이클립스 웹 개발 2014

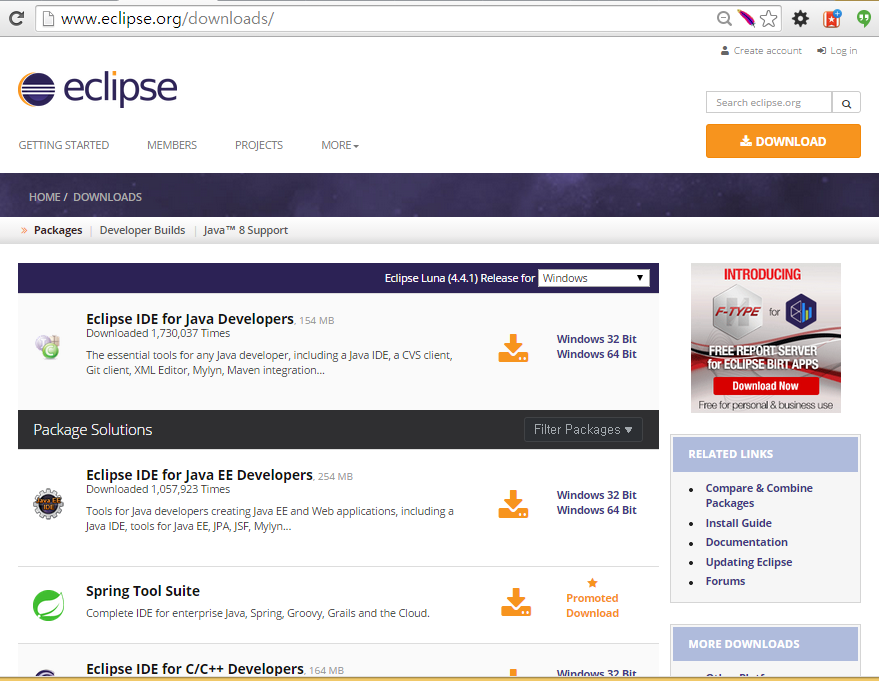
이클립스라는 도구를 알게 된지 벌써 14년 가까이 된다. 14년간 무섭게 성장해 온 이클립스를 보면서 오픈소스의 힘과 수많은 사람들의 열정이 만들어 낸 결과를 감사하게 생각한다.

이클립스라는 도구가 없던 시절에도 자바와 웹은 잘 개발되고 있었다. 당시에는 에디트플러스나 울트라에디트 같은 텍스트편집기가 많은 개발자들에게 사용되는 자바 개발도구였다. 하지만 강력한 기능과 확장성 그리고 무료로 배포되는 이클립스라는 통합 개발 환경을 갖춘 도구 때문에 2014년 현재의 자바 개발 현장에서는 이클립스라는 도구를 기준으로 개발 가이드를 제시하고 있다.

이 글은 필자의 프로젝트 경험을 바탕으로 개발 입문자들이 쉽게 이클립스라는 개발도구와 이클립스 플랫폼을 통해서 만들고자 하는 소프트웨어를 빠르고 정확하게 만들 수 있게 하려고 쓰여졌다.

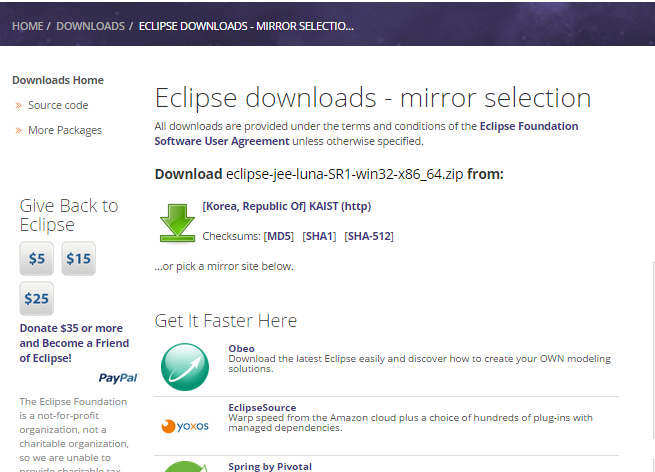
1.1. 설치

이클립스의 설치는 이클립스 웹 사이트에서 다운로드 받아서 압축만 해제하면 쉽게 끝난다. <http://www.eclipse.org/downloads> 페이지에 접속하면 자신의 OS 플랫폼에 맞는 이클립스 배포판들을 볼 수 있다.



<그림1> 이클립스 다운로드 배포판 목록

많은 목록 중에서 자바 웹 개발이 가능한 Eclipse IDE for Java EE Developers 우측의 다운로드 링크를 선택한다. 구형 OS에서는 32Bit를 지원하는 경우가 많으며, 요즘은 64Bit OS가 늘어나는 추세이다.



녹색 화살표를 클릭하면 다운로드가 시작되고, 210MB 정도의 압축된 파일이 다운로드된다.

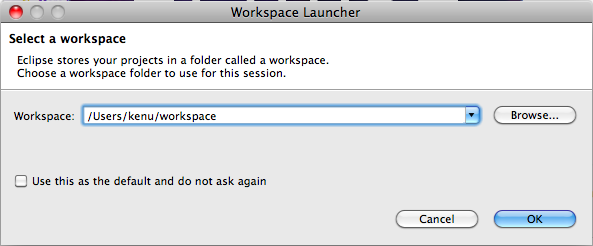
KAIST 미러 링크가 느린 경우는 Japan Advanced Institute Sience Technology 미러 링크를 클릭하시는 것을 추천한다.

압축을 해제하면 eclipse 폴더가 나타난다. 개발 관련된 프로그램들의 기준 폴더를 만들고 그쪽을 위치를 이동하도록 하겠다. 윈도우는 C:\java 폴더를, 맥, 리눅스는 ~/java 폴더를 각각 만들어서 압축이 해제된 eclipse 폴더를 기준 폴더 아래로 옮겨 놓는다.

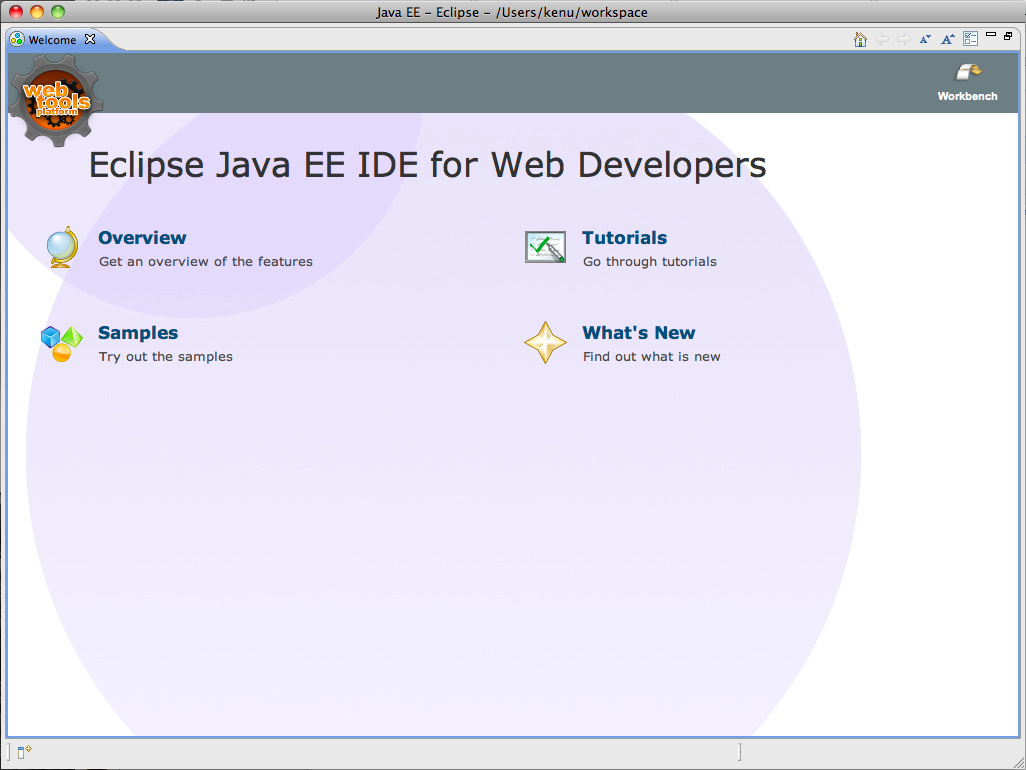
설치는 이로서 완료되었다.

