스프링(Spring) 개발

(4) 스프링 프로젝트 생성하기

간단히 스프링 프레임워크를 살펴보겠습니다.

1. 스프링은 POJO (Plain Old Java Object) 방식의 프레임워크로서, 일반적인 J2EE 프레임워크에 비해 특정 라이브러리를 사용할 필요가 없어서 개발이 쉬우며, 기존 라이브러리의 지원이 용이합니다.

2. 스프링은 관점지향프로그래밍, AOP(Aspect Oriented Programming)를 지원합니다. 트랜잭션, 로깅, 보안 등 여러 모듈, 여러 계층에서 적용되는데, 이런 코드들을 실제 비지니스 로직과 분리할 수 있도록 도와줍니다. 한때, AOP가 OOP(Object Oriented Programming)를 대체하는 기술로 생각되기도 했지만, 실제로 AOP는 OOP를 더욱 OOP스럽게 보완해 주는 기술입니다.

3. 스프링은 의존성 주입, DI (Dependency Injection)를 지원합니다. 이는 객체간의 의존관계를 관리하는 기술이라고 생각하면 됩니다. 어떤 객체가 필요로 하는 객체를 자기 자신이 직접 생성하는 것이 아니라, 외부에 있는 다른곳에서 자신이 필요로 하는 객체를 주입받는 것을 말합니다.

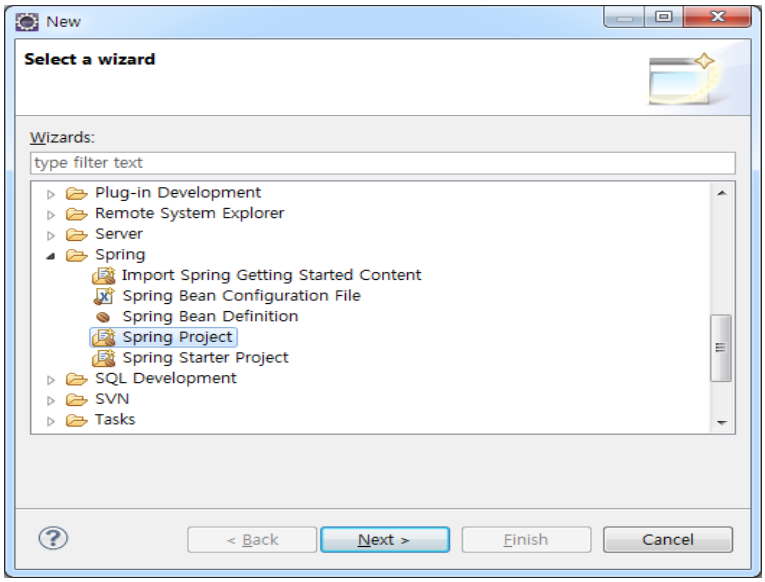
4. 스프링은 제어 반전, IoC (Inversion of Controller)를 지원합니다. 컨트롤의 제어권이 개발자가 아니라 프레임워크에 있음을 말합니다. 즉, 객체의 생성부터 모든 생명주기의 관리까지 객체의 제어권이 바뀐 것을 의미합니다.

스프링의 가장 큰 특징은 AOP, POJO, DI, PSA (Portable Service Abstraction) 를 꼽을 수 있습니다.

1. 스프링 프로젝트 생성

1. File > New > Other 를 선택한다.

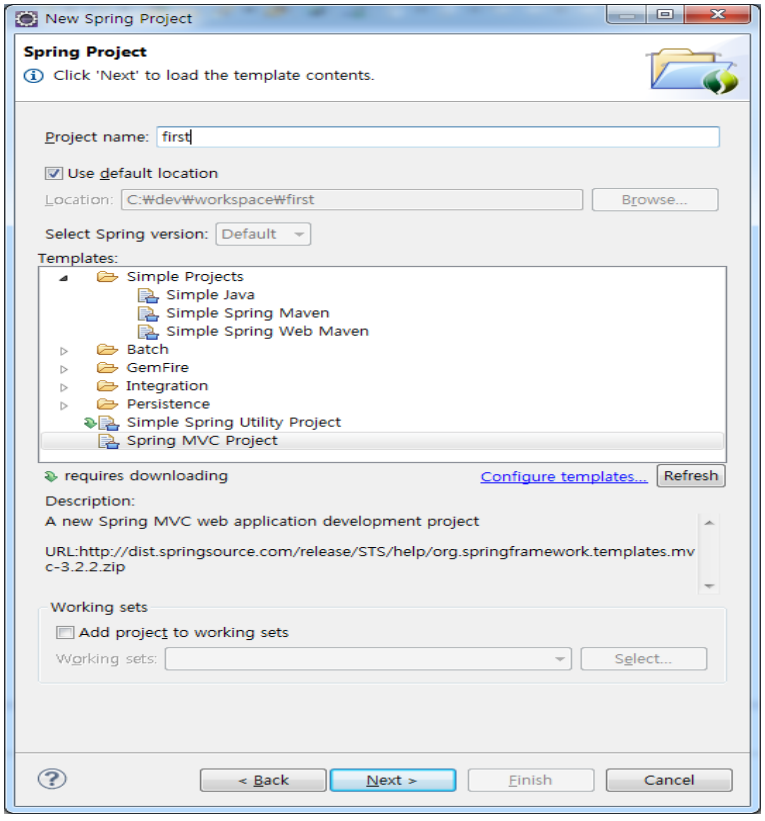
2. Spring > Spring Project를 선택한다.



