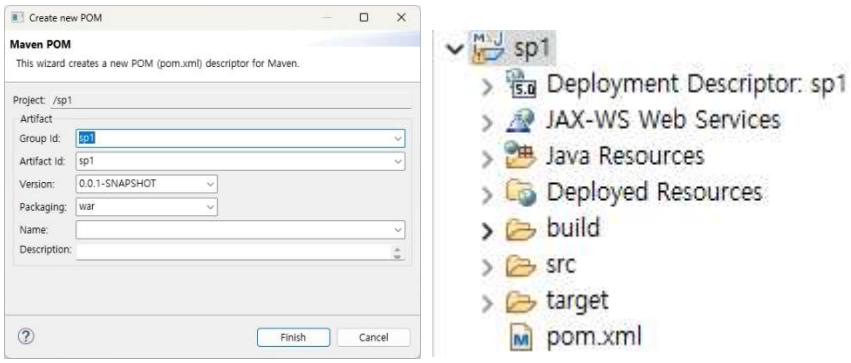


생성한 sp1 프로젝트가 스프링 프로젝트가 되기 위해서는 여러 종류의 라이브러리가 필요합니다. 이러한 라이브러리 관리는 Maven 이나 Gradle 과 같은 빌드 도구를 이용하는 것이 편리합니다. 다행히 Eclipse 의 경우 Maven 을 포함하고 있기 때문에 별도의 설정 없이 Maven 빌드 도구를 이용할 수 있습니다.

생성된 프로젝트를 선택하고 추가적으로 'Configure -> Convert to Maven Project'를 선택해서 sp1 프로젝트를 Maven 을 이용하도록 지정합니다(Eclipse 의 경우 Maven 에 대해서는 추가적인 설정이 필요하지 않습니다.).



Maven 빌드 도구를 이용하도록 지정되면 pom.xml 파일이 프로젝트에 추가됩니다.



Maven 을 이용한 라이브러리 추가

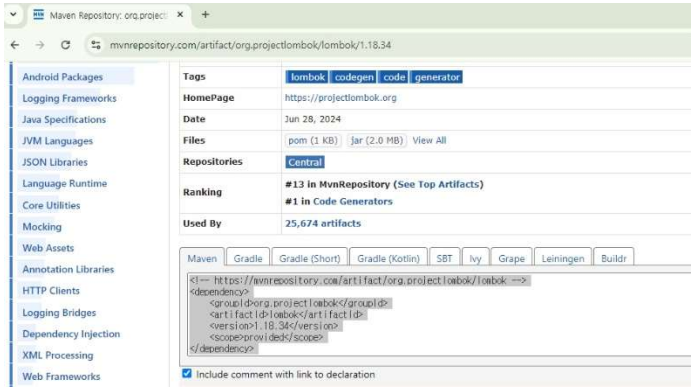
Maven 프로젝트의 전환의 결과로 생성된 pom.xml 파일의 내부에는 프로젝트에 필요한 라이브러리를 설정하는 <dependencies> 태그를 <build>전에 추가해 두는데 <dependencies> 태그 내부에는 앞으로 필요한 라이브러리들을 지정하게 됩니다(저장 메뉴 혹은 Ctrl +S 를 이용해서 변경된 내용을 저장합니다.).

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns
  <modelVersion>4.0.0</modelVersion>
  <groupId>sp1</groupId>
  <artifactId>sp1</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <dependencies>

  </dependencies>

  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
```

Lombok 과 스프링 6 라이브러리 추가

가장 먼저 추가할 라이브러리는 프로젝트내에서 사용되는 Lombok 라이브러리와 스프링 6 관련 라이브러리(spring-core, spring-context, spring-webmvc)입니다.

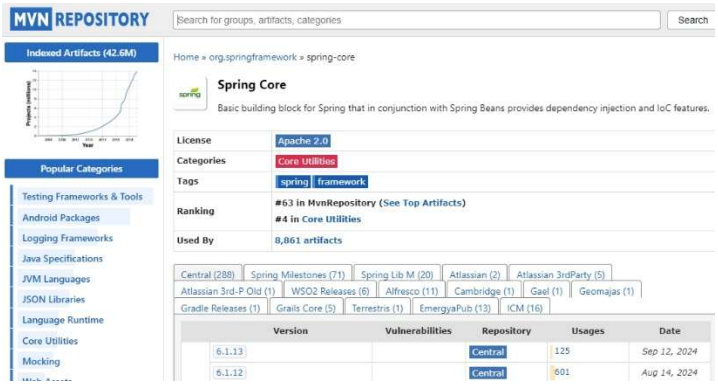
검색 엔진을 이용해서 'Lombok maven'이라고 검색하면 대부분 'Maven Repository'라는 저장소를 알려주는데 여기서 Maven 항목을 지정해서 복사한후에 pom.xml 에 추가합니다.