1.2. 이클립스 실행

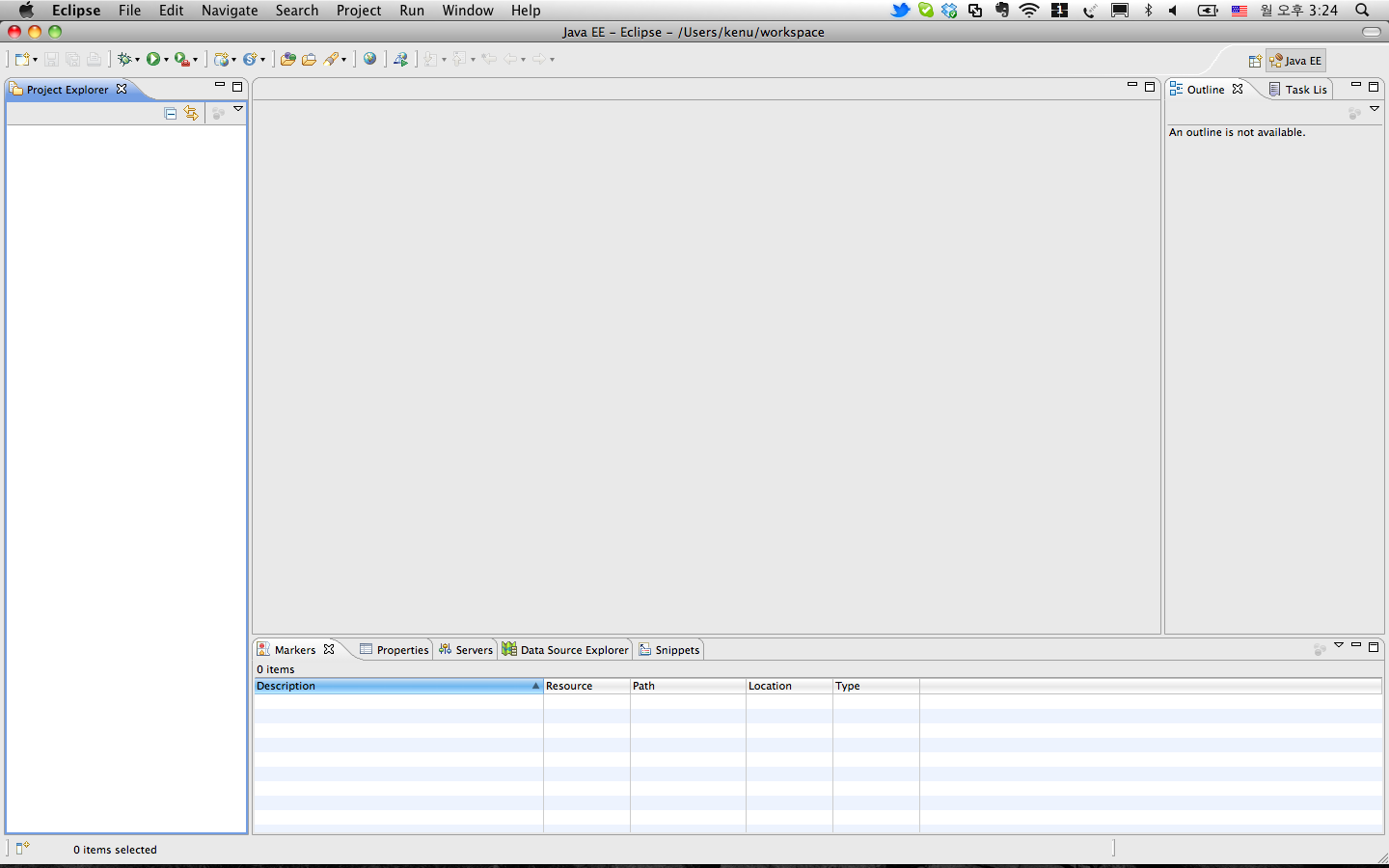
윈도우는 eclipse.exe, 맥과 리눅스는 eclipse를 실행하면 다음과 같은 화면을 볼 수 있다. workspace는 소스를 관리하는 기준 경로이고, 이 아래에 프로젝트의 소스들이 위치하게 된다. 이클립스의 모든 설정 정보는 workspace/.metadata 폴더 아래에 저장이 된다. 이클립스 업그레이드가 되어도 이 폴더는 그대로 연결이 되며 모든 설정 정보가 그대로 보전된다.



이클립스 JEE의 첫 화면은 다음과 같이 Welcome 화면에서 시작된다. 좌상단의 Welcome 탭의 x 를 클릭해서 탭을 닫고 Workbench 화면으로 전환해보자.



<그림> 이클립스 JEE Welcome 화면



<그림> 이클립스 JEE 워크벤치

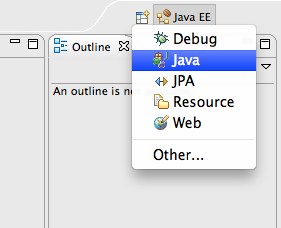
이클립스 워크벤치의 화면 구성은 다음과 같다. 제일 상단에 메뉴가 있고, 그 아래 아이콘으로 구성된 툴바가 위치한다. 툴바의 우측에는 퍼스펙티브(화면 레이아웃)를 선택할 수 있는 퍼스펙티브 셀렉터가 있다. 화면 가운데를 중심으로 좌측과 우측 하단에 뷰들이 놓여있고, 화면 중앙에는 코드를 편집할 수 있는 에디터가 위치한다.

1.3. 자바 프로젝트 만들기

자바 웹 개발에 앞서 자바 프로젝트로 시작해 보자. 자바 웹 개발에는 서블릿 컨테이너가 필요하며 자바 프로젝트는 자바 프로그래밍을 할 수 있는 최소 단위의 프로젝트이다. 프로그램 개발 환경이 제대로 되어 있는지는 관례적으로 “Hello World” 문자열을 프린트해보는 것으로 시작한다.

1.3.1 퍼스펙티브 전환

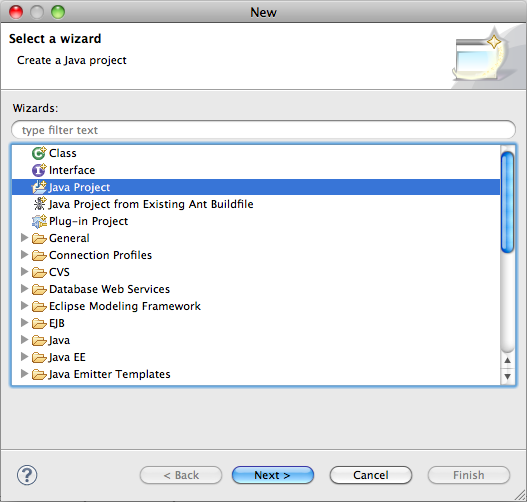
자바 프로젝트를 위해서 JavaEE 퍼스펙티브에서 자바 퍼스펙티브로 전환하는 것이 좋다. 화면 우측상단의 퍼스펙티브 셀렉터에서 Open Perspective 아이콘을 클릭해서 Java를 선택한다.



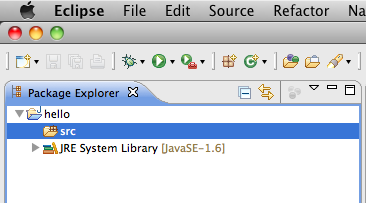
화면 레이아웃이 다음과 같이 순수한 자바를 개발할 수 있도록 바뀌었다. 왼쪽 뷰는 패키지 익스플로러(Package Explorer)로 바뀌었고, 하단의 뷰도 프로블럼(Problems) 등 3개로 뷰 탭 목록이 변경되었고, 우측은 태스크리스트(Task List)와 아웃라인(Outline) 뷰로 나뉘어졌다.

1.3.2 자바 프로젝트 만들기

이클립스는 파일별로 다루는 텍스트 편집기와는 달리 프로젝트가 기본 단위이다. 자바 파일 하나를 만들기 위해서 기반이 되는 프로젝트를 먼저 만들어야 한다. 좌측 상단의 New 아이콘을 클릭하면 다음과 같은 창이 뜬다.



Java Project 항목을 선택하고 하단의 Next > 버튼을 클릭한다. 이어서 프로젝트명을 입력할 수 있는 화면이 보인다. 그림과 같이 hello라고 입력하고 하단의 Finish버튼을 클릭한다. hello라는 이름으로 자바 프로젝트가 만들어진다.



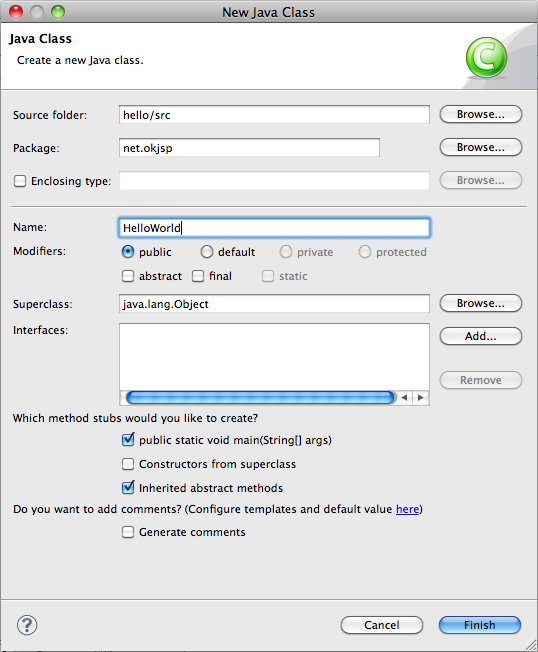
hello 프로젝트 아이콘에 J라는 작은 글자는 해당 프로젝트가 자바 프로젝트임을 표시한다.

1.3.3 자바 클래스 만들기

프로젝트를 확장하면 src 폴더가 보이고 이곳이 java 소스 파일이 위치하는 기준이 된다. src를 선택하고 툴바에 있는 녹색의 New Java Class 아이콘을 클릭한다.

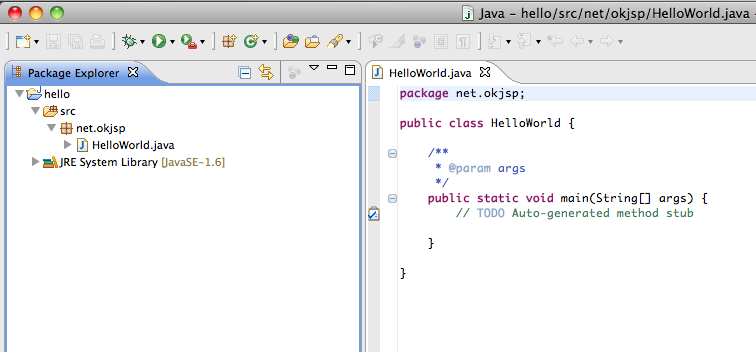


다음 그림과 같이 창이 뜨면 3 부분을 변경한다. ① 커서가 깜박거리는 Name: 항목에 HelloWorld라고 입력한다. 자바는 기본적으로 대소문자를 구분하고 클래스명은 단어의 첫글자를 대문자로 쓰는 낙타등 표기법(Camel Case라고 한다)을 사용한다. ② 하단의 public static void main(String[] args) 앞에 있는 박스에 그림과 같이 체크한다. 실행 가능한 자바 파일에는 반드시 필요한 메소드(함수)이다. ③ 창의 상단에 클래스를 그루핑하는 패키지명을 입력한다. Package: 항목에 net.okjsp 라고 입력한다. Finish 버튼을 클릭하면 HelloWorld.java 파일이 만들어진다.



<그림> New Java Class 위자드 창

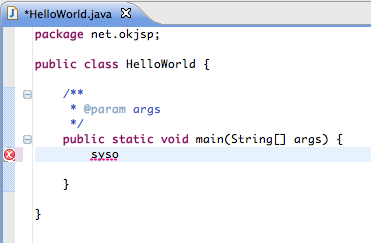
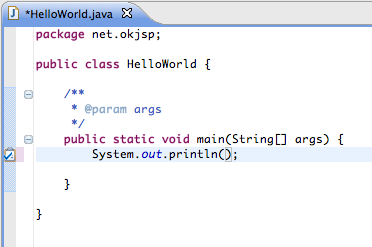
파일이 만들어진 위치는 hello 프로젝트의 src폴더 아래 net/okjsp 패키지에서 정한 폴더가 생기고, 그 안에 HelloWorld.java 파일이 위치한다.



이클립스 상단의 타이틀바를 보면 Java - hello/src/net/okjsp/HelloWorld.java 라고 표시된 것을 확인할 수 있다.