3. 프로젝트의 이름을 입력하고, Spring MVC Project를 선택한다.

단, 프로젝트 이름은 반드시 소문자로만 작성할 것. (스프링은 프로젝트명의 소문자만 인식함)

여기서는 최초의 스프링 프로젝트이니, 프로젝트의 이름을 first 로 지정해 본다.

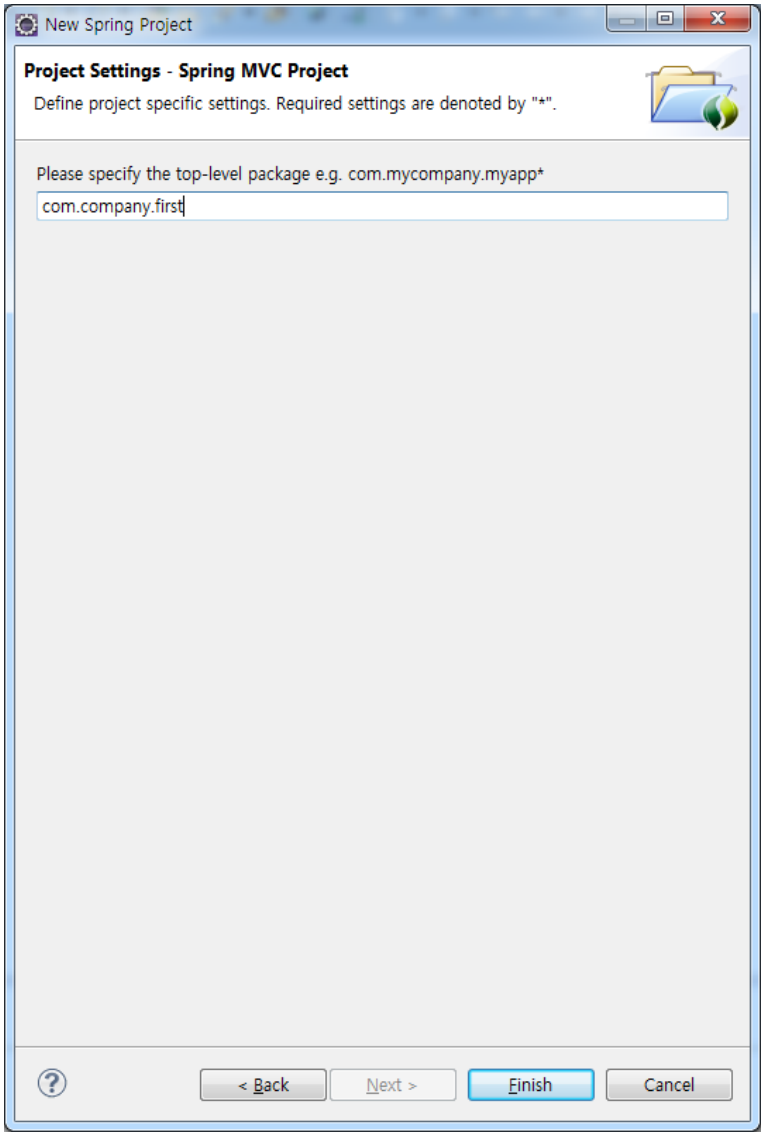


4. package를 입력한다.

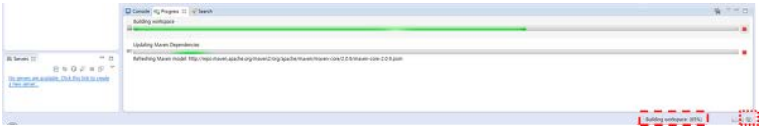
package는 최소 3레벨 이상 ([1레벨].[2레벨].[3레벨])로 구성하게끔 되어있다. 이는 자바 코딩 규칙 (Coding Convention을 찾아보면 확인할 수 있다.) 여기서는, "com.company.first"라는 package를 사용하기로 한다.

주의 : 3레벨의 이름은 프로젝트명과 동일하게 지정할 것.