```
<dependencies>

 <dependency>
     <groupId>org.projectlombok</groupId>
     <artifactId>lombok</artifactId>
     <version>1.18.34</version>
     <scope>provided</scope>
 </dependency>


  </dependencies>
```

pom.xml 을 저장하면 자동적으로 Lombok 라이브러리를 다운로드 하게 되고, 이를 프로젝트 구조에서 확인할 수 있습니다.

스프링과 관련해서는 'spring-core, spring-context, spring-webmvc'를 Maven 저장소를 검색해서 추가합니다(라이브러리들을 추가할 때는 동일한 버전을 사용하도록 주의합니다.).



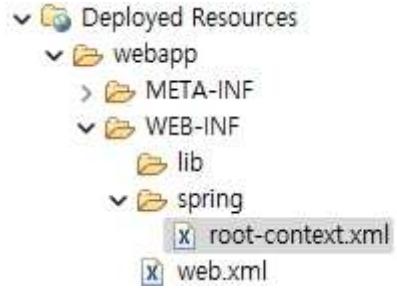
저장후에는 자동으로 아래의 오른쪽과 같이 jar 파일들이 다운로드 된 것을 확인할 수 있습니다.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>6.1.13</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.1.13</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>6.1.13</version>
</dependency>
```

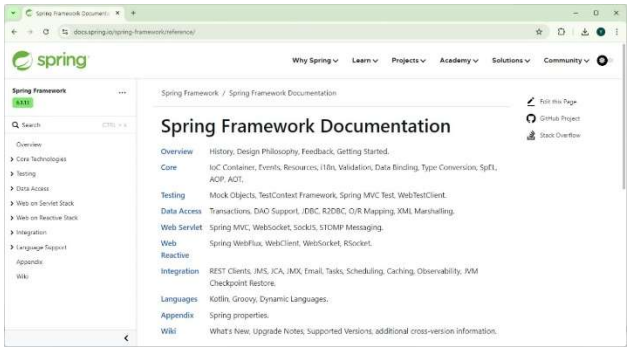
스프링 XML 설정 파일

스프링 레거시 프로젝트의 경우 XML 을 이용해서 필요한 설정을 추가하는 경우가 일반적입니다. 흔히 XML 설정이 어렵다고 하지만 단계별로 하나씩 진행해보면 그다지 어려운 일이 아닙니다.

프로젝트의 WEB-INF 밑에 spring 이라는 폴더를 하나 생성해서 필요한 XML 파일을 저장하는 폴더로 이용합니다. spring 폴더에는 root-context.xml 파일을 추가합니다.



스프링의 XML 과 관련된 정보는 'https://docs.spring.io/spring-framework/reference/index.html'를 통해서 얻을 수 있습니다. 사용하는 라이브러리에 맞게 문서의 버전을 지정하면 해당 버전의 설정 문서를 찾을 수 있습니다. Core Technologies 의 Appendix 항목을 선택하면 XML 문서의 기본 틀을 구할 수 있습니다.



작성한 root-context.xml 의 내용은 'XML Schemas'라는 항목을 찾아보면 어떤 설정이 필요한지 파악할 수 있습니다.

