프레임워크 프로그래밍

2. Maven

2020. 2. 15

컴퓨터공학

허훈식

Learning Objectives

1. Maven의 기본 정의를 이해한다.
2. Maven의 프로젝트 관리 방법을 이해한다.
3. Maven의 빌드 자동화 기능을 이해한다.
4. Maven 프로젝트를 만들어본다.

2.1 Maven 소개

➢ Maven 정의
- 프로젝트를 관리하는 도구
- 빌드 자동화 기능과 프로젝트 관리 기능을 제공

➢ 프로젝트(라이브러리) 관리
- pom.xml 파일을 이용하여 프로젝트 관련된 jar 파일을 다운로드하고 관리
- 프로젝트 산출물을 일관된 구조로 관리

➢ 빌드 자동화
- 빌드 작업들을 간단하고 쉽게 그리고 일관성 있게 수행할 수 있는 통합 환경을 제공
- 빌드는 소스 코드 파일을 실행 코드로 변환하여 배포하는 과정

2.2 프로젝트 관리 기능

➢ 프로젝트 관리 기능

- 일반 개발자 프로젝트 관리 설정들을 메이븐이 미리 정의한 설정들로 대체한다.

- 정형화된 프로젝트 디렉토리 구조 관리 (pom.xml)
  - Convention over Configuration(CoC) 패러다임을 따름 : 설정보다는 규범

- 의존성 관리기능 : 편리한 라이브러리 관리 기능 (pom.xml, Repositorty)
  - 프로젝트 빌드에 필요한 라이브러리, 플러그인을 개발자 PC에 자동으로 다운로드
  - 중앙 저장소(http://mvnrepository.com/)
    (http://search.maven.org/)
  - 로컬 저장소 (USER_HOME\.m2\repository)

- 빌드 프로세스를 관리 (pom.xml)
  - 플러그인 설정을 통해 빌드 자동화

2.2 프로젝트 관리 기능

Apache Maven Project
http://maven.apache.org/

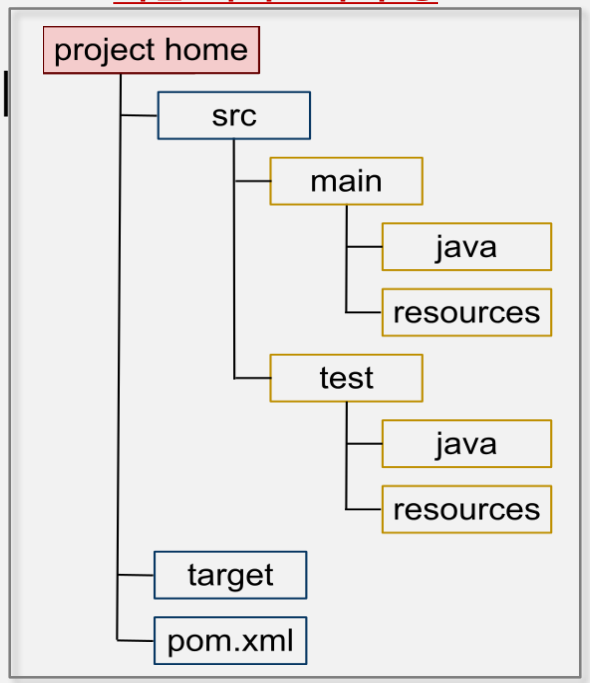
➢ 프로젝트 디렉토리 기본 설정 (pom.xml)

- src/main/resources : 클래스 패스로 사용되는 디렉토리

  • 주요 설정 정보 (xml, properties 파일 위치)

Item	Default
source code	${basedir}/src/main/java
resources	${basedir}/src/main/resources
Tests	${basedir}/src/test
distributable JAR	${basedir}/target
Complied byte code	${basedir}/target/classes

기본 디렉토리 구성



웹 디렉토리 구성

자바 소스와 리소스 디렉토리는 메이븐 기본 디렉토리를 유지하고 웹 자원을 관리하는 별도의 src/main/webapp 디렉토리를 사용한다. 이 값은 maven-war-plugin의 warSourceDirectory에 설정되어 있음.

2.2 프로젝트 관리 기능

➢ 의존관계(라이브러리) 설정 (pom.xml)
- 프로젝트 당 한 개의 pom.xml 파일 관리
- 최상위 엘리먼트(root element) : project
- 3개의 필수 필드를 가짐(groupId: artifactId: version)
  - groupId : 프로젝트 조직 고유 도메인 예) org.kpu
  - artifactId : 프로젝트 명 예) myhomework
  - version : 프로젝트 버전
- 프로젝트 의존관계의 라이브러리 관리 : dependency

➢ 프로젝트 빌드 설정
- 프로젝트 기본 정보, 저장소, 프로퍼티, 디렉토리 구조
- 플러그인(plugins)
- 골(goals)

2.2 프로젝트 관리 기능

➢ POM(Project Object Model).xml

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

    <groupId>org.kpu</groupId>
    <artifactId>myhomework /artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>maven</name>
    <url>http://maven.apache.org</url>

    <properties>
       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
      </dependency>
    </dependencies>
</project>
```

모든 POM은 Super POM(기본 POM)으로부터 상속받는다.
- Super POM은 기본 설정 정보를 포함한다.
- Super POM을 수정하려면 POM에서 오버라이드한다.

2.2 프로젝트 관리 기능

➢ Maven Repository

- 메이븐 저장소는 프로젝트에 사용되는 프로젝트 jar 파일, 라이브러리 jar 파일들이 위치하며 3가지 타입이 있다.

1) 지역(local) 저장소 : 로컬 저장소는 메이븐을 빌드할 때 다운로드하는 라이브러리, 플러그인을 관리하는 저장소이다. 기본 로컬 저장소는 USER_HOME/.m2/repository 디렉토리이다.

2) 중앙(central) 저장소 : 오픈 소스 라이브러리, 메이븐 플러그인, 메이븐 아키타입을 관리하는 저장소이다.

3) 원격(remote) 저장소 : 메이븐 기반으로 프로젝트를 진행하는 경우 프로젝트에 필요한 모든 라이브러리가 메이븐 중앙 저장소에 존재하는 것이 아니다. 이처럼 중앙 저장소에 존재하지 않는 라이브러리를 관리하기 위하여 별도의 메이븐 저장소를 설치해 관리하는 것이 가능하다.

2.2 프로젝트 관리 기능

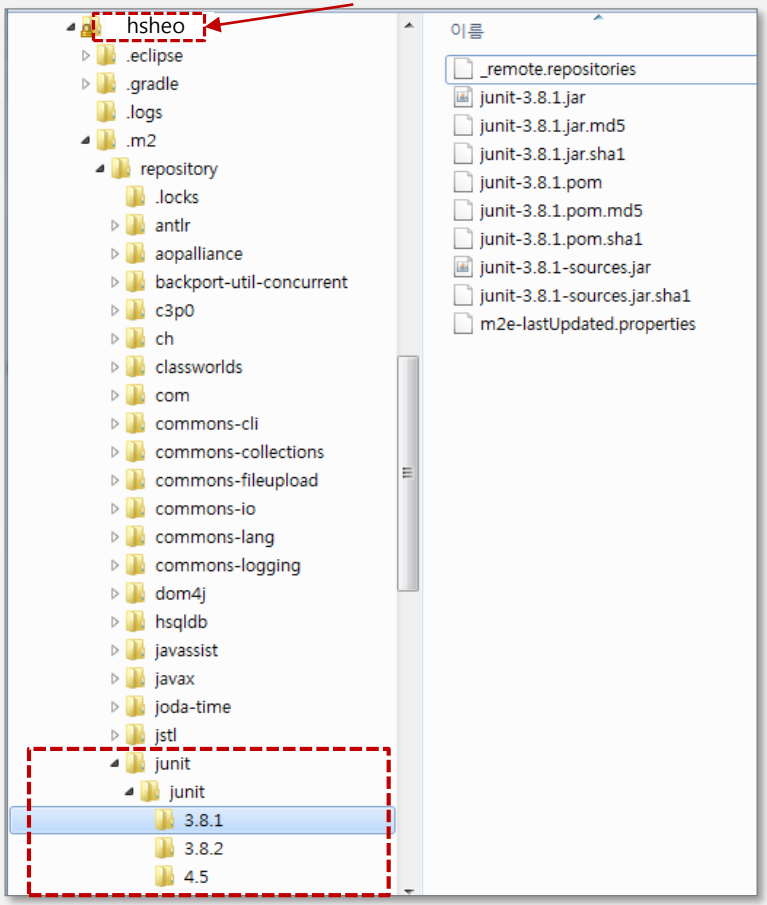
Apache Maven Project
http://maven.apache.org/

- 메이븐 지역 저장소 구조
  - 다운로드한 라이브러리를 지역 저장소에서 관리할 때의 디렉토리 구조는 그림과 같다
  - groupId와 artifactId 아래에 각 버전에 따른 라이브러리를 관리한다.

pom.xml

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
 </dependencies>
```

사용자 홈 디렉토리



2.2 프로젝트 관리 기능

➢ 메이븐 의존성 검색 절차(Maven Dependency Search Sequence)

- 메이븐의 의존 라이브러리 검색 동작 절차는 다음과 같다.

