<http://kjunine.tistory.com/35>

**메이븐으로 자바 프로젝트를 관리해 보자.**

이전부터 자바 프로젝트를 시작할 때 제일 먼저하는 것은 이클립스를 띄우고, 자바 프로젝트를 만들고 라이브러리들을 lib 폴더에 추가하는 것이었다. 그런데 비슷비슷한 라이브러리들을 똑같이 사용하는 자바 프로젝트를 여러 개 만들다보면 비슷한 노가다를 계속해야 했다. 그러다가 조금 더 발전한 것이 여러 프로젝트에서 공통으로 사용하는 라이브러리 프로젝트를 만들어서 참조하게 한 것이다. 조금 편해졌지만 그래도 각 프로젝트마다 클래스패스를 잡아주는 것은 고역이었다.

그러다 [**메이븐**](http://maven.apache.org/)이라는 것을 알게 되었다. 메이븐에서 라이브러리 혹은 프로젝트 의존 관계(Dependency)를 설정하는 것은 매우 간단하다. 설정 파일에 그 의존 관계를 선언하기만 하면 된다. 이클립스와의 연동도 가능하다. 메이븐에서 이클립스 프로젝트를 만들 수도 있고, 이클립스에서 메이븐 플러그인을 설치하여 메이븐 프로젝트를 만들 수도 있다.

메이븐은 사실 의존 관계 관리 도구라기보다는 **프로젝트 관리 도구**이다. 메이븐은 빌드 생명주기(Build Lifecycle)에 따라 프로젝트 표준을 제공하고 의존 관계를 관리하고 플러그인이 제공하는 부가 기능을 사용할 수 있게 하는 도구이다. 더 자세한 것은 [메이븐 홈페이지](http://maven.apache.org/)나 Sonatype에서 공개한**메이븐 책(**[**Maven: The Definitive Guide**](http://www.sonatype.com/products/maven/documentation/book-defguide)**)**를 참조하길 바란다. 메이븐 책은 무료로 다운로드 받을 수 있고, 영어임에도 불구하고 매우 쉽게 쓰여져 읽는 데 큰 어려움은 없다. Thanks, Sonatype. 메이븐 책을 3장까지만 읽어보면 감을 잡을 것이다. 사실 메이븐 책 3장까지 봤으면 이 포스트는 볼 필요도 없다. ㅎ\_ㅎ (그래도 이건 한글이다. +\_+)

**일단 따라해보자.**

애초에 라이브러리 관리를 더 편하게 하기 위해 메이븐을 사용하려고 했던 것이니 다른 라이브러리를 사용하는 프로젝트를 만들어 보자. 단위 테스트를 위한 **JUnit** 라이브러리와 자바 API를 보완하는 여러 기능들을 제공하는**Commons Lang** 라이브러리 정도만 사용해 보자.

**1. 메이븐을 설치한다.**

[메이븐 홈페이지](http://maven.apache.org/)에서 메이븐을 [다운로드](http://maven.apache.org/download.html)하고 적당한 폴더에 압축을 푼 뒤, 메이븐 폴더 내 bin 폴더를 PATH에 추가해준다. **mvn --help** 명령어를 쳤을 때 사용법이 제대로 나온다면 다음을 진행할 수 있다. mvn은 메이븐을 실행하는 명령어이다.

**2. 메이븐 프로젝트를 만든다.**

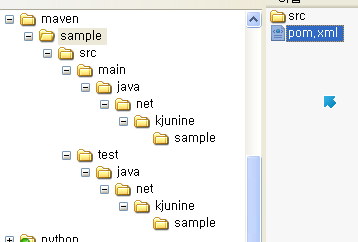
작업할 폴더에서 다음 명령어를 실행한다. 도중에 옵션을 물어보기도 하는데 그냥 엔터를 쳐서 기본값으로 넘어갈 수 있다.

mvn archetype:generate -DgroupId=net.kjunine -DartifactId=sample -Dpackage=net.kjunine.sample -Dversion=1.0-SNAPSHOT

위 명령어에서 archetype:generate는 archetype 플러그인의 generate 골(Goal)이라는 것인데, 플러그인과 골은 지금은 몰라도 좋다. 그냥 프로젝트를 생성하는 옵션이라고 생각하면 된다. -D로 시작하는 것들은 생성할 프로젝트에 대한 정보를 나타내는 파라메터들이다. 각 파라메터는 다음을 의미한다.

* groupId
  + 프로젝트를 만드는 그룹, 조직, 회사 등을 나타내는 유일한 이름, 보통 자바 패키지처럼 URI를 거꾸로 써서 나타낸다.
* artifactId
  + 아티팩트(artifact) 즉, 프로젝트를 나타내는 유일한 이름, 그룹 내 다른 아티팩트와 이름이 같아서는 안된다.
* package
  + 프로젝트의 최상위 패키지
* version
  + 프로젝트의 현재 버전

프로젝트 생성이 성공하면, 명령어를 수행한 폴더 내에 artifactId 이름과 동일한 폴더가 생긴다. 그 폴더로 들어가면 다음과 같이 src 폴더와 **pom.xml** 파일을 볼 수 있다.



pom.xml 파일은 바로 **Project Object Model(POM)**이라는 것으로 메이븐 프로젝트에 대한 모든 설정을 저장하는 파일이다. 메이븐 프로젝트의 모든 것이다. 그 파일을 열어보면 다음과 같을 것이다. 앞서 프로젝트 생성 시 설정한 프로젝트 옵션 값들이 저장되어 있는 것을 볼 수 있다.

01.<project xmlns="<http://maven.apache.org/POM/4.0.0>"xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

02.xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0><http://maven.apache.org/maven-v4_0_0.xsd>">

03.<modelVersion>4.0.0</modelVersion>

04.<groupId>net.kjunine</groupId>

05.<artifactId>sample</artifactId>

06.<packaging>jar</packaging>

07.<version>1.0-SNAPSHOT</version>

08.<name>sample</name>

09.<url>http://maven.apache.org<;/url>

10.<dependencies>

11.<dependency>

12.<groupId>junit</groupId>

13.<artifactId>junit</artifactId>

14.<version>3.8.1</version>

15.<scope>test</scope>

16.</dependency>

17.</dependencies>

18.</project>

**3. 라이브러리 의존 관계를 설정한다.**

POM에서 지금 관심을 가져야 할 부분은 바로 **<dependencies /> 부분**이다. 여기에서 프로젝트의 의존 관계를 설정할 수 있다. 현재 JUnit 라이브러리는 자동으로 추가되어 있는 것을 볼 수 있다. 여기에 Commons Lang 라이브러리도 추가하자. <dependencies /> 부분을 다음과 같이 바꾸면 된다. <dependency /> 항목을 하나 추가하는 것으로 라이브러리를 추가할 수 있다. 정말 편하지 않은가?! +\_+