1.3.4 코딩과 실행

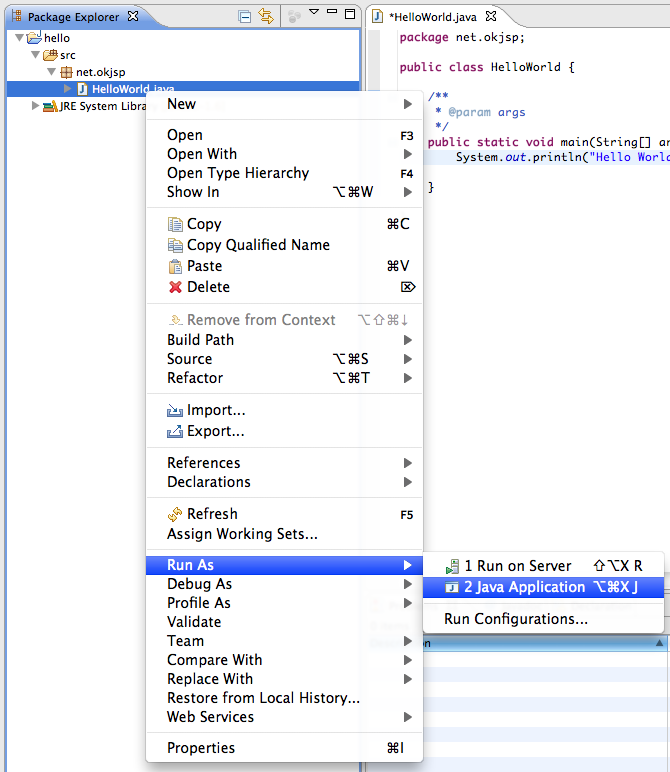
HelloWorld.java 파일 편집 영역에서 // TODO 라인을 지우고, 그림처럼 syso 라고 입력해보자. syso 끝 위치에서 ctrl+space를 치면 System.out.println();으로 자동 변환된다. 관련된 기능은 이후에 소개하겠다.

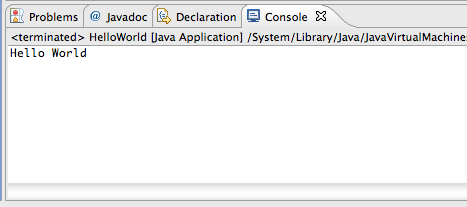
우선 System.out.println(); 괄호 안에 "Hello World"라고 입력하고 윈도우와 리눅스에서는 ctrl+s를, 맥은 command+s를 눌러서 저장하면 다음과 같이 가장 기본적인 자바 코드가 완성된다. 이클립스가 저장하는 시점에 자동으로 컴파일된 class파일을 만들어낸다.



패키지 익스플로러의 src아래에 있는 HelloWorld.java 파일을 선택하고, 오른버튼을 누르면 나오는 컨텍스트 메뉴 하단에 보이는 Run As > Java Application 항목을 선택하면 실행된다.



실행 결과는 하단에 Console뷰가 새로 뜨면서 여기에 Hello World라는 출력결과가 나타난다.



간단하게 자바 클래스를 만들고 실행하는 방법을 알아보았다.

다음 절에서는 간단한 JSP 페이지를 만들어서 실행하는 법을 소개하겠다.

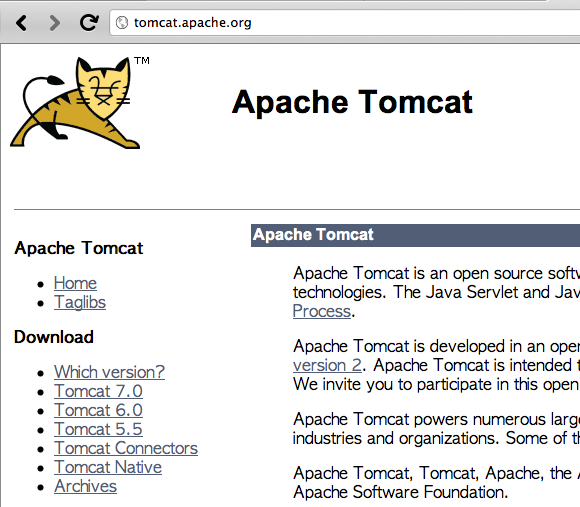
1.4. 다이나믹 웹 프로젝트 만들기

이번에는 JSP나 서블릿 같은 자바 웹 개발을 위해서 이클립스를 사용하는 방법을 알아보도록 하겠다. main() 메소드를 통해 개별적으로 실행되는 자바 애플리케이션과 달리 자바 웹 프로그래밍은 서블릿을 실행하는 서블릿 컨테이너가 필요하다. 대표적인 것이 아파치 톰캣이 있으며, 표준 스펙을 따르는 다양한 경쟁 제품들도 있다. 우선 톰캣을 이클립스에 연결하고 JSP와 서블릿을 실행해 보도록 하겠다.

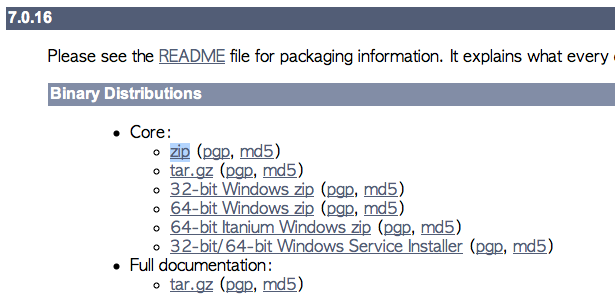
1.4.1 톰캣 설치

아파치 재단에서 관리되는 톰캣은 <http://tomcat.apache.org/> 사이트에서 구할 수 있다. 윈도우용 설치버전과 리눅스 계열의 압축버전이 바이너리로 제공이 된다. 설명은 압축된 버전으로 할 것이다.

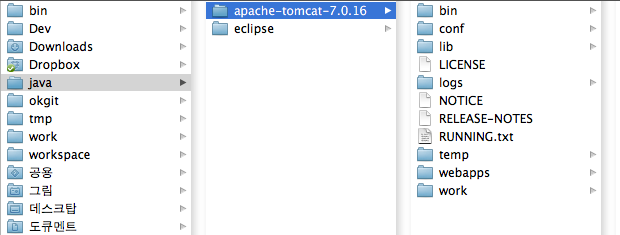
톰캣 홈페이지 좌측에 있는 Download 섹션에서 Tomcat 7.0 링크를 클릭한다.



다운로드 페이지에서 Binary Distibutions의 Core에 있는 zip 파일을 다운로드한다. OS에 상관없이 사용이 가능한 버전이다. 톰캣 7.0은 JDK1.6 이상에서만 동작하며 7.0.x에 대한 자세한 내용은 README 링크를 확인하기 바란다.



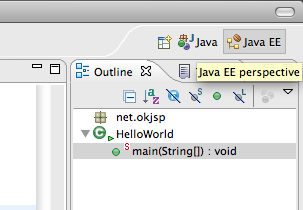
zip파일 압축을 해제하면 나오는 폴더(apache-tomcat-7.0.16)를 앞서 기준으로 삼았던 java 폴더 아래로 이동한다. 윈도우는 C:\java\apache-tomcat-7.0.16 가 될 것이고, 맥과 리눅스는 ~/java/apache-tomcat-7.0.16 처럼 경로가 될 것이다. 다음 그림과 같이 구성되었는지 확인해 보자. 주의할 점은 압축 해제 프로그램에 따라 apache-tomcat-7.0.16 하위에 또 하나의 apache-tomcat-7.0.16 폴더가 생기는 경우가 있는데, 중복되지 않게 하자.



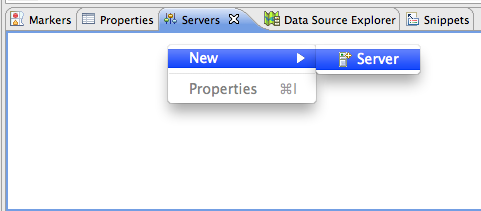
톰캣은 설치가 완료되었다. 이제 이클립스에서 톰캣을 연결해보도록 하겠다.

1.4.2 이클립스에서 톰캣 연결

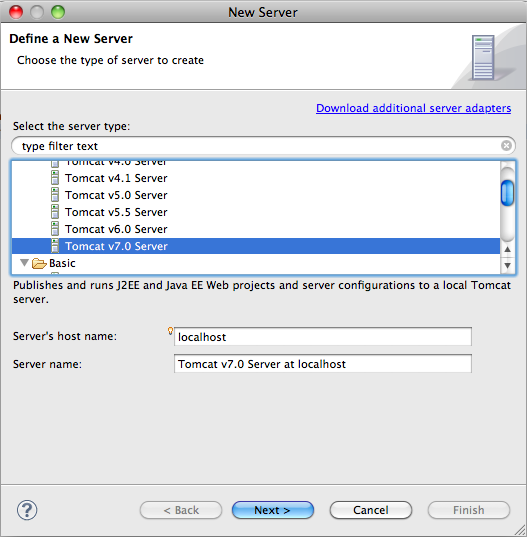
이클립스 퍼스펙티브를 Java EE로 변경한다. 우측 상단의 Java EE라고 쓰여진 아이콘을 클릭하면 퍼스펙티브가 웹 개발 레이아웃으로 전환된다.



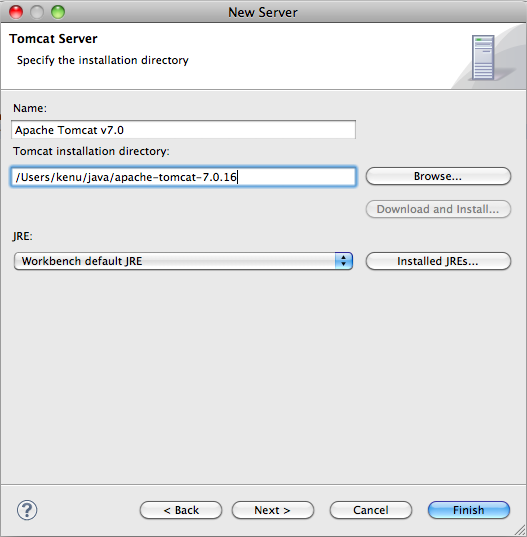
하단에 있는 Servers 탭을 선택한다. 아래 영역에서 컨텍스트 메뉴를 열고 New > Server 항목을 선택한다.



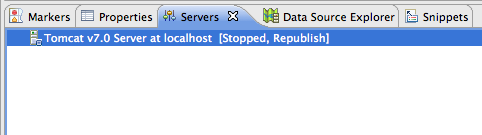
서버를 선택하는 창에서 Tomcat v7.0 Server를 선택하고 하단의 Next > 버튼을 클릭한다. 톰캣 이외의 서버는 이 책의 후반에서 더 자세히 다루도록 하겠다.