(스프링에서 context root 명을 3레벨 이름으로 자동 설정하도록 되어 있음. 실행시 404 에러 발생의 이유가 되기도 함)



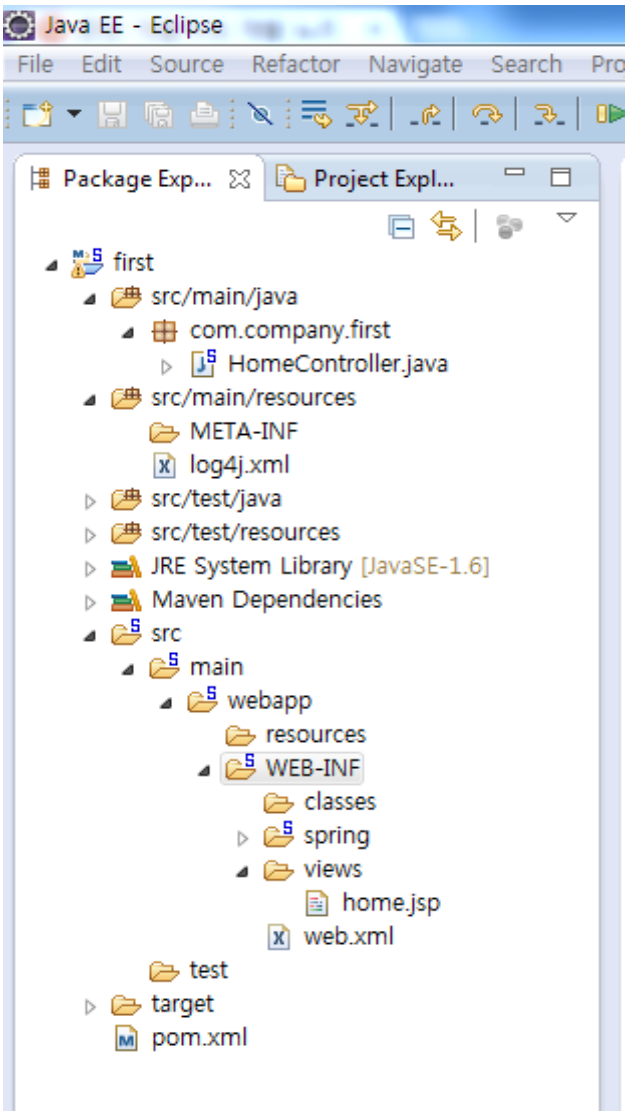
5. Finish를 누르면 프로젝트가 생성이 되고, 인터넷에서 스프링 프로젝트에 필요한 라이브러리를 자동으로 다운받을 것인지 묻는 창이 나타나면 ok 한다, 그러면 자동으로 다운받기 시작한다.



↑ 위 스크린샷의 오른쪽 하단 구석에 있는 버튼을 누르면, 화면과 같이 라이브러리의 다운이 Background에서 이루어지며, 진행 상황을 Progress View에서 볼 수 있다.

우리가 생성한 Spring MVC Project에는 여러가지 라이브러리들이 필요한데, 프로젝트의 생성과 동시에 메이븐이 자동적으로 인터넷에서 필요한 라이브러리를 다운받는 과정이다.

6. 다운로드가 완료된 것을 확인하고, 생성된 프로젝트에 에러가 없는 것을 확인한다.



이와같은 프로젝트가 생성된다.

Package Explorer에서 보이는 것을 기준으로 설명한다.

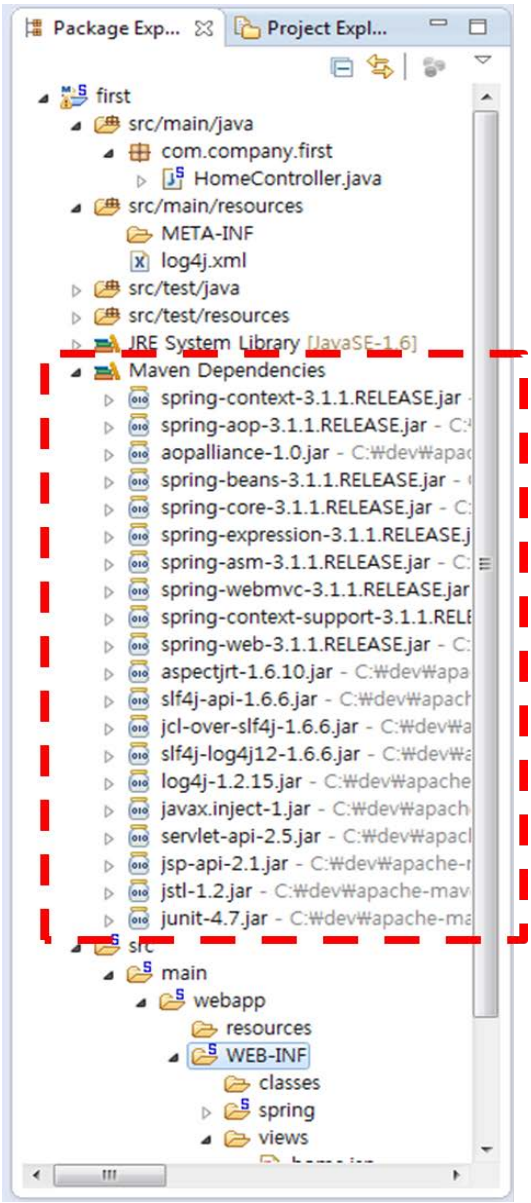
※ 만약 Package Explorer가 보이지 않는다면,

1) Window > Open View > Other를 선택한다.