01.<dependencies>

02.<dependency>

03.<groupId>commons-lang</groupId>

04.<artifactId>commons-lang</artifactId>

05.<version>2.4</version>

06.</dependency>

07.<dependency>

08.<groupId>junit</groupId>

09.<artifactId>junit</artifactId>

10.<version>3.8.1</version>

11.<scope>test</scope>

12.</dependency>

13.</dependencies>

해당 라이브러리의 groupId와 artifactId는 어떻게 알 수 있는지 궁금할 것이다. 의외로 간단하다. [Maven Repository 홈페이지](http://mvnrepository.com/)에서 검색하면 된다. 다만 가끔 비슷한 게 많이 나와서 어떤 걸 써야할지 모를 때도 있다. (예를 들면, springframework) 그럴 때는 해당 라이브러리 홈페이지에서 정확한 메이븐 groupId와 artifactId를 찾아보던가 구글링할 수 밖에 없는 것 같다. 아직 내공이 부족하다. ㅠ\_ㅠ

**4. 자바 소스와 테스트를 작성한다.**

프로젝트 폴더 내 src 폴더에 보면 **main** 폴더와 **test** 폴더가 있는데 각각 자바 소스 파일과 테스트 파일들이 존재하는 곳이다. 메이븐은 이렇게 정해진 폴더 구조와 이름을 강제로 사용하게 해서 프로그래머의 고민을 덜어준다. 하지만 바꿀 수도 있는 것 같다.

각 소스 폴더와 테스트 폴더에는 프로젝트 생성 시에 설정한 패키지 이름대로 폴더 구조가 만들어져 있고, 각각 App.java 파일과 AppTest.java 파일이 존재한다. 두 파일을 사용할 것은 아니니 삭제하자.

그리고 소스 폴더의 해당 패키지에 다음 클래스 파일를 작성하자.

**Sample.java**

01.package net.kjunine.sample;

02.

03.import org.apache.commons.lang.StringUtils;

04.

05.public class Sample {

06.

07.public static String reverse(String s) {

08.return StringUtils.reverse(s);

09.}

10.

11.public static void main(String[] args) {

12.for (int i = 0; i < args.length; i++) {

13.System.out.println(reverse(args[i]));

14.}

15.}

16.

17.}

Commons Lang 라이브러리의 StringUtils 유틸 클래스가 제공하는 reverse 메소드를 사용하여 입력받은 문자열들을 거꾸로 출력하는 프로그램을 만들었다.

테스트 폴더의 해당 패키지에 위 클래스에 대한 테스트 클래스 파일도 작성하자.

**SampleTest.java**

01.package net.kjunine.sample;

02.

03.import junit.framework.TestCase;

04.

05.public class SampleTest extends TestCase {

06.

07.public void testReverse() {

08.String actual = Sample.reverse("hello");

09.String expected = "olleh";

10.assertEquals(expected, actual);

11.}

12.

13.}

JUnit 라이브러리를 사용한 간단한 테스트 클래스이다.

**5. 프로젝트를 빌드한다.**

프로젝트를 빌드하는 것은 간단하다. 프로젝트 폴더에서 다음 명령어를 실행하면 된다.

 mvn install

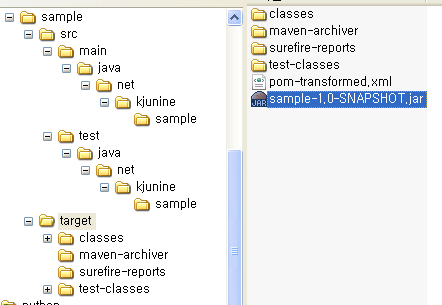
이 명령어를 이해하려면 메이븐에서 말하는 **빌드 생명주기(Build Lifecycle)**라는 개념을 알아야 한다. 빌드 생명주기는 일련의 프로젝트 **빌드 단계(Build Phase)**들의 (순서있는) 집합이다. 빌드 단계는 compile, test, package, deploy 등과 같은 프로젝트를 빌드하는데 거치는 단계들을 의미한다. 그리고 각 빌드 단계마다 무슨 일을 해야하는지가 정해져 있다. 허접한 설명이라 더 자세한 내용은 [여기](http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html)를 참조하자. (메이븐 책을 봐도 좋다.)

위 명령어는 생명주기에서 install 단계까지의 전 단계를 실행하라는 것으로 compile, test, package, install 등의 단계들을 포함한다.

그래서 위 명령어를 실행하면 중간에 다음과 같이 테스트 결과가 나온다. 그리고 빌드가 성공하면 "BUILD SUCCESSFUL"이 보일 것이다.

-------------------------------------------------------  
 T E S T S  
-------------------------------------------------------  
Running net.kjunine.sample.SampleTest  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.031 sec  
  
Results :  
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
  
[INFO] [jar:jar]  
[INFO] Building jar: D:\dev\workspaces\maven\sample\target\sample-1.0-SNAPSHOT.jar  
[INFO] [install:install]  
[INFO] Installing D:\dev\workspaces\maven\sample\target\sample-1.0-SNAPSHOT.jar to ...  
[INFO] ------------------------------------------------------------------------  
[INFO] BUILD SUCCESSFUL  
[INFO] ------------------------------------------------------------------------

또, 프로젝트 폴더 내 **target** 폴더에 생성된 클래스 파일들과 테스트 파일들 등이 들어있는 폴더들과 함께 jar 형태로 묶인 프로젝트 패키지 파일이 보인다.



**6. 프로그램을 실행한다.**

테스트도 성공하고 패키지 파일도 생성했으니 이제 실행만 하면 된다. 다음 명령어를 실행하라.

java -cp target/sample-1.0-SNAPSHOT.jar;"%HOME%/.m2/repository/commons-lang/commons-lang/2.4/commons-lang-2.4.jar" net.kjunine.sample.Sample hello kjunine

생각보다 실행 방법이 까다로운데 이것은 아직 플러그인을 배우지 않았기 때문이니 지금은 그냥 참고 넘어가자.**%HOME%/.m2/repository** 폴더는 메이븐이 라이브러리와 플러그인을 다운로드해서 저장하는 로컬 저장 폴더이다. 그리고 Commons Lang 라이브러리는 위와 같은 경로에 존재한다. 실행 시에 Commons Lang 라이브러리를 사용하기 때문에 클래스패스에 추가한 것이다.

실행 결과는 다음과 같다.

olleh  
eninujk

굳~

**이제 겨우 걸음마를 시작했을 뿐..**

아직 메이븐에 대해 공부할 것은 많이 남아 있다. 플러그인과 골의 개념을 이해하고 POM에 설정할 수 있어야 한다. 그리고 이클립스에서 개발하기 위해 메이븐 이클립스 플러그인(maven-eclipse-plugin)을 사용하거나, 아니면 m2eclipse, Q4E와 같은 이클립스 플러그인을 사용할 수 있어야 한다. 실제 프로젝트를 진행할 때는 지금처럼 명령어에서 실행하기보다는 이클립스 플러그인을 사용하는 것이 훨씬 쉽고 간편할 것이다.