1단계 : 지역 저장소를 검색한다. 찾는 라이브러리가 없을 경우 2단계로 넘어간다.
2단계 : 중앙 저장소를 검색한다. 찾은 라이브러리는 지역 저장소에 저장한다. 만약 찾는 라이브러리가 없을 경우 3단계로 넘어간다. 만약 원격 저장소가 존재하지 않을 경우 에러를 발생시키고 종료한다.
3단계 : 원격 저장소를 검색한다. 찾은 라이브러리는 지역 저장소에 저장한다. 만약 찾는 라이브러리가 없을 경우 에러를 발생시키고 종료한다.

pom.xml

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

중앙저장소에서 찾을 경우
groupId/artifactId/version에 존재하는
파일명: ${artifactId}-${version}.jar

최종적으로
http://repo1.maven.org/maven2/junit/junit/3.8/junit-3.8.jar 파일을 다운로드한다.

2.2 프로젝트 관리 기능

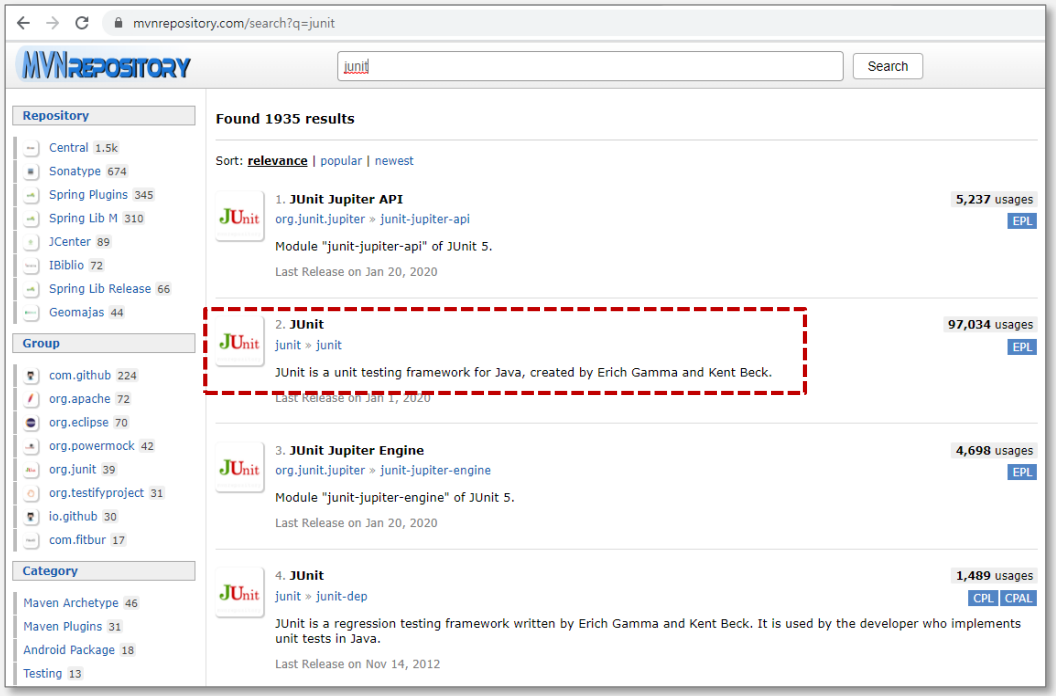
Apache Maven Project
http://maven.apache.org/

➢ 메이븐 중앙 저장소 라이브러리 검색

- 메이븐 중앙 저장소에서 관리하는 라이브러리의 수는 한두 개가 아니다. 한 프로젝트에서 사용하는 라이브러리의 수가 많아서 위와 같은 방법으로 라이브러리를 추가하는 데는 한계가 있다. -> 검색 필요

- 메이븐 중앙 저장소에서 관리하는 라이브러리를 검색 :

http://mvnrepository.com/



2.2 프로젝트 관리 기능

Apache Maven Project
http://maven.apache.org/

➢ 의존 라이브러리 적용 스코프

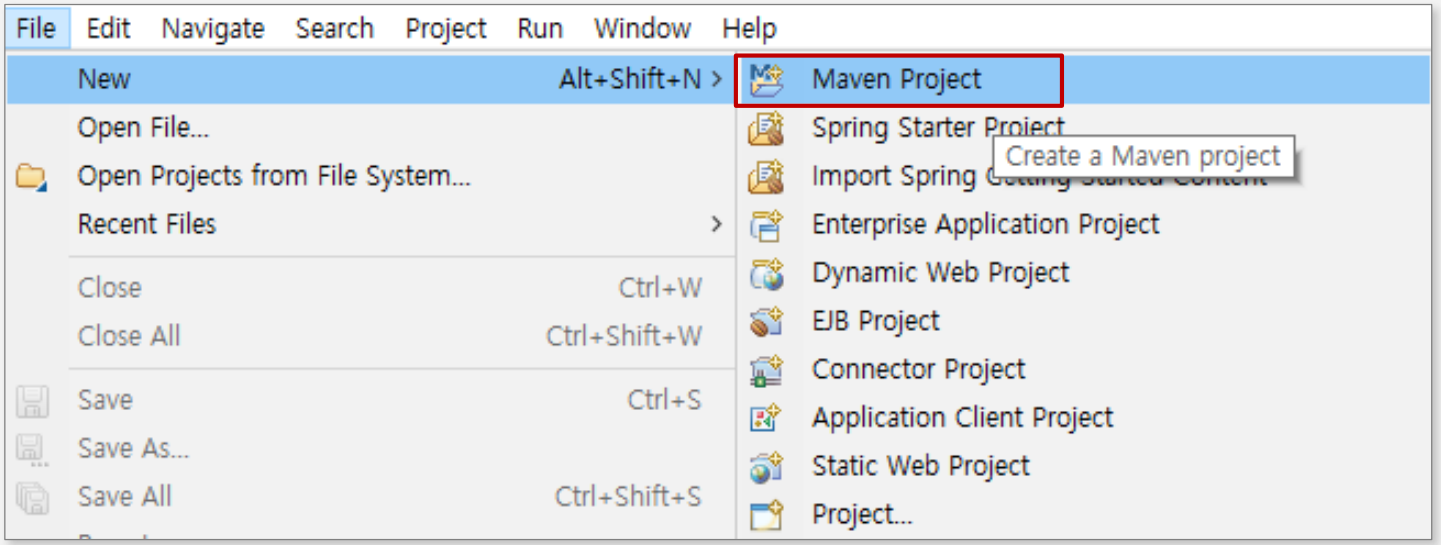
- 의존 라이브러리를 적용할 수 있는 시점을 제한할 수 있다. – Scope 설정

scope	Description
compile	스코프를 설정할지 않았을 때의 기본 스코프
provided	컴파일시에는 직접 의존성을 참조하고 런타임시에는 다른 환경에서 의존성을 제공받는다.(해당 컨테이너의 서블릿 API)
runtime	컴파일 시에는 사용되지 않지만 애플리케이션을 실행할 때 사용되는 라이브러리일 경우 설정한다.
test	테스트하는 시점에만 사용하는 라이브러리에 대한 스코프를 설정할 때 사용한다. (예: JUnit)
system	provided와 비슷하다. 단지 우리가 직접 jar 파일을 제공해야 한다.
import	import(메이븐 2.0.9 이후 버전부터 사용 가능): 다른 POM 설정 파일에 정의되어 있는 의존 관계 설정을 현재 프로젝트로 가져온다. 이 범위는 <dependencyManagement/> 엘리먼트에서만 사용 가능하다.