2) Package Explorer를 검색하여 추가한다.

1.1 Maven Dependencies

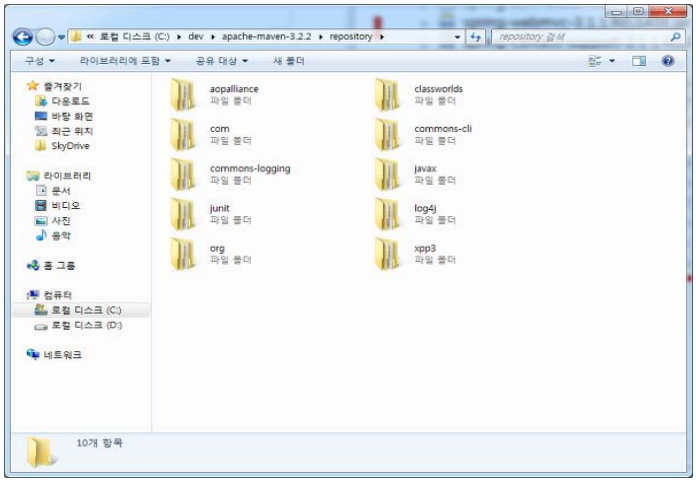
앞에서, Maven이 자동적으로 필요한 라이브러리를 받아온다고 했는데, 그 받아온 라이브러리들은 Maven Dependencies라는 곳에서 확인할 수 있다.



이 프로젝트를 진행하며 필요한 라이브러리들이 다운받아진 것을 알 수 있다.

그러면, 이 라이브러리들이 저장되어 있는 위치는, 개발환경을 설정하면서

C:₩spring₩dev₩apache-maven-3.2.2 밑에 repository라는 폴더를 만들고, LocalRepository를 변경했었다. 그 폴더로 들어가 보자.



그러면 다음과 같이 몇몇 폴더가 생긴것을 알 수 있다.

각각의 폴더에 우리가 필요한 라이브러리들이 위치한다.

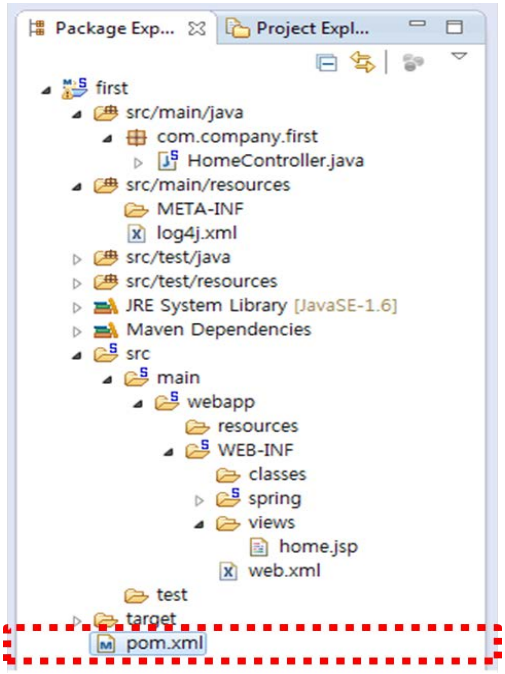
예를 들어, Maven Dependencies 맨 위에 있는 "spring-context-3.1.1.RELEASE.jar"를 찾아보자. 이 jar 파일은 org₩springframework₩spring-context₩3.1.1.RELEASE 폴더에 위치하고 있다.

이런 식으로 인터넷에서 찾아온 라이브러리들이 관리된다.

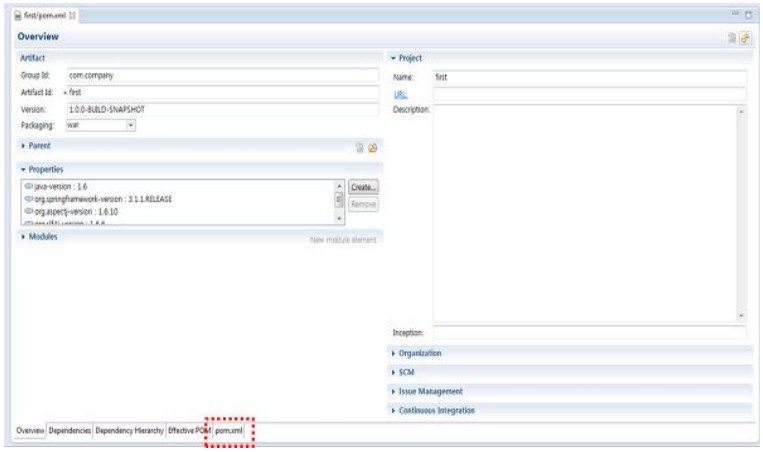
그러면, 프로젝트 내에서 라이브러리의 추가 및 삭제, 관리는 어디서 되는지를 살펴보자.