방금 말한 내용들은 다음 포스트들에서 다룰 것이다. 이미 제목은 써놓았다. 다만 언제 완성될지는.. ^^;

 참조:

[Maven](http://maven.apache.org/index.html)  
[Maven - Getting Started Guide](http://maven.apache.org/guides/getting-started/index.html)  
[Maven - Introduction to the Build Lifecycle](http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html)  
[Maven: The Definitive Guide](http://www.sonatype.com/products/maven/documentation/book-defguide) (Chapter 3. A Simple Maven Project)

[지난 번 포스트](http://kjunine.net/entry/getting-started-with-maven)에서 메이븐으로 자바 프로젝트의 라이브러리를 관리하는 법을 살펴보았다. 이번 포스트에서 살짝 더 나아가서 메이븐의 꽃이라고 할 수 있는 플러그인에 대해서 알아보자.

**메이븐 플러그인은 메이븐의 핵심이다.**

일단 개념부터 짚고 넘어가자. 알아야하는 개념은 플러그인(Plugin)과 골(Goal)이다.

플러그인은 메이븐의 핵심 구성 요소이다. 다시 말하면 메이븐은 플러그인들의 집합이라고 할 수 있다. 참고로 [메이븐 홈페이지의 플러그인 페이지](http://maven.apache.org/plugins/index.html)에서 사용 가능한 플러그인들의 목록을 볼 수 있다. 그리고 플러그인은 관련있는 골들의 집합이다. 골은 메이븐에서 작업의 단위이라고 할 수 있다. 앤트(Ant)를 안다면 앤트의 태스크(task)와 유사한 것으로 생각해도 될 것이다. 메이븐 생명주기의 관점에서 보면 빌드 단계(Build Phase)에는 그 단계가 실행하는 골이 정해져있다. 예를 들면, compile 빌드 단계는 compiler 플러그인의 compile 골을 실행하고, package 빌드 단계는 jar 플러그인의 jar 골을 실행한다.

정리하면 메이븐에는 무수한 플러그인들이 있고, 각 플러그인들은 관련된 골들을 묶어 놓은 것이다. 그리고 각 골들은 메이븐 빌드 단계에서 수행되는 하나의 작업인 것이다. 더 자세한 설명은 역시 [메이븐 홈페이지](http://maven.apache.org/)나 [메이븐 책(Maven: The Definitive Guide)](http://www.sonatype.com/products/maven/documentation/book-defguide)를 참고하자.

**메이븐 플러그인을 사용해 보자.**

사실 [지난 번 포스트](http://kjunine.net/entry/getting-started-with-maven)에서 이미 메이븐 플러그인을 사용해 보았다.

mvn **archetype:generate** -DgroupId=net.kjunine -DartifactId=sample -Dpackage=net.kjunine.sample -Dversion=1.0-SNAPSHOP

위 명령어를 기억하는가? 위 명령어에서 'archetype:generate'는 archetype 플러그인의 generate 골을 실행하라는 메이븐 명령어이다. 참고로 archetype 플러그인은 자바 프로젝트를 생성하는 것과 관련있는 골들의 모음이다. 이 플러그인에 대해 자세한 설명은 [여기](http://maven.apache.org/plugins/maven-archetype-plugin/)를 참고하자. 여기서 보여주고 싶은 것은 플러그인의 골을 실행하는 방법이다. mvn 명령어에 플러그인:골 형식의 옵션과 -D를 붙인 key=value 형식의 파라메터들을 덧붙여서 실행하면 된다.

그리고 [지난 번 포스트](http://kjunine.net/entry/getting-started-with-maven)에서 완성된 프로젝트를 실행하는 방법이 좀 까다로웠던 것을 기억하는가?

java -cp target/sample-1.0-SNAPSHOT.jar;"%HOME%/.m2/repository/commons-lang/commons-lang/2.4/commons-lang-2.4.jar" net.kjunine.sample.Sample hello kjunine

위와 같이 클래스패스를 복잡하게 설정하면서 실행했었는데 이것을 **exec 플러그인**을 사용하면 간단하게 실행시킬 수 있다. exec 플러그인의 상세한 설명은 [여기](http://mojo.codehaus.org/exec-maven-plugin/)를 참조하자.

mvn **exec:java** -Dexec.mainClass=net.kjunine.sample.Sample -Dexec.args="hello kjunine"

exec 플러그인이 POM 파일을 참조해 라이브러리와 프로젝트의 의존 관계를 파악해서 알아서 클래스패스를 설정해주므로 명령어가 매우 간단해졌다. 결과는 다음과 같이 나온다.

[INFO] Scanning for projects...  
[INFO] Searching repository for plugin with prefix: 'exec'.  
[INFO] ------------------------------------------------------------------------  
[INFO] Building sample  
[INFO]    task-segment: [exec:java]  
[INFO] ------------------------------------------------------------------------  
[INFO] Preparing exec:java  
[INFO] No goals needed for project - skipping  
[INFO] [exec:java]  
**olleh  
eninujk**[INFO] ------------------------------------------------------------------------  
[INFO] BUILD SUCCESSFUL  
[INFO] ------------------------------------------------------------------------  
[INFO] Total time: < 1 second  
[INFO] Finished at: Fri Oct 09 21:51:55 KST 2009  
[INFO] Final Memory: 5M/9M  
[INFO] ------------------------------------------------------------------------

플러그인을 실행하는 방법을 알아봤으니 이번에는 POM 파일에서 플러그인의 설정을 하는 방법을 알아보자.

메이븐으로 자바 프로젝트를 빌드할 때 compiler 플러그인의 설정을 바꿔주지 않으면 기본적으로 JDK 1.4 버전을 기준으로 컴파일한다. 그래서 지난 번 포스트에서는 JUnit 라이브러리의 버전이 3.8.2였던 것이다. 이번 기회에 JDK 버전과 JUnit 버전을 각각 1.5와 4.5로 올리고, @Test 어노테이션(Annotation)을 사용해 보자.