The context Schema

The `context` tags deal with `ApplicationContext` configuration that relates to plumbing — that is, not usually beans that are important to an end-user but rather beans that do a lot of the "grunt" work in Spring, such as `BeanfactoryPostProcessors`. The following snippet references the correct schema so that the elements in the `context` namespace are available to you:

```xml
<?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 https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd">

    <!-- bean definitions here -->

</beans>
```

root-context.xml 의 내용

```xml
<?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
  https://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/context
  https://www.springframework.org/schema/context/spring-context.xsd">


</beans>
```

web.xml 의 설정

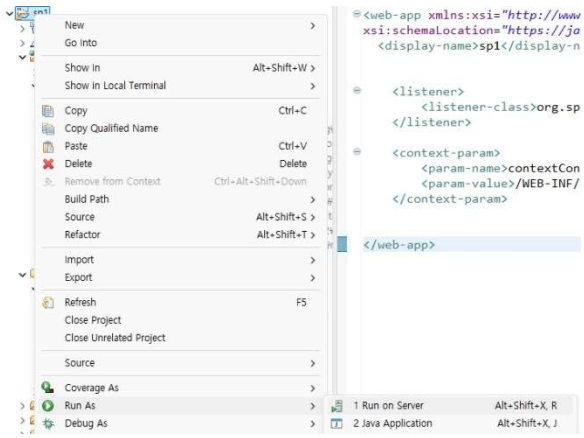
웹 프로젝트에서 스프링의 설정을 확인하기 위해서는 Tomcat 에서 실행될 때 root-context.xml 을 읽어서 실행하도록 하는 리스너(listener)설정이 필요합니다.

WEB-INF/web.xml

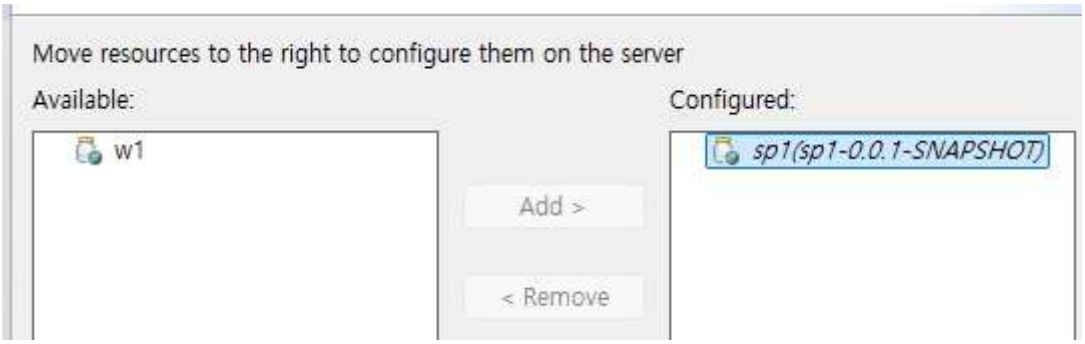
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="https://jakarta.ee/xml/ns/jakartaee" xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd" id="WebApp_ID" version="6.0">
  <display-name>sp1</display-name>

<listener>
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>
</web-app>
```

Tomcat 실행을 통한 설정 확인

스프링과 관련된 설정이 정상적으로 처리되었는지 확인하는 방법은 Tomcat 을 통해서 현재까지 작성된 프로젝트를 실행해 보는 것입니다. sp1 프로젝트를 선택하고 'Run As -> Run on Server'를 통해서 Tomcat 에서 프로젝트를 실행해 봅니다.



Eclipse 에서 프로젝트를 실행할 때는 가능하면 하나의 프로젝트만을 실행하는 것이 안전합니다. 이전에 실행된 'w1'대신에 'sp1'을 지정해서 실행합니다.



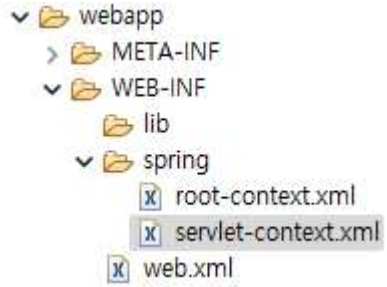
프로젝트가 실행되었을 때 아래의 그림과 같이 'springframework'라는 단어들이 나왔다면 설정에 문제가 없는 것입니다.