2.3 Maven 프로젝트 실습

Apache Maven Project
http://maven.apache.org/

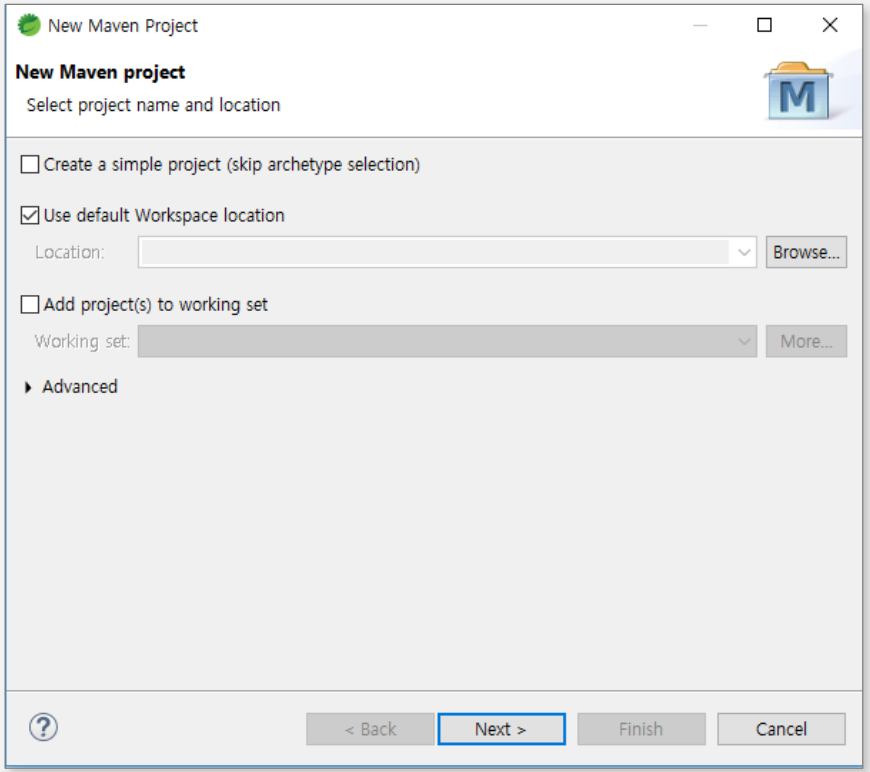
➢ [기본실습] 메이븐 프로젝트 생성 (mymaven)