**1. POM 파일(pom.xml)을 수정하자.**

01.<project xmlns="<http://maven.apache.org/POM/4.0.0>"xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

02.xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0><http://maven.apache.org/maven-v4_0_0.xsd>">

03.<modelVersion>4.0.0</modelVersion>

04.<groupId>net.kjunine</groupId>

05.<artifactId>sample</artifactId>

06.<packaging>jar</packaging>

07.<version>2.0-SNAPSHOT</version>

08.<name>sample</name>

09.<url>http://maven.apache.org<;/url>

10.<dependencies>

11.<dependency>

12.<groupId>commons-lang</groupId>

13.<artifactId>commons-lang</artifactId>

14.<version>2.4</version>

15.</dependency>

16.<dependency>

17.<groupId>junit</groupId>

18.<artifactId>junit</artifactId>

19.<version>4.5</version>

20.<scope>test</scope>

21.</dependency>

22.</dependencies>

23.<build>

24.<plugins>

25.<plugin>

26.<groupId>org.apache.maven.plugins</groupId>

27.<artifactId>maven-compiler-plugin</artifactId>

28.<configuration>

29.<source>1.5</source>

30.<target>1.5</target>

31.</configuration>

32.</plugin>

33.</plugins>

34.</build>

35.</project>

주목할 부분은 세 부분이다. 첫째, 프로젝트의 version 을 '2.0-SNAPSHOT'으로 올렸다. 둘째, junit 라이브러리의 버전을 4.5로 올렸다. 셋째, <build /> 부분에 그 안에 compiler 플러그인의 설정을 추가하였다. 여기서 플러그인을 설정하는 방법을 잘 봐두자. 다른 플러그인도 마찬가지 방법으로 설정할 수 있다. compiler 플러그인에 대한 상세한 설명은 [여기](http://maven.apache.org/plugins/maven-compiler-plugin/)를 참조하자.

**2. SampleTest 클래스를 수정하자.**

01.package net.kjunine.sample;

02.

03.import static org.junit.Assert.\*;

04.

05.import org.junit.Test;

06.

07.public class SampleTest {

08.

09.@Test

10.public void testReverse() {

11.String actual = Sample.reverse("hello");

12.String expected = "olleh";

13.assertEquals(expected, actual);

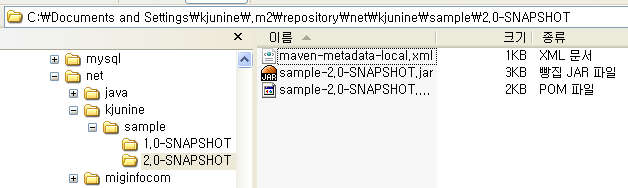
14.}

15.

16.}

JUnit 4.5의 @Test 어노테이션을 사용하도록 수정하였다.

**mvn test** 명령어를 실행하면 빌드와 테스트가 성공하는 것을 볼 수 있다. 그리고 **mvn install** 명령어를 실행하면 설치가 성공하는 것을 볼 수 있고, 로컬 리포지토리에 이전 버전인 1.0-SNAPSHOT 버전과 최신 버전인 2.0-SNAPSHOT 버전이 모두 설치되어 있는 것을 볼 수 있다.



**메이븐 플러그인을 맛보다..**

이 포스트에서 메이븐 플러그인을 살펴보았다. 그러나 여기서 살펴본 것은 거의 맛만 본 수준이다. 메이븐에는 무수한 플러그인이 있고, 또 직접 플러그인을 만들 수도 있다. 메이븐을 제대로 알려면 최대한 많은 프러그인들을 사용해보고 또, 직접 플러그인을 만들어 보는 것이 좋을 것이다.

포스트를 쓰다보니 길어져서 또 나누게 되었다. 다음 포스트에서는 메이븐과 이클립스의 연동을 주제로 '메이븐 시작하기' 시리즈를 마무리하고자 한다. 그럼 다음 포스트에서..

**이클립스에서 메이븐 프로젝트를 해보자.**

개발을 하는데 이클립스와 같은 IDE를 사용할 수 없다면 생산성이 급격히 낮아질 것이다. 메이븐 프로젝트 역시 마찬가지다. 메이븐이 편리한 도구라서 쓰고 싶은데 IDE를 지원하지 않는다면 무용지물일 것이다. 하지만 메이븐은 좋은 도구답게 이클립스를 비롯한 다양한 IDE를 지원한다. 여기에서도 플러그인이 사용되는데, 조금 재밌는 것은 메이븐에도 이클립스 플러그인이 있고, 이클립스에도 메이븐 플러그인이 있다는 것이다. 두 가지의 차이는 프로젝트를 어디에서 시작을 하는 것이냐는 것이다. **메이븐 이클립스 플러그인**은 메이븐에서 먼저 시작하고, **이클립스 메이븐 플러그인**은 이클립스에서 먼저 시작한다. 두 가지 모두 사용 가능하지만 결국에는 이클립스의 메이븐 플러그인을 많이 사용하게 될 것이다. 명령어를 치는 것보다 IDE에서 클릭 몇번으로 프로젝트를 생성하고 관리하는 것이 편리하니까..

먼저 메이븐 이클립스 플러그인을 사용하여 보자.

**1. 프로젝트를 생성한다.**

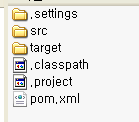
지금은 이전에 만들어 놓았던 프로젝트를 그냥 쓰기로 하자. ^-^

**2. mvn eclipse:eclipse**

프로젝트 폴더에서 eclipse 플러그인의 eclipse 골을 실행하는 명령어를 실행한다.

 mvn eclipse:eclipse

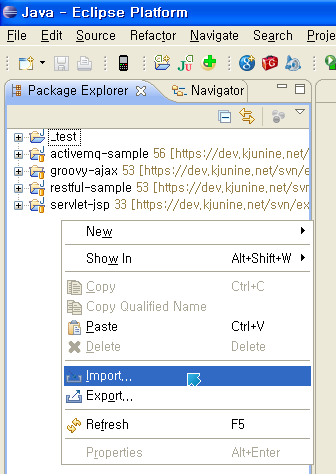
'BUILD SUCCESSFUL' 이 보이면서 완료되면 다음과 같은 이클립스 프로젝트 관련 파일들이 생긴다.

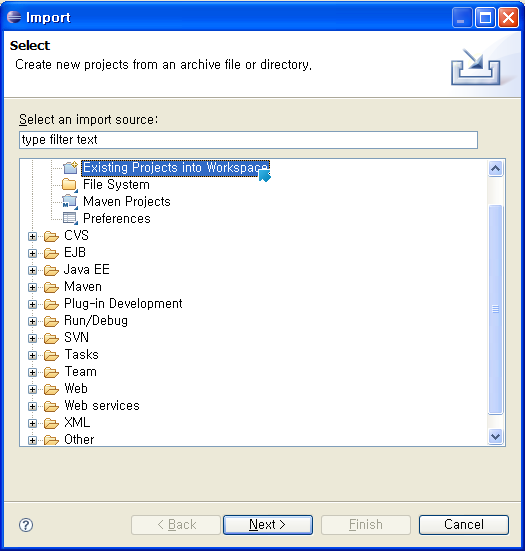


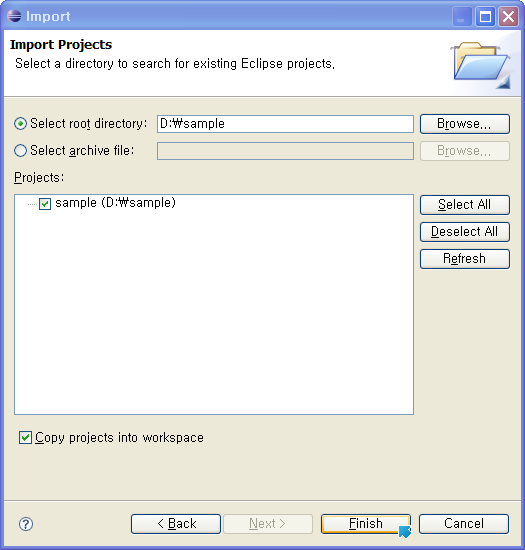
* .project: 프로젝트 설정 파일
* .classpath: 클래스패스 설정 파일
* .settings: 기타 설정 파일들을 포함하는 폴더