```
정보: Initializing Spring root WebApplicationContext
10월 03, 2024 4:43:48 오후 org.springframework.web.context.ContextLoader initWebApplicationContext
정보: Root WebApplicationContext: initialization started
10월 03, 2024 4:43:48 오후 org.springframework.web.context.ContextLoader initWebApplicationContext
정보: Root WebApplicationContext initialized in 241 ms
10월 03, 2024 4:43:48 오후 org.apache.coyote.AbstractProtocol start
정보: 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작합니다.
```

브라우저에서는 아직 웹과 관련된 설정이 완료된 것이 아니기 때문에 아래와 같이 404 화면만 보이게 됩니다.

0.4. Spring MVC 설정

스프링 프레임워크의 경우 web 관련된 설정은 별도의 설정 파일로 분리해서 운영합니다. WEB-INF₩spring 폴더에 servlet-context.xml 파일을 생성합니다.



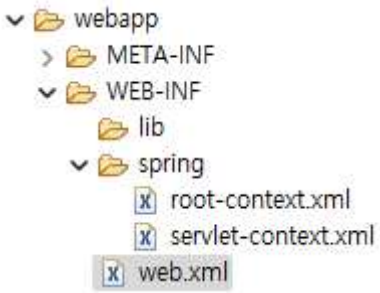
servlet-context.xml 파일의 내용은 스프링 문서에서 'Web on Servlet Stack'에서 찾을 수 있습니다.

servlet-context.xml 파일의 내용

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:mvc="http://www.springframework.org/schema/mvc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  https://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/mvc
  https://www.springframework.org/schema/mvc/spring-mvc.xsd">

 <mvc:annotation-driven/>

</beans>
```

web.xml 의 수정

마지막으로 Tomcat 에서 스프링 프레임워크가 같이 실행되도록 web.xml 에서 서블릿 설정을 아래와 같이 추가합니다.




```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="https://jakarta.ee/xml/ns/jakartaee" xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd" id="WebApp_ID" version="6.0">
  <display-name>sp1</display-name>

 <listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
 </listener>

 <context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/root-context.xml</param-value>
 </context-param>


    <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/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>
```

servlet-context.xml 과 web.xml 을 수정한 후에는 다시 Tomcat 을 실행해서 실행에 문제가 없는지를 확인하도록 합니다. 실행 과정에서 기존과 조금 다른 메시지의 내용이 출력되는 것을 확인할 수 있습니다.

```
10월 07, 2024 9:09:04 오후 org.springframework.web.context.ContextLoader initWebApplicationContext
INFO: Root WebApplicationContext: initialization started
10월 07, 2024 9:09:04 오후 org.springframework.web.context.ContextLoader initWebApplicationContext
INFO: Root WebApplicationContext initialized in 229 ms
10월 07, 2024 9:09:04 오후 org.apache.catalina.core.ApplicationContext log
INFO: Initializing Spring DispatcherServlet 'appServlet'
10월 07, 2024 9:09:04 오후 org.springframework.web.servlet.FrameworkServlet initServletBean
INFO: Initializing Servlet 'appServlet'
10월 07, 2024 9:09:04 오후 org.springframework.web.servlet.FrameworkServlet initServletBean
INFO: Completed initialization in 320 ms
10월 07, 2024 9:09:04 오후 org.apache.coyote.AbstractProtocol start
INFO: 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작합니다.
10월 07, 2024 9:09:04 오후 org.apache.catalina.startup.Catalina start
INFO: 서버가 [1657] 밀리초 내에 시작되었습니다.
```

0.5. Log4j2의 설정

프로그램이 정상적으로 실행되고 있는지 확인하기 위해서는 System.out.println( )대신에 Log4j2 를 이용해서 로그를 기록하는 방식이 편리합니다. 이를 위해서 프로젝트의 시작단계에서 라이브러리와 설정을 미리 추가해 두는 것이 좋습니다.