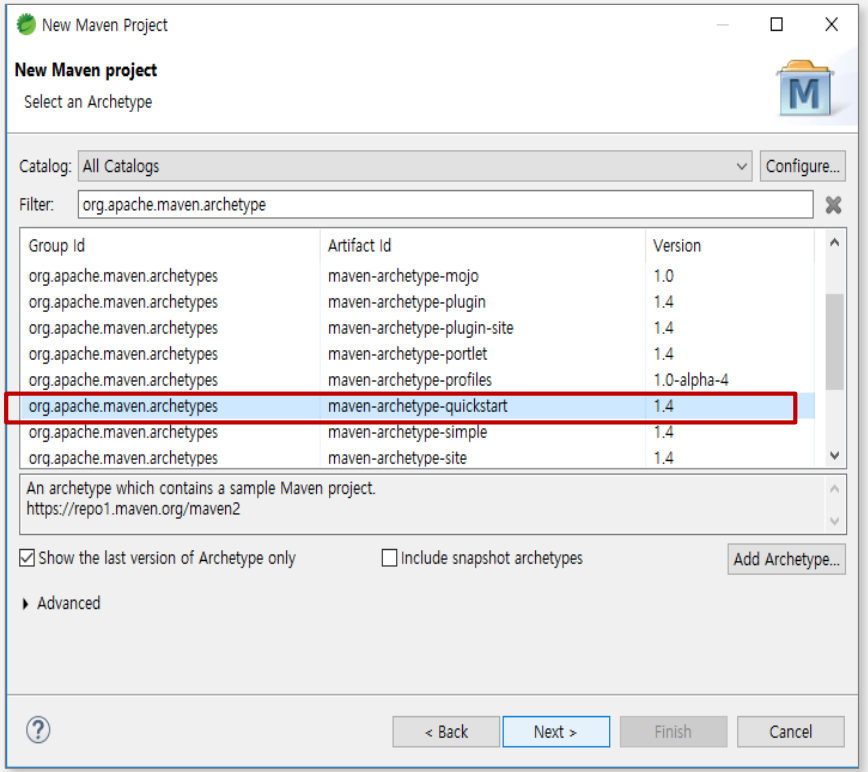


2.3 Maven 프로젝트 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트 생성 (mymaven)
- 프로젝트 디렉토리 위치, 메이븐 프로젝트 템플릿 설정

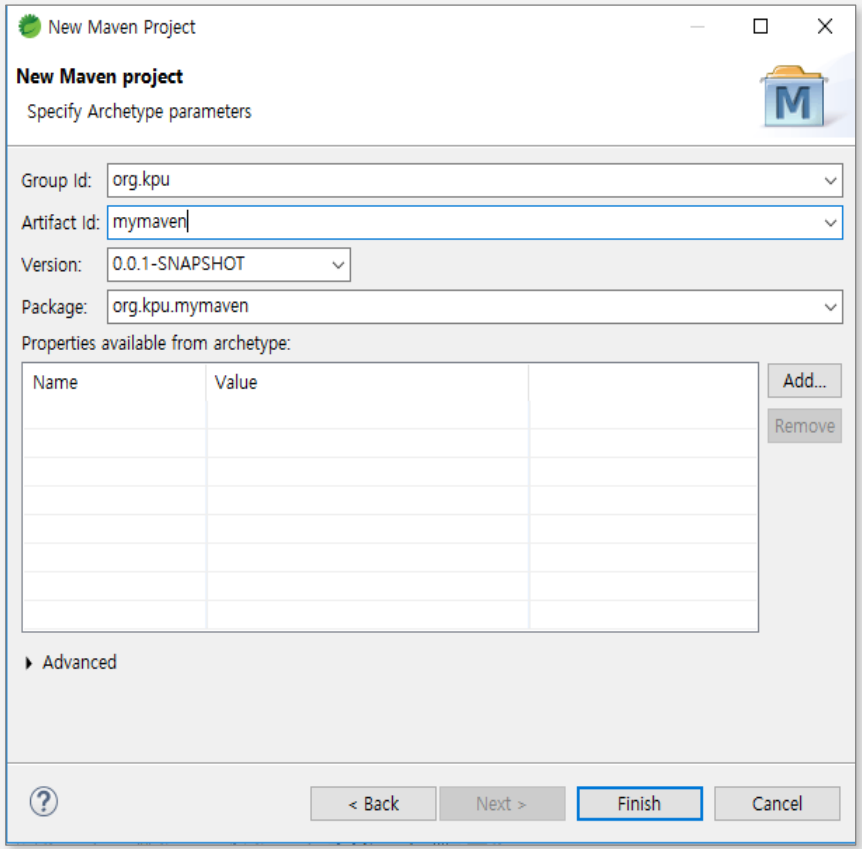


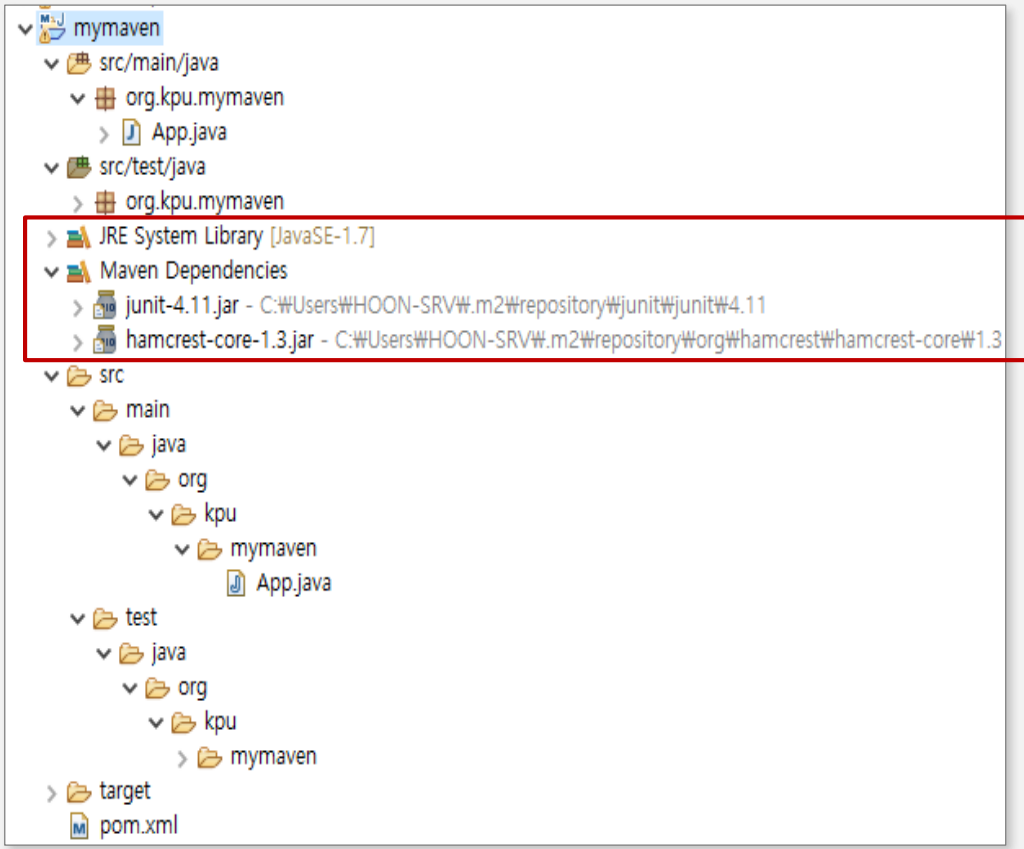


2.3 Maven 프로젝트 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트 생성 (mymaven)

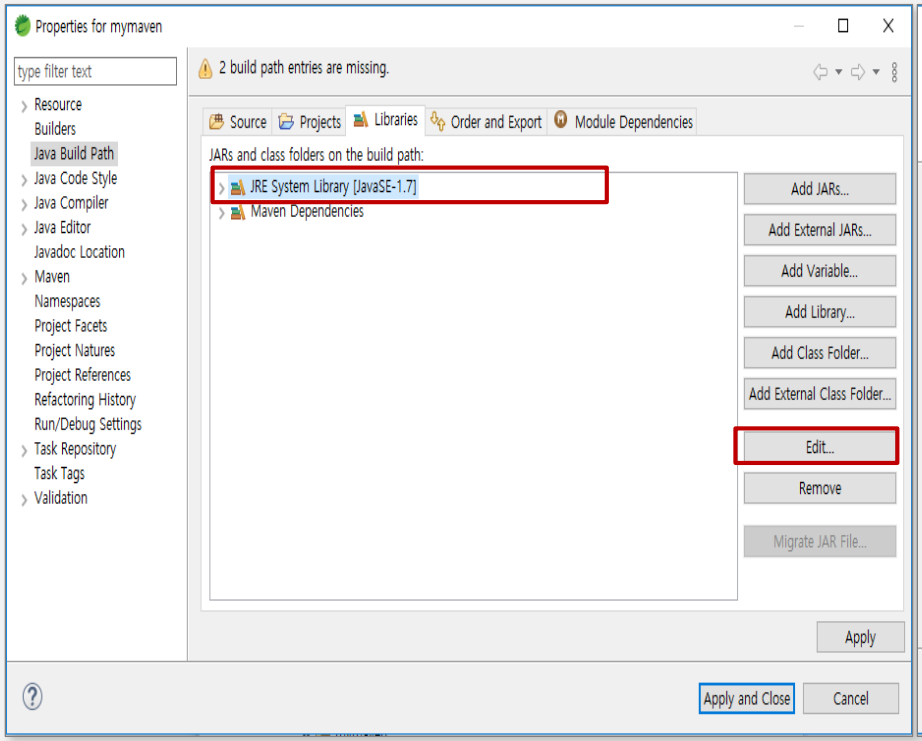