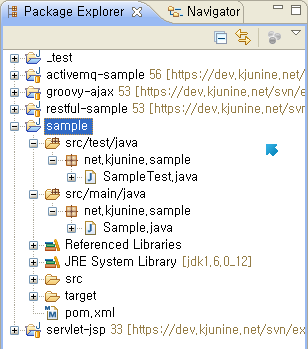
3. Import Project

이클립스에서 Import Project.. 를 통해 프로젝트를 워크스페이스(Workspace) 내부로 가져온다.









'Existing Projects into Workspace'를 선택하고 메이븐 프로젝트가 있는 경로를 선택해주면 된다.

메이븐으로 이미 만든 프로젝트는 위와 같이 메이븐의 이클립스 플러그인으로 간편하게 이클립스 프로젝트로 만들면 된다.

**이클립스에서 전부 다 하자.**

이제는 처음부터 이클립스에서 메이븐 프로젝트를 만드는 방법을 알아보자. 그러기 위해서는 이클립스의 메이븐 플러그인을 이클립스에 설치해야 한다. 메이븐 플러그인으로는 [**M2Eclipse**](http://m2eclipse.sonatype.org/) 나 [**Q4E**](http://code.google.com/p/q4e/) 등이 있다. 둘 중에 아무거나 사용하면 될 것 같다. 나는 이전에 m2eclipse를 설치했는데 요즘엔 q4e가 더 좋다는 것 같기도.. @\_@

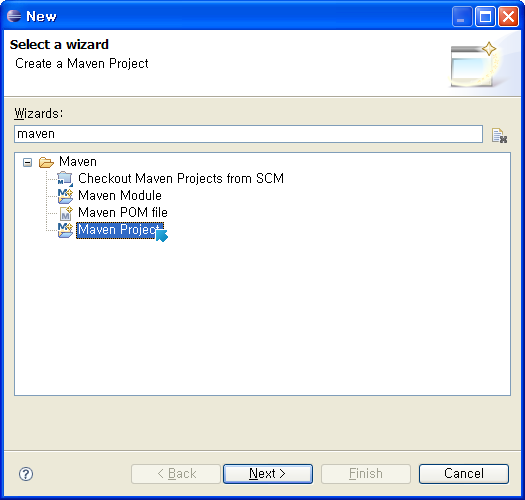
이클립스의 Software Update에서 다음 사이트(Site)를 추가하고 설치하면 된다.

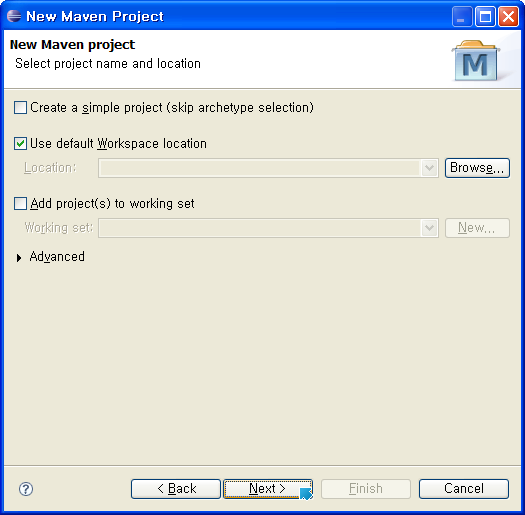
* m2eclipse
  + <http://m2eclipse.sonatype.org/update/>
* q4e
  + <http://q4e.googlecode.com/svn/trunk/updatesite-iam/>

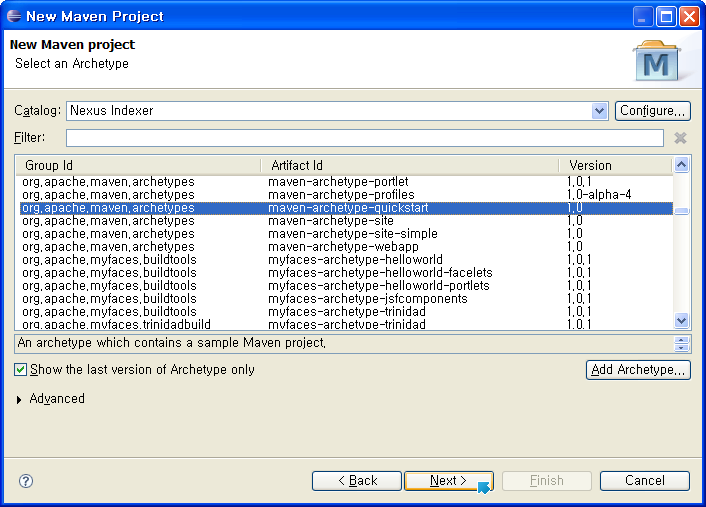
메이븐 플러그인을 설치한 후에 다음을 진행하도록 하자.

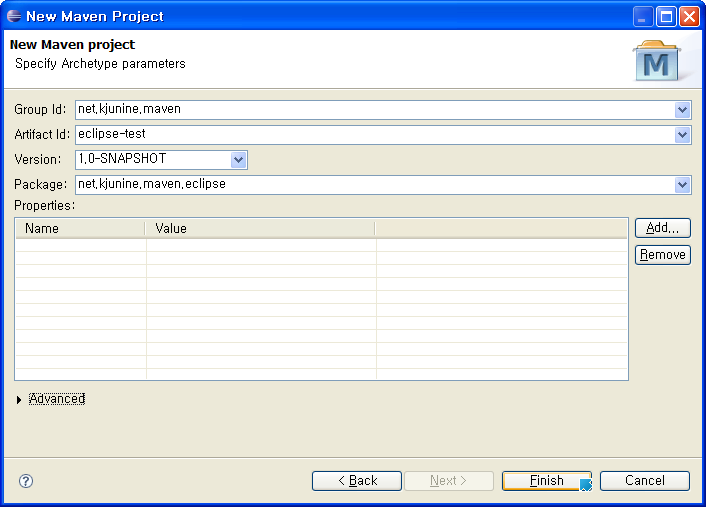
**1. 프로젝트를 생성한다.**

이클립스에서 다음과 같이 프로젝트를 생성한다. 옵션은 필요에 따라 바꾸면 될 것이다.