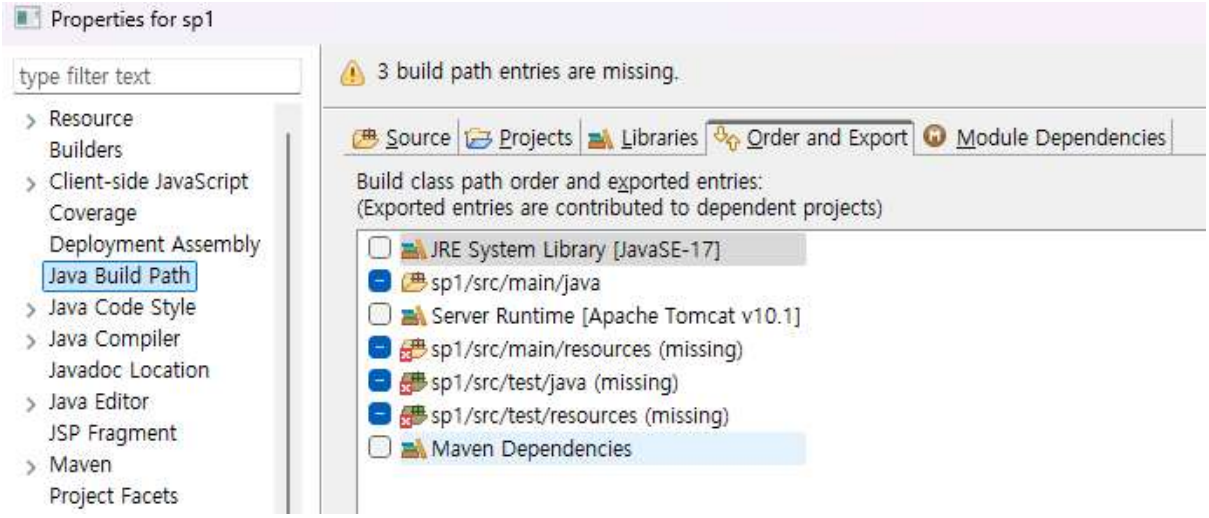
Log4j2 라이브러리는 Maven 저장소를 이용해서 pom.xml 의 <dependencies>내에 아래의 항목들을 추가합니다.

pom.xml 에 추가하는 라이브러리들

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.24.1</version>
</dependency>

<!-- Log4j2 Core -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.24.1</version>
</dependency>

<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.24.1</version>
</dependency>
```

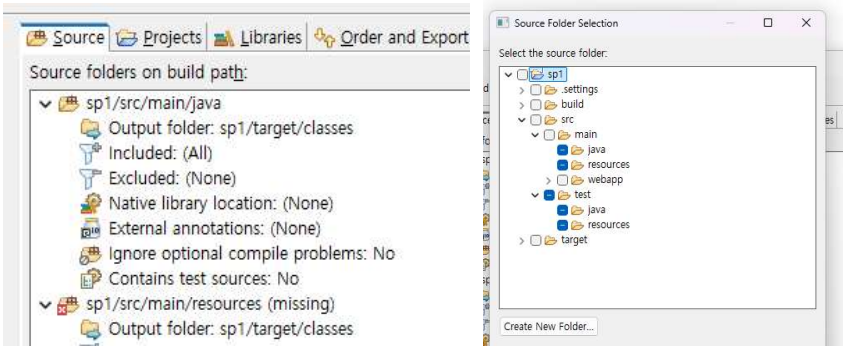
프로젝트에 필요한 폴더 추가하기

sp1 프로젝트의 'properties(마우스 오른쪽 버튼 클릭 시 마지막 항목)'에는 'Java Build Path'라는 항목이 존재합니다.



프로젝트의 생성시에는 위의 그림에서 'x'표시가 된 폴더들이 없기 때문에 missing 이라고 보이게 됩니다.

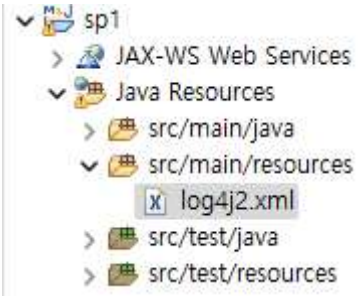
화면 상단의 탭 중에서 'Source'탭을 선택하고 'Add Folder'를 선택합니다. 이때 실제로 폴더를 추가할 필요는 없습니다.



위와 같은 화면을 확인한 후에는 별도의 처리 없이 메뉴를 종료합니다. 이후 프로젝트를 살펴보면 해당 폴더 들이 생성되어 있는 것을 확인할 수 있습니다.

log4j2.xml 파일의 추가

생성된 src/main/resource 폴더에는 log4j2.xml 파일을 추가합니다.



log4j2.xml 파일

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

    <!-- Appender, Layout 설정 -->
    <Appenders>
        <Console name="console" target="SYSTEM_OUT">
            <PatternLayout  pattern="%c %l %5p %m%n"/>
        </Console>
    </Appenders>

    <!-- Logger 설정 -->
    <Loggers>
```

```
        <Logger name="org.springframework" level="DEBUG" additivity="false">
            <AppenderRef ref="console"/>
        </Logger>

        <Root level="INFO">
            <AppenderRef ref="console"/>
        </Root>
    </Loggers>

</Configuration>
```