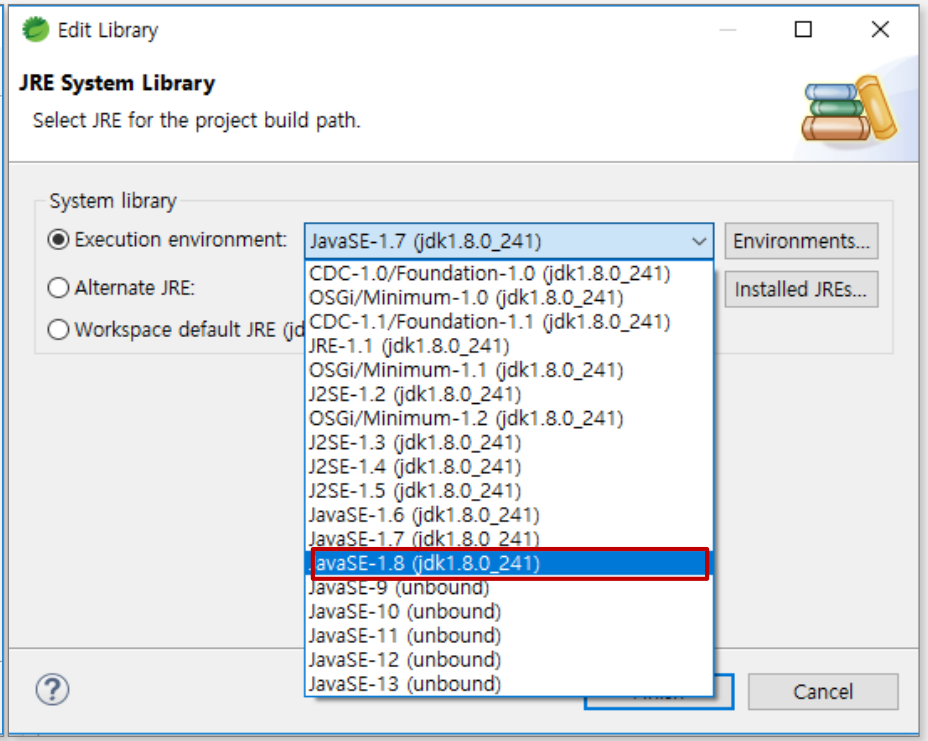


2.3 Maven 프로젝트 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트 – Java Build Path 확인(mymaven)
- jdk1.8.0.241로 변경

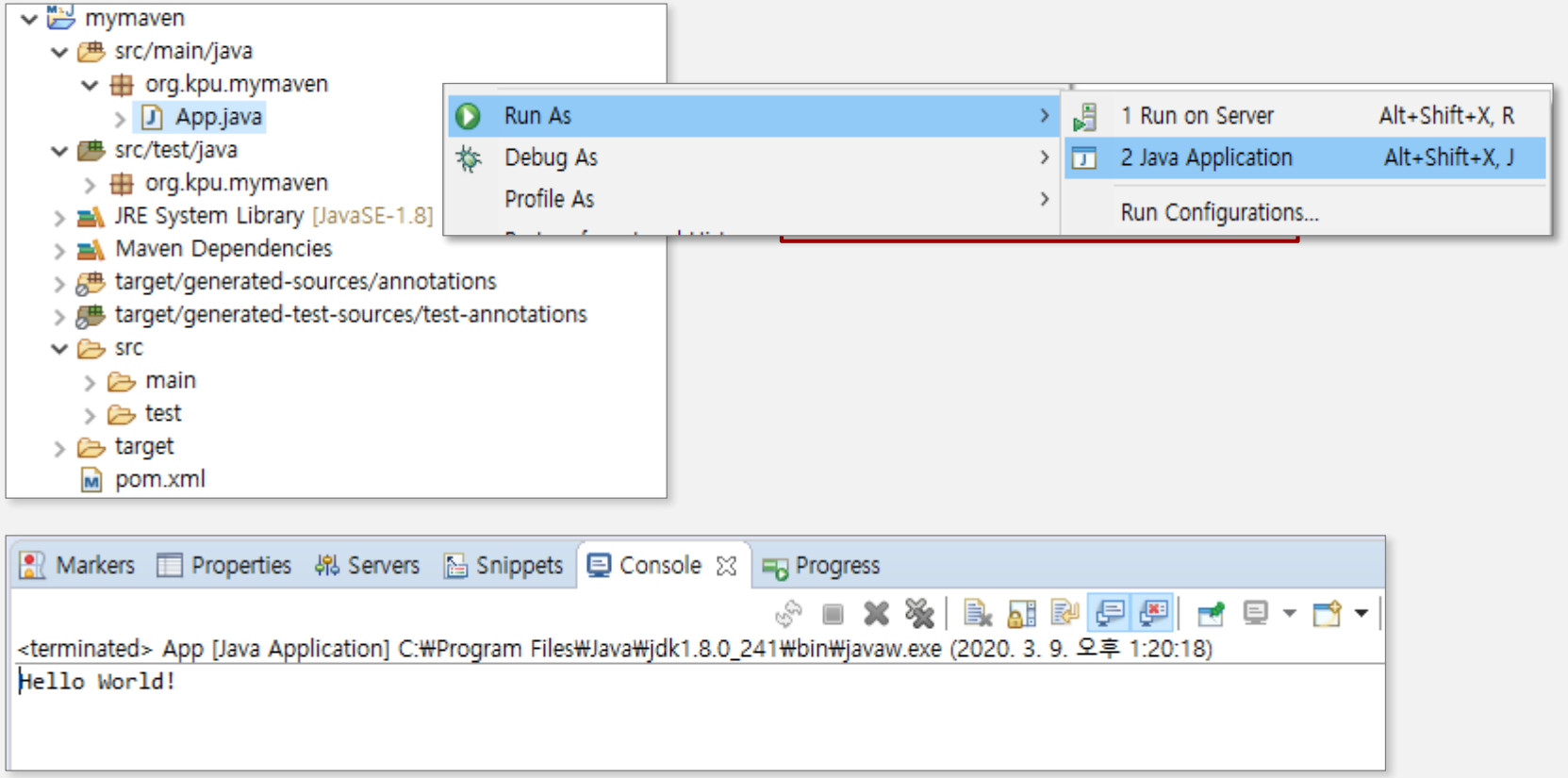




2.3 Maven 프로젝트 실습

Apache Maven Project
http://maven.apache.org/

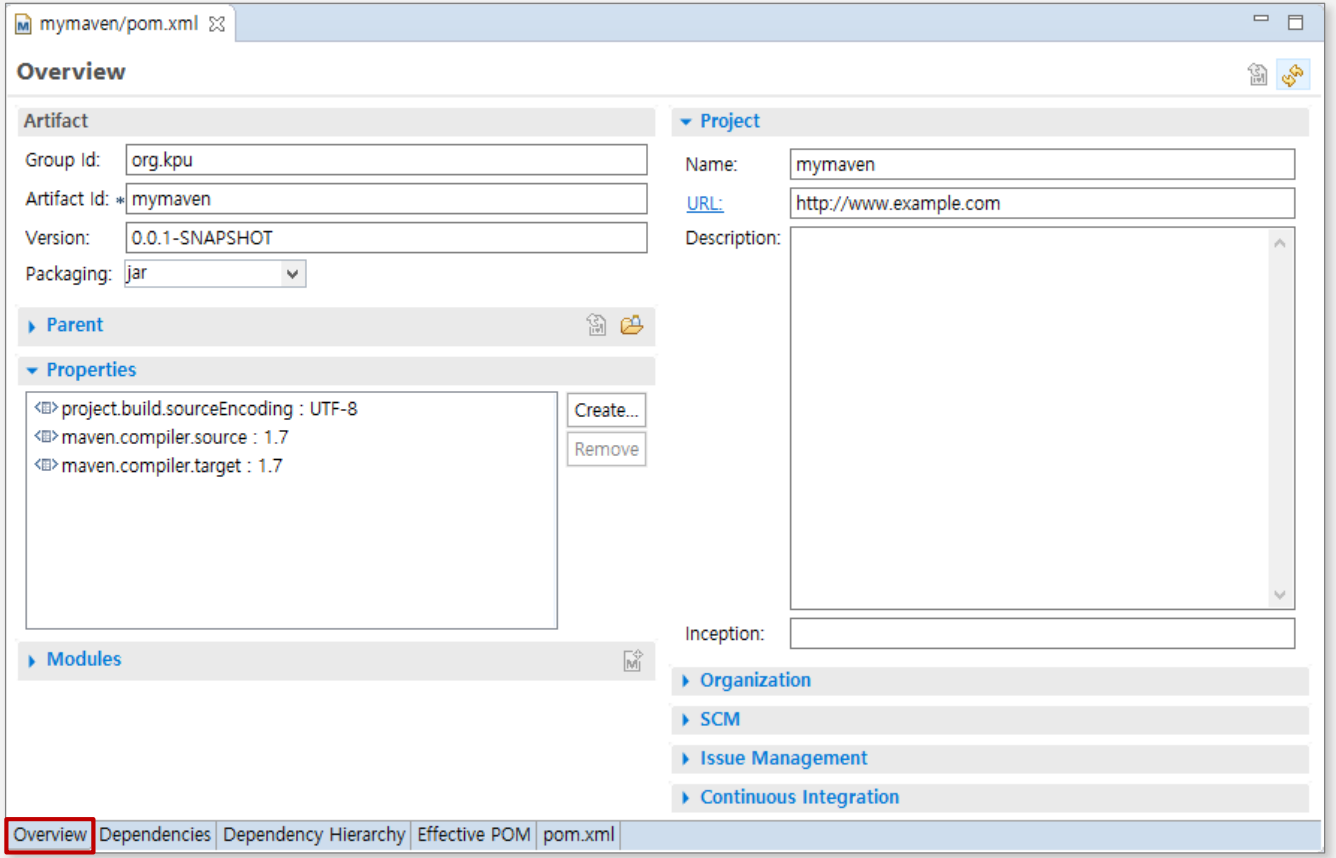
➢ [기본실습] 메이븐 프로젝트 실행 (mymaven)