1.2 POM.xml

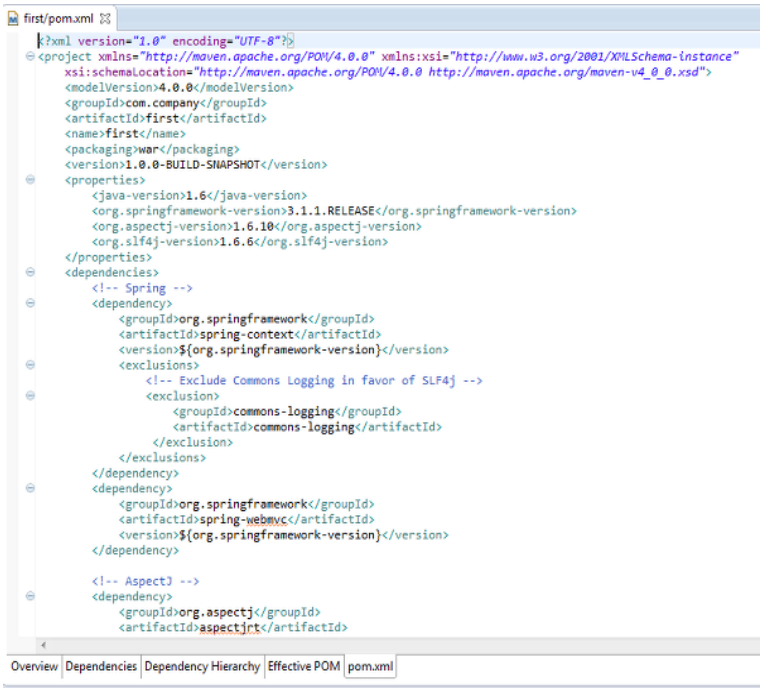
1. pom.xml 파일을 더블클릭하여 열어보자.



그러면 다음과 같은 화면이 보이고, 중간쯤에 있는 pom.xml 탭을 선택한다.



2. pom.xml을 확인하자.



↑ 이와 같은 xml 파일을 볼 수 있다. 우리가 필요한 라이브러리의 관리는 모두 pom.xml에서 관리한다.

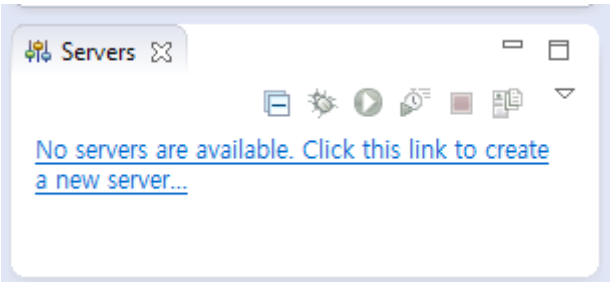
여기서 <dependency> </dependency> 라는 태그를 확인할 수 있는데, 이 태그가 하나의 라이브러리를 의미한다. 만약 라이브러리를 추가하고 싶으면, <dependency></dependency>라는 태그를 추가함으로써, 새로운 라이브러리를 추가할 수 있다.

2. first 프로젝트 실행

이제 우리가 만든 first 프로젝트가 제대로 실행되는지 확인할 차례다.

1. Servers view에서 새로운 서버를 만든다.

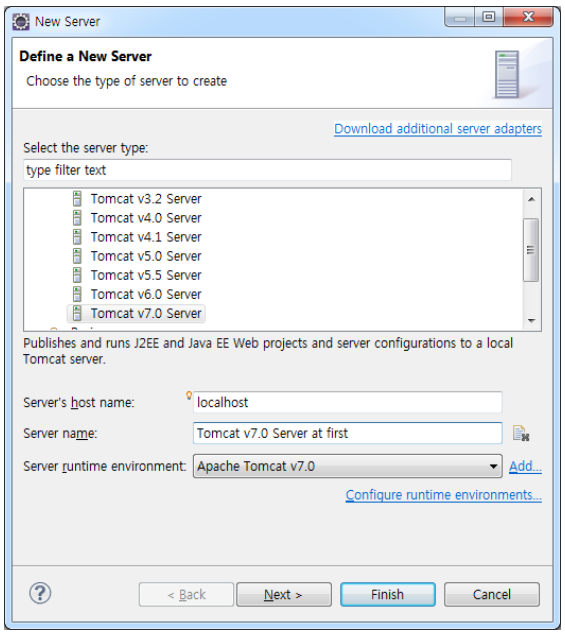
앞에서 톰캣이 제대로 설치되었는지 확인하기 위해서 만든 서버가 남아있다면 지우자.