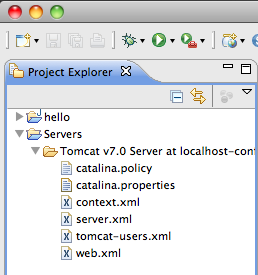
톰캣 서버 설정하는 화면에서는 우측의 Browse... 버튼을 클릭해서 압축을 해제한 톰캣의 경로를 선택한다. Tomcat installation directory: 항목에 apache-tomcat-7.0.16의 전체 경로가 지정되면 하단의 Finish 버튼을 클릭해서 연결을 완료한다.



이클립스 하단의 Servers탭과 좌측 프로젝트 익스플로러 탭에 Servers 프로젝트가 추가되어 다음과 같이 보여지면 된다.

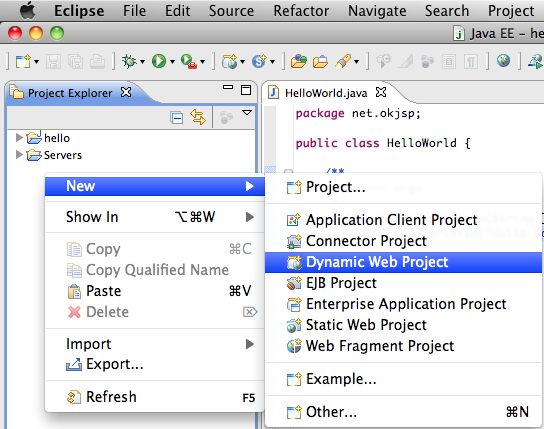


이클립스에 연결된 톰캣은 설정파일을 이클립스에서 따로 관리한다. 그 설정 파일들은 Servers 프로젝트 아래에 위치한다.

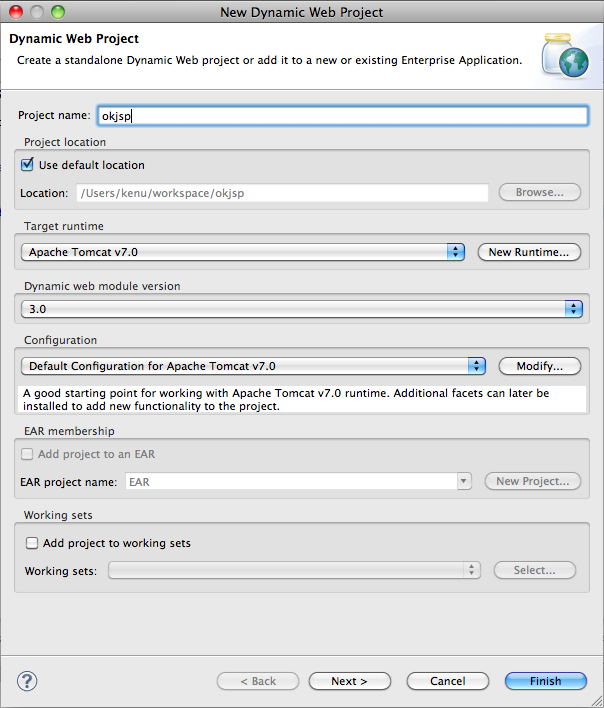


1.4.3 다이나믹 웹 프로젝트 만들기

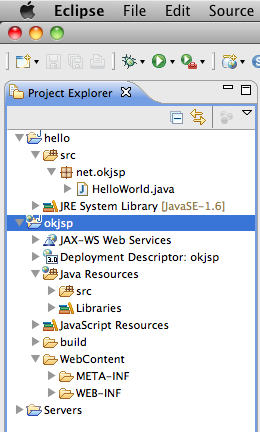
톰캣의 연결이 완료되었으면 이제 자바 웹 개발을 할 준비가 거의 다 되었다. JSP파일을 만들어 실행해보기 위해서 필요한 프로젝트가 다이나믹 웹 프로젝트(Dynamic Web Project)이다. Java EE 퍼스펙티브에서 좌측의 프로젝트 익스플로러 뷰 내에 마우스 오른버튼을 클릭해서 나오는 컨텍스트 메뉴에서 New > Dynamic Web Project를 선택한다.



이어 나오는 창에 프로젝트 이름을 okjsp 라고 입력한다. 이것은 웹 애플리케이션에서 컨텍스트 명이라고 불리는 이름이고, 브라우저 상에 노출되는 경로가 된다.



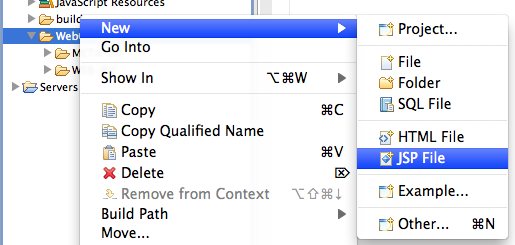
Finish 버튼을 클릭해서 프로젝트를 생성한다. 프로젝트 익스플로러에 보이는 okjsp 프로젝트를 펼쳐보면 다음과 같다.



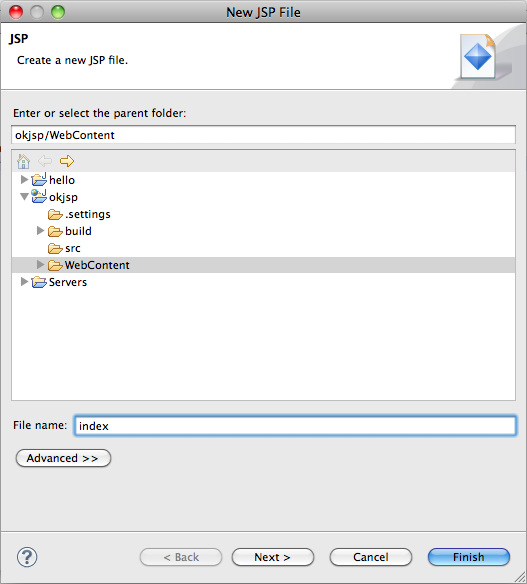
1.4.4 JSP 만들기

마음이 급한 분들을 위해서 일단 먼저 JSP 파일을 만들어서 실행해 보자.

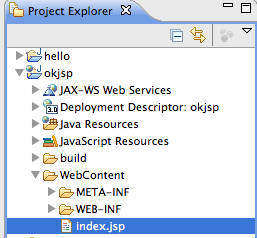
WebContent 폴더를 선택하고 마우스 오른버튼을 클릭해서 New > JSP File 항목을 선택한다.



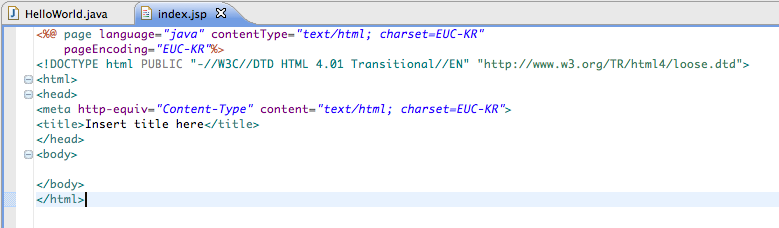
JSP 파일명은 index라고 입력한다. JSP 생성하는 창이기 때문에 index.jsp라고 확장자는 자동을 붙여진다. Finish 버튼을 클릭해서 index.jsp 파일을 생성한다.



WebContent 폴더 아래에 index.jsp 파일이 생긴 것을 확인한다.



에디터 영역에 자동으로 열린 index.jsp 파일은 다음과 같다. 가장 기본적인 html 템플릿으로 내용이 채워진 것을 확인할 수 있다.



1.4.5 JSP 실행하기

내용을 수정해서 실행해보자. <title>Insert title here</title> 을 <title>Hello JSP</title> 로 변경한다. <body> 태그 아래 <h1>Server Time: <%= new java.util.Date() %></h1> 이라고 입력한다. 코드는 다음과 같이 변경된다. 굵게 표시된 영역이 변경된 부분이다.

<%@ page language="java" contentType="text/html; charset=EUC-KR"

pageEncoding="EUC-KR"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>**Hello JSP**</title>

</head>

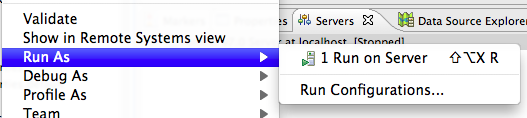
<body>

**<h1>Server Time: <%= new java.util.Date() %></h1>**

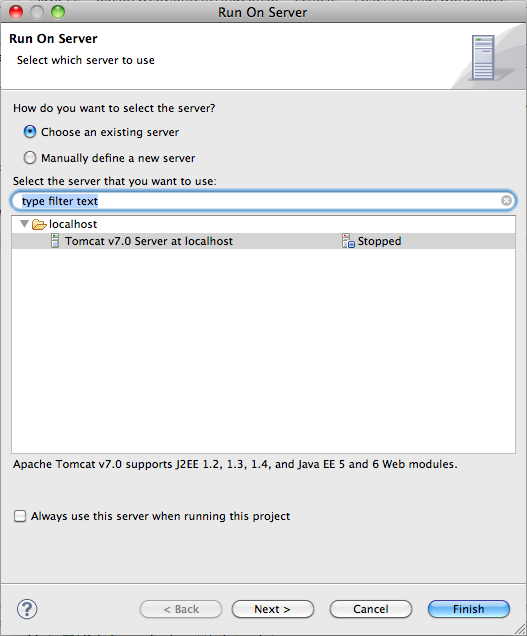
</body>

</html>

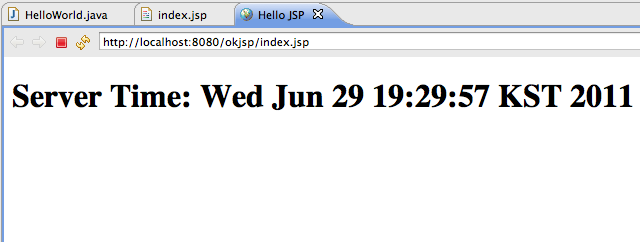
이제 이 JSP파일을 실행해보자. index.jsp 파일을 선택하고 컨텍스트 메뉴 하단의 Run As > Run on Server 항목을 선택한다.