2.3 Maven 프로젝트 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트(mymaven) pom.xml



2.3 Maven 프로젝트 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트(mymaven) pom.xml
- Junit 버전 변경(현재 4.1.1)

mymaven/pom.xml

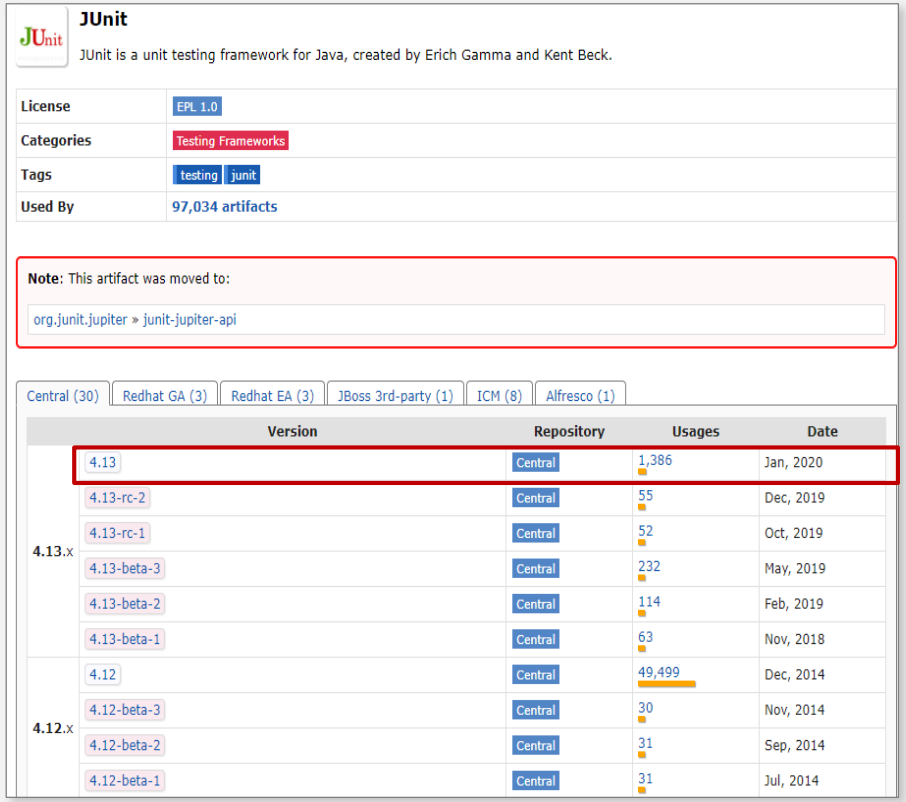
```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5    <modelVersion>4.0.0</modelVersion>
6
7    <groupId>org.kpu</groupId>
8    <artifactId>mymaven</artifactId>
9    <version>0.0.1-SNAPSHOT</version>
10
11   <name>mymaven</name>
12   <!-- FIXME change it to the project's website -->
13   <url>http://www.example.com</url>
14
15   <properties>
16     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17     <maven.compiler.source>1.7</maven.compiler.source>
18     <maven.compiler.target>1.7</maven.compiler.target>
19   </properties>
20
21   <dependencies>
22     <dependency>
23       <groupId>junit</groupId>
24       <artifactId>junit</artifactId>
25       <version>4.11</version>
26       <scope>test</scope>
27     </dependency>
28   </dependencies>
29
30   <build>
31     <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
32       <plugins>
33         <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean_Lifecycle --
```

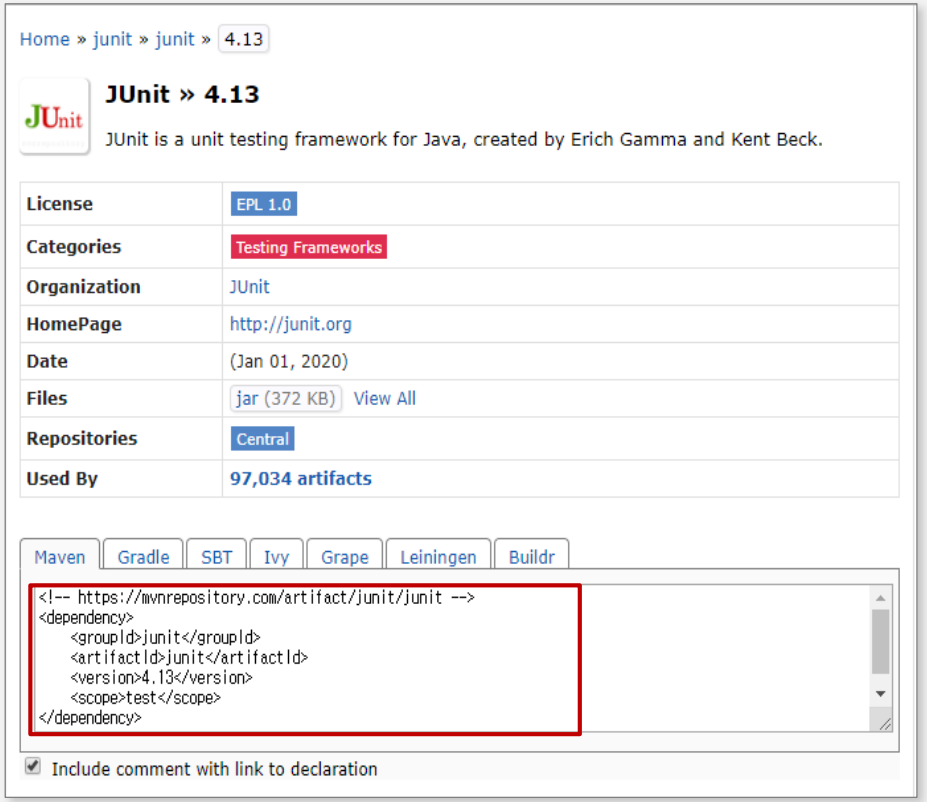
Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

2.3 Maven 프로젝트 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트(mymaven) : Junit 최신버전 검색하기
- http://mvnrepository.com/ 검색



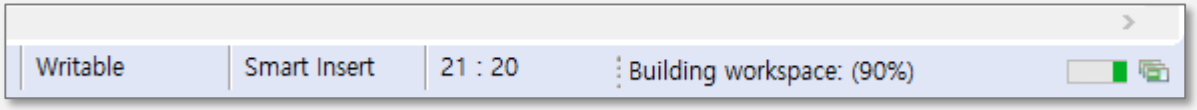


2.3 Maven 프로젝트 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트(mymaven) : Junit 버전 변경하기

- 4.13 로 변경 후 저장 : 자동 다운로드됨



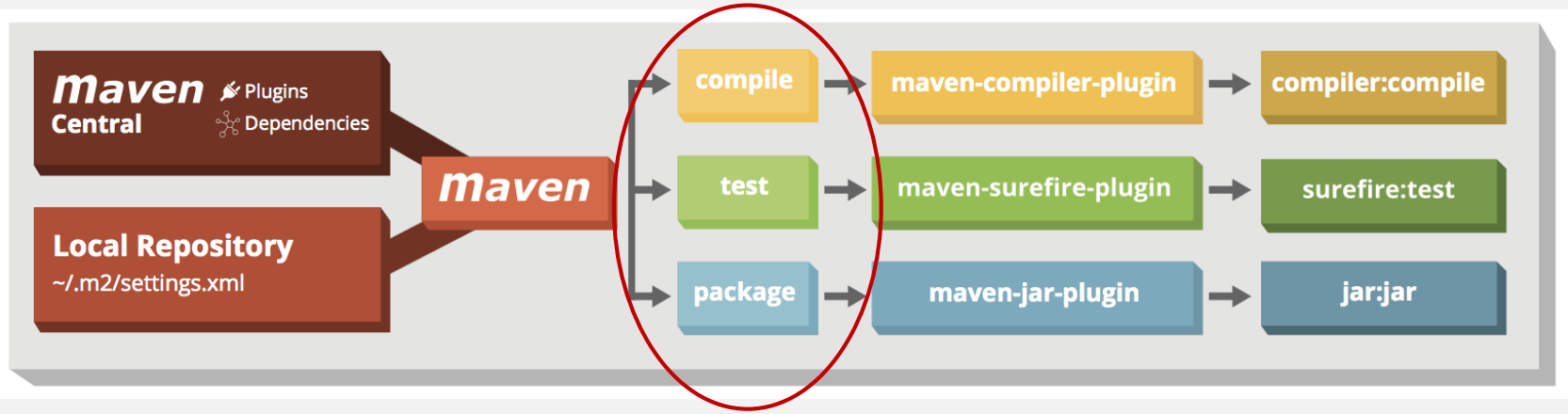
- 지역 저장소에 junit-4.13.jar 확인

2.4 빌드 자동화 기능

Apache Maven Project
http://maven.apache.org/

➢ 메이븐은 소프트웨어 빌드를 위한 공통 인터페이스를 제공하는 프레임워크
- 플러그인 설정을 통해 기능을 위임

- 빌드 단계(컴파일, 테스트, 패키징, 배포)들을 빌드 라이프 사이클이라고 한다.
- 각 빌드 단계에서 수행되는 작업을 골(Goal)이라고 한다.
- 실제 골은 그 단계에 연결된 플러그인(Plugin)에 의해 실행된다.



2.4 빌드 자동화 기능

➢ Maven 빌드 라이프사이클(lifecycle)

- Maven은 기본, clean, site 3개의 라이프사이클이 있다.
- 기본 라이프사이클은 여러 단계의 페이즈(Phase) 로 나뉘어져 있으며,
  각 페이즈는 의존관계를 가진다. compile, test, package deploy 순서로 진행된다.

- clean 라이프사이클은 clean 페이즈를 이용하여 이전 빌드에서 생성된 모든 파일(target 디렉토리)들을 삭제한다.