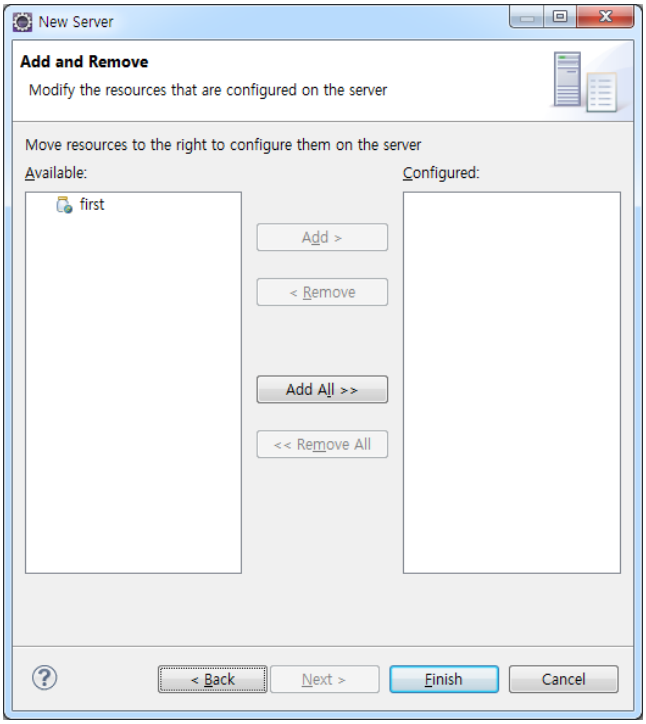
Servers 뷰에서 마우스 우클릭 > New > Server를 선택해도 된다.

2. Tomcat 8.0을 선택하고, 서버 이름은 first로 바꾼다.

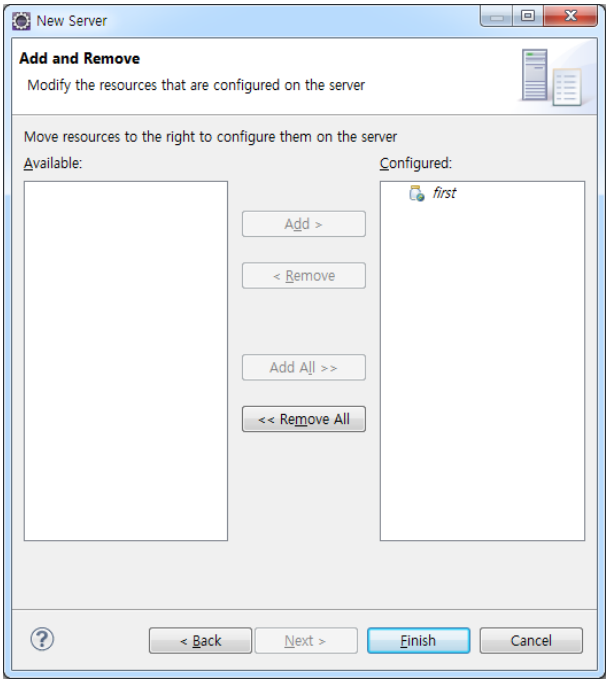
만약 여러 개의 프로젝트가 있을 경우, 하나의 프로젝트에는 하나의 서버만 할당되어야 한다. 즉, 여러개의 서버를 생성해야 할 경우, 같은 이름으로는 서버를 여러개 생성할 수도 없고, 어떤 서버가 어떤 프로젝트인지를 알 수 없기 때문에, 본인은 서버의 이름과 프로젝트의 이름을 동일하게 생성한다. 꼭 바꿀 필요는 없음.



3. Next > 를 누르자.



이와 같은 화면이 나온다. 여기에서는 first라는 프로젝트 하나밖에 없기 때문에 상관없지만, 프로젝트가 여러개가 존재할 경우, 그 모든 프로젝트의 목록이 나온다. first를 더블클릭하거나 Add > 버튼을 눌러서 Configured 쪽으로 옮기고 Finish를 누른다.



4. 서버가 생성되었을 테니, 서버를 실행시키자.