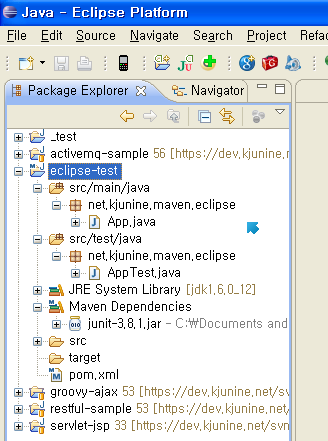








다음과 같이 프로젝트가 생성된다.



자동으로 App.java 파일과 AppTest.java 파일이 생성되었고, JUnit 3.8.1과의 의존성에 따라 프로젝트 클래스패스도 만들어졌다.

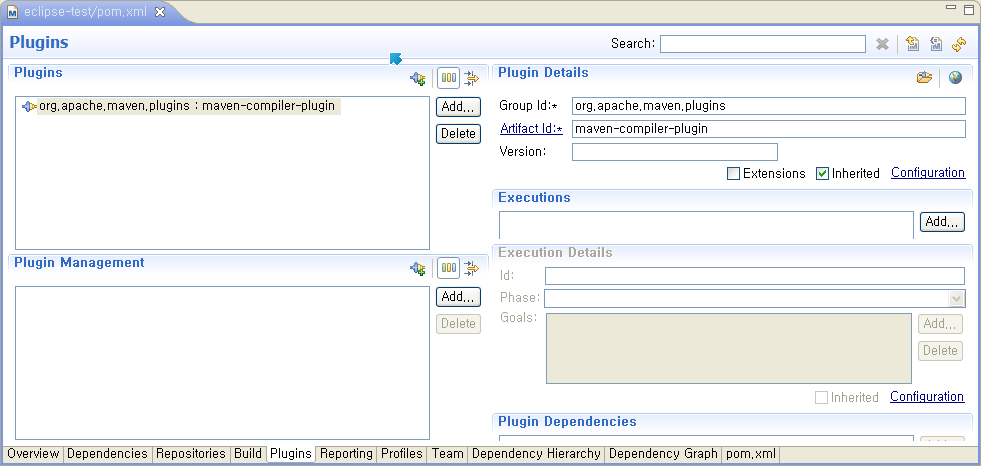
자동으로 만들어진 App.java, AppTest.java 파일들은 가뿐하게 지워주자.

**2. 자바 버전을 5로 올리자.**

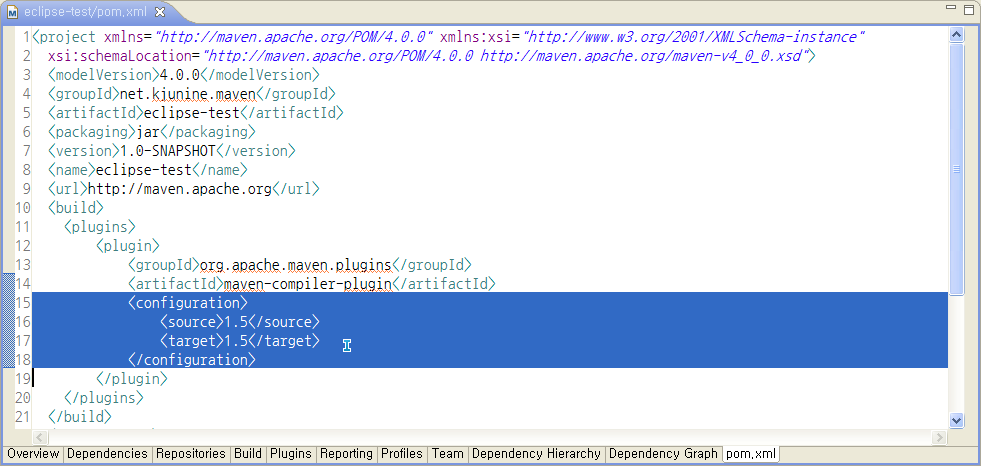
자바 1.4보다는 자바 5 이상의 버전이 좋다. 메이븐 프로젝트를 처음 생성하면 기본적으로 자바 1.4에 맞춰져 있기 때문에 바꿔주어야 한다.

이전에 했던 대로 POM 파일에서 maven-compiler-plugin 설정을 추가해주면 된다. 직접 파일을 수정할 수도 있겠지만 이클립스의 도움을 받아보자.

이클립스에서 pom.xml 파일을 열면 자동으로 **Maven POM editor**로 열린다. 에디터 하단에서 **plugin 탭**을 선택하자. Plugins 부분에서 **Add.. 버튼**을 클릭하고, Plugin Details 부분에서 다음 그림과 같이 내용을 채워준다.



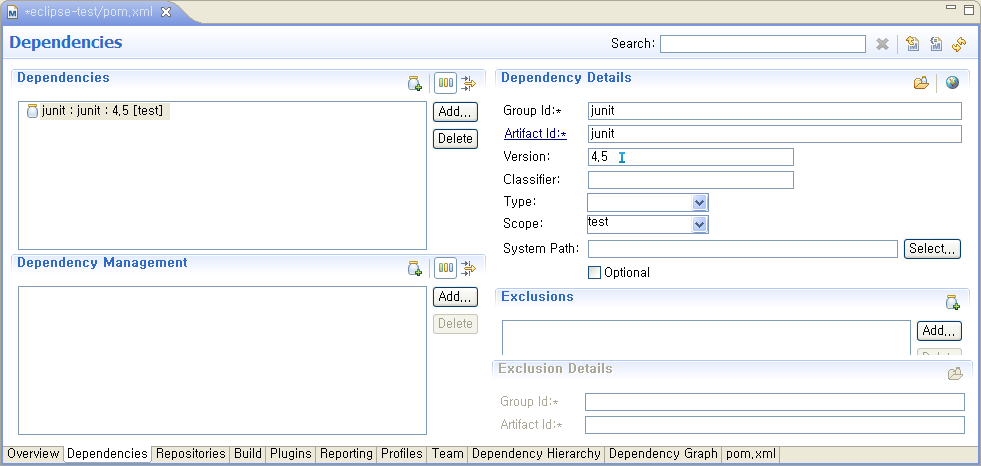
그리고 Executions 위에 있는 **Configuration 링크**를 클릭하여 다음 그림과 같이 나머지 설정을 넣어준다.



플러그인을 설정하는 것은 각 플러그인마다 설정하는 항목이 다르므로 결국 수동으로 작성해야 하는 불편함이 있는 것 같다.

이번엔 **Dependencies 탭**으로 가보자. JUnit 3.8.1이 라이브러리로 등록되어 있는데 자바 5로 올렸으니 거기에 맞게 JUnit도 4.5로 올려주자.