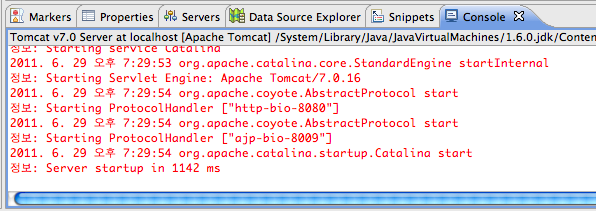
아직 okjsp 프로젝트와 톰캣이 연결되지 않았기 때문에, 다음과 같은 창이 뜨게 된다.



가볍게 Finish 버튼을 클릭하면 okjsp 프로젝트가 톰캣에 연결되고, 톰캣이 실행된다. 톰캣 실행이 끝나면, 이클립스 내에 브라우저가 뜨면서 index.jsp의 실행결과를 확인할 수 있다.



톰캣이 실행될 때 나타나는 시스템 로그는 하단의 콘솔(Console)뷰에서 확인할 수 있다.



상식이지만 JSP는 서블릿 컨테이너에서 서블릿 자바코드로 변환해서 컴파일한 뒤에 실행이 된다. 다음 절에서는 톰캣의 숨겨진 동작을 보면서 좋은 팁 한 가지를 소개하겠다.

1.4.6 숨겨진 동작 구조

이클립스 도구를 쓰기 전, 톰캣을 실행해서 개발할 때는 apache-tomcat-7.0.16/webapps/ROOT 폴더 아래에 jsp 파일을 넣고 브라우저로 실행해서 개발을 했다. 변환된 서블릿 java파일과 class파일은 apache-tomcat-7.0.16/work/ 하위의 폴더에 숨겨져 있어서 심한 경우 이 경로를 뒤져서 디버깅을 하기도 한다.

이클립스 웹 개발을 하는 경우는 이와는 조금 다른 메커니즘을 갖는다. 기본적인 설정 경로가 다르기 때문에 이에 대해서 확인해보자.

index.jsp 파일에 현재 파일의 물리적 경로를 표시하는 구문을 추가한다.

<body>

<h1>Server Time: <%= new java.util.Date() %></h1>

**<%= application.getRealPath("/") %>**

</body>

실행 결과를 예상해 본다면 현재 프로젝트 파일이 있는 경로가 보일 것이라 생각된다. 하지만 결과는 다음과 같이 전혀 생각지 못한 경로로 나온다.



workspace 하위의 .metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/okjsp/ 경로로 나타난다. 전체 경로를 복사해서 해당 위치로 이동하면 다음과 같은 파일들을 볼 수 있다.

kenuheoyimacbookpro-4:~ kenu$ **cd /Users/kenu/workspace/.metadata/ .plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/okjsp/**

kenuheoyimacbookpro-4:okjsp kenu$ **find .**

.

./index.jsp

./META-INF

./META-INF/MANIFEST.MF

./WEB-INF

./WEB-INF/classes

./WEB-INF/lib

kenuheoyimacbookpro-4:okjsp kenu$

수정한 index.jsp 파일이 보인다. 내용을 확인해보자. 윈도우는 cat index.jsp 대신 type index.jsp 라고 입력하면 된다.

kenuheoyimacbookpro-4:okjsp kenu$ **cat index.jsp**

<%@ page language="java" contentType="text/html; charset=EUC-KR"

pageEncoding="EUC-KR"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>Hello JSP</title>

</head>

<body>

<h1>Server Time: <%= new java.util.Date() %></h1>

<%= application.getRealPath("/") %>

</body>

</html>kenuheoyimacbookpro-4:okjsp kenu$

이 경로의 상위로 두 번 올라가면 work 폴더가 보인다.

kenuheoyimacbookpro-4:okjsp kenu$ **pwd**

/Users/kenu/workspace/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/okjsp

kenuheoyimacbookpro-4:okjsp kenu$ **cd ../..**

kenuheoyimacbookpro-4:tmp0 kenu$ **ls**

conf logs webapps wtpwebapps

lib temp work

work 폴더 내용을 보면 다음과 같다.

kenuheoyimacbookpro-4:tmp0 kenu$ **find work**

work

work/Catalina

work/Catalina/localhost

work/Catalina/localhost/\_

work/Catalina/localhost/okjsp

work/Catalina/localhost/okjsp/org

work/Catalina/localhost/okjsp/org/apache

work/Catalina/localhost/okjsp/org/apache/jsp

work/Catalina/localhost/okjsp/org/apache/jsp/index\_jsp.class

work/Catalina/localhost/okjsp/org/apache/jsp/index\_jsp.java

index\_jsp.java 파일이 자동으로 만들어졌고, 이 파일이 컴파일된 index\_jsp.class 파일도 보인다. index\_jsp.java 파일 내용은 대략 다음과 같다.

kenuheoyimacbookpro-4:tmp0 kenu$ cat work/Catalina/localhost/okjsp/org/apache/jsp/index\_jsp.java

/\*

\* Generated by the Jasper component of Apache Tomcat

\* Version: Apache Tomcat/7.0.16

\* Generated at: 2011-07-03 04:36:16 UTC

\*/

package org.apache.jsp;

import javax.servlet.\*;

import javax.servlet.http.\*;

import javax.servlet.jsp.\*;

public final class index\_jsp extends org.apache.jasper.runtime.HttpJspBase

implements org.apache.jasper.runtime.JspSourceDependent {

...중략...

try {

response.setContentType("text/html; charset=EUC-KR");

pageContext = \_jspxFactory.getPageContext(this, request, response,

null, true, 8192, true);

\_jspx\_page\_context = pageContext;

application = pageContext.getServletContext();

config = pageContext.getServletConfig();

session = pageContext.getSession();

out = pageContext.getOut();

\_jspx\_out = out;

out.write("\n");

out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\" \"http://www.w3.org/TR/html4/loose.dtd\">\n");

out.write("<html>\n");

out.write("<head>\n");

out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=EUC-KR\">\n");

out.write("<title>Hello JSP</title>\n");

out.write("</head>\n");

out.write("<body>\n");

out.write("<h1>Server Time: ");

out.print( new java.util.Date() );

out.write("</h1>\n");

out.print( application.getRealPath("/") );

out.write("\n");

out.write("</body>\n");

out.write("</html>");

} catch (java.lang.Throwable t) {

if (!(t instanceof javax.servlet.jsp.SkipPageException)){

out = \_jspx\_out;

if (out != null && out.getBufferSize() != 0)

try { out.clearBuffer(); } catch (java.io.IOException e) {}

if (\_jspx\_page\_context != null) \_jspx\_page\_context.handlePageException(t);

}

} finally {

\_jspxFactory.releasePageContext(\_jspx\_page\_context);

}

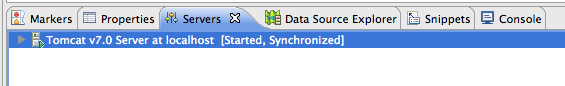
}

}

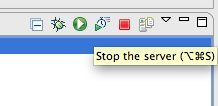
kenuheoyimacbookpro-4:tmp0 kenu$

index.jsp 내용들이 out.write() 안에 감싸진 것을 확인할 수 있다.

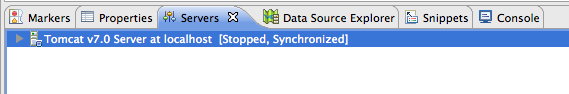
wtpwebapps/okjsp 경로의 파일들을 war로 압축하면 서버에 바로 전송해서 서비스가 가능한 구조를 갖고 있다. 이클립스 프로젝트 하위에 있는 파일들이 컴파일된 후 이곳으로 복사가 된다. 이것을 퍼블리쉬(publish)라고 한다. 단점은 프로젝트의 파일이 1,000개를 넘어서면 작업에 불편할 정도로 이클립스가 느려지기도 한다. 때문에 이런 퍼블리쉬 과정을 제외시키는 옵션이 있다.

이클립스 하단의 Servers탭을 선택한다. 

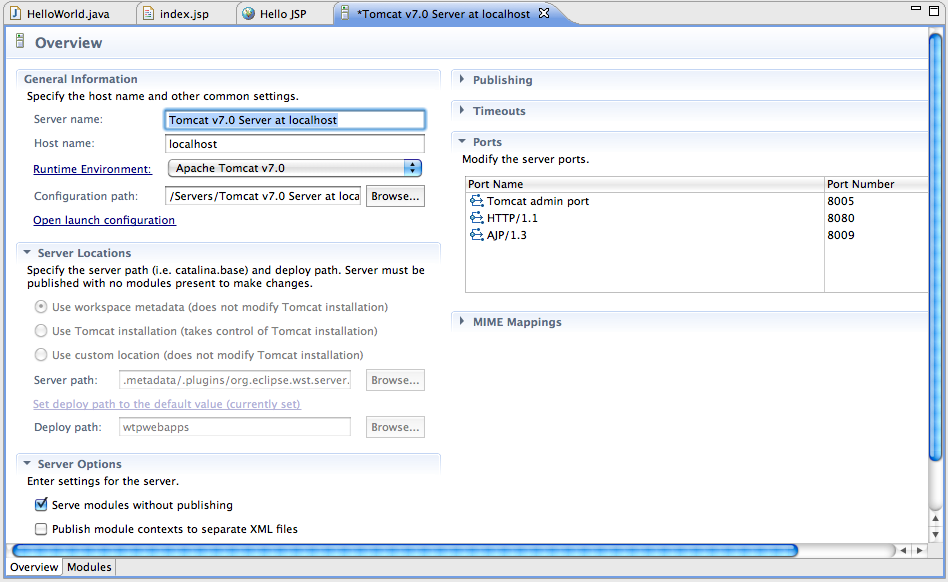
Tomcat v.7.0 Server at localhost [Started, Synchronized] 항목을 선택하고 탭 우측의 빨간 네모로 보이는 정지 아이콘을 클릭한다.



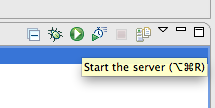
정지가 된 화면은 다음과 같이 [Stopped, Synchronized]로 표시된다.



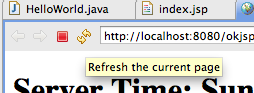
Tomcat v7.0 Server at localhost 항목을 더블클릭하면 에디터 영역에 다음과 같이 설정 화면이 보인다. 좌측 하단의 Server Options에서 Serve modules without publishing 항목을 체크하고 이클립스 상단 툴바의 좌측에 있는 디스크 아이콘을 클릭해서 저장한다. (단축키 ctrl+s, command+s)



서버를 다시 실행시킨다. Servers 탭의 우측에서 녹색 원의 플레이 아이콘을 클릭하면 된다.



에디터 영역의 Hello JSP 탭을 열고 페이지를 다시 불러온다. 주소줄 좌측의 노란 화살표 아이콘을 클릭하면 된다.



페이지 결과로 나온 바뀐 경로를 확인해보자.