1) 서버 선택후 Ctrl + Alt + R을 누르거나,

2) 서버 우클릭 > Start를 선택하거나,

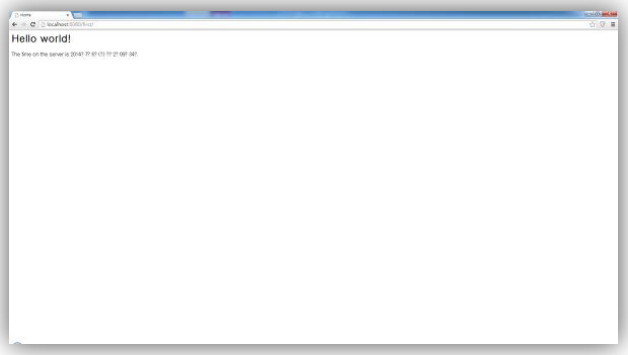
3) 녹색 동그라미안에 ▷가 있는 아이콘을 눌러서 서버를 실행시키자.

그러면 Console에 다음과같은 로그가 출력되면서 서버가 정상 작동된 것을 확인할 수 있다.

5. 브라우저를 실행시키고, 주소창에 http://localhost:포트번호/first/ 를 입력한다.

first는 우리가 생성한 프로젝트의 이름으로, 만약 다른 이름으로 프로젝트를 생성하였다면, first 대신 작성한 프로젝트 명을 입력하면 된다.

그러면 다음과 같은 화면이 나온다.



이렇게 에러없이 화면이 나오면 문제없이 프로젝트의 생성 및 실행이 된 것이다.

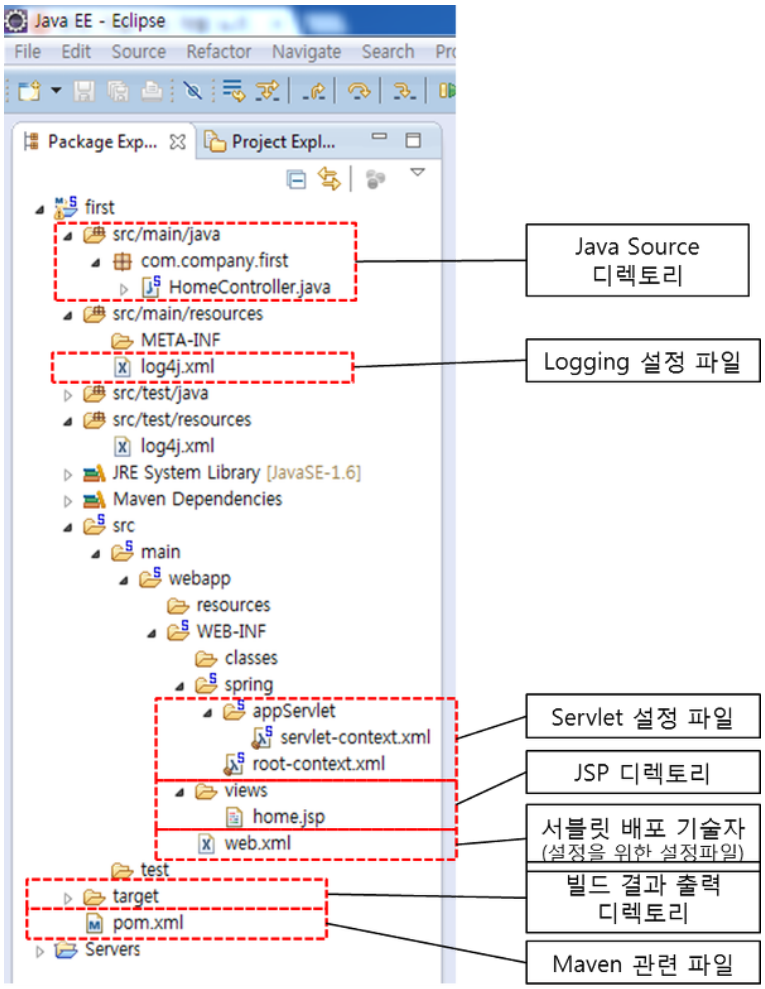
만약, 404 에러가 발생하였다면 먼저 톰캣서버를 stop하고 해당 프로젝트의 속성창을 열어 context root 명이 맞는지 확인한 다음, Servers 뷰에서 톰캣서버를 더블클릭하여 나타난 속성화면 아래쪽에 있는 Modules 탭을 선택한다. Path가 프로젝트명 또는 context root 명과 일치하는지 확인하고 다르면 수정한다. 다시 실행시켜 봄.

500번 에러의 경우 jstl core 라이브러리 관련 에러가 발생하였다면, jstl1.2.jar 파일을 복사해서 WEB-INF 폴더 아래에 lib 폴더를 만들고, 거기에 복사해 넣고 다시 실행해 봄.

(5) 스프링 기본 프로젝트 분석

1. 폴더 구조 파악

스프링 프로젝트는 다음과 같은 구조를 가지고 있다.



이 구조를 간단히 살펴보자.

1) **src/main/java**는 java 파일이 모여 있는 디렉토리이다. 우리가 앞으로 만들 java 파일은 전부 이 디렉토리에 구성된다.

2) **src/main/resources**는 스프링 설정 파일이나 쿼리가 저장될 디렉토리이다.