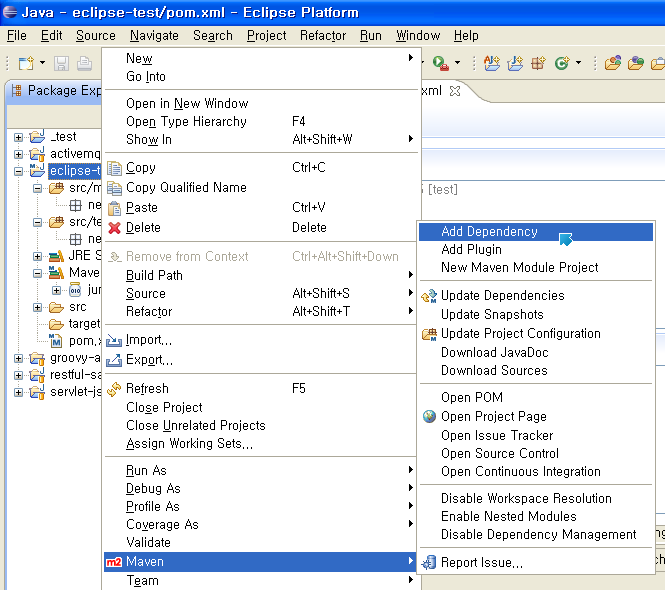
Dependencies 부분에서 junit을 선택하고 Dependency Details 부분에서 버전을 4.5로 바꿔준다.



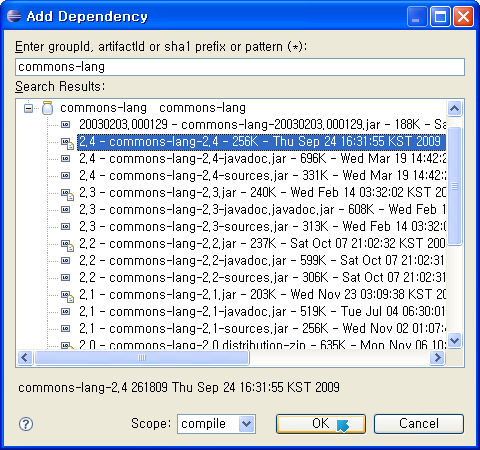
**3. 다른 라이브러리를 더 추가하자.**

여기서는 역시 Commons Lang 라이브러리를 사용해보자. POM 에디터의 dependencies 탭에서 직접 추가할 수도 있겠지만.. 조금 더 편하게 라이브러리를 검색해서 추가하는 방법도 있다.

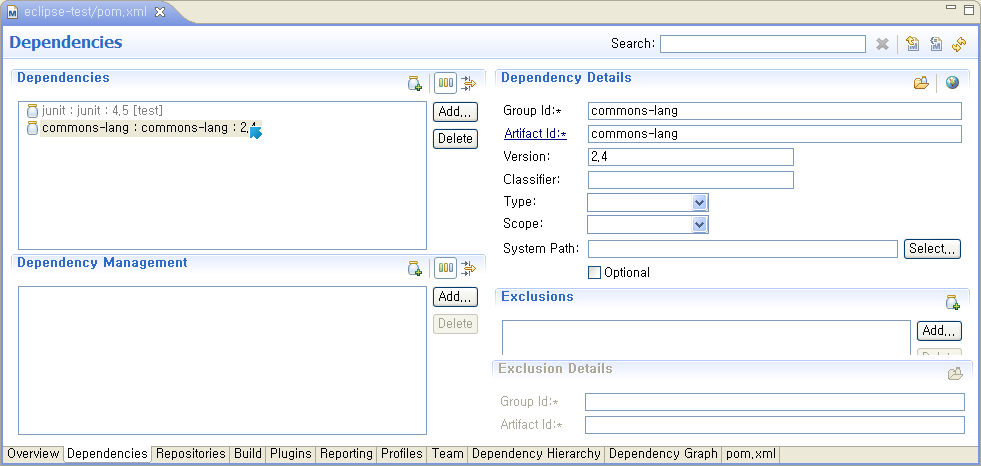
프로젝트에서 마우스 오른쪽 클릭 후 나오는 메뉴에서 Maven > Add Dependency 를 선택한다.



다음처럼 나오는 창에서 라이브러리를 검색하고, 원하는 라이브러리가 보이면 선택한다. 원하는 버전을 골라서 선택할 수 있다.



다음과 같이 commons-lang 라이브러리가 추가된 것을 볼 수 있다.



필요한 라이브러리를 모두 추가했으면 개발을 시작하자.

**4. 소스와 테스트 파일을 추가한다.**

**Sample.java**

01.package net.kjunine.maven.eclipse;

02.

03.import org.apache.commons.lang.StringUtils;

04.

05.public class Sample {

06.

07.public static String reverse(String s) {

08.return StringUtils.reverse(s);

09.}

10.

11.public static void main(String[] args) {

12.for (int i = 0; i < args.length; i++) {

13.System.out.println(reverse(args[i]));

14.}

15.}

16.

17.}

**SampleTest.java**

01.package net.kjunine.maven.eclipse;

02.

03.import static org.junit.Assert.\*;

04.

05.import org.junit.Test;

06.

07.public class SampleTest {

08.

09.@Test

10.public void testReverse() {

11.String actual = Sample.reverse("hello");

12.String expected = "olleh";

13.assertEquals(expected, actual);

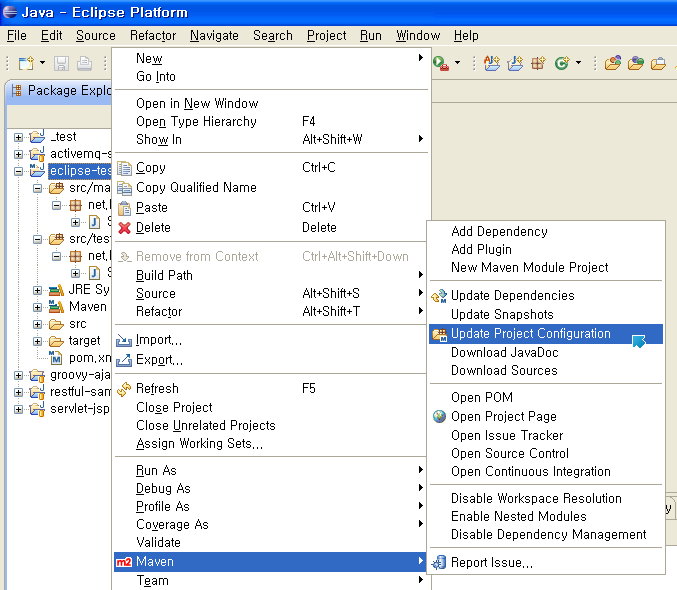
14.}

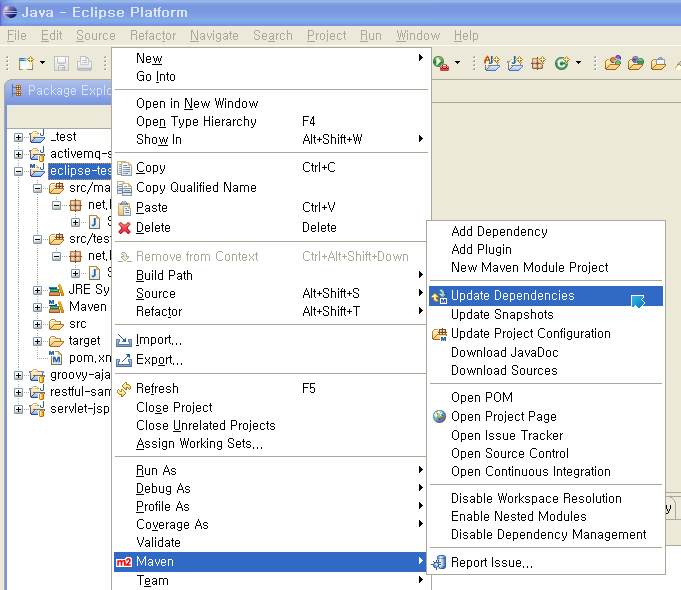
15.