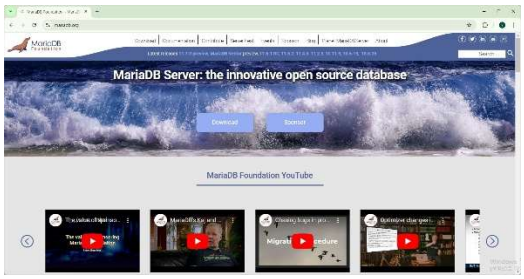
log4j2.xml 파일의 설정하고 Tomcat 을 다시 실행하면 기존과 달리 많은 양의 로그가 기록되는 것을 확인할 수 있습니다.

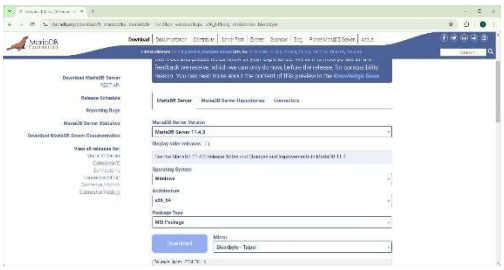
```
org.springframework.web.context.support.XmlWebApplicationContext org.springframework.context.su
org.springframework.beans.factory.xml.XmlBeanDefinitionReader org.springframework.beans.factory
org.springframework.ui.context.support.UiApplicationContextUtils org.springframework.ui.context
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter org.springfr
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter org.springfr
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.web.servlet.mvc.method.annotation.ExceptionHandlerExceptionResolver org.spr
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.beans.factory.support.DefaultListableBeanFactory org.springframework.beans.
org.springframework.web.servlet.DispatcherServlet org.springframework.web.servlet.DispatcherSer
org.springframework.web.servlet.DispatcherServlet org.springframework.web.servlet.DispatcherSer
org.springframework.web.servlet.DispatcherServlet org.springframework.web.servlet.DispatcherSer
org.springframework.web.servlet.DispatcherServlet org.springframework.web.servlet.DispatcherSer
org.springframework.web.servlet.DispatcherServlet org.springframework.web.servlet.FrameworkServ
org.springframework.web.servlet.DispatcherServlet org.springframework.web.servlet.FrameworkServ
10월 07, 2024 9:11:23 오후 org.apache.coyote.AbstractProtocol start
INFO: 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작합니다.
10월 07, 2024 9:11:23 오후 org.apache.catalina.startup.Catalina start
INFO: 서버가 [2054] 밀리초 내에 시작되었습니다.
```

0.6. MariaDB 설치와 설정

예제에서 사용하는 데이터베이스는 관계형 데이터베이스 중에서 무료로 사용이 가능한 MariaDB 를 이용하도록 합니다(MySQL 을 이용하는 것도 가능합니다만 MySQL 은 상업적인 용도로 사용하는 경우 라이센스가 필요합니다.).

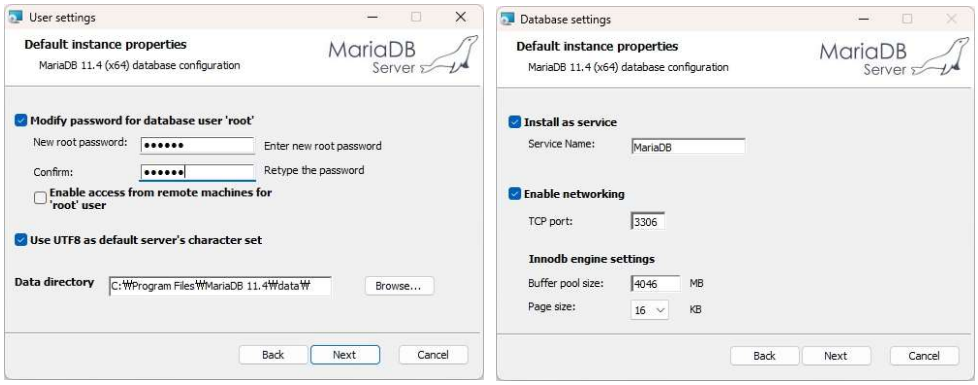
MariaDB 는 mariadb.org(com 도 존재하므로 주의)를 통해서 다운로드 받을 수 있습니다.