3) **src/test/** 관련 폴더는 test 관련 폴더인데, 우리는 TDD(Test Driven Development) 방법론이나 테스트 코드를 따로 작성하는 방식은 아직까지는 사용하지 않기 때문에, 현재로써는 필요없는 폴더이다. 과감히 삭제.

4) 우리는 메이븐을 사용하는데, 메이븐의 기본 폴더는 **src/main/webapp** 폴더가 기본 폴더이다. **webapp** 폴더 밑에 모든 jsp 및 js 등의 파일이 포함된다.

5) **servlet-context.xml, root-context.xml**은 서블릿(Servlet) 관련 설정 파일이다.

2. HomeController.java

Controller는 웹 클라이언트에서 들어 온 요청을 해당 비지니스 로직으로 분기시켜 주고, 수행 결과의 응답을 해 주는 Dispatcher의 역할을 담당하는 클래스이다.

```
@RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}.", locale);

        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale);

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate );

        return "home";
    }
```

먼저 @RequestMapping 이라는 부분이 웹 클라이언트 (jsp)에서 들어온 요청에 해당하는 비지니스 로직을 찾아주는 역할을 한다. 뒤에 method는 이 요청이 POST 인지, GET 방식인지를 말해주는데, 우리는 앞으로 거의 대부분을 POST로 보낼 것이기 때문에, 이 부분은 지울 계획이다.

그 다음은 return "home"; 이라는 부분이다. 이 부분은 수행 결과의 응답을 어디로 보낼지를 명시해준다. 나중에 서블릿(Servlet) 설정에서 다시 설명하겠지만, "home"이라는 것은 jsp 파일명을 의미한다.

서블릿 설정에서 자동으로 앞에 "/WEB-INF/views"를 붙여주고 (prefix),

뒤에 ".jsp"를 붙여주도록 되어있다.(suffix)

따라서 우리가 위에서 본 src/main/webapp/WEB-INF/views/home.jsp가 호출되게 되는 것이다.

세번째로, model.addAttribute("serverTime", formattedDate); 부분이다.

이는 비지니스 로직에서 수행한 결과를 화면으로 보내주기 위한 부분이다. serverTime이라는 이름으로 formattedDate를 전송함을 의미한다. 이를 사용하는 방법은 home.jsp에서 이야기 한다.

3. home.jsp

home.jsp는 다음과 같이 작성되어있다.

```
1       <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2       <%@ page session="false" %>
3       //중략
4
5           <title>Home</title>
6
7       //중략
8       <h1>
9           Hello world!
10      </h1>
11
12      <p>  The time on the server is ${serverTime}. </p>
13      <br>
```

첫번째로 볼 부분은 The time on the server is ${serverTime}. 이라는 부분이다.

우리는 Controller에서 model.addAttribute("serverTime", formattedDate); 라는 방식으로 화면으로 결과값을 보내주었다고 했었다. 여기서 serverTime이라는 이름으로 보낸 부분이 ${serverTime}이라는 방식으로 사용됨을 볼 수 있다.

${} 는 EL(Expression Language)를 사용한 부분이다.

${serverTime}이라는 부분이 서버에서 넘어온 결과를 화면에 보여준다.

3. web.xml

web.xml을 위에서 서블릿 배포 기술자라고 했다. 영어로는 DD (Deployment Descriptor)라고 한다. web.xml은 WAS (Web Application Server)(여기서는 Tomcat)이 최초 구동될 때, WEB-INF 디렉토리에 존재하는 web.xml을 읽고, 그에 해당하는 웹 애플리케이션 설정을 구성한다. 다시 말해, 각종 설정을 위한 설정 파일이라고 이야기 할 수 있다.

4. servlet-context.xml

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

    <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven>

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the
${webappRoot}/resources directory -->
    <resources mapping="/resources/**" location="/resources/">

    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-
INF/views directory -->
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/">
        <beans:property name="suffix" value=".jsp">
    </beans:property></beans:property></beans:bean>

    <context:component-scan base-package="com.company.first">



</context:component-scan></resources></annotation-driven></beans:beans>
```

servlet-context는 서블릿 관련 설정이다. 우리가 여기서 주목해야하는 부분은

<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<beans:property name="prefix" value="/WEB-INF/views/" />

<beans:property name="suffix" value=".jsp" />

</beans:bean>

↑ 이 부분이다.

Controller을 설명할 때, 서블릿 설정이 자동으로 prefix와 suffix를 붙인다고 해줬는데, 그 역할을 담당한다. 즉, 우리가 일일이 전체경로와 .jsp를 붙이지 않아도 되도록 도와준다.

그 다음은 <context:component-scan base-package="com.company.first" /> 이다.

이 부분은 스프링에서 사용하는 bean을 일일이 xml에 선언하지 않고도 필요한 것을 어노테이션(Annotation)을 자동으로 인식하게 하는 역할을 한다.