16.}

이전 예제에서 사용한 예제 그대로이다. 다른 예제를 만들기가 귀차나서.. ㅠ\_ㅠ

에러가 생기면 POM 에디터에서 수정한 사항이 프로젝트에 반영이 제대로 안 된 것이니 프로젝트에서 마우스 오른쪽 클릭 후  나오는 메뉴에서 Maven > Update Dependencies 와 Update Project Configuration 을 실행하여 반영시키자.

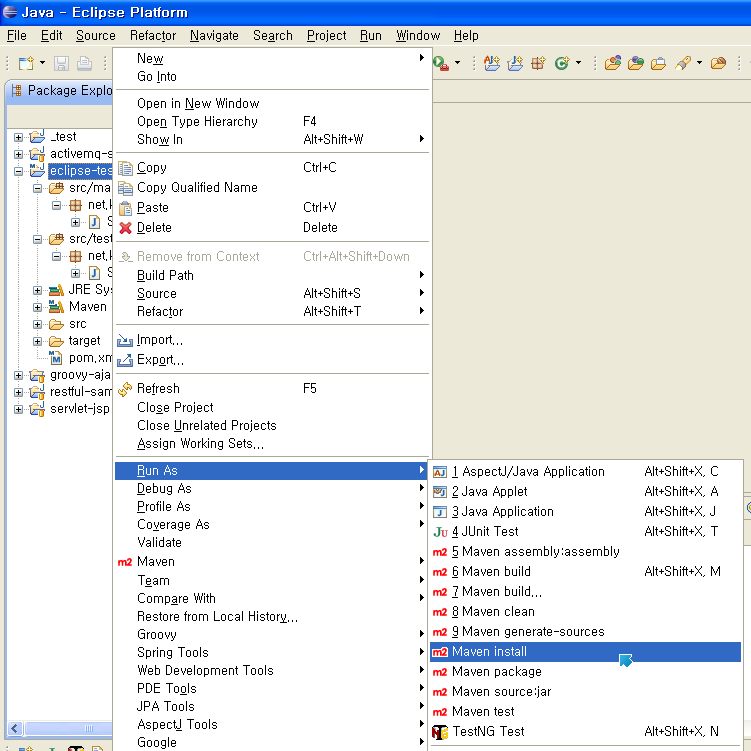




**5. 프로젝트를 테스트하고 빌드한다.**

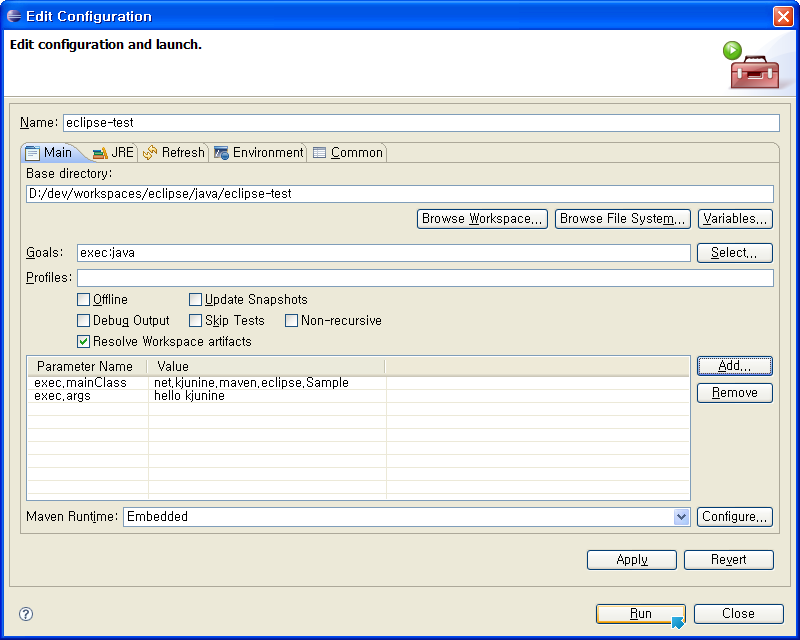
프로젝트에서 마우스 오른쪽 클릭 후 나오는 메뉴에서 Run As > Maven 으로 시작하는 많은 메뉴 아이템들을 볼 수 있다. 콘솔에서 직접 입력해서 실행하던 메이븐 명령어를 방금 언급한 메뉴를 선택하는 것으로 쉽게 실행할 수 있다.

빌드, 테스트 후 그리고 로컬 리포지토리에 설치하려면 아래 메뉴들 중에서 'Maven install' 메뉴를 선택한다.



콘솔 창에 'BUILD SUCCESSFUL' 이라고 메시지가 뜨면 성공한 것이다.

'Maven build...' 메뉴를 선택하면 다음과 같은 팝업창이 떠서 다른 플러그인의 골을 실행할 수 있다. 다음과 같이 exec:java 골을 실행하여 위에서 만든 프로그램을 실행하여 보자.



콘솔 창에 'BUILD SUCCESSFUL' 이라는 메시지와 함께 결과가 제대로 나오는 것을 볼 수 있을 것이다.

**자바 프로젝트는 항상 메이븐과 함께 하자.**

이제 메이븐의 기초를 배웠으니 자바 프로젝트를 만들 때는 항상 메이븐으로 시작하자. 이클립스 플러그인의 도움을 받으면 몇번의 클릭으로 거의 모든 것을 할 수 있으니 어려워하지 않아도 될 것 같다.

이것으로 첫 시리즈인 메이븐 시작하기 시리즈를 마친다. 자세하게 쓸려고 노력했지만 의미가 충분히 전달되지 않은 것도 많은 것 같다. 잘 이해가 안 가거나 잘 못된 부분은 댓글로 남겨주길 바라고, 더 자세히 알고싶으면[메이븐 홈페이지](http://maven.apache.org/)나 [메이븐 책](http://www.sonatype.com/products/maven/documentation/book-defguide)을 참고하길 바란다.

앞으로는 더 높은 수준의 포스트를 기대하며..