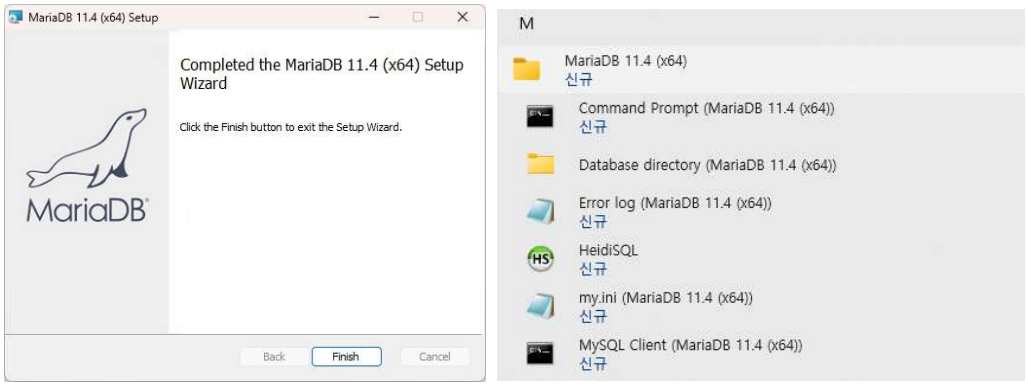


설치 과정은 별도의 추가적인 설정 없이 진행할 수 있으나 중간에 root 계정의 패스워드를 지정하는 부분은 주의해야 합니다.

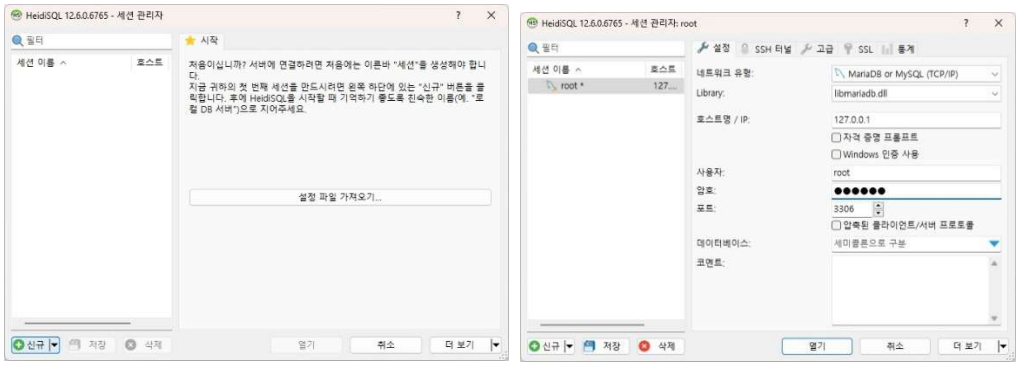
root 계정의 패스워드는 나중에 복구가 힘들기 때문에 처음 설정값을 정확히 기억해 두어야 합니다.



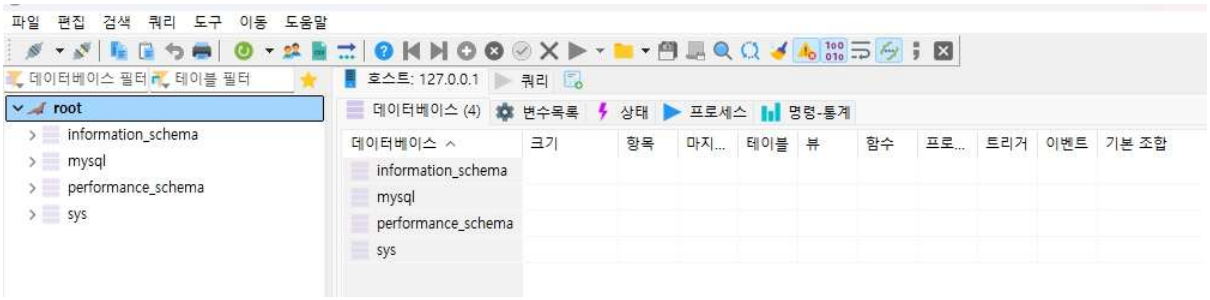
MariaDB 가 설치되면 추가적으로 HeidiSQL 이라는 추가적인 프로그램이 설치됩니다. HeidiSQL 은 데이터베이스에 접속하고 실행할 수 있는 편집툴입니다.