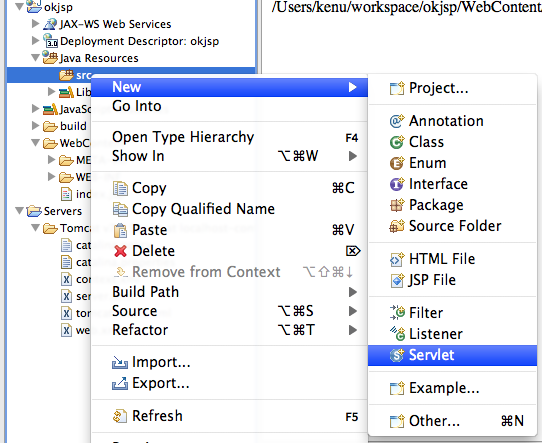
경로가 프로젝트의 WebContent로 변경된 것을 확인할 수 있다. 이렇게 톰캣을 설정하면 퍼블리쉬로 인해 지연되는 시간을 제거해서 쾌적하게 작업이 가능하다.

다음 절에서는 서블릿을 만들어 보겠다.

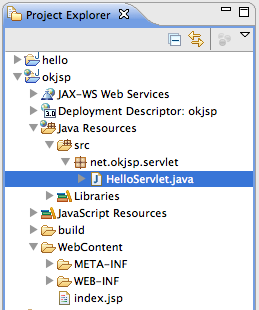
1.4.7 서블릿 만들기

서블릿은 JSP보다 먼저 만들어졌다. 하지만 html을 처리할 때 매번 다시 컴파일해줘야 하는 번거로움 때문에 화면처리를 위해서 JSP가 만들어졌고, 서블릿은 비즈니스 로직을 담당하는 모델부분과 JSP를 연결하는 컨트롤러 역할을 주로 담당하게 되었다. 여기에 관해서는 직접 서블릿을 만들어 보고 다시 설명하겠다.

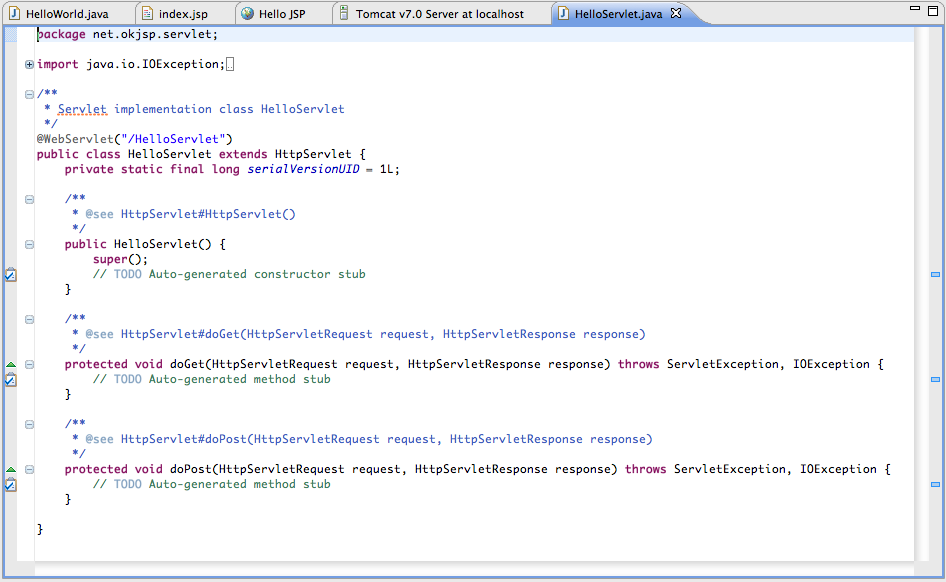
다이나믹 웹 프로젝트인 okjsp에서 Java Resources > src 폴더를 선택한다. 마우스 오른 버튼 클릭으로 나오는 컨텍스트 메뉴에서 New > Servlet을 선택한다.



이어 나오는 창에서 패키지는 net.okjsp.servlet 으로 클래스명은 HelloServlet 으로 입력한다. Finish를 버튼을 클릭하면 서블릿이 만들어진다.



서블릿으로 생성된 HelloServlet.java 파일의 내용은 다음과 같다.



서블릿을 이해하기 위해서는 라이프사이클을 알 필요가 있다. main() 메소드로 실행되는 자바 애플리케이션과는 달리 서블릿은 컨테이터가 서블릿을 컨트롤한다. 이 때 상태에 따라서 실행되는 메소스들이 달라지게 되는데, 이것을 도표로 나타내면 다음과 같다.

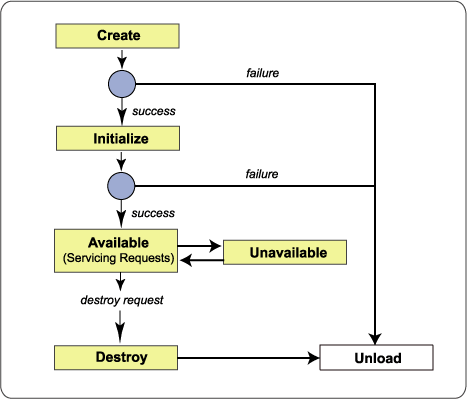
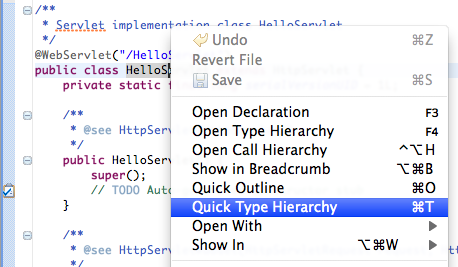
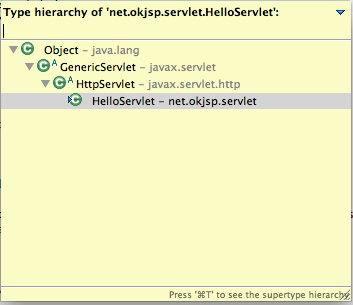


image from: <http://wiki.eeng.dcu.ie/ee557/g2/426-EE.html> <qqq한글로 번역 필요>

에디터 영역의 HelloServlet을 선택하고 컨텍스트 메뉴에서 Quick Type Hierarchy 메뉴를 선택한다. 단축키인 ctrl+T 또는 command+T를 눌러도 된다.



다음과 같은 상속 구조를 확인할 수 있다.



모든 자바 클래스의 할아버지인 java.lang.Object 가 제일 상위에 있고, 추상(abstract)클래스인 javax.servlet.GenericServlet 이 그 아래 놓인다. 때문에 C 아이콘에 A가 붙어있다. 이어서 javax.servlet.http.HttpServlet 클래스가 보이고 이것을 상속받은 net.okjsp.servlet.HelloServlet 클래스를 볼 수 있다.

화면 우측 Outline 뷰에서 doGet()을 선택하면 커서가 HelloServlet 파일의 doGet() 메소드로 옮겨간다. 다음과 같이 doGet() 메소드의 내용을 수정한다.

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

**response.setContentType("text/html");**

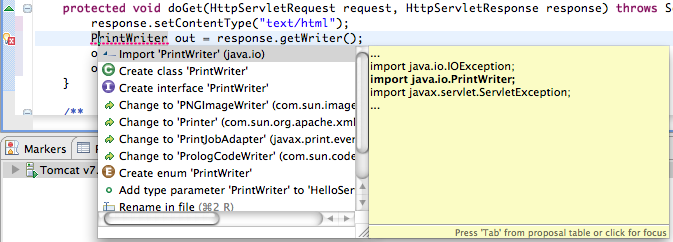
**PrintWriter out = response.getWriter();**

**out.println("<h1>Hello World</h1>");**

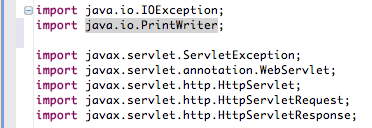
**out.close();**

}

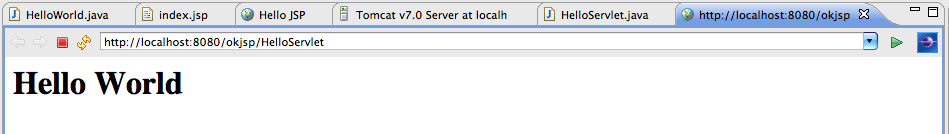
입력을 마치면 PrintWriter에 에러를 표시하는 빨간줄이 다음과 같이 보인다. 클래스에 대한 import 가 선언되지 않았기 때문이고, 커서를 PrintWriter out 줄에 옮겨놓고 퀵 픽스(Quick Fix) 단축키인 ctrl+1(맥은 command+1)을 누른다.



첫째 항목을 선택하면 자동으로 클래스 상단에 import java.io.PrintWriter; 라인이 추가된다.



코드를 실행할 준비가 되었다. 이클립스 좌측에 있는 프로젝트 익스플로러 뷰에서 HelloSerlvet.java를 선택하고 컨텍스트 메뉴에서 Run As > Run on Server를 선택한다. 톰캣이 실행중이라면 Restart Server를 선택한다.



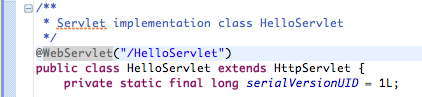
주소줄을 보면 다음과 같다.

http://localhost:8080/okjsp/HelloServlet

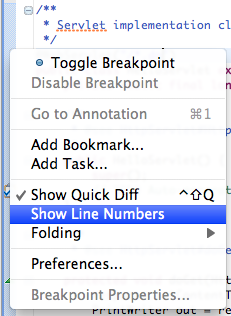
서블릿 호출 주소를 풀어보면 다음과 같다.

|  |  |
| --- | --- |
| http:// | 프로토콜 |
| localhost:8080 | 서버 도메인 또는 IP와 포트 번호 |
| /okjsp | 컨텍스트명 |
| /HelloServlet | 서블릿 URL매핑 |

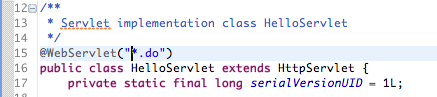
/HelloServlet 경로의 매핑은 서블릿 스펙 3.0(톰캣7)부터는 어노테이션 @WebServlet 으로 지정된다. 그 전까지는 WEB-INF/web.xml에 만들어져 있다. 서블릿 매핑을 "/HelloServlet" 에서 "\*.do" 로 변경해보자.