- site 라이프사이클은 site, site-deploy 페이즈를 이용하여 생성된 문서들을 대상 사이트에 배포한다.

2.4 빌드 자동화 기능

Apache Maven Project
http://maven.apache.org/

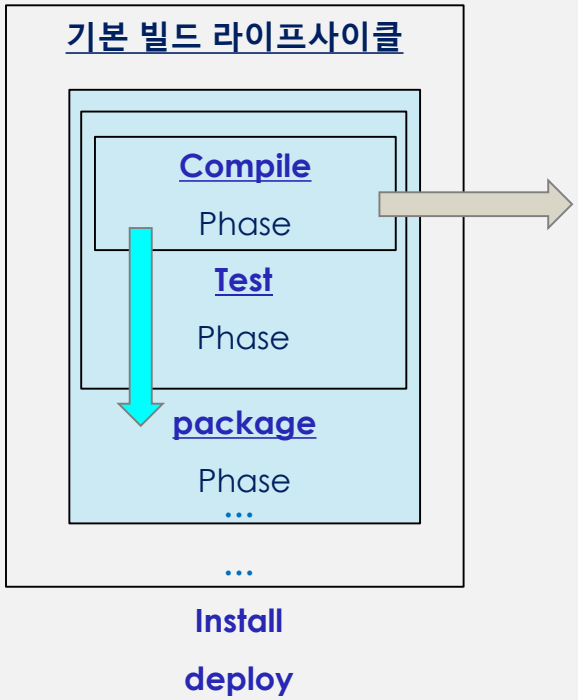
➢ Maven 빌드 라이프사이클(lifecycle)
- 기본 빌드 라이프사이클 일부 단계

기본 Phase	Description
리소스 준비(resource)	리소스가 준비되어 복사된다.
컴파일(compile)	소스 코드를 컴파일한다.
테스트(test)	Junit과 같은 테스트 프레임워크로 단위 테스트한다.
패키지(package)	pom.xml의 <packaging /> 엘리먼트 값(jar, war, ear 등)에 따라 압축한다.
인스톨(install)	패키지를 로컬 저장소에 배포한다
배포(deploy)	원격 메이븐 저장소에 압축한 파일을 배포한다

2.4 빌드 자동화 기능

Apache Maven Project
http://maven.apache.org/

- Maven Phase & Goal
  - 빌드 라이프사이클은 하나 이상의 골을 수행하는 페이지(Phase)들로 구성.
  - 각 페이즈 별로 플러그인이 작업을 수행한다. 이 작업을 Goal이라고 한다.




```
<plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.0</version>
    <executions>
      <execution>
        <id>default-compile</id>
        <phase>compile</phase>
        <goals>
          <goal>compile</goal>
        </goals>
      </execution>
      <execution>
        <id>default-testCompile</id>
        <phase>test-compile</phase>
        <goals>
          <goal>testCompile</goal>
        </goals>
      </execution>
    </executions>
</plugin>
```

pom.xml

2.4 빌드 자동화 기능

➢ Maven Plugin
- 대부분의 기능들을 플러그인을 통해서 제공된다.
- 예) maven-compiler-plugin, maven-clean-plugin, maven-surefire-plugin

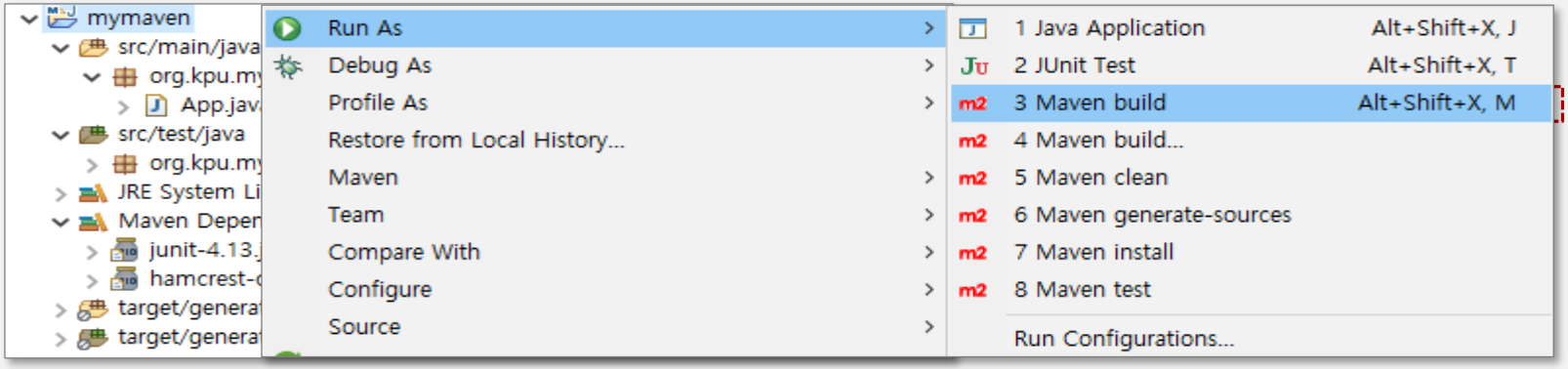
Plugin	Description
clean	빌드 이후에 target 디렉토리를 삭제한다.
compiler	자바 소스 파일을 컴파일한다.
surefire	JUnit 단위 테스트를 실행한다. 리포트를 생성한다.
jar	현재 프로젝트로부터 JAR file 를 생성한다.
war	현재 프로젝트로부터 WAR file 를 생성한다.
javadoc	프로젝트의 Javadoc 문서를 생성한다.
antrun	빌드 페이지 단계에서 특정 ant 작업을 실행한다.