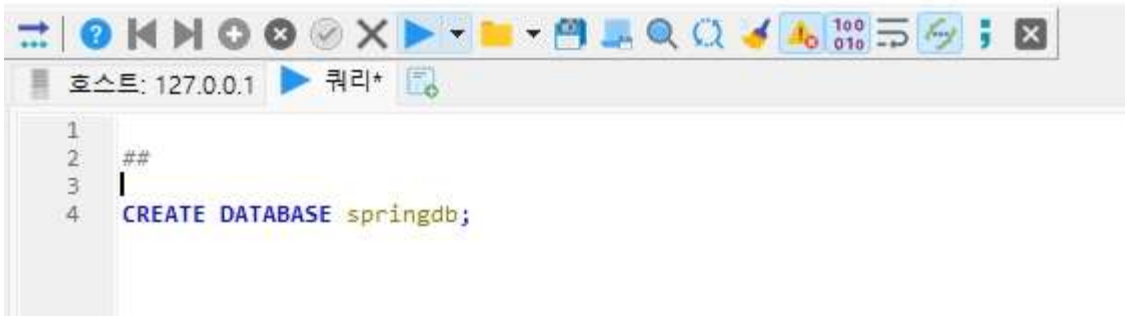
프로젝트에 사용할 예제를 위해서 root 를 사용하는 것은 위험하기 때문에 root 계정을 통해 새로운 스키마와 계정을 생성해 주도록 합니다.



root 계정으로 접속되면 기본적으로 설치된 스키마들을 확인할 수 있습니다.



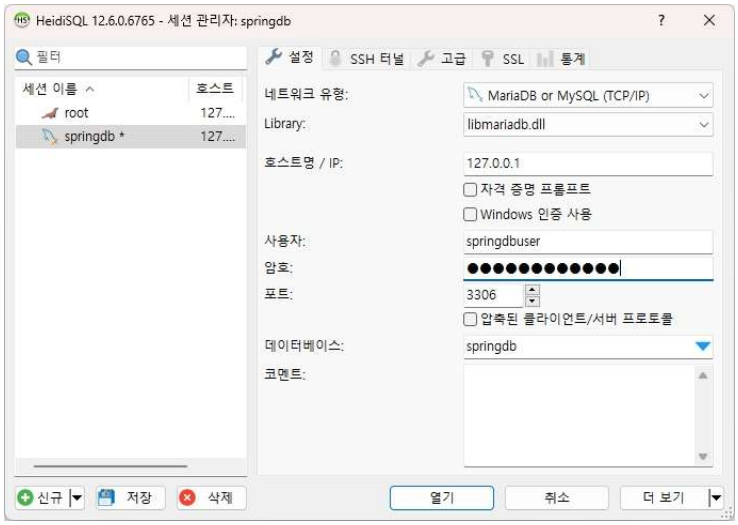
예제 프로젝트에서 사용하려는 스키마를 생성합니다.



HeidiSQL 의 메뉴를 이용해서 사용자를 추가하게 되면 암호화되기 때문에 명시적으로 아래와 같이 SQL 을 직접 작성해서 실행하도록 합니다(아래의 SQL 문을 작성하고 한 라인씩 실행(Ctrl+F9)해 주도록 합니다.).

```
CREATE DATABASE springdb;
CREATE USER 'springdbuser'@'localhost' IDENTIFIED BY 'springdbuser';
CREATE USER 'springdbuser'@'%' IDENTIFIED BY 'springdbuser';
GRANT ALL PRIVILEGES ON springdb.* TO 'springdbuser'@'localhost';
GRANT ALL PRIVILEGES ON springdb.* TO 'springdbuser'@'%';
```

예제를 위한 스키마와 계정이 생성되었다면 다시 HeidiSQL 을 실행해서 생성한 계정으로 연결이 가능한지 확인해 줍니다.



생성한 사용자의 경우 springdb 에만 권한이 있으므로 아래 화면과 같이 springdb 항목만 출력되는 것을 확인합니다.