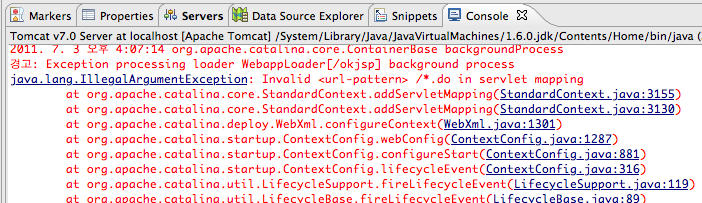
행 번호를 보이도록 설정을 바꾸자 코드 좌측의 여백에 마우스를 놓고 컨텍스트 메뉴를 보면 Show Line Numbers 항목이 보인다. 이것을 선택하면 에디터 영역 소스의 줄번호가 보이게 된다.



다음과 같이 15번 라인이라는 것을 쉽게 알 수 있다.



매핑이 잘못 설정된 경우는 다음과 같이 Console뷰에 이유가 나타나게 된다. "\*.do" 대신 "/\*.do"로 /가 있는 경우에 발생한 에러메시지이다.

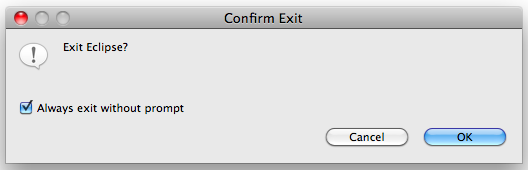


정상적으로 된 경우 브라우저 주소줄에 <http://localhost:8080/okjsp/hello.do> 라고 입력을 하면 HelloServlet 실행결과를 볼 수 있다. 이 때 hello.do의 hello는 와일드카드(\*) 패턴이기 때문에 아무 단어를 써도 된다.

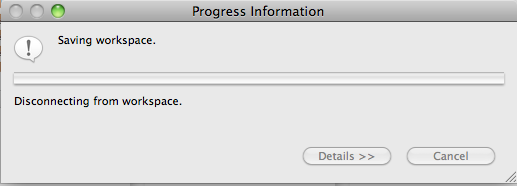
1.4.8 이클립스의 종료

이클립스를 정상적으로 종료하면 모든 설정이 이때 저장된다. 이클립스가 강제 종료된 경우에는 이전에 정상종료시의 설정이 불려지게 된다. 윈도우에서는 alt+f4 맥에서는 command+q를 누르면 이클립스를 종료할 수 있다. 이때 실행중인 톰캣 서버는 자동 종료된다.

종료 전에 종료 확인 창이 뜨는데, Always exit without prompt 체크 박스를 체크하고 종료하면 다음부터는 물어보지 않는다.



종료 시 Process Information 창이 뜨고 종료 프로세스를 마치면 이클립스가 정상적으로 종료된다.



지금까지 이클립스 웹 개발의 기본이 되는 설치와 톰캣의 연결, 프로젝트, JSP, 서블릿 만들기를 맛보았다. 이제 웹개발을 위한 기본 환경 설치를 마친 것이다.

2. JSP

2.1 웹 애플리케이션

JSP는 웹 애플리케이션을 개발하기 위한 서버 사이드 스크립트이다. JSP 파일 하나에 HTML, 자바스크립트, CSS와 같은 클라이언트 언어와 JSP, EL(Expression Language), 자바와 같이 서버에서 실행되는 언어를 모두 포함하기 때문에 실행 시점에 대한 혼란이 생기기 쉽다.

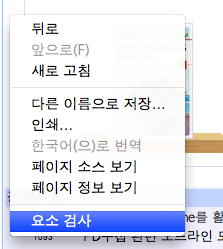
|  |  |  |
| --- | --- | --- |
| 구분 | 서버 | 클라이언트 |
| 언어 | JSP, EL, 자바 | HTML, CSS, JavaScript |
| 실행위치 | 서버 | 브라우저 |
| 디버깅 | 서버 | 브라우저 개발도구 |

웹 애플리케이션의 기본 입출력은 request와 response로 나뉜다. 클라이언트에서 request를 보내면 해당 요청에 대한 response를 서버에서 클라이언트 쪽으로 보내주는 구조이다.



request와 response에 대한 형식은 웹에서는 HTTP(Hypertext Transfer Protocol)를 따른다. 이에 대한 스펙은 <http://www.w3.org/Protocols/rfc2616/rfc2616.html> 에서 확인할 수 있다.

request와 response로 오고 간 정보는 크롬 브라우저( <http://www.google.com/chrome> )에서 쉽게 확인이 가능하다. 브라우저 내에서 컨텍스트 메뉴를 열어보면 요소 검사 메뉴를 통해서 웹인스펙터를 열 수 있다.



웹인스펙터는 파이어폭스의 파이어버그 플러그인과 유사하다. 헤더의 request, response를 보려면 Network 탭을 지정하고 페이지를 새로고침하면 좌측에 호출된 주소들이 보여지고, 해당 항목을 선택하면 Headers 탭에서 내용을 확인할 수 있다.



2.1.1 웹 헤더

웹 브라우저를 통해서 서버로 요청하는 내용들과 서버에서 보내오는 response의 내용을 보면 다음과 같다.

1. **Request URL:** http://www.okjsp.pe.kr/bbs?act=RECENT
2. **Request Method:** GET
3. **Status Code:** 200 OK
4. **Request Headers**
   1. **Accept:** application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5
   2. **Accept-Charset:** windows-949,utf-8;q=0.7,\*;q=0.3
   3. **Accept-Encoding:** gzip,deflate,sdch
   4. **Accept-Language:** ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4
   5. **Cache-Control:** max-age=0
   6. **Connection:** keep-alive
   7. **Cookie:** sid=163; okid=kenu; ACEGI\_SECURITY\_HASHED\_REMEMBER\_ME\_COOKIE=a2VudToxMzEwNz...; memo=true; okwriter=kenu; layer\_delpass=none; JSESSIONID=FEE0A8EF830854A08605CCBA3FD91BC9; passwd\_layer=block; \_\_utma=69148431.8649599.1309425115.1310535357...
   8. **Host:** www.okjsp.pe.kr
   9. **User-Agent:** Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_6\_8) AppleWebKit/534.24 (KHTML, like Gecko) RockMelt/0.9.58.494 Chrome/11.0.696.71 Safari/534.24
5. **Query String Parameters**
   1. **act:** RECENT
6. **Response Headers**
   1. **Content-Type:** text/html;charset=euc-kr
   2. **Date:** Wed, 13 Jul 2011 09:10:24 GMT
   3. **Server:** Apache-Coyote/1.1
   4. **Transfer-Encoding:** chunked

2.2 JSP

서버에 요청되는 것을 처리하는 대표적인 자바 기술이 JSP(JavaServer Pages)이다. 서블릿이 먼저 나온 기술이었지만, HTML을 쉽게 다룰 수 있다는 장점 때문에 JSP가 서블릿보다 더 많이 쓰이게 되었다. 하지만 최근에는 모델2라고 불리는 모델, 뷰, 컨트롤러 패턴을 지원하는 웹 프레임워크들이 자리를 잡으면서 순수 자바로 코딩하는 영역이 많아지고, JSP는 뷰 영역만 담당하는 쪽으로 정리되고 있다.

톰캣과 같은 서블릿 컨테이너는 JSP파일이 요청이 되면 변환된 서블릿이 있는지 확인하고 JSP와 변환된 서블릿과의 업데이트 시간을 비교해서 서블릿이 더 최신이면 그 서블릿의 실행결과를 response로 클라이언트에게 보내준다.

JSP가 변경되고 저장되면 해당 주소 요청시 JSP를 다시 서블릿으로 변환하고 컴파일해서 실행한다. 해당 주소에 해당하는 파일이 없으면 Response의 Status 값으로 404 Page Not Found 를 보내주게 된다.

2.2.1 디렉티브

JSP의 처음에는 서블릿으로 변환하기 위한 메타정보를 지정하는 지시자(directive)가 위치한다.

2.2.2 스크립틀릿

스크립틀릿은 자바코드가 실행되어 클라이언트에서 받게 되는 코드에 영향을 주는 부분이다. 스크립틀릿은 서버에서 실행되기 때문에 클라이언트에서

2.2.3 액션

2.2.4 Expression Language

3. 서블릿

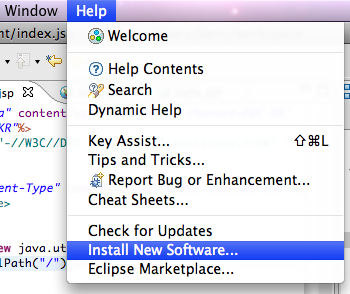
4. 이클립스 플러그인

이클립스는 플러그인을 통해서 도구의 기능을 확장할 수 있다. 플러그인은 두 가지 방식으로 설치된다. 플러그인의 URL을 통해서 다운로드 받아서 설치하는 법과 압축된 플러그인 파일을 따로 받아서 이클립스 폴더에 위치시키는 방식이다. URL을 통해서 받는 것은 이클립스의 배포 프로젝트(3.7은 Indigo, 3.6은 Helios, 3.5는 Galileo, 3.4는 Ganymede, 3.3은 Europa, 3.2는 Calisto)에서 관리하는 플러그인 주소록 그룹이 있고, 직접 입력해서 추가할 수 있는 형태로 되어 있다.

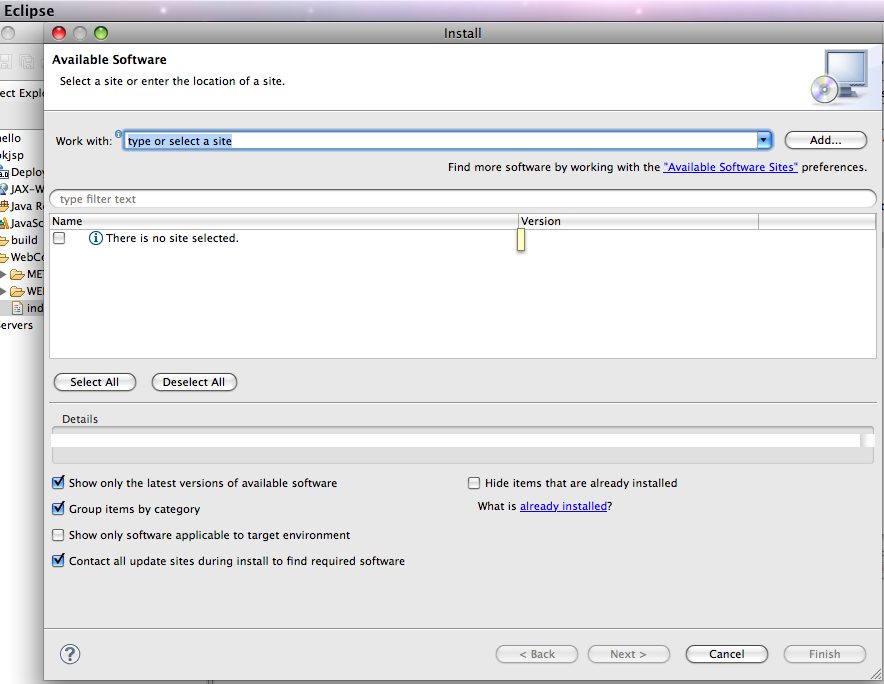
4.1 svn 플러그인 설치

버전 관리 시스템에서 요즘도 많이 사용되고 있는 것이 서브버전이다. 서브버전의 이클립스 플러그인은 크게 두 가지가 있다. 서브버시브와 서브클립스인데, 서브버시브는 이클립스의 공식 프로젝트이며 다음과 같이 설치할 수 있다.