2.5 Maven 빌드 기능 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트(mymaven) : Goal 설정하기

- 패키지 페이지 골 설정 - jar:jar



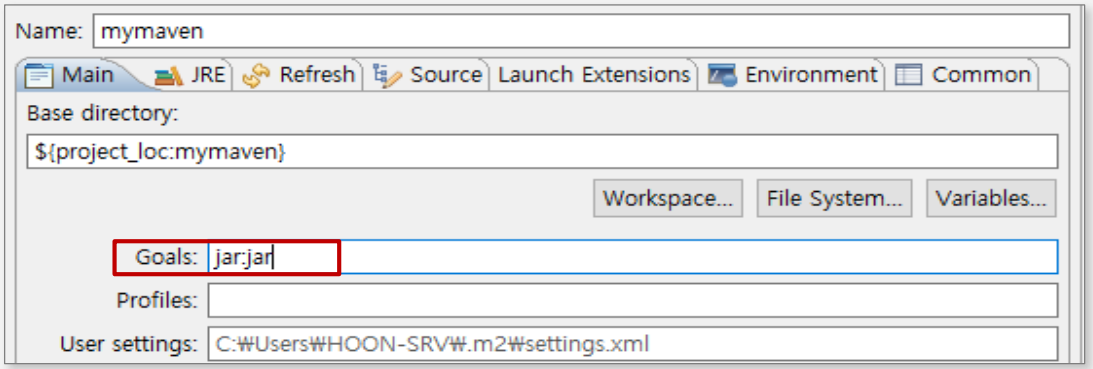
GOAL

compiler:compile

surefire:test

jar:jar

[플러그인 prefix:골] 형식으로 표현

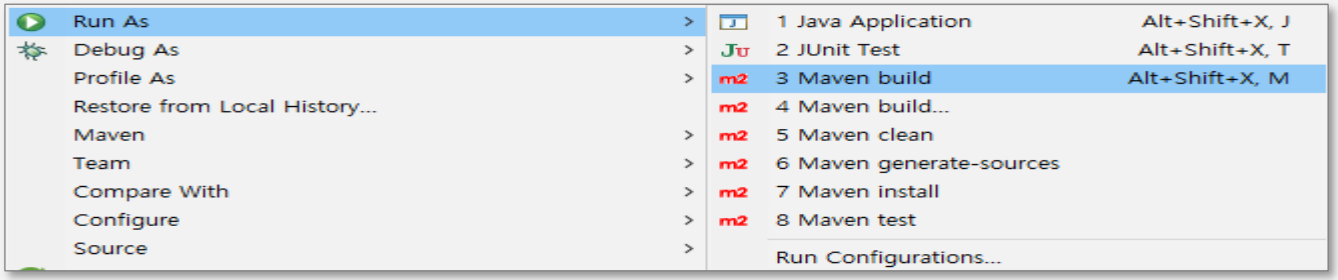


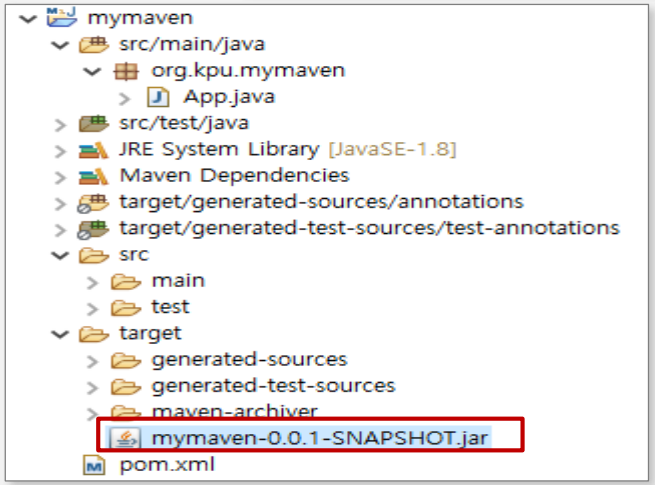
2.5 Maven 빌드 기능 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트(mymaven) : Goal 실행하기

- jar:jar Goal 실행 후 refresh -> target 디렉토리 확인





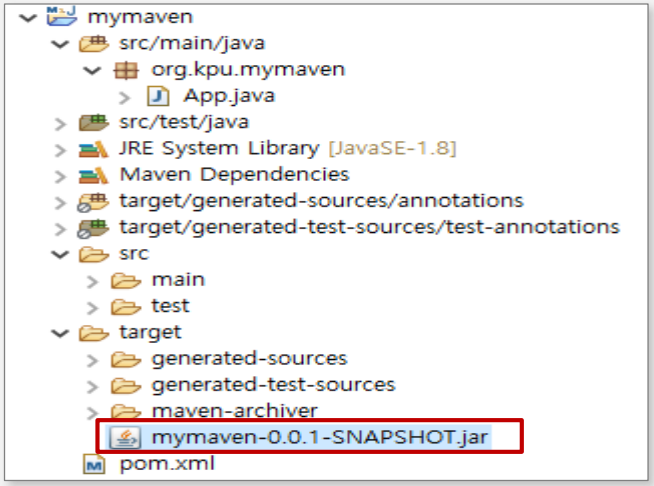

```
<terminated> mymaven [Maven Build] C:₩Program Files₩Java₩jdk1.8.0_241₩bin₩javaw.exe (2020. 3. 9. 오후 1:58:12)
[INFO] Scanning for projects...
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-
[INFO]
[INFO] -------------------------< org.kpu:mymaven >-------------------------
[INFO] Building mymaven 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-jar-plugin:3.0.2:jar (default-cli) @ mymaven ---
[INFO] Building jar: C:\Dev\workspace-sts-451\mymaven\target\mymaven-0.0.1-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.676 s
[INFO] Finished at: 2020-03-09T13:58:16+09:00
[INFO] ------------------------------------------------------------------------
```

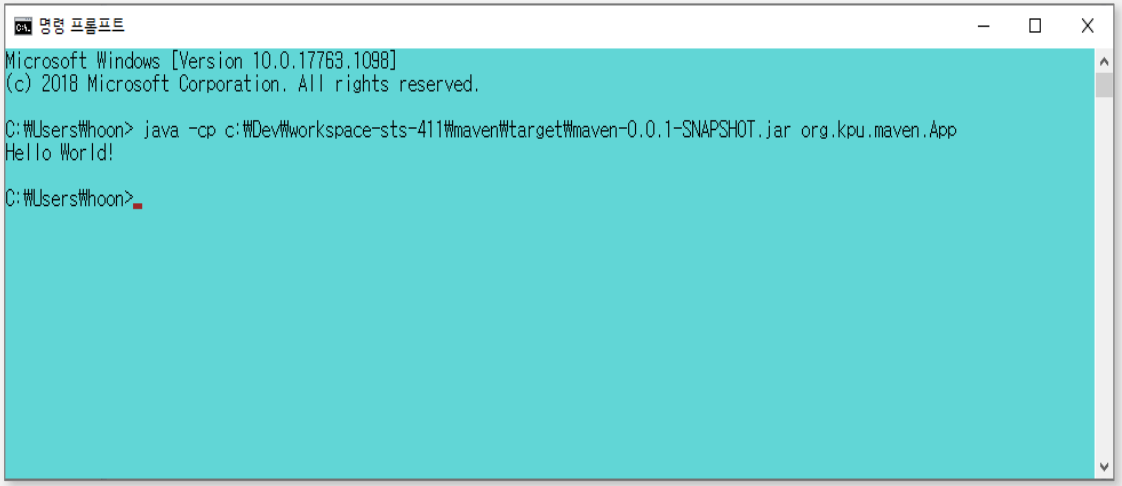
2.5 Maven 빌드 기능 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트(mymaven)
- maven-0.0.1-SNAPSHOT.jar 실행해보자.

```
java –cp c:\Dev\workspace-sts-451\maven\target\maven-0.0.1-SNAPSHOT.jar org.kpu.maven.App
```



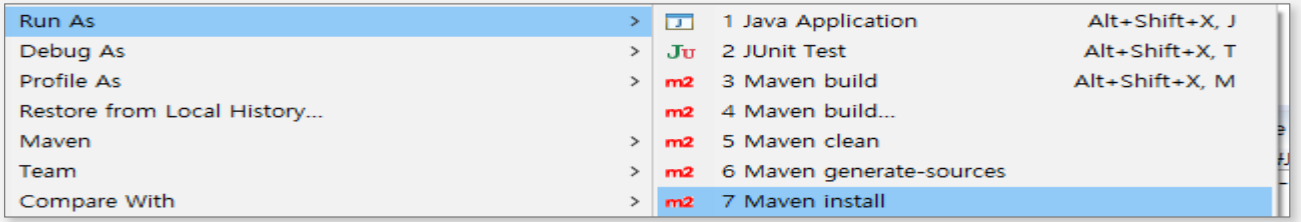


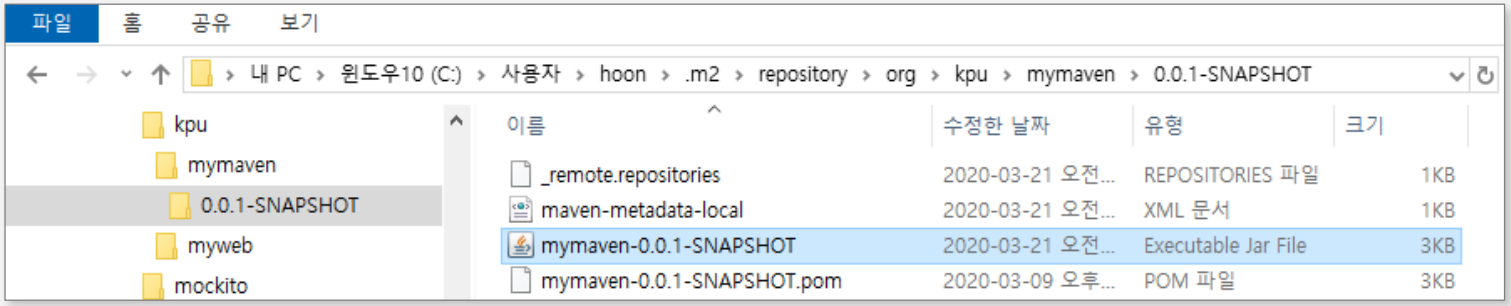
2.5 Maven 빌드 기능 실습

Apache Maven Project
http://maven.apache.org/

➢ [기본실습] 메이븐 프로젝트(mymaven) : Goal 실행하기

- Maven install 실행 후 지역 저장소 디렉토리에 배포된 파일 확인





- Maven clean 실행 후 refresh -> 생성된 파일들 모두 삭제한다.