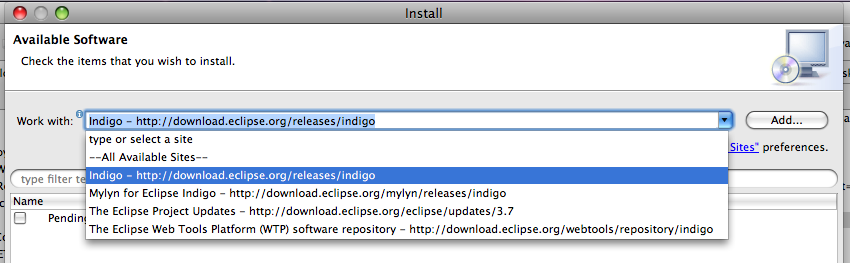
그림과 같이 메뉴의 Help > Install New Software... 를 선택한다.



Work with 항목을 보면 이클립스 버전에 따른 사이트 주소가 나오게 되는데, 여기에서 플러그인의 배포 정보를 입력하면 된다.



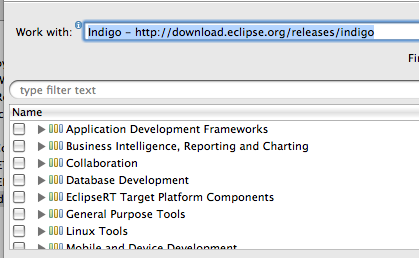
3.7인 경우는 Indigo 로 시작하는 항목을 선택하고 기다린다. 아래 목록에 Pending... 표시가 사라지고 관련 패키지들이 나온다.



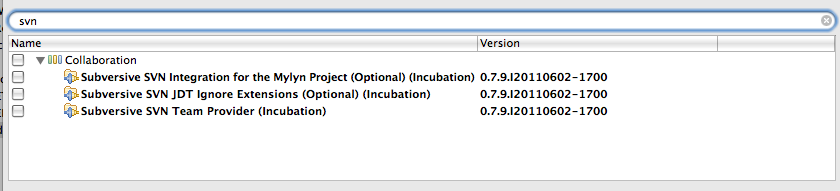
관련 정보를 받아오는 동안 이클립스 창의 우측 하단을 보면 진행상황을 표시하는 아이콘이 표시된다. 이 아이콘을 클릭하면 Progress뷰가 뜨면서 조금 더 자세한 내용을 확인할 수 있다. 우측 하단의 진행상황을 볼 수 있다면 기다림의 답답함을 조금 덜 수 있을 것이다.



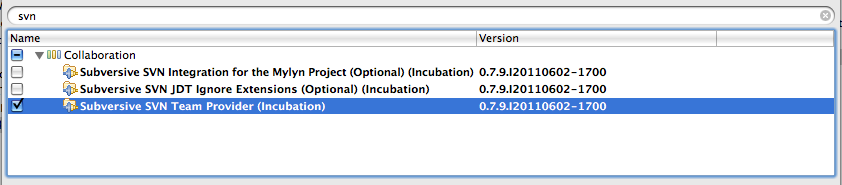
목록을 다 가져오면 아래 그림과 같이 그룹 목록들이 보이게 된다. type filter text 부분에 svn 이라고 입력한다.



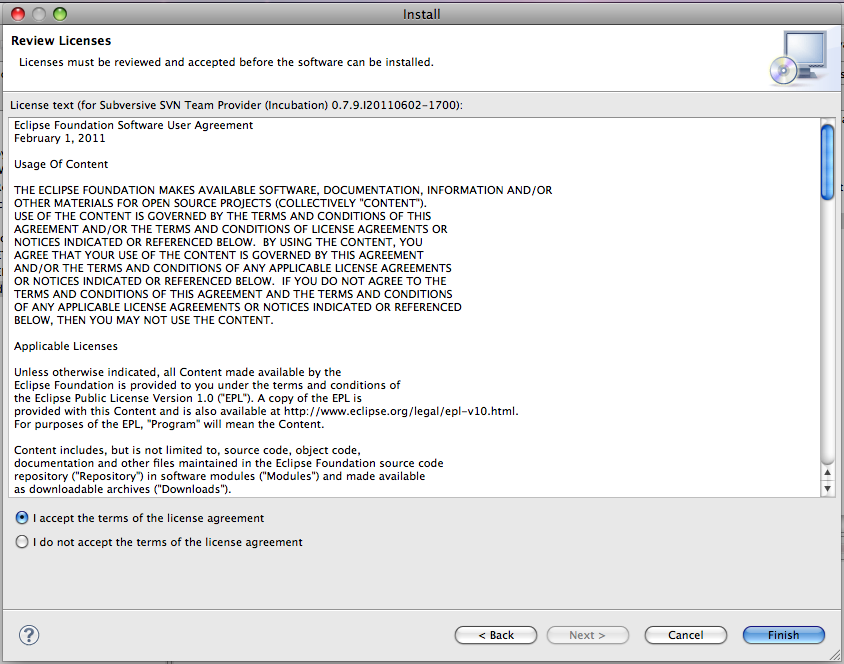
svn으로 검색한 결과가 보여지게 되는데, 검색 속도는 살짝 느린 것을 감안해야 한다.



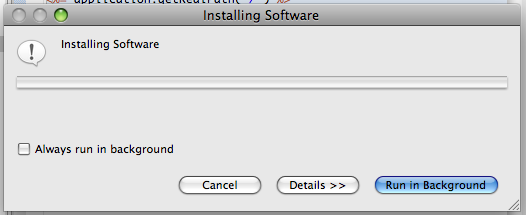
Subversive SVN Team Provider 항목만 체크하고 설치하도록 하자. SVN의 기본 기능을 다 사용할 수 있기 때문에 다른 플러그인은 필요에 따라 나중에 설치해도 된다.



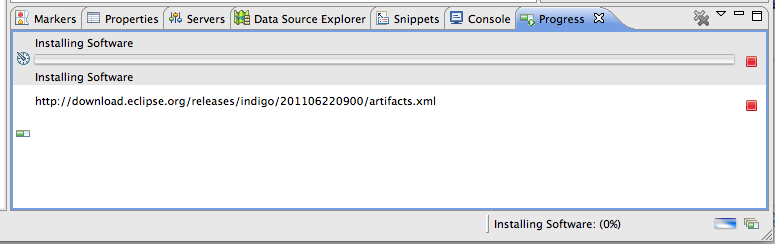
라이선스를 보여주는 창이 뜬다. I accept the terms … 항목을 체크해야 다음 과정으로 진행할 수 있다.



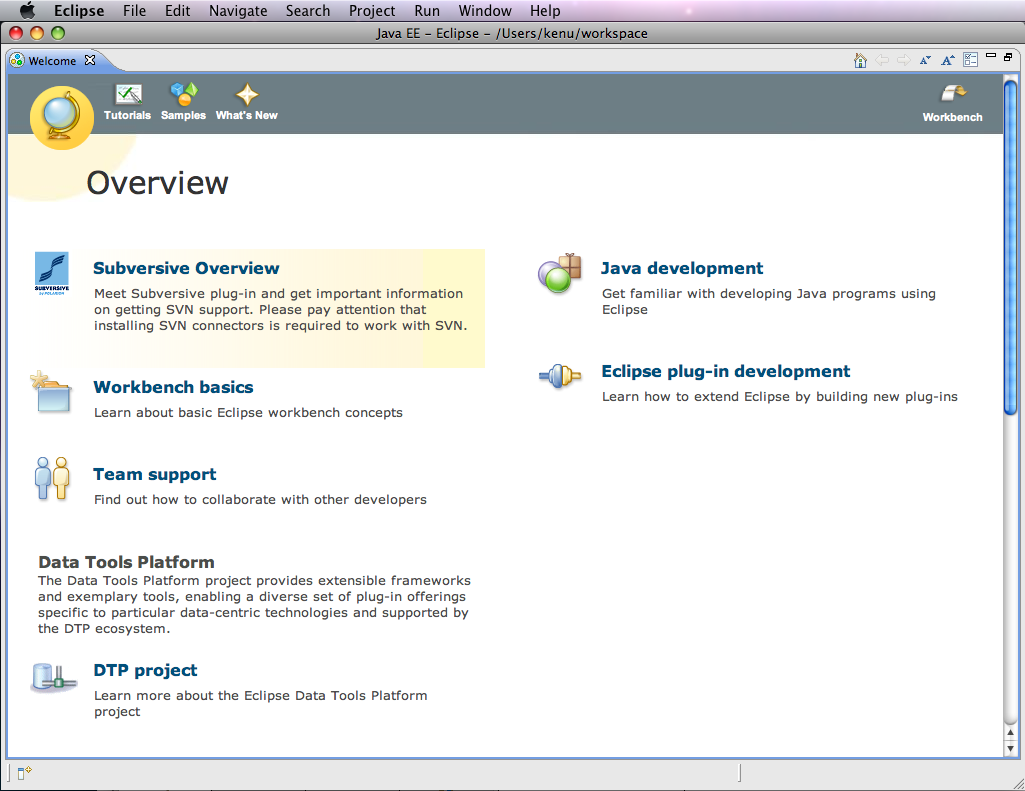
소프트웨어를 설치하는 프로그레스 창이 뜬다. Run in Background 버튼을 클릭하면 다른 작업을 하면서 설치를 진행할 수 있다.



목록을 가져올 때와 마찬가지로 하단의 Progress 뷰를 통해서 진행상황을 확인할 수 있다.



설치를 완료하면 이클립스 재시작을 요청하는 창이 나타나고, 다시 시작하면 Subversive Overview가 있는 Welcome 창이 뜬다. 물론 Welcome탭의 x를 누르고 작업 화면으로 다시 들어갈 수 있다.



SVN 프로젝트 가져오기

원격 서버 접속하기